Advanced Distributed Simulation through the Aggregate Level Simulation Protocol

Richard M. Weatherly
Annette L. Wilson
Bradford S. Canova
Ernest H. Page
Anita A. Zabek

The MITRE Corporation
7525 Colshire Drive
McLean, Virginia 22102

Mary C. Fischer

U.S. Army STRICOM
12350 Research Parkway
Orlando, Florida 32826

Abstract

The venerable problem solving technique of simulation finds itself in the midst of a revolution. Where once it was regarded as a "technique of last resort" for systems analysis, today simulation is widely applied to support myriad purposes, including: training, interaction, visualization, hardware testing, and decision support in real-time.

Advanced distributed simulation (ADS) is the U.S. Department of Defense (DoD) nomenclature used to describe the cooperative utilization of physically distributed simulations toward a common objective. The Aggregate Level Simulation Protocol (ALSP), under the auspices of ADS, provides a mechanism for the integration of existing simulation models to support training via theater-level simulation exercises. Consisting of a collection of infrastructure software and protocols for both inter-model communication through a common interface and time advance using a conservative Chandy-Misra based algorithm, the ALSP has supported an evolving "confederation of models" since 1992.

A review of the history and design of ALSP is presented and serves to outline directions for future investigation.

1 Introduction

Simulation is among the most well studied and widely utilized digital computer applications in existence. Emerging during the mid 1950s, simulation has evolved from a "technique of last resort" into a cost-effective mechanism for system investigation. Where once it was utilized exclusively for systems analysis, simulation in the 1990s is expected to provide support for myriad purposes, including: training, interaction, visualization, hardware testing, and decision support in real-time.

Advanced distributed simulation (ADS) is the nomenclature emanating from the U.S. Department of Defense (DoD) to describe the cooperative utilization of physically distributed simulations toward a common objective. The Aggregate Level Simulation Protocol (ALSP), under the auspices of ADS, provides a mechanism for the integration of existing simulation models to support training via theater-level simulation exercises. Consisting of a collection of infrastructure software and protocols for both inter-model communication through a common interface and time advance using a conservative Chandy-Misra based algorithm, the ALSP has supported an evolving "confederation of models" since 1992.

This paper describes the evolution and design of ALSP – one of the largest "real world" applications of parallel and distributed simulation theory. Section 2 presents a brief history of protocol development. Section 3 outlines the protocol design, focusing on the architecture, components and communication scheme. Conclusions and directions for future investigation are given in Section 4.

2 A brief history of ALSP

Historical references for the Aggregate Level Simulation Protocol and the confederations of models that it supports may be found in [1, 2]. A brief outline of this history is given below.

2.1 Motivating factors

In the fall of 1989, around the time of the Berlin Wall collapse, the Warrior Preparation Center (WPC) in Einsiedlerhof, Germany hosted the computerized military exercise *ACE-89*. The Advanced Research Projects Agency (ARPA, at that time the Defense Advanced Research Projects Agency) used *ACE-89* as a technology insertion opportunity by funding the deployment of the Defense Simulation Internet (DSI). The packetized video teleconferencing made possible by the DSI brought general officers of all the NATO nations face-to-face during a military exercise for the first time and was well received. On the other hand, the major computational application of the DSI, the distribution of the combat model Ground Warfare Simulation (GRWSIM), realized fewer accolades. Unreliability of the central portion of the GRWSIM game and inconsistencies in the distributed

GRWSIM database had profound effects on the state of combat. Players on both the sides of the game were frequently confronted with tactical and strategic challenges that were the result of software anomalies rather than enemy action.

In this same time period, ARPA funded the development of a distributed tank trainer system called SIMNET. SIMNET captured the imagination of many in the military training community by creating an environment where individual, computerized, tank-crew trainers could be connected over local area networks and the DSI to cooperate in a single, *virtual* battlefield. The success of SIMNET, the disappointment of *ACE-89*, and the desire to combine existing combat simulations to form more comprehensive systems prompted ARPA to initiate the research that lead to the Aggregate Level Simulation Protocol.

2.2 Fundamental principles

In early 1990, ARPA sponsored The MITRE Corporation to investigate the design of a general interface between large, existing, aggregate-level combat simulations. Designed using Lanchestrian models of combat (see [3]) rather than individual physical weapon models, aggregate-level combat simulations are typically used for high-level training. Despite significant representational differences, several of the underlying principles of SIMNET are applicable to aggregate-level simulations [1]:

- *Dynamic configurability.* Simulations are permitted to join and depart a simulation exercise without restriction.
- *Geographic distribution.* Simulations can be located in different geographic locations yet exercise over the same logical terrain.
- *Autonomous entities.* Each simulation controls its own resources, fires its own weapons and when hit, conducts damage assessment locally.
- *Communication by message passing.* Information from one simulation is distributed to all other simulation using a message-passing protocol.

The ALSP challenge has several unique requirements beyond those of SIMNET:

- *Simulation time management.* Typically, simulation time is independent of wall-clock time. For the results of a distributed simulation to be "correct," time must be consistent across all "processes" involved in the simulation.[1]
- *Data management.* The schemes for internal state representation may differ widely among existing simulations. A common representational system and concomitant mapping and control mechanisms are needed.
- *Architecture independence.* The architectural characteristics, e.g. implementation language, user interface, and time flow mechanism, of existing simulations may differ widely. The architecture implied by ALSP should be unobtrusive to existing architectures.

[1] *Absolute* consistency of time across distributed simulations is not always necessary, depending on the objectives of the simulation, and is theoretically impossible in certain circumstances; see Lamport [4].

2.3 Initial experiments

Three experiments were performed in 1991 using the Ground War Simulation (GRWSIM) in an effort to verify the feasibility of implementing the fundamental ALSP concepts. The first phase of experimentation dealt with ground-to-ground combat. Of particular interest were unit movement and attrition, and the GRWSIM time advance mechanism. The primary objective was to characterize the nature of the code modifications required to permit GRWSIM to participate in a *confederation* of models. The second demonstration used GRWSIM and the Air Warfare Simulation (AWSIM) to demonstrate the additional ability to perform air-to-air, air-to-ground, and ground-to-air combat. This demonstrated the ability to integrate objects and interactions from two different simulations. The final experiment used the Corps Battle Simulation (CBS) as the ground simulation and the current interaction capabilities were further expanded to demonstrate the capability to have interactions occur between objects owned by different simulations (i.e. CBS air defense units shooting down aircraft flying in AWSIM).

2.4 Fielding the software

Successful demonstrations within the laboratory provided the motivation to use ALSP in support of a major exercise, *Reforger 92*. To prepare for *Reforger 92*, U.S. Army in Europe (USAREUR) selected two exercises for initial use of the ALSP Joint Training Confederation (JTC). The first was the US V Corps exercise *Central Fortress 92* which was conducted in Germany in July. For this exercise, CBS was executed at V Corps facilities in Frankfurt, Germany and AWSIM was executed at WPC. The second exercise was *Ulchi Focus Lens 92*, a Combined Forces Command (CFC) exercise conducted in Korea in August. For this exercise, CBS executed in Korea while AWSIM and the Navy's Research, Evaluation and Systems Analysis Facility (RESA) executed at WPC. For *Reforger 92*, communication reliability challenges faced in the first two exercises were reduced by executing both CBS and AWSIM at WPC. The exercise was regarded as a success and demonstrated that many of the *ACE-89* problems had been rectified by the ALSP approach.

Since its initial configuration, the JTC has expanded to include seven primary models. In addition to CBS, AWSIM and RESA, the confederation now includes MTWS (littoral combat), JECEWSI (electronic warfare), TACSIM (intelligence collection and distribution), and CSSTSS (logistics). Apart from the JTC, ALSP has been used by the SHAPE Technical Centre in the Netherlands, by the Army at Fort

Leavenworth and Huntsville, and the Air Force at Hanscom Field.

3 Design

A review of the ALSP design is given in two areas: (1) software architecture, and (2) communications. To provide the context for the presentation of design, a discussion of the world view implied by ALSP is warranted.

3.1 The ALSP conceptual framework

A *conceptual framework* (CF) is an organizing structure of concepts that facilitates simulation model development [5]. Also referred to as a *simulation strategy* and *world view,* some common CFs include: event scheduling, activity scanning and process interaction.

The ALSP CF is best described as *object-based:* a *model* is comprised of *objects;* an object is characterized by *attributes* to which values are assigned. Within a confederation, object *classes* are organized hierarchically in much the same manner as with object-oriented programming languages, however the inheritance mechanism for ALSP is less powerful (see [6]).

Missing from the above description is a key concept, that of a *confederation.* A confederation is the entity that ALSP supports: a collection of existing simulations in support of a single, common model.

To design a mechanism that permits *existing* simulations to interact, two strategies are possible: (1) define an infrastructure capable of translating between the representations of any confederated simulation, or (2) define a common representational scheme and require all simulations to map to that scheme. The former approach has the advantage that very few perturbations to the existing simulations are required; interaction is facilitated entirely through the interconnection infrastructure. However, this solution does not scale well. Because of an underlying requirement for scalability, the ALSP design adopts the latter strategy. ALSP prescribes that a *translator* be constructed to provide mappings between the representational scheme of the confederation and the representational scheme of a simulation.

The construction of a translator represents one of the three ways in which a simulation must be fundamentally altered to participate in an ALSP-supported confederation. The remaining modifications involve:

- Recognition that not all objects that a simulation perceives are owned by the simulation.
- Modification of the internal time advance mechanism to work cooperatively with the other simulations within the confederation.

In the familiar use of simulation, between initialization and termination objects come into (and go out of) existence with the passage of simulation time; the disposition of these objects is solely the purview of the simulation. When acting within a confederation, the simulation-object relationship is more complicated. The simulation-object ownership property is dynamic, i.e. during its lifetime an object may be *owned* by more than one simulation. In fact, for any value of simulation time, several simulations may own different attributes of a given object. A simulation is said to *own* an attribute if the simulation owns a lock associated with the attribute. By convention, a simulation is said to own an object if the simulation owns the lock on the "identifying" attribute associated with that object. Owning an object implies that the simulation is responsible for calculating and reporting attribute value changes for all (otherwise unowned) attributes of that object. Objects not owned by a simulation but within the area of perception for the simulation are known as *ghosts.* Ghosts are local copies of objects owned by other simulations.

When an object is created, the creating simulation reports this fact to the confederation to enable the creation of ghosts (as applicable) within other simulations. Likewise, when an object is deleted, the deleting simulation must report this fact to enable ghost deletion. Whenever an action is taken between an object and a ghost, the owning simulation must report this to the confederation. In the parlance of ALSP, this is known as an *interaction.*

These fundamental concepts provide the basis for the remainder of the presentation. In the following discussion the terms model and simulation are used synonymously. Where appropriate the term *confederation model* is used to describe the object hierarchy, attributes and interactions supported by a confederation.

3.2 The ALSP infrastructure software

The object-based conceptual framework adopted by ALSP defines *what* classes of information must be distributed. The ALSP Infrastructure Software (AIS) provides *how* data distribution and process coordination are accomplished. In its current incarnation, the AIS consists of four major components: (1) the ALSP Common Module (ACM), (2) the ALSP Broadcast Emulator (ABE), (3) the ALSP Control Terminal (ACT), and (4) the Confederation Management Tool (CMT). Figure 1 depicts a simple configuration for the AIS. The ABE provides message distribution capabilities; time synchronization and object management are performed by the ACM, which provides the interface between an actor/translator pair and the rest of the confederation. The term *actor* refers to any simulation or simulator that is used in a confederated environment.

3.2.1 ALSP Common Module

The ALSP Common Module (ACM) provides a common interface for all translators and contains the essential functionality for ALSP. The services provided by the ACM include:

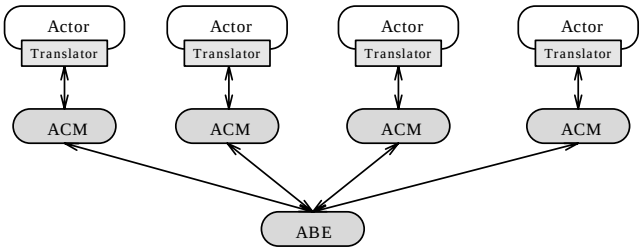

Figure 1: Current ALSP architecture.

- Coordinating the processes of joining and departing from the ALSP confederation.
- Coordinating actor local time with confederation time.
- "Filtering" incoming messages, i.e. only messages of interest are received by the translator.
- Coordinating the ownership of object attributes, and permitting ownership migration.
- Enforcing the attribute ownership relation; actors may report values only for attributes they own.

These services define two classes of management activity: (1) time management, and (2) object management.

Time management. Joining and departing a confederation is an integral part of the time management process. When an an actor joins a confederation all ACMs currently in the confederation are notified to permit the creation of input message queues (that support the Chandy-Misra time synchronization algorithm). Conversly, when an actor departs a confederation the input message queues for that actor are deleted from the other ACMs.

The ALSP time management facilities are designed to support discrete event simulation using either asynchronous (next-event) or synchronous (time-stepped) time advance mechanisms (see Nance [7]). To support next-event simulations the event request mechanism is used as follows:

1. The actor sends an `event_request` message to the ACM with a time parameter corresponding to simulation time t, of the next local event it would like to process.
2. If there are any messages with timestamps less than or equal to t the ACM will send the smallest one to the actor. If all messages are time stamped with times greater than t the ACM will send a `grant_advance` to the actor, giving it permission to process its local event at time t.[2]
3. The actor sends all output (if any) resulting from the event to the ACM.
4. The actor repeats from Step (1).

In order to support time-stepped simulations the AIS provides an advance request mechanism (basically) as follows:

1. The actor processes all events for some time interval $(t, t + \Delta t]$.
2. When the actor has processed all its local events for the time interval it sends an `advance_request` to the time $t + \Delta t$.
3. The ACM sends all messages with timestamps on the interval $(t, t + \Delta t]$, followed by a `grant_advance` to $t + \Delta t$.
4. The actor sends all output (if any) for the interval $(t, t + \Delta t]$ to the ACM.
5. The actor repeats from Step (1).

The AIS provides a deadlock avoidance mechanism using null messages. Such a mechanism requires that the intrinsic processes posses exploitable lookahead characteristics. For the primary application of ALSP, the JTC, this is the case. Mechanisms for deadlock detection/recovery are being investigated.

Object management. The ACM administers the *attribute database* and *filter* information. The attribute database maintains the objects known to the actor, either owned or ghosted, and the attributes of those objects which the actor currently owns. The attribute database employs several *sets* to facilitate its function. For any object class, attributes may be members of: (1) a *create set,* (2) an *interest set,* and (3) an *update set.* The create set defines those attributes minimally required to represent an object. Useful, but not mandatory, information is determined by the interest set. Object attribute values reported by an actor to the confederation are determined by the attributes and object classes specified in the update set.

Information flow (across the supporting network) is further restrictable through the use of *filters.* Filtering provides: (1) discrimination by object class, (2) discrimination by attribute value or range,[3] and (3) discrimination by geographic location, e.g. polygonal playboxes. Filters are also used to define the interactions relevant to an actor (see Section 3.3).

When an ACM receives an update, it determines if the update passes any filter criteria. If so, the ACM determines if the object is known to the actor. If the object is known to the actor, the ACM sends the new attribute values to the actor. If the object is not known to the actor, the ACM determines if enough information is present for the actor to create a ghost, i.e. all attributes in the create set are present in the update. If enough information is present, the object manager sends the actor a create message. If there is not enough information for the actor to create the ghost object, the object manager stores the information and sends a request to the confederation for

[2] Pursuant to the Chandy-Misra synchronization scheme, the actor may block if one or more input message queues are empty.

[3] Filtering on attribute values is only provided for certain attribute *types.*

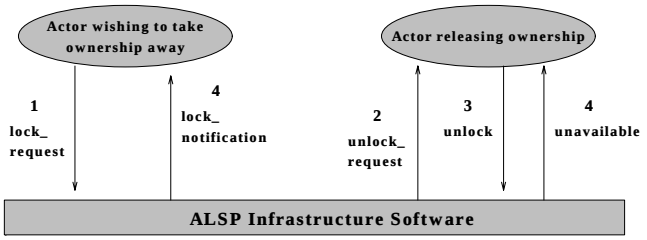

Figure 2: Actor acquiring attribute ownership.

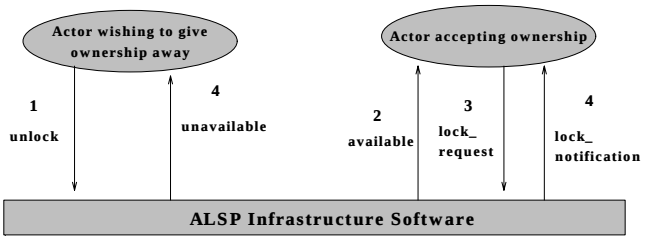

Figure 3: Actor relinquishing attribute ownership.

the missing information. If the update does not pass the actor's filter, the object manager checks to see if the object is known to the actor. If the object is known to the actor, the object manager sends the actor a delete message for the object. If the object is not known to the actor, the object manager discards the update.

The ownership and filtering information maintained by the ACM provide it with the information necessary to coordinate the transfer of attribute ownership between actors. Figure 2 shows the exchange that occurs between actors when one actor is trying to take ownership away from another. Figure 3 shows the exchange when one actor attempts to give away ownership.

In Figure 2 the actor on the left is attempting to acquire ownership of an object attribute owned by the actor on the right. The actor initiates the transaction, in Step 1, using the `lock_request` message. When the ACM receives this message, it verifies (1) that the actor knows about the object (either owned or ghosted), and (2) that the attribute(s) specified are included in the update set for the object class. The AIS locates the current owner of the attribute, in Step 2, by sending a query to the confederation. Each ACM in the confederation receives this message and determines if the actor associated with the ACM currently owns the attribute. If the actor owns the attribute, the ACM sends an `unlock_request` message providing the attribute owner with the name of the desired attribute and, optionally, the acquirer's reason for wanting to own it. If the attribute owner decides to release ownership, it sends, in Step 3, the `unlock` message. The ACM receives the `unlock`, and changes the state of the attribute and sends the information to the confederation. Lastly, if the acquiring actor has not canceled its ownership acquisition request, the AIS informs actors capable of locking the object attribute that it has been locked by an actor, and informs the acquiring actor that it now owns the attribute (Step 4).

In Figure 3, the actor on the left owns an object attribute that it is attempting to transfer to some other actor. In Step 1, the actor sends an `unlock` message with the name of the attribute to be transferred. When the ACM receives the `unlock` message, it verifies that the actor has the attribute(s) locked. The ACM informs the confederation that the attribute has been unlocked. When each ACM receives the message, it determines if the actor knows about the object and whether it is capable of locking the attribute (i.e. the attribute is in the update set for the object's class). If both of these conditions are met, the ACM sends an `available` message to the actor. Upon receipt of the available message, an actor can send a `lock_request` message, Step 3, to compete for ownership of the attribute. The AIS selects a recipient for the attribute ownership from the set of actors that make ownership requests and completes the transfer, in Step 4, as was done in the previous example. In this example, an `unlock_request` message is not needed because the attribute is already unlocked.

3.2.2 ALSP Broadcast Emulator

An ALSP Broadcast Emulator (ABE) is a process that facilitates the distribution of ALSP information. Its primary function is to receive a message on one of its communications paths and retransmit the message on all of its remaining communications paths. This permits configurations like that depicted in Figure 1 where all ALSP components are local to one another (on the same computer or on a local area network). It also permits configurations where sets of ACMs communicate with their own local ABE with inter-ABE communication over wide area networks.

3.3 The ALSP communication scheme

The ALSP communication scheme consists of four parts: (1) an inter-component communications model which defines the transport layer interface used to connect each of the ALSP components, (2) a layered protocol that provides actor-to-actor communication, object management, and time management, (3) a programmable message filtering scheme used to define the information of interest to an actor, and (4) a mechanism for providing intelligent message distribution.

3.3.1 Inter-component communications model

The AIS employs a persistent connection communications model (see Boggs [8]) to provide the inter-component communications for the AIS. The transport layer interface used to provide inter-component communications is dictated by actor

Figure 5: Actor protocol and actor/ACM protocol messages.

information is then distributed, via the AIS, to any other actors in accordance with their respective create and interest sets.

`Interaction`. Interactions between objects are identified by *kind*. Interaction kinds are parameterized, i.e. interactions are described by parameters, just as objects are described by attributes. When an object owned by an actor engages either an object owned by another actor or a geographic area, an `interaction` message is generated by the initiating actor. The `interaction` message contains the kind of interaction and a set of appropriate parameters.

`Refresh_request`. An actor can, at any time, request an update of a set of attribute values for any specific object or class of objects by sending a `refresh_request` message to the confederation.

`Delete`. When an object owned by an actor ceases to exist (within the domain of the owning actor) a `delete` message is used to inform confederation members that the object has gone out of existence. When a `delete` message is received by an actor, the appropriate ghost, if one exists, should be deleted.

The actor protocol is a text-based protocol rather than a binary encoded protocol, e.g. the DIS Protocol Data Unit (PDU). The actor protocol is defined by an LALR(1) context-free grammar (see [6]). The semantics of the protocol are confederation dependent, i.e. the set of classes, class attributes, interactions, and interaction parameters are variable. Therefore, the syntactical representation of the actor protocol may be defined without *a priori* knowledge of the semantics of the objects and interactions of any particular confederation.

3.3.4 Actor/ACM connection protocol

The actor/ACM connection protocol is used to provide a set of services for managing the state of the connection between the ACM and the actor, actually translator, and to provide a method of information exchange between the ACM and the actor (Figure 5). The actor/ACM connection protocol provides two services for the distribution of actor protocol messages: *events* and *dispatches.* Messages sent as events are time-stamped and delivered in a temporally consistent order. Dispatch messages are delivered as soon as possible, without regard for simulation time. The additional actor/ACM connection protocol messages are used to provide: connection state, filter registration, attribute lock control, confederation save control, object resource control, and time control services.

3.3.5 Object management protocol

The object management protocol is a peer-level protocol that sits below the actor protocol and provides object management services. The object management protocol is used solely by the ACMs to support object attribute creation, acquisition, release, and verification (of the consistency of the distributed object database). These services provide the mechanism by which the AIS manages distributed object ownership, which is a key concept in ALSP.[5]

The concept of distributed object ownership is based on the premise that no single simulation needs to own all the objects in a confederation, however many simulations may require knowledge of any particular object. Via actor protocol update messages, objects owned by a simulation may be *discovered* by other simulations in the confederation. If these objects are of interest to these simulations they can be ghosted. The ghosting process gives a simulation knowledge that an object exists in the confederation, thus providing the opportunity for interactions between owned and ghosted objects.

Locks are utilized to implement attribute ownership; a primary function of the object management protocol is to ensure that a simulation only updates attributes for which it has acquired a lock. The object manager in the ACM is responsible for the management of the objects and object attributes of the owned and ghosted objects known to the ACM. Services provided by the actor/ACM protocol are used by the simulations in the confederation to interact with the ACM's attribute locking mechanism. The coordination of status, request, acquisition, and release of object attributes, between ACMs, is controlled via the object management protocol.

Each attribute (of each object) known to a given ACM has a corresponding status which assumes one of three values:

Locked. The actor controls the attribute and may update the attribute value. An actor "owns" the attribute if it has that attribute locked. An actor "owns" the object if it has the id attribute locked.

Unlocked. No simulation currently controls the attribute. Any simulation asking for control will be granted control.

Gone. The state of control is held elsewhere in the confederation.

From the ACM's perspective, objects come into existence through the registration process performed by the attached actor or through the discovery of objects registered by other actors. The initial state attribute locks for registered objects and discovered objects are as follows:

Object Registration causes each object-attribute pair to be placed in the locked state. The actor may optionally specify attributes to be in the unlocked state.

Object Discovery adds an object to the object database as a ghosted object. All of the attributes for this object are marked with a status of gone.

3.3.6 Time management protocol

The time management protocol is also a peer-level protocol that sits below the actor protocol. It provides time management services for the synchronization of simulation time between ACMs. The time management protocol provides three main services for the distributed coordination (among ACMs) of an actor's entrance into the confederation, time progression, and confederation saves.

The join/resign services and time synchronization mechanisms are described in Section 3.2.1. The save mechanism is provided for fault tolerance. Coordination is required to produce a consistent snapshot of all ACMs, translators and simulations for a particular value of simulation time.

3.3.7 Message filtering

The AIS uses filtering for two purposes. First, filtering is used by the ACM to determine if a particular message is pertinent to the actor that is attached to it. In addition, the AIS uses filtering to perform intelligent message distribution.

3.3.8 Actor message filtering

Actor message filtering is the process by which the ACM evaluates the content of a message received from the confederation. Messages that are of interest, and pass the filtering criteria are delivered to the actor and those that are not of interest are discarded. Filtering is performed by the ACM on two types of messages: update messages and interaction messages.

[5] Recall from Section 3.1 that in ALSP, "ownership" is defined at the attribute level. Object ownership is provided by convention and utilized to govern object creation/deletion.

Update messages. `Update` messages are evaluated by the ACM based on the actor's update message filtering criteria, which the actor provides through the `filter_class`, `filter_attr`, and `filter_polygon` messages. As discussed in Section 3.2.1, when an ACM receives an `update` message there are four possible outcomes: (1) the ACM discards the message, (2) the ACM sends the actor a `create message`, (3) the ACM sends the actor the `update` message, or (4) the ACM sends the actor a `delete` message.

Interaction messages. Interaction messages may be discarded because of the kind parameter. The kind parameter has a hierarchical structure similar to the object class structure. The actor informs the ACM of the interaction kinds that should pass or fail the interaction filter by using the `filter_interaction` message.

3.3.9 Message distribution

In order to minimize message traffic between components in an ALSP confederation the AIS employs an form of intelligent message routing that is implemented by using the Event Distribution Protocol (EDP). Described in [10], the EDP allows the ACMs to inform the other AIS components about the update and interaction filters registered by their actors.

In the case of update messages, the distribution of this information allows the ACMs to only distribute data on classes (and attributes of classes) that are of interest to the confederation. The ABE also use this information to send only information that is of interest to the components it serves. For interaction messages, the process is similar, with the exception that the `kind` parameter in the interaction message is used to determine where the message is sent.

Expected bandwidth savings due to the EDP have been estimated at 30% - 50% for a typical exercise involving the Joint Training Confederation. These savings have only been approximated, however, since the computational requirements (with respect to the computational facilities) of most JTC exercises preclude extensive data collection; unobtrusive data collection techniques are the subject of continuing investigation.

4 Conclusions

Initiating with prototype efforts in 1990, the Aggregate Level Simulation Protocol, and the confederations it supports, has grown into one of the largest practical applications of distributed simulation; the 1995 Joint Training Confederation consists of seven warfare models which collectively, along with the infrastructure and support software, yield an environment consisting of millions of lines of code.

Based on a scalable architecture, explicitly designed to support the introduction of reliable multicast distribution, the ALSP is poised to accommodate increasing demands for detail and fidelity. Using a conservative synchronization algorithm, the ALSP supports interactivity by bounding the temporal relationships among actors in a confederation, and provides a platform for both analysis and training.

The avenues for future investigation in this arena are many. Foremost on the horizon is the Defense Modeling and Simulation Office (DMSO) sponsored High Level Architecture (HLA) initiative. Designed to support and supplant both DIS and ALSP, efforts are underway to prototype an infrastructure capable of supporting these disparate applications.

Performance considerations have dominated DIS research in the past, and will occupy a more prominent role in ALSP as the transition to HLA occurs. Many of the solutions to the performance problem must be realized in advancing hardware technologies, however some key software issues require directed research.

- *Conservative versus optimistic synchronization.* Within the PDES community optimistic protocols (e.g. time warp) have been shown to outperform conservative mechanisms for a wide variety of problems (see Nicol and Fujimoto [9]). However, in an interactive training environment the results of all incorrect computation (and subsequent rollback) must be hidden from the training audience. Coupling the simulation output displays with GVT provides a solution, but for a typical training environment (e.g. hundreds to thousands of displays) can the GVT calculation, state saving and garbage collection problems be handled efficiently?

- *Dynamic fidelity.* For simulations with real-time requirements (e.g. DIS, and envisioned for ALSP) some mechanism must be provided to allow a simulation not meeting its minimum performance requirements to reduce its computational load. A simple solution is simply to forego all computation for which there isn't sufficient time. This solution is less than ideal. Methodologies – either fully- or semi-automatable – that permit well-defined, on-the-fly fidelity adjustments need to be defined so that both correctness and timing requirements may be satisfied.

Finally, VV&A for ALSP, DIS and HLA remains a critical open issue. Traditional statistically-based verification and validation techniques are generally not applicable to interactive simulations. Furthermore, whereas face validation for the model *may* itself be feasible, in an interactive environment where multiple failures are common – perhaps unavoidable – over a multi-week training exercise, each *use* of the simulation(s) must be subjected to validation. Methodologies for this type of evaluation are currently not well-defined.

Advanced distributed simulation – ALSP, DIS, HLA and the like – portends a significant transition in the day-to-day use of computer simulation. To realize success, the evolution of ADS must be a combined effort of government, industry and academia. The purveyors of ALSP look forward to contributing to this arena, and welcome input from each of these communities.

Acknowledgements

This work was partially supported under contract number DAAB07-93-C-N651. ALSP is the product of the efforts of many talented individuals. The work of Dave Bellino, Judith Dahmann, Scott Emeret, Laura Feinerman, Sean Griffin, Monica Grose, Annette Janeway, Ben King, Frank Maginnis, Gordon Miller, Jeff Opper, Dave Prochnow, Dan Sandini, Dave Seidel, John Tufarolo, and Sophia Yu is acknowledged and appreciated. The authors would also like to thank the anonymous referees for their valuable comments and suggestions.

References

[1] Seidel, D. (1993). "Aggregate Level Simulation Protocol (ALSP) Program Status and History," The MITRE Corporation, McLean, VA, 22102, March.

[2] Wilson, A.L and Weatherly, R.M. (1994). "The Aggregate Level Simulation Protocol: An Evolving System," In: *Proceedings og the 1994 Winter Simulation Conference*, pp. 781-787, Lake Buena Vista, Fl, 11-14 December.

[3] Taylor, J.G. (1983). *Lanchester Models of Warfare*, Operations Research Society of America, Arlington VA, March.

[4] Lamport, L. (1978). "Time, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM*, **21**(7), pp. 558-565, July.

[5] Balci, O., Nance, R.E., Derrick, E.J., Page, E.H. and Bishop, J.L. (1990). "Model Generation Issues in a Simulation Support Environment," In: *Proceedings of the 1990 Winter Simulation Conference*, pp. 257-263, New Orleans, LA, 9-12 December.

[6] The MITRE Corporation. (1994). Aggregate Level Simulation Technical Specification, McLean, VA, March.

[7] Nance, R.E. (1971). "On Time Flow Mechanisms for Discrete Event Simulations," *Management Science*, **18**(1), pp. 59-93, September.

[8] Boggs, D.R. Shoch, J.F., Taft, E.A., and Metcalfe, R.M. (1979). "PUP: An Internetwork Architecture," Report CSL-79-10, XEROX Palo Alto Research Center, July.

[9] Nicol, D.M. and Fujimoto, R. (1994). "Parallel Simulation Today," *Annals of Operations Research*, **53**, November.

[10] Weatherly, R.M., Wilson, A.L. and Griffin, S.P. (1993). "ALSP – Theory, Experience, and Future Directions," In: *Proceedings of the 1993 Winter Simulation Conference*, pp. 1068-1072, Los Angeles, CA, 12-15 December.