Building Scalable Distributed Simulations: Design Patterns for HLA DDM

Björn Möller
bjorn.moller@pitch.se

Martin Johansson
martin.johansson@pitch.se

Fredrik Antelius
fredrik.antelius@pitch.se

Mikael Karlsson
mikael.karlsson@pitch.se

Pitch Technologies
Repslagaregatan 25
582 22 Linköping, Sweden

Keywords: Scalability, HLA, DDM, Design patterns

ABSTRACT: *Over the last decades the size of scenarios in distributed simulation has grown considerably, for example in defense training. There is also a demand for larger number of federates within exercises. This means that federation scalability is an area of growing importance. The developers of HLA foresaw this and introduced not only class-based filtering, but also the HLA Data Distribution Management (DDM) for instance filtering. This is a very general and flexible mechanism for filtering. The challenge for many beginners has been to understand DDM and to develop efficient designs.*

This paper presents some design patterns for DDM and discusses their pros and cons as well as implementation and efficiency. One design pattern is Uniform DDM where all attributes of an object class have the same DDM dimensions available. This makes the use of DDM much easier in federations. Design patterns for filtering based on static properties (like the fuel type of a vehicle) and dynamic properties (like the position of a vehicle) are then covered.

A number of best-practices are also discussed, for example FOM design, handling of objects going in and out of scope as well as the usefulness of advisories. Life cycle challenges, like how to mix federates with and without DDM support are covered.

Finally, some thoughts are given on the design of general and reusable DDM schemes. As an example a number of DDM schemes are proposed for the RPR FOM.

1. Introduction

During the last decade, there has been a growing demand for scalability in distributed simulations. Defense simulation scenarios have grown and become more complex, for example in international civilian-military exercises. The number of simultaneous platform trainers in the same federation is also growing. While early High-Level Architecture (HLA) [1] integrations focused on integrating existing monolithic simulations, today federations are developed in a more modular way, using a larger number of smaller components. And federations developers, like any other community, are always trying to push the envelope.

The lack of scalability, from a bandwidth and CPU perspective, was one of several reasons for developing HLA as a successor of Distributed Interactive Simulation (DIS) [2]. Today, an increasingly common architectural pattern for reusing existing DIS simulations is to create an HLA backbone to which islands of DIS simulations are connected.

1.1. Where are the bottlenecks?

When building large distributed simulations there are many factors that can limit the scalability. In practice, two of the most common are:

Network bandwidth limitations. While Gigabit networks are now common in many Local Area Networks, long distance links still have limited capabilities. Simple math shows that a one-megabit link cannot reasonably carry more than 1250 updates/second of 100 bytes (a common update size for updating entity positions).

CPU limitations. In a distributed simulation it is necessary both to produce data, and to receive and process data from other simulations. Many simulations have limited capability for processing incoming updates, in particular if this feature was added later, rather than in the original design of the system. This problem gets worse as the federation grows.

Consider ten simulations that send 1000 updates/second each. If every simulation subscribes to all of the shared information, they will thus receive 9000 updates/second. Now consider increasing the number of simulations to 100. They will now receive 90 000 updates/s while still only sending 1000 updates/s. The bottleneck for processing incoming data is usually CPU, although graphical sub-systems and databases may also be a constraint.

What program code that causes the CPU constraint is generally not very well understood. Many developers believe that, when bandwidth is abundant, the processing done by a communication framework, like an RTI, is extensive compared to the simulation model. In reality, very little CPU is used by the RTI to transfer information from the network to the receiving simulation. In the next step, for example when a new aircraft position is received, extensive processing may be needed for determining the relative position and angle of that aircraft and all other aircrafts.

Understanding how and when incoming updates are processes may be crucial for optimizing a federation. In some cases, "lazy" strategies may work well, like avoiding calculations until data is actually needed, or until all data for a particular time frame has been received.

1.2. General approaches for improving scalability

There are a number of general approaches for increased scalability. The most obvious one, and easy to implement, is to increase the available bandwidth and CPU resources. Another is to refactor and optimize the system code implementation. Optimized federation design and smart use of services for distribution of data will also increase performance and scalability by allowing infrastructure implementations to perform sender-side filtering and other dynamic optimizations during runtime.

For bandwidth limitations, there are also some common approaches, like compression. This can be handled by the network equipment, or by the sending and receiving CPU. In the latter case, some CPU processing is traded for increased bandwidth.

Bundling is another approach, where several messages are sent in one bundle. This reduces the impact of the networking overhead, since it takes less effort to send ten messages of 100 bytes bundled together as one single 1000-byte message, compared to sending them separately.

In some network topologies it is possible to replace networks hubs, where all local systems share the same bandwidth, with switches, where each combination of senders and receivers can use the full bandwidth.

Beyond these general approaches it is hard to achieve any optimizations without deeper insights into the information exchange, for example what information that is needed by each simulator, and what the characteristics of the simulation data are.

1.3. Add domain specific information for scalability

If more domain specific information is available for the filtering, better scalability may be achieved. The most obvious example is the publish/subscribe scheme used in HLA Declaration Management (DM). Each HLA federate subscribes to the object classes and attributes it is interested in. The RTI only delivers updates for a particular class and attributes to interested federates. The same scheme is also used for interactions. This scheme improves scalability when different federates have different interests. If all federates subscribe to all classes, little optimization can be achieved.

To further optimize the information that is delivered to each federate, it may be desirable to deliver data only for a subset of the instances of a given class. An aircraft simulator may only be interested in other aircraft in the same geographical area. A command and control simulation may only require the positions of ground vehicles belonging to a certain force. If such criteria can be provided to the RTI, it is possible to reduce how much data that a federate needs to process and how much data that needs to be delivered over the network. There is a service group called Data Distribution Management (DDM) in HLA that provides this type of filtering. This paper seeks to describe how to use DDM in practical applications and discuss the optimal way to use it.

It shall also be mentioned that there are several other domain-specific approaches. One example is dynamic aggregation and de-aggregation. In this case we chose to describe a number of entities as an aggregate, for example a platoon, battalion or brigade. When required, for example during a particular phase of the scenario, the aggregate is de-aggregated into a larger number of entities. This assumes that not all aggregates need to be de-aggregated all of the time, in which case no additional scalability is gained. Aggregation and de-aggregation is extensively used in command and control exercises.

Another example is to use predictive techniques like dead-reckoning. A sender can avoid sending messages when dead-reckoned values, on the receiver side, will be close enough to the real value. This is used in the DIS and Real-Time Platform Reference FOM (RPR FOM) [3,4] standards for the exchanging spatial data for physical entities.

2. Overview of HLA DDM

This section contains a brief introduction to HLA DDM, that also forms a basis for the design patterns.

2.1. General principle

The HLA Data Distribution Management services enables developers of a federation to perform filtering on any data that they need. Figure 1 shows how DDM extends upon class-based subscriptions.

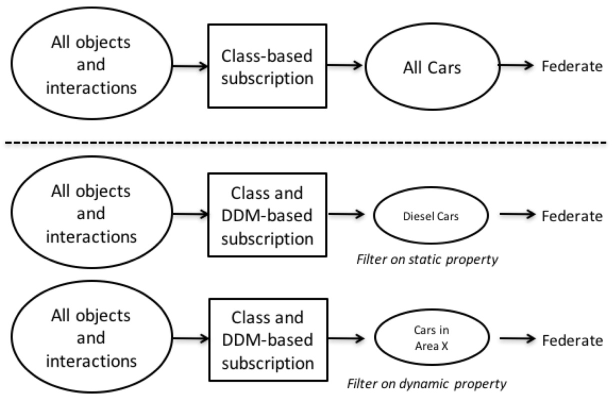

All objects and interactions
Class-based subscription
All Cars
Federate
All objects and interactions
Class and DDM-based subscription
Diesel Cars
Federate
Filter on static property
All objects and interactions
Class and DDM-based subscription
Cars in Area X
Federate
Filter on dynamic property

Figure 1: Subscription without and with DDM

If class-based subscriptions are used, a federate can choose to subscribe, for example, to all cars, but avoid to subscribe to aircrafts. When DDM is added, the federate can subscribe to diesel cars only, or cars in a selected geographical area. In the first case, filtering is done based on a static property of a car instance. In the second case, filtering is done based on a dynamic property of a car instance.

2.2. The normalization function

The key to understanding DDM is the Normalization Function. The purpose of the Normalization Function is to map any domain specific data in a federation into data with a generic format, in this case integer ranges, that the RTI can use. The RTI cannot reasonably be required to have any knowledge about a particular application domain. Detailed aspects like data types, enumerations, geospatial positioning need to be hidden. The usage of the Normalization function is shown in Figure 2.

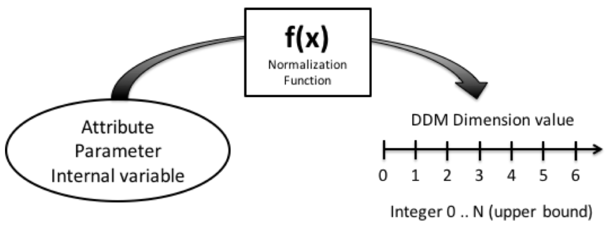


Figure 2: The Normalization function

The Normalization function takes input data, which could be attribute values, parameters, or any variable in a program, and converts it into an integer range in a user-defined Dimension. It is up to the developer to specify and implement a normalization function that meets his needs.

Consider the case of different types of fuel. The developer can introduce a Fuel Type Dimension. All types of fuel that are used are then mapped into Ranges in this dimension, as shown in figure 3. When sending updates and interactions, or when subscribing, a DDM Region is used which specifies one or more Ranges, each one related to a Dimension.

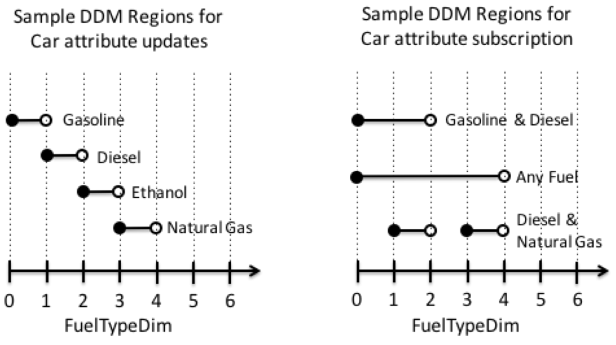
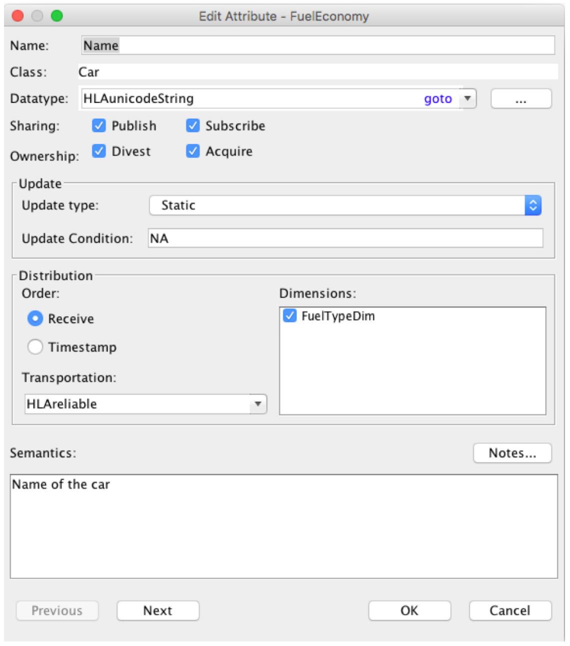


Figure 3: Sample Regions in the Fuel Type Dimension

In the left part of Figure 3 we can see that Gasoline is specified as the range [0..1), meaning that the range goes from 0 up to, but not including, 1. Diesel is specified as the Range [1..2). The value goes up to 4, which is the Dimension Upper Bound. The right part of Figure 3 shows regions that are used for subscribing, which will be covered in the next section.

2.3. Filtering at runtime

For each attribute and interaction class that needs to use DDM, the available Dimensions must be specified. Figure 4 shows how the Fuel Type Dimension is specified for one attribute of the Car class.


Edit Attribute - FuelEconomy

Name:	Name
Class:	Car
Datatype:	HLAunicodeString
Sharing:	✓ Publish ✓ Subscribe
Ownership:	✓ Divest ✓ Acquire
Update type:	Static
Update Condition:	NA
Order:	● Receive ○ Timestamp
Transportation:	HLAreliable
Dimensions:	✓ FuelTypeDim
Semantics:	Name of the car

Figure 4: Specifying available dimensions

To perform filtering at runtime, Regions must be used as follows:

For the **federate that updates** an attribute (or sends an interaction), a Region shall be associated. As an example, when updating an attribute of a car, the Diesel Region (see Figure 3, left side) could be specified.

For the **federate that subscribes** to that attribute (or interaction), a subscription Region shall be provided. As an example, the Gasoline & Diesel Region (see Figure 3, right side) can be used.

The RTI will then compare these Regions when the update is sent. If the Regions of the update and the subscription **overlap**, then the update will be delivered.

To conclude, the DDM services enable federation developers to filter on any data that they have available. Any data can be used as input to Normalization Functions, which are used to determine Regions in one or more dimensions. The subscription requirements, expressed as Regions, are compared to the Regions of the updates or interactions.

2.4. Scope and Advisories

Consider a federation using DDM, where a federate that displays a map subscribes to gasoline & diesel cars. The federate then changes the subscription to gasoline cars only. This means that no more updates are received for diesel cars. This is known as the diesel cars going **out of scope**. All these cars will now freeze if displayed on a map display. In order to make it easier for the map display federate to handle this, for example by removing, or greying out these cars on the map, the RTI sends **out-of-scope** callbacks to the federate. Should they later come into scope, there are corresponding **in-scope** callbacks. These callbacks are called advisories. The updating federate gets the **turn-updates-on** and **turn-updates-off** advisories, to let it know if there are any federates that will receive any updates that it makes. In the above example, out-of-scope happens since the subscription region was changed. It may be just as common that an object and its attributes go out of scope since the updating federate changed the associated DDM regions.

2.5. Why isn't DDM more widely used?

DDM has proven very useful in some large federations. All major RTIs support it. Still, it is not extensively used. Some reasons for this may be:

- Many modest-size federations do not have a scalability problem.
- Federations with legacy simulations, where the source code may not be available, cannot implement DDM in

all participating federates. This limits the degree of filtering that can be made and thus the value of DDM.

- The DDM configuration – dimensions, ranges and normalization functions – needs to be agreed upon in the entire federation. This limits the opportunity to reuse a federate in a different federation, or at least requires coordination across organizations or standardization.
- It is not very clear from the HLA specification, or other documents, how to implement good DDM. The key to a good DDM design is the normalization function, which only has a short and formal description in the standard.
- The most commonly used reference FOM in the defense domain, the Real-time Platform Reference FOM (RPR FOM), does not provide any Dimensions or Normalization Functions. One explanation for this is that the RPR FOM maps to the older DIS standard, where no DDM is available.

The authors of this paper hope to make some improvements with respect to the two latter reasons.

3. Design Patterns for DDM

Design patterns are the re-usable form of a solution to a design problem. The idea was first introduced by the architect Christopher Alexander [5] for architecture and urban design. He describes a pattern as follows: "Each pattern describes a problem that occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice." The concept of design patterns has since been used in many disciplines, not the least computer science, where it has made a big impact through the works of Erich Gamma et.al., also known as the "Gang of Four" [6].

3.1. Two general recommendations

Before looking at the design patterns, two recommendations are provided. The first and most important recommendation to DDM designers is to **keep the DDM design simple**. If just one simple Dimension with ten regions is used, and the data is evenly distributed across these regions, there is a potential of removing, on average, 90 percent of the incoming data for each federate, which means a big leap in scalability. This assumes that the most optimal Normalization Function is used for the particular domain and scenario.

A second, related recommendation, is to **avoid inventing "nice-to-have" DDM schemes** in a federation. Make sure that the design patterns truly divide the simulation data space into partitions that are meaningful to participating federates. Each additional DDM scheme that is required in a federation imposes some work on federate developers.

3.2. Pattern 1: Uniform DDM

Let's assume that we use different Dimensions and/or Normalization functions for each attribute of an object class. The result will be that different attributes will go in and out of scope at different times. This becomes very complicated to handle in a program. Parts of the remote object are up to date and parts are out of date. The parts that are up to date varies all of the time. A clearer design is achieved if all attributes go in and out of scope at the same time. The design pattern Uniform DDM is defined as follows:

"All attributes of a given object class shall have the same available Dimensions. Federates that update any of these attributes, shall provide regions for all Dimensions, using the Normalization Function associated with each attribute."

For the Fuel Type example this means that the entire Car instance, with all attributes, will be either in or out of scope.

3.3. Pattern 2: Static DDM

The Fuel Type pattern, described in the previous chapter, is a good example of the Static DDM pattern. There is a fixed set of regions along one Dimension, in this case regions for Gasoline, Diesel, Ethanol and Natural Gas along the Fuel Type Dimension. These regions are never modified. Each Car instance is associated with one region. This association never changes.

Another example is the Force Identifier in the RPR FOM. Platform instances can be associated with Regions connected to the Force Identifier Dimension. It will then be easy to subscribe to entities that are associated with the Friendly, but not the Opposing forces.

The Static DDM pattern, when applied to a class, is defined as follows:

"A fixed set of Regions with static Ranges are used. The object instance attributes are associated with the same Region throughout the federation execution."

This pattern is very efficient since there is no need for the RTI to recalculate the region overlaps after the initial calculation. The limitation is that the Normalization Function needs to be based on an input variable that is constant, like the fuel type of a car.

3.4. Pattern 3: Dynamic Checkerboard DDM

In many cases, we want to filter on geographical position. Since the position of a car is not expected to be constant during the federation execution, we cannot use the previous design pattern. For best efficiency we still want to use a fixed set of Regions. We design it so that it creates a grid or a "checkerboard" across the map, as shown in Figure 5.

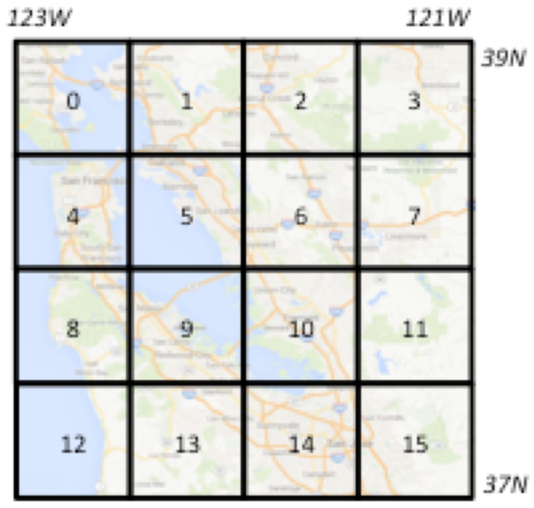


Figure 5: Checkerboard DDM regions

In this case we use a four by four grid and get sixteen Regions, from [0..1) to [15..16). Each Car instance is associated with a Region based on its position. A subscribing federate can select which Regions that it is interested in. Figure 6 shows how a car is associated with square 5. A federate has a subscribing region of square 0, 1, 4, 5 and will thus receive updates for that car.

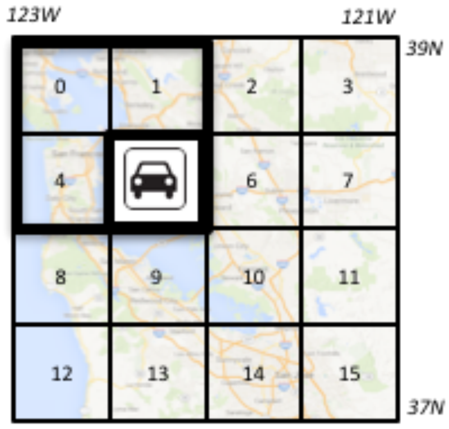


Figure 6: Checkerboard regions for updates and subscriptions

The Dynamic Checkerboard DDM pattern, when applied to a class is defined as:

"A fixed set of Regions with static Ranges are used. The object instance attributes are associated with one Region at a time. This association may change throughout the federation execution."

This patterns handles input variables that change over time, like the position of a car. Even with modest grid sizes, the filtering can be very powerful. A ten by ten grid can give an average update reduction of 99 percent. The recalculation of the Region overlap is usually very limited, and is caused by federates changing their subscriptions. You may also design other sets of static Regions, for example the States of USA.

The limitation is that it may be difficult to design a grid that fits any scenario. The example in Figure 5 has one square (number four) that contains San Francisco, which could be expected to contain considerably more cars than other squares.

3.5. Pattern 4: Dynamic Floating DDM

In order to make the filtering more exact than with the fixed checkerboard grid, we can tailor the region to each actual car. We will now introduce one Latitude Dimension and one Longitude Dimension. We may for example use a Normalization Function that maps a latitude range to a Latitude dimension with a Dimension Upper Bound of 100, i.e. values of [0..100). Each car is associated with its own Region that is constantly updated to match the position of the car. This is shown in Figure 6.

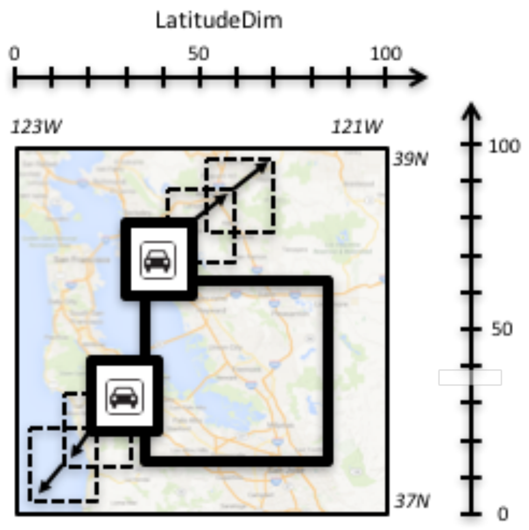


Figure 7: Dynamic floating regions

In the figure we can see two smaller Regions associated with two cars that move. There is also a larger Region, used by a subscribing federate. There is an overlap between the subscribing Regions and the two Regions associated with the cars. Note that the RTI has no insight in exactly where in the updating Region a car is located. If there is even the slightest overlap between the Regions of the updating federate and the subscriber, updates will be passed to the subscriber.

The Dynamic Floating DDM pattern, when applied to a class is defined as:

"One Region per object instance is used. The attributes of the object instance are associated with this Region. The Ranges of this Region may change throughout the federation execution".

Determining the size of the Regions used when updating the cars may be a challenge. They should be reasonably large, so that we do not need to modify them too often. At the same time, they should be kept small, in order to get higher accuracy when another federate subscribes to a region.

As a rule of thumb, consider the update rate for the cars and make sure that the regions are not updated more often than this, since changing the filtering conditions more often than sending data is suboptimal. However, the biggest limitation with this design pattern may be that it uses a lot of Regions where the Ranges are frequently recalculated. This may cause recalculation within the Local RTI Component across all federates.

3.6. Comparison

The following table summarizes the three design patterns.

Pattern	**Instance Attributes**	**Regions**
Static DDM	Statically associated with one region	Fixed set of regions with static, predefined ranges
Dynamic Checkerboard DDM	Associated to one region at a time. This association may change to other regions.	Fixed set of regions with static, predefined ranges
Dynamic Floating DDM	Each object instance is associated with its own region	One region per object instance. Ranges in the regions may change dynamically.

There are of course many other potential design patterns for DDM. The purpose of this section is to show a few proven patterns that federation developers can start with and apply as is.

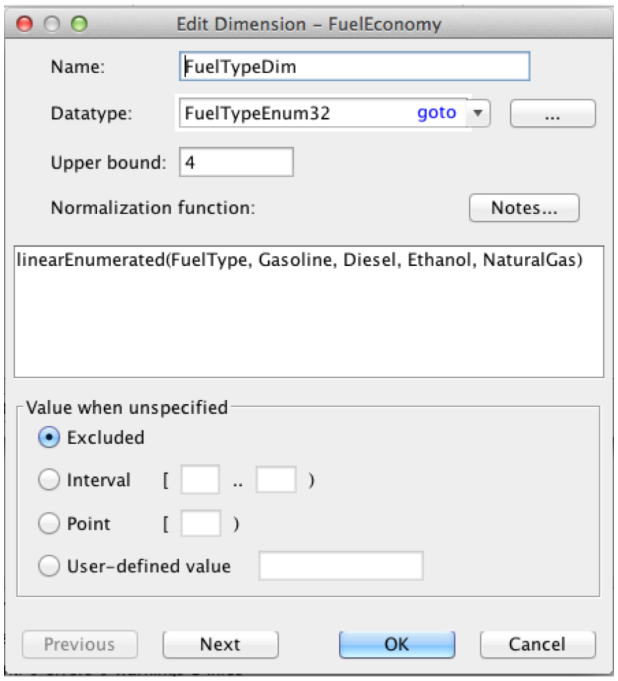
It should also be noted that the discussion above builds on the assumption that it is computationally more expensive to modify regions (and thus recalculate region overlap) than to change the association of attributes to region.

4. Tool Support

This section shows how the implementation of DDM is supported in three commercial tools.

4.1. Developing FOMs with DDM

In Pitch Visual OMT, Dimensions are specified in a dedicated editor, as shown in the example in Figure 8.


Edit Dimension – FuelEconomy

Name: FuelTypeDim

Datatype: FuelTypeEnum32

Upper bound: 4

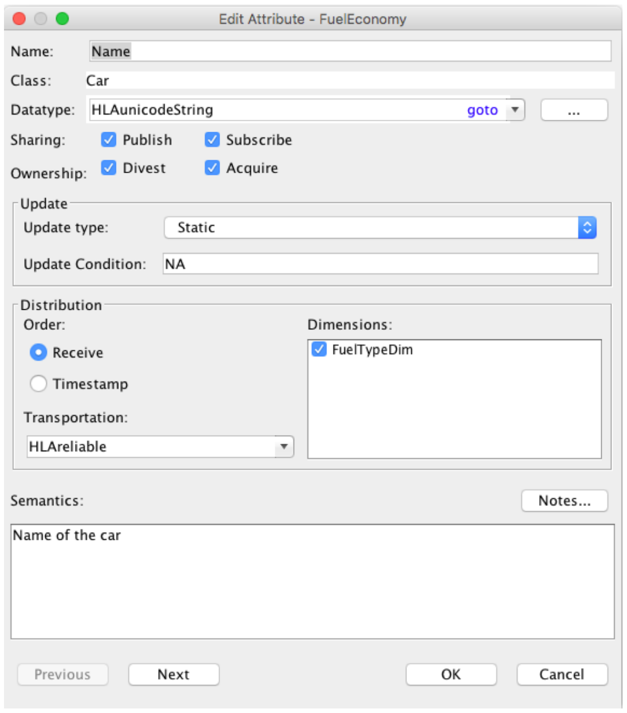
Normalization function:

linearEnumerated(FuelType, Gasoline, Diesel, Ethanol, NaturalGas)

Value when unspecified: [✓ Excluded] Interval Point User-defined value

Figure 8: Defining a Dimension

When Dimensions have been specified they can be selected for attributes and interactions, as shown in the example in Figure 9.



Edit Attribute - FuelEconomy

Name: Name

Class: Car

Datatype: HLAunicodeString

Sharing: [✓ Publish] [✓ Subscribe]

Ownership: [✓ Divest] [✓ Acquire]

Update

Update type: Static

Update Condition: NA

Distribution

Order: [✓ Receive] Timestamp

Transportation: HLAreliable

Dimensions: [✓ FuelTypeDim]

Semantics:

Name of the car

Figure 9: Specifying Dimensions for an Attribute

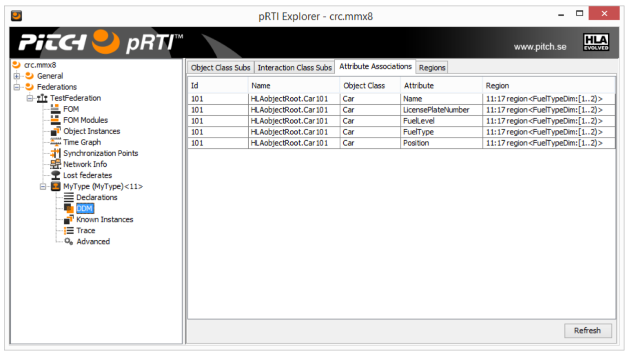
Pitch Visual OMT also provides an advanced tool that makes it possible to quickly specify Uniform DDM in large FOMs with many classes and attributes.

4.2. Federate Development in C++ and Java

Pitch Developer Studio generates C++ and Java code based on FOMs. It explicitly supports DDM based on the above design patterns.

4.3. Verifying and Debugging at Runtime

Pitch pRTI offers the ability to inspect and verify the DDM properties used in a federation. Figure 10 illustrates how to inspect the Region associations of Car attributes.



pRTI Explorer - crc.mmx8

Id	Name	Object Class	Attribute	Region
101	HLAobjectRoot.Car101	Car	Name	11:17 region<FuelTypeDim:[1..2)>
101	HLAobjectRoot.Car101	Car	LicensePlateNumber	11:17 region<FuelTypeDim:[1..2)>
101	HLAobjectRoot.Car101	Car	FuelLevel	11:17 region<FuelTypeDim:[1..2)>
101	HLAobjectRoot.Car101	Car	FuelType	11:17 region<FuelTypeDim:[1..2)>
101	HLAobjectRoot.Car101	Car	Position	11:17 region<FuelTypeDim:[1..2)>

Figure 10: Region to Attribute inspection in Pitch pRTI

There are also views to inspect the Ranges of all Regions as well as the Regions used by subscribing federates.

5. DDM for RPR FOM

A standard scheme for RPR FOM would be highly desirable to facilitate the development of large federations in the defense and security domain. As previously mentioned, DDM and Normalization Functions should be kept simple. Based on discussions around the RPR FOM and the extension in the NATO Education and Training Network (NETN) design, the following four DDM approaches would be the most helpful ones for many federations. Note that it should be possible to freely combine them.

Position using dimensions for Latitude and Longitude. The most obvious filtering criteria for many simulations is to get information only from a selected part of the battlefield.

Force Identifier. It is very common for a simulation to only process information about platforms for selected forces.

Domain (Air/Ground/Sea). This is another commonly used discriminator. It is particularly interesting when used together with the two previous approaches.

Echelon. This is important in particular in command and control applications.

6. Challenges and Considerations

This section discusses some challenges when implementing DDM as well as some particular aspects that need to be considered.

6.1. Mixing federates that use and don't use DDM

There may be challenges when building federations where not all federates use DDM. The challenge for federates that use DDM is that federates that do not use DDM will send data without associated Regions. This data will be sent in the Default Region, which is a region that matches all other regions. Federates that subscribe using DDM will still need to handle incoming data that would otherwise be filtered, thereby reducing the performance advantage that DDM would normally provide. A workaround can be to add Regions for older federates using RTI plug-ins or using a federation-to-federation bridge.

Federates that subscribe without using DDM will receive all data that is sent, regardless of any regions used by the sending federates. The challenge for these subscribing federates is that they may not able to process all the incoming data in a federation where DDM is assumed since they don't take advantage of the load-reducing filtering that DDM-enabled federates do.

6.2. Designing reusable Normalization Functions

When implementing DDM in a federate it is an advantage if it can be reused in different federations. Consider design pattern 3 and 4 in this paper. As described above, they are hard-coded to a particular geographical area. A better idea is to make them configurable, or to come up with a generic Normalization Function.

6.3. Understand the scenario

In many cases, the optimal Normalization Function depends on the scenario. Consider a scenario using checkerboard DDM for a large number of entities in a large geographical area. Now consider a similar scenario in a small geographical area. If we need to filter out five percent of the entities, then the size of the grid needs to be different in these two cases. Two possible solutions are to either use variable parameters for the grid resolution, or to have fixed, fine grained update regions, and vary the size of the subscribing regions.

Similar cases may occur for other types of Normalization Functions. In some scenarios, subscribing federates may only need to distinguish between entities in the air, land and sea domain, other scenarios need to distinguish between friendly, neutral and opposing entities. For some scenarios, the platform type may even be of interest.

Federation developers need to strike a balance between these requirements and the recommendation earlier in this paper, to avoid inventing “nice-to-have” DDM schemes.

6.4. Know your RTI

Another challenge is that different RTIs implement DDM in different ways. This paper will not go into this topic, since it would require a paper on its own. Developers are encouraged to study the specifics and configuration options of RTIs they are considering.

6.5. Towards cutting-edge scalability

This paper focuses on a set of good design patterns for getting started with DDM. Advanced DDM scheme developers may consider studying work done in JLVC and NCTE [7] that implements a world-wide DDM grid using both geographical and other dimensions, resulting in 55 million regions.

7. Conclusions

The size of simulation scenarios has grown considerably over the years. DDM is one of the most important tools for achieving scalability. This paper presents some experiences, some advice and four design patterns as a starting point for developers that want to start exploring DDM.

References

[1] IEEE: "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)", IEEE Std 1516-2010, IEEE Std 1516.1-2010, and IEEE Std 1516.2-2010, www.ieee.org, August 2010.

[2] IEEE: “IEEE Standard for Distributed Interactive Simulations”, IEEE Std 1278.1-2012, www.ieee.org, December 2012

[3] SISO: “SISO-STD-001.1-2015, Standard for Real-time Platform Reference Federation Object Model (RPR FOM)”, www.sisostds.org, September 2015.

[4] Björn Möller et al.: “RPR FOM 2.0: A Federation Object Model for Defense Simulations”, 2014 Fall Simulation Interoperability Workshop, (paper 14F-SIW-039), Orlando, FL, 2014.

[5] Alexander, Christopher (1977). “A Pattern Language: Towns, Buildings, Construction”. Oxford University Press. ISBN 0-19-501919-9.

[6] Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1995). “Design Patterns: Elements of Reusable Object-Oriented Software”. Addison-Wesley. ISBN 0-201-63361-2.

[7] Andy Ceranowicz et al: “Revisiting Interest Management”, 2014 Fall Simulation Interoperability Workshop, (paper 14F-SIW-041), Orlando, FL, 2014.

Author Biographies

BJÖRN MÖLLER is the Vice President and co-founder of Pitch Technologies. He leads the development of Pitch's products. He has more than twenty-five years of experience in high-tech R&D companies, with an international profile in areas such as modeling and simulation, artificial intelligence and web-based collaboration. Björn Möller holds a M.Sc. in Computer Science and Technology after studies at Linköping University, Sweden, and Imperial College, London. He is currently serving as the chairman of the Space FOM Product Development group and the vice chairman of the SISO HLA Evolved Product Development Group. He was recently the chairman of the SISO RPR FOM Product Development Group.

FREDRIK ANTELIUS is a Senior Software Architect at Pitch and is a major contributor to several commercial HLA products, including Pitch Developer Studio, Pitch Recorder, Pitch Commander and Pitch Visual OMT. He holds an M.Sc. in Computer Science and Technology from Linköping University, Sweden.

MARTIN JOHANSSON is Systems Developer at Pitch Technologies and is a major contributor to several commercial HLA products such as Pitch Developer Studio and Pitch Visual OMT 2.0. He studied computer science and technology at Linköping University, Sweden.

MIKAEL KARLSSON is the Infrastructure Chief Architect at Pitch overseeing the world's first certified HLA IEEE 1516 RTI as well as the first certified commercial RTI for HLA 1.3. He has more than ten years of experience of developing simulation infrastructures based on HLA as well as earlier standards. He also serves on several HLA standards and working groups. He studied Computer Science at Linköping University, Sweden.