Proceedings of the 1995 Winter Simulation Conference
ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman

ARCHITECTURAL OPTIMIZATIONS TO ADVANCED DISTRIBUTED SIMULATION

Larry Mellon
Darrin West

Science Applications Int. Corp.
4301 N. Fairfax Dr., Suite #370
Arlington, VA 22203

ABSTRACT

Current DoD mechanisms to support distributed simulations have reached their limits in terms of size and fidelity. Several projects are underway to improve the state of the art in the DoD, defining a new class of distributed simulation: Advanced Distributed Simulation (ADS). This paper presents an architectural view of the problem area, i.e. identifying the conceptual objects in an ADS system, and describing their responsibilities and interactions. Classes of data transmission optimizations are identified and discussed in terms of scalability, flexibility, and current research efforts.

1 INTRODUCTION

ADS defines the next-generation of military distributed simulation systems, intended to support executions distributed across LANs and/or WANs, with up to 100k entities at up to 50 sites (DoD M&SMP 1994). An ADS architecture should consist of:

> *"The structure of the components of a program / system, their interrelationships, and principles and guidelines governing their design and evolution over time." (DoD M&SMP 1994)*

Such an architecture is intended as a reference for use by designers of more detailed architectures, such as a collection of training simulators, or a communications modeling architecture. It should also provide the basis for standardization of terms, and clear identification of roles and responsibilities for distributed simulation components. The ADS architecture is further tasked with defining an interaction paradigm which will support higher degrees of fidelity, interoperability, and numbers of simulation entities than current DoD paradigms.

1.1 Background

We identify three uses of distributed simulation by the military. The test and evaluation community stimulates real devices within a synthetic environment. Warfighter-in-the-loop simulators may be linked together for training purposes. Abstract models, either continuous or discrete event, are used to investigate military systems and doctrine. These different uses require differing amounts of linkage to the progress of real time. The first requires hard real time deadlines, the second requires soft real time deadlines within the several hundred millisecond human perception tolerance. The third is not bound to real time except with regard to producing results within the time frame of the study. Any of these classes of simulations may be distributed across local, high-speed communications (such as FDDI or a shared memory ring), or slower, nation-wide networks.

The ADS community loosely classifies such simulations into one of three categories:

> *"Live simulation involves real people operating real systems. Virtual simulation involves real people operating simulated systems. Constructive simulation involves simulated people operating simulated systems." (DoD M&SMP 1994)*

For our purposes, distributed simulation is then defined as a networked combination of independently executing live, virtual, and constructive entities that share a common view of simulation time, interacting via a prearranged set of data types and events (while outside the current definition, we also include environmental entities, such as cloud models, into the definition).

Two standards currently exist to link together simulations which meet this definition, DIS and ALSP. Below, we briefly describe and critique these approaches in terms of the ADS system goals.

1.2 DIS

Live, virtual, and constructive entities have been successfully integrated using the Distributed Interactive Simulation (DIS) protocol (DIS 1994). DIS provides a standard definition of data to be exchanged between simulations and an unreliable protocol for transmission. Each entity in a simulation continuously produces entity state descriptors, which are broadcast to all other simulation hosts. Specific event types are defined, such as detonations and collisions, which are also broadcast to all simulation hosts. Repeated broadcasts are done to

address dropped packets and late joiners which thus receive up-to-date copies of all entities' published states. Simulation time is loosely tied to the advancement of wallclock time. No causal ordering is required, other than the dropping of packets older than current time minus 250 milliseconds. Dead reckoning of an entity's position is used to minimize the frequency of entity state broadcasts.

DIS has been shown to reliably support small exercises of hundreds of entities. Significant scaling problems have been encountered in exercises with greater numbers of entities, primarily due to the linear increase of entity state broadcasts. As entities are added, two key bottlenecks have been identified: the bandwidth of the connecting network, and the compute cycles spent servicing communications I/O at each host. The largest DIS exercise to date has been the approximately two thousand entities supported in STOW-E, which used an application gateway to compress and reduce data packets at the WAN level (Van Hook et al. 1994).

No standard tools or techniques are currently defined to support the determination of valid interoperability via DIS. Work is in progress to extend the DIS protocol for more sophisticated interactions between entities.

1.3 ALSP

The Aggregate Level Simulation Protocol (ALSP) described in Weatherly et al. (1991) provides an interoperation mechanism for simulations of combat entities modeled at a combat unit level. The protocol provides time synchronization and data transfer between simulations.

The simulations cooperate to maintain a distributed database of public attributes, while private attributes of simulations are maintained within the simulations themselves. Public attributes are written only by their owners, and mechanisms exist to migrate ownership of attributes. Translators are implemented for each simulation to convert private attributes to public attributes, and to provide reflections called ghosts of remotely simulated attributes to the local simulation.

Globally consistent time is provided to the federated simulations by blocking the advance of local simulation time until it is safe to advance. The simulations must provide a value called lookahead which is used to allow a slight difference between local clocks and enable more than one simulation to run concurrently. ALSP is based on a modified Chandy-Misra conservative synchronization algorithm (Chandy and Misra 1981), with null messages for deadlock avoidance. The time advances tend to be very coarse grained.

Gateways are used to interconnect the simulations, passing attribute updates and events, and to coordinate time. The communication mechanism is a reliable broadcast protocol.

ALSP simulations tend to have infrequent time steps, and infrequent exchanges of attributes and events. This coarse grained advancement of simulation time is not appropriate for virtual (i.e. human-in-the-loop) simulation without significant modification. In addition, the synchronization algorithm may not scale to the size required of ADS systems.

1.4 Current Techniques: Summary

DIS and ALSP embody the majority of interoperability standards for independently developed simulations in the DoD community. They allow simulations the freedom to be implemented in any way the developer sees fit, provided the public data and events are generated according to the protocol. Modelers have the additional burden of understanding and implementing the distributed systems aspects of a distributed simulation. Vendors and researchers have developed libraries that are coming into more common usage, but there are no architectural distinctions between the system modeling and data distribution aspects of a distributed simulation.

The level of understanding currently required of modelers is acceptable, since the DIS and ALSP data distribution/management schemes are very simple, and are moderately easy for numerous model developers to implement in an interoperating fashion. However, the distributed system protocols required to meet size, fidelity and reliability requirements for ADS will be substantially more complex.

2 THE ADS ARCHITECTURE

The ADS architecture outlined in this paper describes the results of an ARPA-sponsored project to define a new standard for distributed simulation interoperation which would allow a greater number of entities, operating at a higher level of fidelity, than is possible with current DoD standards. Support for a VV&A process was also required as part of ARPA's interoperability goals.

Since a DoD-wide standard was desired, the presented architecture had the additional requirement of being able to support multiple classes of *federations*, where a federation is defined as a number of independent models sharing a common data dictionary, defined entity actions, and compatible views of simulation time. Federations will likely have differing requirements for data throughput, synchronization, and realtime performance.

The final ARPA goal was to define a flexible architecture, capable of encapsulating continual improvements to technology at the modeling, distributed system infrastructure and networking service layers. This paper addresses only the runtime performance optimizations of the architecture, as outlined in the following three components.

Simulation Runtime Support (SRS): responsible for startup, execution, synchronization and control of a distributed simulation. All communication between ADS simulations is via the SRS. The SRS is the primary focus of this paper.

Execution Manager: performs high-level flow control using exercise context information, such as number of entities versus the fidelity of data. For example, if the SRS has signaled that realtime performance goals are not being met, the Execution Manager may reduce fidelity in non-critical sections of the battlefield, or remove players from the system until the volume of data being distributed is low enough to permit the achievement of performance goals.

Data Collector: responsible for the archiving of data. It is primarily a log of data collected for post-mortem analysis. The volume of such data is usually large, and is separated out from SRS-provided data because of the significantly different performance and availability requirements.

3 ARCHITECTURE DESIGN APPROACH

To meet the ADS goals of flexibility, common code reuse, and evolutionary development of components, a philosophy of strict encapsulation of functionality was followed. A decision was made to clearly separate the simulation and distributed system functionalities in the architecture. This approach is justified as follows:

1) The complexity of distributed data management for ADS will be sufficiently more complex than DIS/ALSP that a common-code library approach may be required to ensure consistent execution of protocols at each host.

2) Since many of the techniques required to make an ADS system execute efficiently are still in the experimental stage, all distributed system functionality should be encapsulated. Additionally, the network and compute resources available are constantly changing. Given that the optimization of a distributed system consists primarily of trading off between local compute resources and network bandwidth, the best optimizations for a given federation will continually evolve.

To address the ADS goal of increased interoperability, we first define interoperability as requiring: 1) a valid set of modeling behaviors (i.e. a fully defined set of data types and entity actions); and 2) synchronized distributed computing operations (i.e. data/event exchange and execution control). DIS currently address these sections via a single protocol definition. This ADS architecture proposes two separate standards for these two distinct areas, where the first is defined by each federation using the ADS architecture, and the second is defined by the SRS component of the architecture.

The level of fidelity and number of entities in an ADS system will require substantial optimizations at the network transport layer, in particular, the sharing of global state. Such optimizations are discussed in: *Section 5.0: Optimizations of the GTD.*

The existing DIS standard was used as the basis for this architecture, extended as required for ADS requirements. A top level comparison of DIS and ADS architectural responsibilities is given in Table 1.

Table 1: Architectural Views of ADS and DIS

DIS	ADS Federations	ADS SRS
Federation-wide definition of data types and events	Same	N/A
Blind-push of entity states into the public view	Same	N/A
Unreliable transport mechanism to publish state/events	Publish mechanism assumed reliable	Many internal transport mechanisms
Dead reckoning used to reduce transmissions of state	*Shared-state usage* information given to the SRS to allow network optimizations	Many classes of network optimizations may be used internally, dynamic adaptations possible
Changes slowly	Will change slowly	May change quickly to support R&D and incorporation of new technology

3.1 Entity Definition

The primary clients of the SRS are *entities*: simulation objects consisting of encapsulated state and behavior. Further, an entity's total state consists of both public and private data.

- Private data is created, changed and maintained strictly by the entity, for use only by that entity.
- Public data is created and changed strictly by the owning entity, for use by any entity.

Entities interact (via the SRS) by:

- Reading other entities' public states, and changing their own public state for others to read. e.g.: position data.
- Generating public events. e.g.: fire events.
- Specialized communication to support distributed sub-entity modeling and/or model infrastructure support. e.g.: invocation of modeling services, experimental interfaces between models.

3.2 Global Ground Truth Data Definition

Global Ground Truth (GT) data is defined as the union of all entities' public states. Further, we assume:

- Events may be considered as GT data (with a short temporal existence).
- Entities at any host may access any GT data.
- The reading and writing of Ground Truth data is the primary communication mechanism for ADS simulations.

Thus, from the viewpoint of the architecture, entities may be treated primarily as producers and consumers of Global Truth Data.

4 SIMULATION RUNTIME SUPPORT

The SRS encapsulates the distributed and real-time support tools required for a federation execution. Where possible, the SRS will be composed of libraries and tools reusable across ADS models, although host-optimized implementations of the SRS are allowed.

The SRS is separated internally into a number of internal components, each of which addresses one specific area of the SRS's responsibilities. Of the three entity interaction types, specialized communications are expected to be a small portion of network traffic, and a relatively simple service to provide. The remainder of this paper focuses on the GTD, an internal SRS component which is responsible for the sharing of global Ground Truth (GT) data.

4.1 The Ground Truth Database (GTD)

The GTD presents a consistent memory model of global Ground Truth data to ADS simulations across distributed hosts. It formally defines a layer between entity behavioral modeling and the mechanics required to distribute information in a distributed system. This allows incremental improvements to how information is distributed, with no changes required at the modeling layer.

Consistent memory models across a network have been presented in many non-ADS systems, such as Linda (Carriero and Gelernter 1986), and ORCA (Bal et al. 1990). It has also been discussed informally in the DIS community. This architecture formalizes the definition of such a paradigm for ADS, and outlines the characteristics of ADS systems which may be used to optimize its performance.

To access GT data, entities register with the GTD the types of data they require. Published data items meeting the stated requirements are then available on a blocking read basis from the GTD. Interrupt-driven access is also provided: an entity may register triggers with the GTD, based on data types or values occurring in the GTD. To publish GT data, entities create a data item in the GTD, and write to it whenever it changes value.

We isolate this class of data (GT) from others in the system because of the stringent performance requirements, the volume (and mapping) of data items, and the differing synchronization techniques used. One key design decision is to keep the amount of data in the GTD as small as possible. This restricts the amount of data passing through a potential performance-critical bottleneck. Examples of data types in the GTD: entity positions, dynamic environmental data. Examples of data types not in the GTD: performance data on the runtime system, data logged only for post-mortem analysis, and specialized communication between models.

5 OPTIMIZATIONS OF THE GTD

Clearly, distributing copies of all GT data to all entities is not feasible for the majority of ADS systems. Instead, we require the GTD to provide the potential for access to all GT data, and only distribute copies of specific GT data items to entities that actually require them. Current technology does not permit an efficient, automated mechanism of doing this, thus ADS simulations are required to define the minimum subset of GT data they require for accurate behavioral modeling. Additional optimizations are possible by defining the simulation's characteristics of how the data is to be used, as well as the characteristics of the GT data items themselves. The set of these application-specific characteristics relates only to their view of shared GT data, and thus is referred to as *shared-state usage declarations.*

The use of application-specific characteristics breaks the pure isolation of simulation and distributed system activities, but must exist as a controlled tradeoff in some federations to meet performance goals. A flexible set of optimization techniques is required, as optimizations will differ across federations in both application and hardware characteristics, and the level of optimizations required. Further, we distinguish between federation-level optimizations, such as minimizing the amount of shared data or restricting how it may be used, and distributed system optimizations, such as only shipping data to where it is required and only updating it when required. As outlined in Table 2, the GTD is only responsible for the second class.

Table 2: An Allocation of Distributed Simulation Optimizations to ADS Components

Federation actions	Simulation actions	Distributed system actions (i.e. the SRS)
Define as small a set of (shared) public data as possible	N/A	N/A
Define set of interests, recommended data accuracy levels, other optimizations	Interest declarations, other shared-state usage declarations	Find matching data (on any host)
Define data dictionary	GT data read/write	Transport data to appropriate hosts via best-available mechanism
Define classes of entity actions and events	Generate event	Transport data to appropriate hosts via best-available mechanism
Pre-runtime	*Runtime*	*Runtime*

The distribution of global GT data is treated as an abstract data sharing problem. Given that the overall system may be viewed as a collection of distributed data sources and data sinks, we assume the following:

- each host in the distributed system will consist of zero or more sources and sinks.
- each data source may lead to multiple data sinks.

- each GT data item is single-writer, i.e. changes to a data item may only occur at one point in the distributed system.
- the mapping of data sources to data sinks is both predictable and relatively static, i.e. the mapping of sources to sinks will not change as frequently as the data items change value.
- a weak consistency data model may be used, i.e. data may be allowed slight inconsistencies in both simulation time and value. Further, the amount of inconsistency allowed will vary across sinks.

For our purposes then, the distribution of GT data reduces to a distributed cache management problem, similar to that of shared memory hardware mechanisms (Chaiken et al. 1991). Further, optimizations to network traffic may be accomplished via the weak consistency and single-writer characteristics of the application.

5.1 A Distributed View of the GTD

The GTD as outlined above may be implemented in any number of ways, dependent on underlying network configurations and data delivery requirements. An excellent measure of design abstraction was to examine the GTD interfaces in terms of implementation on both LAN/WAN-connected distributed processors and shared memory multiprocessors. This allowed us to determine what areas of functionality should be encapsulated within the GTD, as they only related to the distributed nature of the GTD. This section presents a view of the GTD in a generic LAN/WAN environment, intended to illustrate the scaling potential of the GTD.

5.1.1 Local Cache (LCache)

The LCache maintains all dynamically changing global Ground Truth data produced by, or required by, entities at the local host. LCaches are implementation dependent, and could consist of one per LAN, one per processor, one per entity, etc. A LCache's primary characteristic is that it is tightly bound to its local entities, i.e. they may exchange data at high volume rates, with low latency costs. Some form of shared address space is expected. LCaches may also be hierarchical in nature: i.e. a LCache may have other LCaches as clients, thus serving as a directory-based caching scheme.

5.1.2 Interest Manager (IM)

The IM determines what set of data is required by all local entities, unifying their individual requirements into a set of requirements for the local host. Entity registration of triggers/actions is provided to support asynchronous data delivery.

5.1.3 Cache Coherence Mechanism (CCM)

The CCM is responsible for maintaining cache coherence for the GTD across the Local Caches at each host. It contains *Transmission Optimizers*, used to reduce the amount of data sent to remote hosts. The shared-state usage information provided by the simulations is used to decide what data is required at which hosts at what level of resolution. The CCM uses the most appropriate transport mechanism to send locally generated data to hosts which have registered an interest in it. Copies of data items in remote caches are only updated when the source data item changes.

Interest declarations are used as a cache pre-fetch control mechanism, thus avoiding the latency of cache miss and fetch solutions.

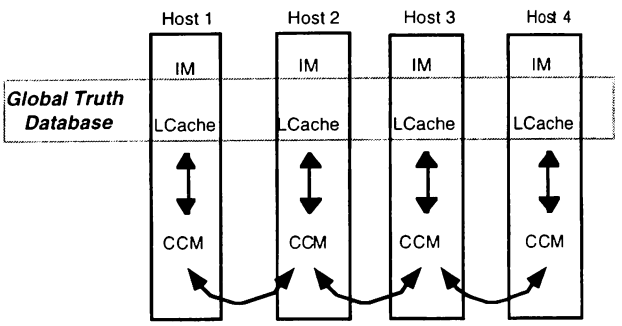

Figure 1: Mapping of GTD Components to Hosts

5.2 Classes of Shared-State Declarations

As stated above, the GTD requires a mechanism for a simulation to describe both the subset of GT data it requires, and characteristics of how the data is produced and used. This section introduces three classes of shared-state declarations that may be used to optimize the network flow of traffic internal to the GTD.

5.2.1 Interest Declaration

In the general case, we assume that not all GT data is required by all entities for the full simulation execution to accurately model their behavior. We also assume it is practical for entities to specify what GT data is required. Clearly, general statements of interest are easier for the modeler to provide, but will result in larger volumes of GT data arriving than is actually required. Precise statements of interest will result in the minimum amount of GT data arriving, but may not be practical to provide due to computational complexity or highly dynamic behavior in the simulation. For any given federation, there will exist an optimal balance between precision of interest statements and available compute cycles or network bandwidth. Note that federations where the majority of GT data is needed at many hosts is

inherently non-scalable. If the amount of required shared data is greater than can be supported via available infrastructure, then that federation must be redefined or more resources acquired.

Given the interest declarations of entities at a host, the GTD is responsible for maintaining at that host the set of GT data items that match the union of the local entities' interests. The GTD must support dynamic changes to that set of GT data items. The GTD must also support changes to the union of entity interests, as the scope of GT data required at a host may change due to changes in an entity's internal state. For example, if an entity has a limited sensing range, it is not interested in entities whose current positions fall outside radius X of the sensing entity's current position.

A flexible predicate-based approach is proposed, where modelers can define either strict or loose predicates, depending on the needs and characteristics of their federation. Predicates must be able to be executed at remote hosts, as the GTD will try to avoid network transmissions of GT data items based on those predicates (i.e. *source-based filtering*). Also note that due to their similarities predicates may be combined at the data source to reduce computational load. It is expected that federations will define a small a set of predicates as is practical, and data classes that lend themselves to simple predicate unification. Also note that super-sets of predicates may be used to further reduce computational loads (at the expense of increased network loading). This may be compared to cache lines, where the data required at a cache will bring with it a larger grain of data unrelated at the application layer, but is related at the OS layer for efficiency.

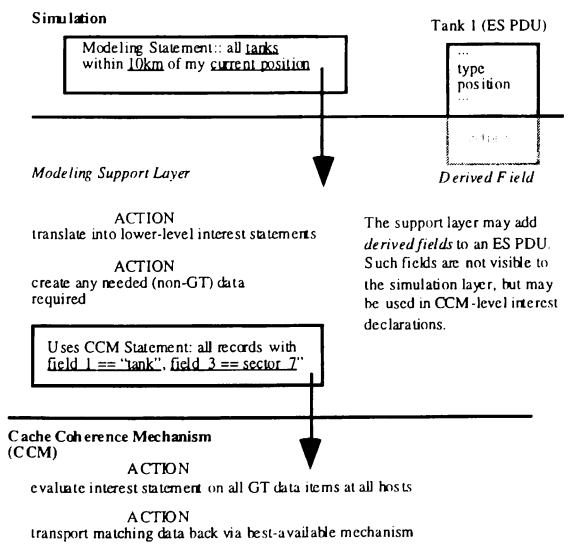

Figure 2. Levels of interest management

Two interest management languages are proposed: one at the modeling support level, and a separate language for internal use by the GTD. Clarity and ease of interest expression at the modeling level requires a dynamic, computationally complex expression syntax. Such expressions may be too expensive for the GTD to evaluate continually on thousands of remote GT data items. An example of interest expression at both the modeling and GTD levels is given in Figure 2.

5.2.2 Variable Resolution Data

In the DIS community, it has been shown that not all consumers of GT data require it at the same level of resolution as it was produced. Given that this holds true for ADS systems, we propose a *weak consistency* data model, where data is allowed to have slight inconsistencies in resolution and time. For example, DIS data is allowed an inconsistency of 250 milliseconds of error before it is considered invalid. Any jitter in this latency will result in positional inaccuracies. Therefore, at any given point in simulation time, it is valid to have the same data item with slightly differing values at multiple hosts in a DIS system. It has also been noted that some DIS models could perform accurately with even lower resolution, and schemes for variable resolution data have been proposed (Cohen 1994, Calvin et al. 1995). Such schemes are generally intended to address the problem of wide-area viewers, which might otherwise defeat interest management optimizations because they can sense across very large areas of the simulated world, and thus are interested in the majority of GT data. We include this principle in ADS via the following approach.

When an entity interest registration is done, the entity is also required to state the preferred resolution for matching GT data items, and a minimum resolution value. If the lower bound of resolution cannot be met, the GTD will throw an exception, which either the entity itself or the Execution Manager component may handle. Dynamic flow control and load balancing through the network may be accomplished, controlled by either the GTD itself and/or the Execution Manager. Entity migration may also occur to alleviate the problem.

For example, an entity simulating the behavior of a satellite may only be able to detect the position of ground-based entities with an accuracy of 100 meters. The GTD thus will only need to update the positions of ground-based entities for the satellite entity when they change by more than 100 meters.

Note that excessive use of this feature places a large computational burden on the GTD, as it must evaluate changes to GT data items at the source to see if they have changed enough to require updating remote copies. While it is possible to unify such comparisons at the source (with possibly overlapping sets of resolution), it is suggested that federations define a small set of valid resolution levels at which simulations subscribe.

5.2.3 Predictive Contracts

Dead Reckoning (DR) (DIS 1994) in the DIS community has been shown to reduce network updates for entities whose changing positions may be predicted. To expand the possible use of DR-like algorithms, we define a parent class: Predictive Contracting.

We postulate the following: any GT data item that tends to change over time in a predictable fashion may be closely matched by a function $f(t)$ which will closely match that data item's probable changing value over time. $f(t)$ is available to the reader, who then only needs the current value of t. Note the GTD may evaluate the $f(t)$ transparently to the reading model, thus providing a best-view of that data item. Given this assumption, the GTD may make the following transmission optimization: changes to GT data items with a predictive contract are only sent if those changes fall outside the range of $f(t)$. Example classes of predictive contracts include: DR, aircraft following a set of waypoints, weather patterns that follow a set of scripted changes.

Three primary advantages are obtained by this approach:

1) Independent processing of a CPU-intensive function is done by the GTD -- thus providing a clean division of tasks for parallel processing.

2) Given that extrapolation of a local copy (at time t) to the current time is done by the GTD, a potentially expensive operation is done only when an entity reads that data item.

3) The best-effort approach provided by variable resolution data is enhanced by a predictive contract which may smooth out variance in resolution introduced by the underlying network.

5.2.4 Ownership Migration

In some instances, it may be advantageous to migrate the source of a data item closer to a sink that requires it very precisely. For example, a missile fired at a plane requires a very accurate view of the plane's location. By migrating ownership of the plane's position value closer to the missile simulation (possibly within the same simulation as the missile), more accurate values may be used without increasing the load of the network. Also note that some simulations may require ownership migration to support V&V -- if an entity moves into a specific region of the battlefield under the control of a different simulation, control of that entity may need to migrate to the new simulation.

5.2.5 Sectorization

The use of spatial sectorization is a well known optimization for parallel simulations (Beckman 1988, Mellon 1994). Van Hook (1994)) offers an approach for applying sectorization to the DIS/ADS problem. Sectorization is the process of breaking simulation space into sectors, then tracking the location of entities to which sector they are currently in. When an entity senses, it only checks against other entities which are currently within their sector (or possibly neighboring sectors). Sectorization may be considered a low cost, first-order pass through entity positions to eliminate entities which are clearly outside sensing range.

From the viewpoint of this ADS architecture, sectorization is a federation-level optimization, in that it determines how data is used. As outlined in Figure 2, a layer in the model may automatically add derived information, such as sector locations, to GT data items. Standard interest expressions may then be done in terms of the derived information.

5.2.6 SRS Optimization Summary

Figure 3 shows the relations between the three SRS optimizations listed above. The federation-level optimization (sectorization), would reduce the amount of GT data being accessed, before interest management is applied.

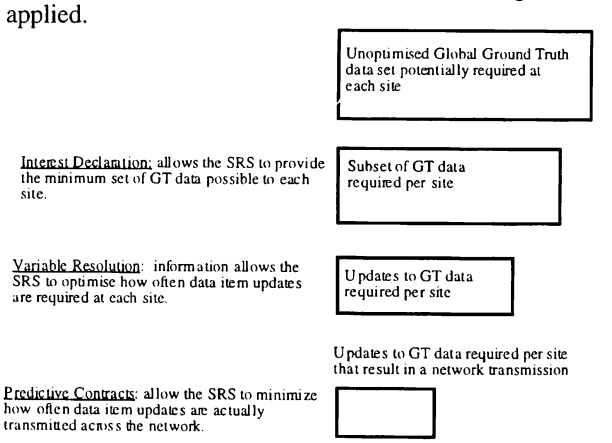

Figure 3: Optimization Summary

6 CONCLUSIONS

This paper has presented an architecture which has the flexibility and modularity to adapt as technology evolves by strict encapsulation of the transport mechanisms and a flexible shared-state view of the interface. It allows for larger numbers of entities interacting at a higher level of fidelity than existing systems. This is primarily accomplished by replacing the broadcast approach of current systems with a minimum data flow approach. Access to a simulation's minimum data requirements is provided to the SRS in the form of interest statements, resolution change sensitivity, and behavior predicting information. As performance depends on application-supplied information, the policy enforced by a particular federation will make or break the realized performance. The SRS must, however, give sufficient feedback about problems and their causes to planners and exercise