# DDM Explained: Lessons for Data Distribution Management Developers and Strategists

**Jennifer Lewis, Khoi Do, Diana Vagiakos**
**Science Applications International Corporation**
**Orlando, FL**
**jennifer.e.lewis@saic.com, khoi.m.do@saic.com, diana.mercedes.vagiakos@saic.com**

## ABSTRACT

The Data Distribution Management (DDM) service provided through the High Level Architecture (HLA) is an extremely powerful data filtration tool. In its simplest incarnations, it can reduce the amount of data a federate must process. With proper design, the service can perform federate functionality architecturally. However, extending a federation's capabilities to use DDM can add layers of conceptual and technical challenges, such as scenario planning, network configuration, and more complicated software implementation. The Army Capabilities Integration Center (ARCIC)-led OmniFusion and NetBCT 2009 experiments used DDM to support an ambitious set of requirements, including modeling large entity counts in a heterogeneous distributed simulation environment. While these experiments only scratched the surface of DDM's full potential, integration testing revealed a wealth of lessons learned that could benefit anyone interested in developing DDM functionality or planning a federation-wide DDM strategy. This paper details these lessons learned and explains them in a way that makes the material approachable for DDM beginners and veterans alike. As a starting point, the paper discusses basic DDM concepts, uses, and Application Programming Interfaces (APIs). However, the paper focuses primarily on more advanced topics, such as ways to design an effective DDM strategy, solutions to specific implementation issues, troubleshooting techniques, and options to improve run-time efficiency. It also examines the cultural issues ARCIC encountered while implementing DDM as well as reasons why DDM is not always a viable solution.

## ABOUT THE AUTHORS

**Jennifer Lewis, CMSP,** is a senior simulation engineer supporting ARCIC's Battle Lab Collaborative Simulation Environment (BLCSE). She holds a Master of Science degree in Computer Science with an emphasis in Telecommunications and Networking from the University of Texas at Dallas. She has designed and implemented network protocols for the telecommunications and defense industries, and spent the past six years transitioning the BLCSE environment from DIS to HLA in support of distributed experimentation.

**Khoi Do** is a senior simulation software engineer. He currently works for Science Applications International Corporation (SAIC) supporting ARCIC's Battle Lab Collaborative Simulation Environment (BLCSE) as the OneSAF lead developer. He also works as a lead technical integration engineer for BLCSE experimental events, supporting the integration of various simulations in the BLCSE HLA environment. He has worked in the military constructive and virtual simulation industry for the past 10 years. He holds a Bachelor of Science degree and is currently pursuing his Master of Science degree in Computer Science from the University of Central Florida.

**Diana Mercedes Vagiakos** is a software engineer supporting ARCIC's Battle Lab Collaborative Simulation Environment. She holds a Bachelors of Science Degree in Engineering Physics from Embry Riddle Aeronautical University. She has been involved in systems engineering of military satellites and missiles system and as a software engineer for modeling and simulation for the past six years.

# DDM Explained: Lessons for Data Distribution Management Developers and Strategists

**Jennifer Lewis, Khoi Do, Diana Vagiakos**
**Science Applications International Corporation**
**Orlando, FL**
**jennifer.e.lewis@saic.com, khoi.m.do@saic.com, diana.mercedes.vagiakos@saic.com**

## INTRODUCTION

The Data Distribution Management (DDM) service provided through the High Level Architecture (HLA) is a powerful data filtration tool. In its simplest forms, it can reduce the amount of data a federate must process. With proper design, the service can perform federate functionality architecturally. However, extending a federation's capabilities to use DDM can add layers of conceptual and technical challenges, such as scenario planning, network configuration, and more complicated software implementation. This paper details the DDM-specific lessons learned from the Army Capabilities Integration Center (ARCIC)-led OmniFusion and NetBCT 2009 experiments. As a starting point, the paper discusses basic DDM concepts, uses, and Application Programming Interfaces (APIs). However, the paper focuses primarily on more advanced topics, such as ways to design an effective DDM strategy, solutions to specific implementation issues, troubleshooting techniques, and options to improve run-time efficiency.

## DDM CONCEPTS

DDM is one of six services provided by the Runtime Infrastructure (RTI). It extends the services of the Declaration Management service by providing subscription and publication, not only to classes, but to the values of the data within those classes. As an example, consider a federation that models triangles. Using the Declaration Management service, a federate can choose whether to subscribe to triangles as a whole. Using DDM, a federate can choose to subscribe only to red triangles.

DDM services may be used by federates to reduce both the transmission and the reception of irrelevant data. Data producers use DDM services to assign properties to their data in terms of user defined spaces. Consumers of data use DDM services to specify their data requirements in terms of these same spaces. The RTI then distributes data from producer to consumer only when these properties and requirements match.

### Routing Spaces and Dimensions

The fundamental concept of DDM is the routing space. Routing spaces are abstract definitions of the data that DDM can filter. Routing spaces contain zero or more dimensions that are used for filtering federation data belonging to that routing space. Both routing spaces and dimensions are defined in the Federation Object Model (FOM). Table 1 defines a generic Triangle routing space and supporting dimensions which could be used in the example "Triangle" federation.

**Table 1- Example Routing Space**

| Routing Space | Dimension | Dimension Type | Dimension Range / Set |
|---|---|---|---|
| Triangle | Type | TriangleTypeEDT | Equilateral, Isosceles, Right [1-3] |
| | Color | ColorEDT | Red, Yellow, Blue [0-2] |

**Regions and Extents**

Federates use routing spaces by declaring regions. A region is an instance of a routing space created by the RTI during runtime. A region can contain multiple extents, which are sets of upper and lower boundaries for each dimension defined in the routing space. The dimension ranges defined in the FOM are human-readable ranges. However, the extent ranges used in RTI APIs are far more granular. The main conceptual difference in the two types of range values is that extent ranges have a constant minimum and maximum. The minimum dimension range for Type is 1and Color is 0. However, the minimum extent range for all dimensions is 0, and the maximum extent value (MAX_EXTENT) for all dimensions is 2^32-1. In essence, the human-readable dimension ranges are used to divide the extent ranges into upper and lower boundary pairs. For example, since there are three valid colors, the lower boundary for Color X is MAX_EXTENT/3*X. The upper boundary is (MAX_EXTENT/3*(X+1))-1.

Please note that the number of divisions is not always clear from the information in the FOM. Developers should also be familiar with their federation's DDM strategy. For example, the Triangle federation may define 10 partitions for the Color dimension, even though the FOM only defines three valid values. This allows for future expansion of the Color dimension range without changing the dimension ranges used for the three existing enumerators.

## USING DDM APIs

Several RTI API calls are required to create, configure and announce regions. The *createRegion* interface takes one parameter, which identifies how many extents the region will contain. Although regions may contain more than one extent, for simplicity, many federates only specify one extent per region. Figure 1 provides sample code to create a Triangle region with one extent from the routing space defined in Table 1.

**Object Handling**

The RTI API *updateAttributeValues* is the same whether the federate uses DDM. However, when DDM is in use, prior to making the call using DDM, the federate will need to associate the object instance with a DDM region. There are two ways to associate an object with a region. First, use the *associateRegionForUpdates* RTI API to associate an object to a region at any point during the lifetime of the object. Second, when the object is created, use the DDM-specific API *registerObjectInstanceWithRegion*. This registers and associates the object in one call. Although functionally the same process, if the application uses the non-DDM API *registerObjectInstance* followed by *associateObjectWithRegion*, the object will temporarily publish to the default region. This means all federates will receive a *discoverObjectInstance* callback for this object in addition to an *attributesInScope* callback for each of the object's attributes. Once the *associateObjectWithRegion* call is made, federates not subscribed to the object's region will then receive *attributeOutOfScope* callbacks for each of the object's attributes. This process introduces significant and unnecessary overhead for other federation members.

If an object changes the region with which it is associated during simulation execution, use the *unassociateRegionForUpdates* RTI API. However, the developer should associate the object with the new region prior to unassociating from the previous region. If an object is not associated with a region for any length of time, it will publish to the default region. Therefore, if the application unassociates the object with its only region, all federates in the federation will receive *attributesInScope* callbacks unnecessarily.

**Figure 1- Example Region Creation**

```
pRtiAmbassador = new RTI::RTIambassador;
RTI::SpaceHandle hSpace = pRtiAmbassador->getRoutingSpaceHandle ("Triangle");
hType = pRtiAmbassador->getDimensionHandle ("Type", handle);
hColor = pRtiAmbassador->getDimensionHandle ("Color", handle);

RTI::Region* pRegion = pRtiAmbassador->createRegion (hSpace, 1);
pRegion->setRangeLowerBound (0, hType, 1431655765);
pRegion->setRangeUpperBound (0, hType, 2863311529);
pRegion->setRangeLowerBound (0, hColor, 1431655765);
pRegion->setRangeUpperBound (0, hColor, 1431655765);
pRtiAmbassador->notifyAboutRegionModification (*pRegion);
```

**Scope Advisories**

The previous section described simple programming mistakes that can trigger unnecessary overhead in the form of attribute scope advisories. Some RTIs allow users to disable scope advisories to reduce federation overhead. However, without scope advisories, federates have no way of knowing if an object instance has stopped sending updates to the federation or if DDM is now filtering out the attribute updates. This is because the RTI triggers the *discoverObjectInstance* and *removeObjectInstance* callbacks only for federates subscribed to a region overlapping the object instance's publication region at the moment the object is instantiated or removed, respectively. Federates not subscribed to a region overlapping the publication region at the moment the object is instantiated or removed will only receive the *attributesInScope* or *attributesOutOfScope* callbacks for the object instance. In addition, if a federate unsubscribes to all regions for a particular object class, the federate will no longer receive *attributesOutOfScope* callbacks for that object class, even for objects that were in scope when the final unsubscribe was called.

**Interaction Handling**

Because interactions are one-time events, sending an interaction with DDM is simple. To send an interaction using DDM, use the *sendInteractionWithRegion* RTI API. Only federates subscribed to a region overlapping the publication region will receive the interaction via the *receiveInteraction* RTI callback. Receiving federates will not receive any other callbacks or notifications. Please note that passing an invalid region to this API will cause all federates subscribed to the interaction class to receive the data.

**Using Multiple Extents**

When possible, many developers prefer to use multiple regions with one extent versus one region with multiple extents. However, some situations require the use of multiple extents. One example is when a detonation needs to be received by both the shooter and target. If two regions are used to transmit the detonation information, one for the target and one for shooter location, the federate will have to send two detonation event messages to the RTI for one event. Using a single region that contains multiple locations will allow the federate to send a single message for a single event.

## TECHNICAL LESSONS LEARNED

The most important lesson learned from more than a year's worth of in-depth DDM work is "Know Your RTI". In many cases, RTIs from different vendors are run-time interchangeable. However, while they provide the same functionality, they do it in very different ways. They have different troubleshooting toolkits, and their internal implementations, especially in the case of DDM, can vary greatly. These differences can significantly impact the overall efficiency of the federation and the federation's technical support teams. The following sections describe specific issues that all revolve around this central lesson.

Please note that because the ACDEP 2009 experiments used the Modeling Architecture for Technology, Research, and Experimentation (MATREX) RTI, these lessons use some MATREX-specific terminology. However, the basic ideas and concepts apply to any HLA-compliant RTI. Readers are encouraged to contact their RTI vendor for more specific information.

**Troubleshooting**

First and foremost, DDM integrators must be well versed in the RTI's troubleshooting toolkit. While this may seem like an obvious statement, more than a year later, ARCIC integrators are still learning new ways to use the MATREX RTI's troubleshooting toolkit. Investing time upfront to thoroughly learn the tools, attend training sessions or discuss options with MATREX's support team would have saved time and frustration during experimentation integration.

Troubleshooting tools are especially important to diagnose DDM problems since most of the work of DDM is handled internally by the RTI. If data is not received as expected, it can be difficult to determine if the error is caused by the sender or the receiver. MATREX offers two primary tools to troubleshoot DDM (and other) issues: RTI logging and RTI console.

**RTI Logging**

The MATREX RTI can create a log identifying all RTI calls made and callbacks received by a federate during runtime. By reviewing the log, developers can determine if the federate is making the expected RTI calls with the expected parameters. Many times, the log can help identify problematic behavior, including, but certainly not limited to, calling the non-DDM APIs and creating DDM regions with invalid extents.

Although it seems straightforward, many DDM interoperability problems ARCIC encountered were a result of federates unexpectedly using a region whose extent ranges include all values for all dimensions, also known as the default region. When the default region is passed to a DDM API, the application behaves as if it is

not using DDM. The following is an example MATREX RTI log setting up a default region with four dimensions:

```
1257523739.156250 | Invoking |
RtiAbstractAmbassador::
notifyAboutRegionModification
(RtiWrapperRegion & theRegion =
_spaceHandle=``11''
_token=`    `22''
_extentSize=``1''
_pExtents=``0260D764''
_pRealRegion=``02E6C8D0''
Extent 0
_numDimensions=``4''
  Dimension: 0, Range=0 to 4294967295
  Dimension: 1, Range=0 to 4294967295
  Dimension: 2, Range=0 to 4294967295
  Dimension: 3, Range=0 to 4294967295
```

By simply scanning RTI logs for regions whose dimensions each range across all extent values, ARCIC integrators could identify use of default regions. Almost without exception, a federate that used a default region in a DDM API was the source of the DDM interoperability problem. In many cases, the problem came from incorrect logic when the federate determined whether it should use DDM. In ARCIC's experience, developers often assume any DDM issue lies in the sometimes complex DDM API calls or in the RTI, itself, rather than first looking at simple logic or coding mistakes in other areas of their code.

**RTI Console**

The MATREX RTI also has a console application integrators can use to more clearly identify what is happening within the RTI. The RTI operator can perform basic checks, such as verifying federates are responsive and joined to the correct federation with the correct FED file. Operators can also verify which object and interaction classes a particular federate has subscribed, a useful piece of information when a federate is not receiving data is expects.

Developers can also use the console to gather a wide array of federation statistics. The statistics most commonly used by ARCIC integrators identify how many messages individual federates have sent or received per DDM routing space and per object/interaction class. The statistics also identify messages that were received by the federate's Local RTI Client (LRC) but subsequently dropped due to subscriptions or DDM data filtering. This information allows integrators to verify how the data is being transmitted, whether the data is reaching the receiver LRC, and if any data is reaching the receiver application. Below are the instructions on how to turn on statistics for the MATREX RTI Console:

1) Run rtiConsole and connect to the federation
2) Navigate to interested federate in the rtiConsole
3) Start collecting three statistics by typing:
   a. "activate_statistic PerClass"
   b. "activate_statistic PerChannel"
   c. "activiate_statistic PerFederate
4) Print the statistic by typing:
   a. "print_statistic PerClass"
   b. "print_statistic PerChannel"
   c. "print_statistic PerFederate"

It is important to note that gathering federation statistics places an extra burden on the RTI. Therefore, users should not activate statistics unnecessarily, from multiple instances of the console, or anytime excellent RTI performance is critical.

**The Most Common Problem**

Correctly specifying a region's machine-readable dimension ranges is critical to a successful DDM implementation. To date, this has been the most common cause of DDM interoperability issues during ARCIC experiment integration.

An important issue to verify is whether every federate is using the same extent scale for each routing space dimension. For example, if one federate uses a human-readable Latitude range of 30-32, and another federate uses 31-32, the extent range that correspond to latitude 31 will be different for the two federates. In this case, even if both federates think they are publishing and subscribing to latitude 31, they will not receive data from the other application. The RTI log provides a simple way to compare the extent ranges of federates' DDM regions.

In addition, simple rounding errors can cause applications to attempt to set a dimension's upper boundary greater than the largest possible unsigned integer value. Because the API *setRangeUpperBound* accepts an unsigned integer as the boundary value, a value larger than 4294967295 will be interpreted as a very small unsigned value. Usually, the rolled-over upper boundary value is less than the lower boundary, which causes the RTI to generate an exception when *notifyAboutRegionModification* is called. In some cases, though, the rolled-over upper boundary value is still greater than the lower boundary. In this case, the only symptom will be that applications do not see data they should see. Using the RTI console, the developer will see that the federate's LRC is receiving the data but

that the LRC is dropping it. The only way to identify the problem is to review the extent values for the dimension ranges in question within the RTI log. The following is an example MATREX RTI log with improperly rounded extent values for dimensions 2 and 3. Although dimension 3 will throw an exception, dimension 2 will not.

```
1257523739.156250 | Invoking |
RtiAbstractAmbassador::
notifyAboutRegionModification
(RtiWrapperRegion & theRegion =
_spaceHandle=``11''
_token=``26''
_extentSize=``1''
_pExtents=``02EAEE3C''
_pRealRegion=``02E6CA80''
Extent 0
_numDimensions=``4''
  Dimension: 0, Range=1521134262 to
1521134262
  Dimension: 1, Range=268435456 to
268435456
  Dimension: 2, Range=0 to 1
  Dimension: 3, Range=3865470565 to 9
```

**Sender Side Filtering**

As mentioned in the previous section, federate LRCs can receive data that they subsequently drop due to DDM data filtering. This is because the MATREX RTI performs DDM's data filtering function within the receiver's LRC. This means that the sender transmits its data to each federate, where the receiving LRC calculates the region intersections and filters out any data to which the federate application has not subscribed. While this approach lightens the receiving federate application's processing load, it does nothing to lessen the bandwidth required for federation operation.

In order to reduce bandwidth, the RTI needs to perform DDM's data filtering function within the sender's LRC. To do this, the MATREX RTI uses optional RID file parameters to create partitions on which data can be directed only to those that subscribe to it. This implementation requires the underlying network to support a many-to-many multicast architecture. It can also negatively impact RTI performance if the partitions are not set up efficiently.

In an attempt to reduce bandwidth for the OmniFusion 2009 experiment, ARCIC integrators configured sender side filtering in the federation RID file without realizing that the underlying Battle Lab Collaborative Simulation Environment (BLCSE) network did not support many-to-many multicast. The result was intermittent multicast traffic from the federation's primary entity modeling site. The Maneuver Battle Lab (MBL) at Ft. Benning, GA, modeled approximately 75% of the OmniFusion 2009's entity count. When using sender-side filtering, federates at other sites periodically removed and then rediscovered MBL's objects. Time between removal and rediscovery was significant, typically an hour or more. During this time of "network silence", the MBL simulations were operational and reported no problems communicating with the RTI.

The BLCSE Network Operations and Security Center (NOSC) troubleshot the problem for several days. However, the RTI's partitioning implementation to support sender side filtering made the network's multicast configuration sufficiently complex that the NOSC was unable to determine the root cause of the problem. Eventually, ARCIC integrators decided against the use of sender side filtering, since hours of network silence from the federation's most populous site was obviously unacceptable.

Once ARCIC integrators discovered the many-to-many multicast requirement, they requested the BLCSE NOSC to support the architecture. However, this is not a simple request. Configuring Bidirectional, the Cisco-recommended many-to-many multicast solution, requires the code on routers and switches to distinguish the traffic requiring such support. In addition to the complexity of the code, itself, network engineers must configure the source routers and switches to identify the traffic type of the data packets they are sending. However, on the BLCSE network, individual sites manage their own Local Area Network's (LAN) routers and switches. Although the NOSC has compelling reasons for this setup, it complicates ARCIC's many-to-many support request that much more.

## COORDINATING RTI AND FEDERATE IMPLEMENTATIONS

The central lesson of "Know Your RTI" does not apply merely to troubleshooting integration problems at the federation level. Because the HLA specification only standardizes the DDM API, the efficiency of a federate's DDM design is highly dependent on the underlying RTI's DDM implementation. This detail is particularly important when considering the fact that a poorly designed DDM implementation can increase network bandwidth and federate processing load. This section describes the MATREX RTI's DDM implementation and how it can affect federates' DDM

design strategies. Developers should be sure to research their specific RTI's DDM implementation for clues to increase DDM efficiency.

**Identifying High-Overhead Callbacks**

In the MATREX RTI, significant DDM overhead is incurred when a federate calls *notifyAboutRegionModification*. During execution of this call, the RTI executive disseminates details of the new or modified region, and each federate's LRC calculates the intersections of the new/modified region with all existing regions. Therefore, every time a federate creates or modifies its own regions, every federate in the federation will incur a performance hit. The significance of the performance hit is proportional to the number of regions defined in the federation as whole, not just the regions the one federate in question as created. That is, the more regions exist in the federation, the more time it will take to calculate all of the intersections.

The benefit to this implementation is that, because all possible intersections are already calculated, the more frequently used DDM calls, such as *updateAttribute* or *sendInteractionWithRegion*, incur no more overhead than their non-DDM counterparts.

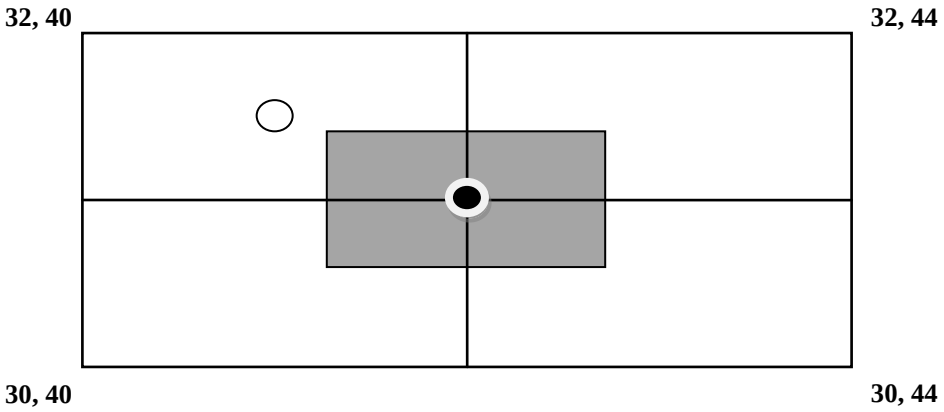
Changing the region with which an object is associated incurs little overhead on its own, either. However, if attribute scope advisories are transmitted, federates will have to process additional RTI callbacks each time an external object changes its associations. These additional callbacks also add to the bandwidth the Wide Area Network (WAN) and LANs will have to carry. Based on this knowledge, ARCIC integrators suggest federates running on the MATREX RTI should minimize:

1. The frequency with which they call *notifyAboutRegionModification* during simulation execution.
2. The number of regions they create.
3. The likelihood that objects will frequently change the regions with which they are associated.

Unfortunately, minimizing one option will often lead to increases in the other options. Therefore, the best way to minimize all options is heavily dependent on the way the federate is used in the federation. For example, in support of #2, federates whose entities are mostly stationary could create only those regions in (or near) which an entity exists at simulation startup. If an entity moved to a new region during simulation execution, the federate would be forced to call *notifyAboutRegionModification*, but the likelihood of this is low. On the other hand, federates modeling fast-moving entities may want to create all possible regions at startup. This design increases the likelihood that unused regions will be created but guarantees the federate will not need to call *notifyAboutRegionModification* during simulation execution.

The following sections discuss some of the basic publication and subscription design options BLCSE federates use. Each design option assumes the federation has defined a DDM playbox divided into a grid of squares. For example, Figure 2 defines at 2x4 degree playbox divided into two latitude partitions and two longitude partitions.

**Figure 2- Example Gridded Playbox Defined by Federation DDM Parameters**



**Center Post Publication**

When using Center Post, federates create point publication regions in the center of each grid square. Entities publish to the point publication region nearest its current location. The design limits the number of possible publication regions, which allows DDM to operate more efficiently. However, the latitude and longitude values used for DDM filtering are not necessarily the same as the entity's physical location. Using the example from Figure 2, a Center Post federate with an entity located at the black circle in the center of grid publish to a DDM region that specified by the Latitude and Longitude of the white circle in the upper left quadrant, more than 100 kilometers from the actual location of the entity. This can make troubleshooting more difficult and limit other federate's DDM designs. For example, a federate could not subscribe to a DDM region defined by the grey rectangle and still see the entity in question, even though the entity is actually located within that range.

**Blanket Post Publication**

When using Blanket Post, federates create publication regions whose Latitude and Longitude dimension ranges are the same as its subscription regions. Depending on their use of Routing Number and Domain, federates may even be able to use the same region for both publication and subscription. Entities publish to the publication region that contains their current location. This design limits the number of possible publication regions while still guaranteeing entities will publish data to their actual physical location. However, publication regions will overlap with more subscriptions than necessary, which can significantly increase network and federate processing load, especially if used by federates modeling large entity counts. Federates using this design should attempt to limit region size as much as possible, especially if sender-side filtering is used for Latitude and Longitude dimensions.

**Minimal Region Subscription**

Minimal Region federates create subscription regions that cover the gridded squares in which it is currently modeling entities. The minimum subscription region size is defined by the experiment's playbox size and number of partitions. However, federates will create larger regions if the region contains an entity with a sensor range that extends beyond the minimum region size. This design limits the number of regions created, which allows DDM to operate more efficiently. However, federates may need to create regions during simulation execution, which causes significant federation-wide run-time overhead.

**Maximum Region Subscription**

Maximum Region Federates use a gridded playbox concept to define subscription region boundaries and create all possible regions prior to the start of simulation execution. Because each subscription region is the minimum region size, federates may subscribe to multiple regions to encompass a single entity's sensor range. This design **e**liminates the possibility of creating or modifying regions during simulation execution, both of which cause significant federation-wide run-time overhead. However, the **f**ederate may create more regions than it needs, reducing the overall efficiency of DDM.

## FEDERATION-WIDE DDM STRATEGY

OmniFusion 2009 was comprised of seven different simulation applications each running multiple instances. Even after correctly and efficiently implementing DDM for one federate, careful DDM strategy planning is required in order use DDM efficiently in the overall federation. Strategy is just as important as implementation. Bad strategy can make things worse. For example, entities that will continually move across a geographic boundary will increase overhead. The following sections describe some of the items that need to be considered when planning a federation's DDM strategy.

**Scenario Planning**

In order to come up with the appropriate size for playbox grids, integrators first need to know the footprint of each federate that will be simulating entities. If the entities are grouped together but cover a large area, the playbox should be divided into large grids that cover an area larger than that federate's footprint. If the entities are located in various pocket areas spread out far away from each other on the map, the playbox grids can be smaller in size. Integrators should also take into account the farthest sensor range of the entities that will be played in the scenario for that federate. The grid dimensions should be at least twice as large as the maximum sensor range to ensure that region subscriptions cover the area that can be sensed by the sensor.

Another important parameter for consideration when planning the DDM playbox grid size is scenario battle movement direction. If it is possible to know the overall direction of movement of entities in the scenario ahead of event execution, integrators should take that into consideration when planning the DDM grid size. They also need to know how far the entities will move before

coming into contact with entities from different federates. The idea is to segregate unwanted network traffic but also keep the publication and subscription to new regions to a minimum. If the general battle direction flow is north and south, the playbox grid size should be longer vertically to account for greater north/south movement on the map.

When sender side filtering is turned on, it is important to plan to have federates modeling entities in similar areas of the map running at the same LAN site. This ensures that multicast data from the local federates stay within the LAN as much as possible. This is not always possible but planning will surely help.

**Federation Data Exchange Planning**

Besides planning the playbox grid partitions, ARCIC integrators used a specific routing space dimension, Routing Number, to partition the HLA network traffic. In this case, the Routing Number dimension included a valid range of 1-24. One strategy ARCIC used was to assign entity side/force to a set number or range of numbers, e.g. OPFOR publishes to routing number 1-7 and BLUFOR publishes to routing number 9-23. The routing number 24 was reserved for Battle Command Management Services (BCMS). If a BLUFOR federate is only interested in interacting with the OPFOR and not other BLUFOR federates, that federate can just subscribe to the OPFOR routing numbers.

Also, when dealing with battle command systems that listen for HLA network traffic, such as BCMS, using routing numbers has proven to be extremely useful in scaling to support large entity counts. Multiple instances of such systems can listen to a smaller range of routing numbers. The entity producing federates will need to publish their entity and interaction data such as salute reports to the correct routing numbers. This design is also useful for data collection applications if performance is an issue or if data collection is interested for certain force side.

Domain is also an available option for segmenting data. Two available domains in the MATREX FOM are Air and Ground. It is available for federates interested in only air or ground entities. However, in this case, it is probably better for federates to subscribe to the entity object type rather than using DDM.

**Federation Publication and Subscription Planning**

As mentioned in earlier sections, there are two basic publication and two subscription design options that were used in ARCIC experimentation. In a larger federation comprised of various type of simulation applications, careful planning is necessary in order to ensure federates work well together based on their DDM implementation.

Center Post publication is ideal for federates modeling large number of entity. All the entities belonging to a partition grid can post updates to the same publication region. Blanket Post publication is ideal for federates modeling small entity counts or modeling entities that contain short sensor ranges. Minimal Regions Subscription is for federates that model entities who are relatively stationary on the map or perform limited movement around the initial startup area. Such entities do not tend to move outside their gridded playbox area, which results in creation of new subscription regions. Finally, Maximum Region subscription is recommended for federates who model highly mobile or fast-moving entities and for federates whose entities are dispersed throughout the DDM playbox.

For federates, such as OneSAF, who usually model a large number of entities but are isolated to an area of the map, it is recommended for that federate to use Center Post publication and Minimum Region subscription. In ARCIC experimentation, there are usually many instances of OneSAF federates, each modeling a group of entities such as a Battalion. For federate, such as EADSIM, who usually models a small set of fast flying aircraft in ARCIC experiments, developers use the Blanket Post publication and Maximum Region subscription.

Location-based filtering has only been mainly used for entity objects. Recently, ARCIC has added the MATREX FOM's Fire Engagement space to its DDM strategy. The implementation mainly deals with the weapon fire and detonation interactions. The weapon fire and detonation interactions are published to the same regions where the shooting federate publishes its entities and the regions where the target of engagement resides. This implementation further reduces network burden since these interactions are classified as reliable messages in the MATREX FOM. Also, it also removes the cases where users may see weapon fire and detonation occurring to locations outside their entity subscription regions on the map where there are no shooters or targets in the vicinity due to those entities being filtered out by DDM.

## CULTURAL CONCERNS

Many users of federates are used to seeing all entities that are on the battlefield and receiving all their

attributes on the network, especially those from the DIS operating environment. When switching to an HLA environment, integrators should explain the need for and complexity of DDM, since its initial introduction may meet some resistance. Some users are used to previous environments where all data was available all the time. In order to achieve larger entity count and meet experiment objectives without degradation to network and simulation applications performance, it is important to communicate to the users why certain data needs to be filtered and to show that it does not affect the experiment result. ARCIC integrators have seen instances where, even if the network can keep up with the data load of more than 25,000 entities, an application cannot keep up with displaying all the entities and processing the data associated with them.

One concern is when federates are running with DDM, not all federates receive the same total federation entity count. Federates which run in cluster mode, such as OneSAF, have various roles. One such role is a Battle Master who has a ground truth view of the battlefield. However, when DDM is enabled, the Battle Master OneSAF view will only see external entities that are in the federate's subscribed regions. When compared to a Battle Master from another OneSAF federate, playing in a different area of the map, the total entity count will be different, and the Battle Masters will see a different picture of external entities. If ARCIC integrators identify a need for site to have a full view of the battlefield, that site runs a federate whose only role is display external entities. That federate is essentially running with DDM turned off and subscribes to everything.

Another concern is users cannot see the DDM location-based regions. Since they know DDM is being used, it is preferable to be able to visualize such boundaries, especially from a Battle Master point of view. To address this, ARCIC has implemented the ability to display publication and subscription regions onto the OneSAF map.

## SUMMARY

DDM is a complex but extremely powerful RTI service. Federate developers should devote sufficient time to understanding their RTI's DDM implementation and designing an efficient DDM architecture that best suits the needs of their RTI. Federation integrators need to be aware of the ways in which different federate RTI implementations affect the efficiency of DDM in the overall federation. Perhaps most importantly, federation strategists need to evaluate the ways in which DDM can assist the overall experiment objectives. While this may seem like a daunting task, the upfront investment needed to learn, implement and configure DDM can help make federations more powerful, efficient and useful.

## REFERENCES

Gupta, P. & Guha R. (2007). *A Comparative Study of Data Distribution Management Algorithms.* The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, Vol. 4, No. 2, 127-146.

Morse, K. & Steinman, J. (1997). *Data Distribution Management in the HLA.* Proceedings of the Simulation Interoperability Standards Organization Spring 1997 Simulation Interoperability Workshop.