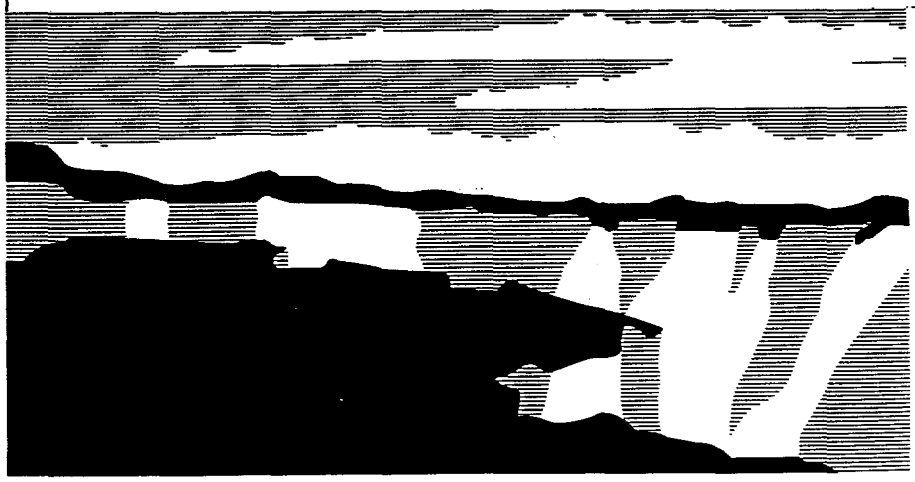
LA-UR- 99-460

Title: COMPOSING SIMULATIONS USING PERSISTENT SOFTWARE COMPONENTS

CONF-990403--

Author(s): J. V. Holland, R. e.Michelsen, D. R. Powell, S. C. Upton, D. r. Thompson

Submitted to: Advanced Simulation Technologies Conf.
San Diego, CA
4/11-15/99



Los Alamos
NATIONAL LABORATORY

Composing Simulations Using Persistent Software Components

Joe V. Holland, Randy E. Michelsen, Dennis R. Powell,
Stephen C. Upton, and David R. Thompson

Los Alamos National Laboratory
P.O. Box 1663
Los Alamos, NM 87545

Keywords: agent, components, composition, framework, simulation

Abstract: *The traditional process for developing large-scale simulations is cumbersome, time consuming, costly, and in some cases, inadequate. The topics of software components and component-based software engineering are being explored by software professionals in academic and industrial settings. A component is a well-delineated, relatively independent, and replaceable part of a software system that performs a specific function. Many researchers have addressed the potential to derive a component-based approach to simulations in general, and a few have focused on military simulations in particular. In a component-based approach, functional or logical blocks of the simulation entities are represented as coherent collections of components satisfying explicitly defined interface requirements. A simulation is a top-level aggregate comprised of a collection of components that interact with each other in the context of a simulated environment. A component may represent a simulation artifact, an agent, or any entity that can generate events affecting itself, other simulated entities, or the state of the system. The component-based approach promotes code reuse, contributes to reducing time spent validating or verifying models, and promises to reduce the cost of development while still delivering tailored simulations specific to analysis questions. The Integrated Virtual Environment for Simulation (IVES) is a composition-centered framework to achieve this potential. IVES is a Java implementation of simulation composition concepts developed at Los Alamos National Laboratory for use in several application domains. In this paper, its use in the military domain is demonstrated via the simulation of dismounted infantry in an urban environment.*

1.0 Introduction

The Integrated Virtual Environment for Simulation (IVES) is a Java implementation of simulation composition concepts developed at Los Alamos National Laboratory for application in the simulation-based study of a variety of domains. IVES is a composition-centered framework supporting the development of discrete-event simulations based on an aggregate-component view of simulation. A simulation is viewed as a top-level aggregate comprised of a collection of components, i.e., simulation entities, that interact with each other in the context of a simulated environment. At this level, a component may represent a simulation artifact (e.g., run-time data collectors), a software agent, or any entity that can generate events affecting itself, other simulated entities, or the state of the system. Every component must satisfy a published interface that specifies its supported functions. The aggregate-component view of a simulation entity is a representation of an entity in a simulation which asserts that the entity is an aggregate of discrete software components which satisfy the interface declaration of the aggregate. One advantage of this method is that the components are relatively easy to verify and validate, as they typically represent significantly smaller functional areas compared to the aggregate. The verification and validation of the aggregation poses additional technical challenges. Because components can be expressed in many ways to achieve the published interface, the user, or composer of the aggregate, has many choices in how to satisfy representational requirements of the aggregate by selecting appropriate components, or by constructing new components.

In this paper, the authors discuss the potential impact component reuse can have on the development of simulations, review general requirements for a component-based approach, describe an infrastructure framework developed by the authors, and discuss an application of the framework to a military simulation.

1.1 Paradigm

In traditional monolithic simulations, modifying the simulation code was time-consuming and often difficult. Thus, when new capabilities were required in the simulation, a common approach was to re-parameterize existing entities to enable them to approximate the new capabilities. While unsatisfying, this was expedient. There is an apocryphal story of an Army simulation that was required to represent the effect of a battleship providing naval fire support of an Army attack. The modelers, not having a battleship representation, merely placed an armored battalion in the "ocean" and overrode the constraint that an Army unit could not be in a water obstacle. This story is representative of how legacy codes have enduring staying power, even when confronted with obvious shortcomings and inadequacies. This has been and continues to be a source of frustration for analysts who must explore new concepts in equipment, organization, doctrine, tactics, and deployment. The High Level Architecture [1] (HLA) attempts to address this issue by allowing federations of models of differing resolution and capability to interact. In this setting it is possible to augment a model with a weak representation of a given feature with a more suitable model which has the requisite capabilities. In such a federation, the weaker model would subscribe to the capabilities computed by the relatively stronger model.

From the IVES perspective, this type of problem would be addressed by replacing the components responsible for the weak representation with components that adequately address the analysis requirements. Because the functionality of the model is modularized into components, each of which must satisfy known interface requirements, the implementation details can be hidden. This enables interoperability among components that satisfy the common interface. It becomes possible to create a new component which can satisfy an alternate implementation, which in many cases, can address either representational, resolution, or fidelity deficiencies compared to another component.

1.2 Software Reuse

Components are designed to be replaceable parts within a software program. Once developed, components can be stored in repositories and maintained for reuse in other simulation compositions. When a component is selected for use from a repository, a copy of the component is supplied, thus maintaining the integrity of the repository. Both partial and full compositions can be stored in the repository and can be edited. Since the component modules are typically small, verification and validation are expected to be comparatively easier than in a large simulation code. The issue of aggregate verification and validation has yet to be fully explored.

1.3 Potential Applications

Currently, IVES has been applied to the military modeling domain to create constructive simulations of soldier agents in a simplified urban environment. Soldiers select paths through the urban environment and use the sensors assigned to them, e.g., human eyeball or thermal sights. Based on the perceptions from the sensors, soldiers may react to threats by firing their assigned weaponry. Although current capabilities are primitive, the extensibility of IVES will allow investigations of future concepts, new tactics, and analysis of alternatives. Other potential application domains include manufacturing, transportation, electronic commerce, and organizational dynamics.

1.4 Exploration of Output Space

Simulations are almost always developed to perform analysis on a domain of interest. Since analysis is closely tied to simulation development, it is natural to include analysis tools as simulation artifacts, i.e., components that aid in monitoring or analyzing simulation outputs. A current interest in constructive simulations is to study the output space of the simulation for "interesting" regions [2, 3]. The simulation can be thought of as a mapping, S, from the space of all inputs, I, to the space of all outputs, O, e.g., $S{:}I\rightarrow O$.

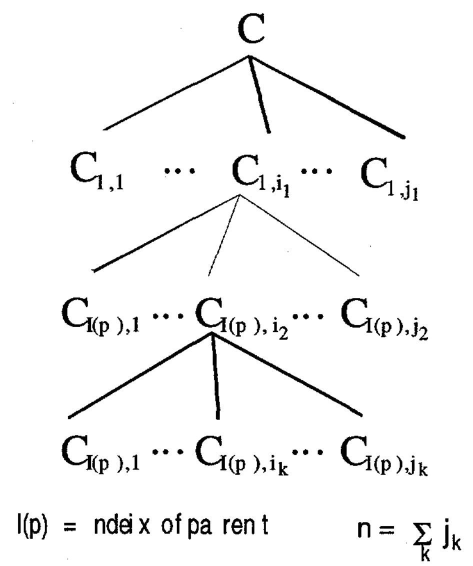
Characterizations of the output space are useful to determine which subset of inputs are important, to assess regions of attraction, and possibly identify ridges which separate adjacent regions. The latter are of interest because they may lead to investigation of what sort of changes in the input, what "nudges," are required to send the process into an adjacent region of attraction. This type of study is of great interest to military analysts.

2.0 Components

A component is an evolving concept without a concise widely accepted defintion. In this paper a component is defined as a reusable unit of composition for a larger architecture that conforms to contractually specified interfaces and reflects explicit context dependencies [4, 5]. Components are based on a simple nesting scheme: any component may contain zero or more components. More complex components can be deeply nested, as shown in Figure 1. In a consistent architecture, a component must satisfy a fundamental interface defined for components, i.e., the component model. Such an interface would specify that a new component can be added to another only through a well defined function that has the responsibility to accept components of user-defined types and reject all others. A component which is composed of other components is an aggregate. The interface should also support a function that returns an enumeration of the components of the aggregate. Components can be instance-based or class-based, although the former may be a more prudent choice. Since a component represents some concept or real world item, a label, representative of the domain type of the modeled concept or item, is attached to the component. It is then convenient to refer to the component by its domain type.

2.1 Context

The concept of components is not fully realized without the notion of context. Context is the collection of facts, state, functionality, and user perspective supported by the component. In addition the component reflects the interface of the component model.



C
$C_{1,1} \cdots C_{1,i_1} \cdots C_{1,j_1}$
$C_{I(p),1} \cdots C_{I(p),i_2} \cdots C_{I(p),j_2}$
$C_{I(p),1} \cdots C_{I(p),i_k} \cdots C_{I(p),j_k}$
I(p) = index of parent
$n = \sum_k j_k$

Figure 1: Top level component with n components and k levels

2.2 Challenges

The first challenge in using a components-based framework is to generate a view (design) of an application using a proper combination of *is-a* (inheritance) and *has-a* (component) relationships. This view is driven by the degree and level of interoperability desired. The process of achieving interoperability begins by defining the interface for a component, which must reflect the component model. The interface is extended as needed to address the functional needs of the domain type represented by the component. The interface must be rich enough to express the functional needs of all types of planned interactions, yet parsimonious to reflect an efficient design. For example, if a vehicle is defined by its components of propulsion and steering, then the vehicle interface must be rich enough to have domain types representing an automobile, a submarine, and an airplane while sharing the same interface. `Dive` would make sense for the airplane and submarine, but not for an automobile. One possible compromise would be to use a `Move` interface based on acceleration, velocity, and location. This would allow interoperability yet still provide appropriate mechanisms for dive-like or takeoff-like functionality.

2.3 Infrastructure

The component infrastructure must be tightly linked to the architecture [5]. In general, infrastructure services can be partitioned into many categories; however two types of services are needed at a minimum for a simulation framework: one for composition and one for simulation. The composition package provides the basic functions to support composition and defines the fundamental component interface. The simulation package provides support for discrete event simulation. A simulation is viewed as a composition of a simulation object containing simulation entities (often agents) and optionally a simulated environment. It should provide a definition for event objects as well as a discrete event simulation scheduler that can sequence events based on time and optionally event priority.

3.0 IVES

The IVES framework is designed with the intent of supporting the construction of simulations in many different domains. A simulation composed in this system is made up of elements that are in part specific to a particular problem domain. In the military simulation described later, the components represent infantry, buildings, roads, weapons, and sensors. A suite of components designed to simulate financial environments would have a very different set of components, messages and representations. Components are designed and implemented in suites to interoperate along contract lines that make sense in a particular domain. These objects might respond to a common set of event messages and share notions about the representation of state and response to those messages.

In addition to this specialized and stylized interaction within a domain specific area there is another layer of objects that are domain independent. This is called the IVES framework. The framework cares about functions that are common to all systems regardless of the domain. They provide such things as an abstract simulation object, object repository services necessary for composition, editing and visualization services as well as the basic composition functionality that can be utilized by objects in many different domains.

The framework imposes a contract that is both syntactical and intentional. The syntactic aspects of the contracts are imposed by the classes and interfaces that make up the IVES framework. Essential to satisfying the contracts is the notion of component-centered autonomy. The responsibility of complying with the contract in the context of certain actions and events falls to the components. This distribution of responsibility is essential to many aspects of the IVES architecture. When an object is added as a component to a composition, the composition could be a simulation or agent or other simulation artifact, both the composition and the component must respond responsibly and comprehensively to the event.

The components that enable the abstract view and capabilities of simulations are in the package **sim**. Compatibility is supported and enabled by the package **comp**. The Java interface **I_IvesPrimitive** that is in **sim** must be implemented by any class which is to be part of a composition, stored in a repository and function within the scope of a simulation. **I_IvesPrimitive** imposes a contract requiring the implementation of several functions and enables the runtime identification as an IVES object. The functions imposed by **I_IvesPrimitive** require that the object support such things as the ability to copy itself completely *deepCopy()*, to be named and to report it's name *getName()* and *setName()* and to initialize it's self and run time, *runTimeInit()*. For a component (object) to be composable in an IVES sense it must be derived from the interface **I_Composable**. **I_Composable** in part requires that an object implement *addComponent()* and *removeComponent*(). There are several classes in **comp** that enable the functionality of composition to be easily added to a class by delegation.

Implicit is the assumption that the object will respond responsibly and comprehensively to a message. When an object is added to another as a component i.e. a sensor is added to an agents sensor system, the sensor must be added to the **SensorSystems** collection of sensors and connected and initialized appropriately as required by the parent component. There is no external agent or functionality that does this work for the component. It must be self-configuring in this sense. This has the side effect of allowing the design to be individualized as needed for the requirements imposed by each component.

4.0 JIVES

JIVES is an application of IVES in a simplified in a military simulation domain ("J" stands for "joint" as in joint military operations). JIVES uses the discrete event simulation defined in the IVES framework. The protocols for the composition and simulation packages are observed and domain specific information is placed in a separate software package. Currently, JIVES simulates the activities of individual military infantry in a two dimensional environment which represents buildings, roads, and other infantry.

4.1 JIVES Architecture

The architecture for an agent in the simulation is a specialization of a general agent architecture as shown in Figure 2, where an agent is composed of a system for perception, a system of effectors that allow the agent to interact with the environment, and a collection of cognitive processes which govern the activities of the agent.

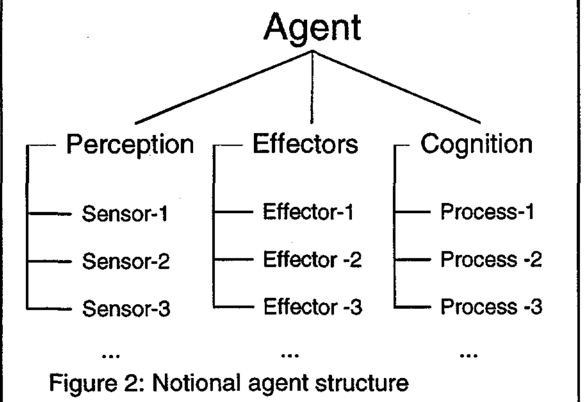


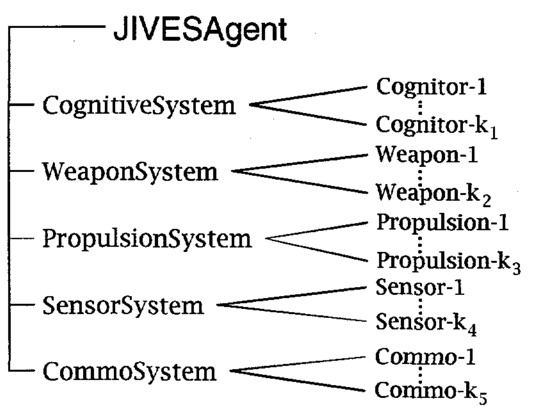
Figure 2: Notional agent structure using components

4.2 Demonstration Goals

A primary goal of JIVES is to represent combat of infantry in the complexity of the urban environment. Using this capability, JIVES would then be used to run computer-based experiments on the utility of alternate tactics for urban operations. A longer term goal is to use evolutionary computation [8,9] as a means of evolving new tactics over a set of specific scenarios. Similarly, new concepts in weapons or platforms could be evaluated and possibly evolved to assess utility in a fixed scenario space. However useful these experiments, there is no assumption that such evolutions would be applicable in the real world. Rather it is hoped that the changes would be suggestive of directions for warfighting research and experiments. Other objectives include experimenting with the modeling of soft factors with respect to combat results and developing a more robust cognitive capability for military agents.

4.3 JIVES Agents

The military entities of interest in JIVES are modeled as agents [7] and are termed `JIVESAgents`. They share a structure similar to the agent architecture of Figure 2. The actual structure is shown in Figure 3. The agent is composed of five systems: cognitive, weapon, propulsion, sensor, and communications (commo) based on the requirements to represent movement, attrition, communication, perception, and rational behavior. There is no inherent limit of the agent to five component systems. Other potential systems have been proposed, such as logistics.



Systems are containers for appropriate components, and have the responsibility for managing the components over which it has purview. The cognitive system is a container for components that support decision-making. For weapon systems, the proper components are **Weapons** and **Munitions**; for sensor systems, the components are **Sensors**, and so on. As examples of management functions, the weapon system can determine the best weapon and munition combination for a specified target. The propulsion system can recommend the most appropriate propulsion mode given the current conditions. Similarly, the sensor system can recommend the most appropriate sensor suite for specified classes of targets and environmental conditions. In each case, the system examines its components and recommends the components, if any, which maximize the evaluation criteria. Other management functions can readily be imposed if needed. The sensor system in JIVES is expected to monitor threat contacts, create tracks (data structures for modeling the motion of the contact), and classify tracks into threat categories.

The cognitive system contains zero or more decision-making components called **Cognitors**. A **Cognitor** is a component that implements the **I_Cognitor** interface that mandates two functions: *think* and *updatePerceptions*. The interface definition is deliberately slim, as it is not well known what functions the interface should require. A **MilRap** class is derived from **Cognitor** in which a partial implementation of the InterRap architecture [6] is functional. Currently in JIVES, all agents use the **MilRap** component as the sole component in their cognitive system.

Although the compositional nature of the agents allows for many cognitive structures, the primary one under development for the initial phase of JIVES is Muller's InterRap (integrated reaction and planning) architecture. This architecture specifies three layers of increasingly complex cognitive capabilities. The first layer is the Behavior Based Layer in which agents recognize situations and respond in a programmed way using patterns of behavior. This is the current level of capability for a **JIVESAgent**. Another way to invoke programmed behavior is to send the agent a message "commanding" the agent to invoke a specific behavior. Most often primitive behaviors are desired so that a commander can dictate arbitrary sequences of behavior and thus generate complex scripts describing desired actions. This capability is termed command-driven behavior. The utility is to allow a user or a command agent in the simulation to give orders to "subordinate" agents who will then execute those commands.

The second layer in the InterRap architecture is the Local Planning Layer which enables the agent to exhibit goal-driven behavior. In this layer, an agent performs means-ends reasoning on its capabilities and assets in order to derive a plan to achieve a set of goals. Goals can simply represent a desired state of the world. Actions that the agent can invoke are already expressed as patterns of behavior, used in the reactive layer. By augmenting the patterns of behavior with pre- and post-condition attributes, they can be used in a means-end reasoner to develop plans.

The architecture also supports second order planning, which matches goals to pre-defined plans which are stored in a plan library. A pre-defined plan can be incomplete in that certain planning variables need to be assigned in the course of fulfilling the plan. In this respect, partial plans resemble low-level tactics. Since partial plans consist of sequences of patterns of behavior with some planning variables, it should be possible to begin to evolve tactics in an evolutionary computation paradigm.

Future task goals would be to develop a means-end planner based on patterns of behavior and to create the infrastructure for a plan library. Initially the plan library will be a repository for command-driven execution. Eventually, the plan library will be accessed by an indexing function based on the goals provided to the Local Planning Layer. JIVES does not yet implement this layer.

A command agent is possible using this layer of the architecture under the strong assumptions of agent cooperation in subordinates and well-understood threat tactics. In other words, the command agent would consider the subordinates to be resources under its control and has implicit knowledge of threat responses.

The third layer of the InterRap architecture is the strategic planning layer. Muller terms this the "cooperative" planning layer, but in a military combat simulation, it seems appropriate to include both cooperative and adversarial planning. In either case, the agent uses a model of other agents in order to derive plans based on its own goals and goals imputed to modeled agents. JIVES does not yet implement this layer.

Agents maintain a set of `PatternsOfBehavior` (PoB) as components within the `MilRap` component. A PoB represents a routine task the agent can perform, based on the agent's primitive actions. A PoB includes an activation condition, a method to assess when the preconditions for the PoB are satisfied, and methods to assess whether the behavior, once invoked, was a success, failure, or encountered an exception condition requiring special processing. `MoveTo` is an example of a PoB in a JIVES agent, and is based on the primitive activity of "moving one step." To move from one location to another, the agent executes a sequence of "moving one step" until the destination is encountered or there is a failure in the movement capability as represented in the propulsion system.

The perceptions object of a cognitor acts as a storage facility for knowledge acquired about the environment. Actions of agents are based on perceptions acquired by using sensors assigned to the agent. Perceived information about the simulated environment is saved in the agent's perceptions. One element of perceptions is the perception map which makes a map-like copy of the environment the agent has perceived, including buildings, roads, and other agents.

4.4 Simulated Environment

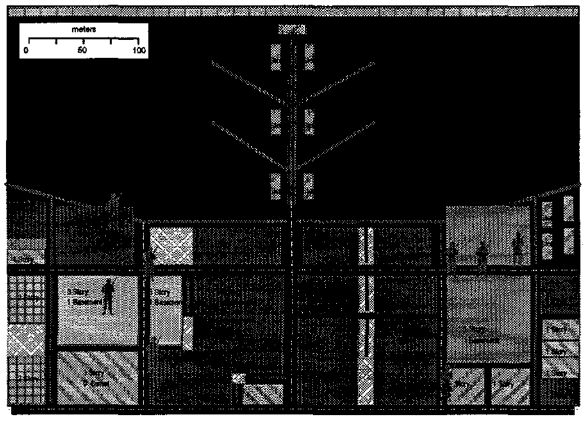
While not every IVES simulation requires an environment, in JIVES, a simulated environment is needed to support the interaction of agents. The environment component represents surface and cultural features, e.g., surface type, vegetation, roads, buildings, bridges, etc., as well as terrain elevations. Functionally, the environment supports line of sight calculation, occupancy determination, access to feature attributes, and other utility functions. Agent components such as sensors and propulsion frequently interact with the environment to update the perceptions of the agent and to constrain movement.

4.5 Arbitrators

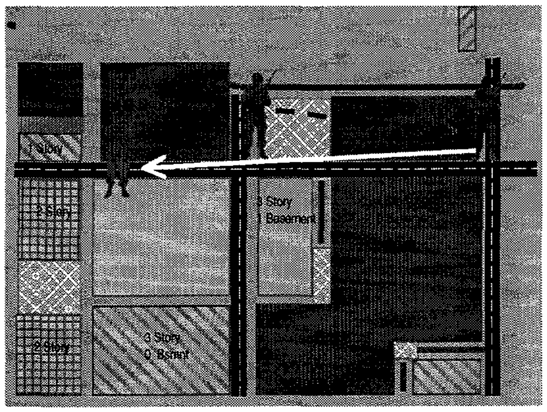
For some actions, agents interact with the simulation environment and update their own state or change the state of the environment. For other actions, such as attrition, where one agent uses a weapon to launch a munition at another agent, the damage received by the victim is determined by a neutral arbitrator. In combat simulations, there is a concern for fair play. If, for example, the attacker computes the damage to be assigned to the victim, the worry is that the attacker may be programmed to be biased towards imposing greater damage. Conversely, if the victim assesses the damage received, the worry is that the bias may be towards computing lesser damage. The approach taken in JIVES is to use an arbitrator mechanism to calculate the damage and impose it upon the victim. This is a pattern which can be used in other situations, such as sensing, where there is concern for fairness in determining an outcome. Like other simulation elements, the arbitrator is a component of the simulation and can be manipulated as any other component.

4.6 Results

The initial scenario examined in JIVES is the movement of individual infantry to an objective. Threat forces are arrayed within the urban environment and when encountered, lethal force is employed against the threat, if possible. Agents are provided only with patterns of behavior to enable moving and shooting. Agents are given a destination to move to and the simulation begins. Figure 4 shows the initial locations for a typical run.



During the simulation, agents begin moving towards the assigned destination and employ sensors to examine the environment. When a possible threat contact is detected, it is tracked until classification exceeds a specified confidence level. If the contact is classified a threat, the agent attempts to engage it with appropriate weapons. Figure 5 shows an agent sensing (dashed line) an opposing agent and another agent shooting at an opponent.



Analysis of simulation output can be accomplished in many ways. For demonstration purposes, 100 runs of JIVES were performed using various combinations of sensor range and weapon range with results shown in Figure 6. This figure depicts the notional relationship between sensor and weapon range with respect to mission success.

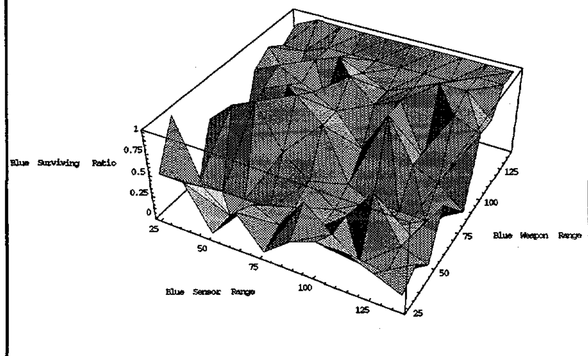


Figure 6. Mission success as a function of weapon range and sensor range (Notional)

5.0 Conclusion

The promise of composable simulations is that they will replace large cumbersome models that deeply embed assumptions about the domain that may no longer be valid. Instead, a simulation can be constructed from appropriate components and specifically focused on the analysis issues of importance. IVES has been demonstrated to be a viable framework and is one approach to developing component-based simulations. A prime requirement for successful use of IVES is a well-understood application domain leading to the definition of component interfaces. The JIVES simulation is an example of how to apply IVES within the rich and complex domain of military modeling.

6.0 References

[1] Defense Modeling and Simulation Office, "HLA Federation Development and Execution Process Model, Version 1.0," 21 August, 1996.

[2] Brandstein, A. and Horne, G., "Data Farming: A Meta-Technique for Research in the 21st Century," in Maneuver Warfare Science 1998, eds. F. Hoffman and G. Horne, pub. U.S. Marine Corps, Combat Development Command, Quantico, VA.

[3] Bankes, S. and Gillogly, J., "Exploratory Modeling: Search Through Spaces of Computational Experiments", The RAND Corporation, RAND/RP-345, Santa Monica, CA.

[4] Brown, A. W., and Wallnau, K. C., "The Current State of CBSE," IEEE Software, September/October 1998, pp. 37-46.

[5] Szyperski, C., Component Software: Beyond Object-Oriented Programming, Addison Wesley Longman, Reading, MA, 1998.

[6] Muller, J. P., The Design of Intelligent Agents: A Layered Approach, Lecture Notes in Artificial Intelligence, Number 1177, Springer-Verlag, Berlin, 1996.

[7] Jennings, N. R., and Wooldridge, M. J., (eds.), Agent Technology: Foundations, Applications, and Markets, Springer-Verlag, New York, NY, 1998.

[8] Back, T., Fogel, D.B., and Michalewicz, Z., (eds), Handbook of Evolutionary Computation, Institute of Physics Publishing, Philadelphia, PA, 1997.

[9] Mitchell, M., An Introduction to Genetic Algorithms, The MIT Press, pub., Cambridge, MA, 1997.