Implementing the Data Distribution Management Services in the SIP-RTI

by

David A. Tudino,
BEng.

A Thesis Submitted to the Faculty of Graduate Studies and Research In Partial Fulfillment of the Requirements For the Degree of

Master of Applied Science

Ottawa-Carleton Institute for Electrical Engineering
Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada

Your file Votre référence
ISBN: 978-0-494-33671-7
Our file Notre référence
ISBN: 978-0-494-33671-7

Abstract

The SIP-RTI was developed to address the issue of interoperability between Run-Time Infrastructures (RTI), by building a Local RTI Component on a Conferencing Infrastructure. This open communication layer consists of a conference-based communication model, whereby applications can participate via attendee objects. The SIP-RTI also makes use of the Session Initiation Protocol to facilitate the initial connection among nodes. An important aspect of the RTI which is often overlooked by non-commercial RTIs are the Data Distribution Management services. These services facilitate communication among federates by filtering messages based on defined regions of interest. This simplifies checking for messages that are unnecessary at the receiving federate end, as well as reducing network communication by sending messages only to federates that need updates.

This research examines the implementation requirements and implications of the Data Distribution Management services. Much of the current research on Data Distribution Management is involved with improving matching algorithms. While the matching algorithm is an important component of the Data Distribution Management services, it is not the only component. This research looks at the how to implement the actual services, along with a matching algorithm, using the IEEE 1516 version of the High Level Architecture RTI specification. Region information sharing is explored to determine how to minimize the amount of data being transferred between RTI nodes while still providing sufficient information for overlap calculations. A novel technique, called Dynamic Region Reduction, for reducing the number of regions to

examine while performing the matching is introduced. Finally, the Conferencing Infrastructure was modified to streamline the filtering requirements of the Data Distribution Management publication and subscription for Attendees within a conference. Tags are added to conference Attendees to provide this filtering. A test federation was created to demonstrate that the Data Distribution Management services were functioning as expected.

Acknowledgements

I would like to thank Professor Pearce for his support and guidance throughout my development and writing process.

My appreciation goes out to Claude Van Ham as well, for answering my questions and offering insights into the inner-workings of the SIP-RTI.

Finally, I would also like to thank my family and friends for their support and understanding over the course of my Graduate studies.

Contents

List of Figures

Chapter 1

Introduction

The High Level Architecture (HLA) was developed by the US Department of Defense (DoD) to address the growing need for simulations in training, and to increase the value of simulation development efforts by using a standardized software architecture while keeping development costs as low as possible [1]. The HLA specification consists of a software Application Programming Interface (API) for a middleware application called the Run-Time Infrastructure (RTI). The RTI consists of six services, one of which is the Data Distribution Management (DDM) services. The DDM services provide a means of filtering messages sent between simulation components ("federates") based on the intersection of regions of interest within an application-specific coordinate space, defined within the RTI for the distributed simulation ("federation"). This filtering serves a dual-purpose of reducing the network traffic by reducing the number of messages when implemented as sender-side filtering.

Using the Session Initiation Protocol (SIP) [2], and an open conferencing model for communication, the SIP-RTI was developed to address some of the interoperability short comings of the currently available RTIs [3, 4]. While federates following the HLA rules will work on any commercially available RTI, a federation can only function if all of the connected nodes are using the same vendor's RTI. The HLA defines an API

that the RTI must implement, however does not require interoperability between RTI implementations. The SIP-RTI addresses this issue by creating a common lower-layer, which serves as a middleware for allowing local RTI components to communicate with each other using a software conferencing paradigm. The SIP-RTI provides an initial RTI implementation supporting the Declaration Management (DM) and Ownership Management (OM) services. One of the suggestions for future development on the SIP-RTI was the inclusion of the Data Distribution Management (DDM) services. These services are often neglected in academic RTIs [5, 6], although they provide a valuable advantage to the RTI.

Current research based on the DDM services explores improvements in region matching algorithms, and falls into two broad camps: Grid-Based algorithms and Region-Based algorithms. An overview of these research streams is provided in Chapter 2, along with the necessary background information on the DDM services. Published research in both of these areas describe the benefits of their approach, and compare and contrast the performance between different algorithms. Little attention is given to the supporting framework of the matching algorithm; the DDM services themselves.

The thesis of this research is that Data Distribution Management is more than just an efficient matching algorithm. The DDM services provide federates with a means of identifying their narrow range of interest on object classes through regions. These regions are managed by the RTI, and are used to determine whether or not updates are sent to subscribing federates using the matching algorithm. Regions can be shared among RTI nodes, and should only be sent where required. The effects of the matching algorithm must be assessed when updates are sent so the correct filtering can be used. These issues have been addressed by this research.

In order to demonstrate the claims of the thesis, a DDM implementation was

developed. The implementation realized three contributions to the body of knowledge surrounding the DDM services. The first is a technique to dynamically reduce the number of regions being processed by the matching algorithm as it determines region overlap, and is called Dynamic Region Reduction. The Dynamic Region Reduction technique was designed as an improvement to the Sort-Based matching algorithm (which is described in detail in Chapter 3) and is discussed in Chapter 5. The second contribution is a modification to the SIP-RTI Conferencing Infrastructure and cRTI Model. Conference attendees are given tags to identify whether or not the federates they represent are using DDM, and if they are in overlap with any publishing regions. These modifications are discussed in Chapter 6. The third contribution provides an implementation of the DDM services following the IEEE 1516 RTI specification, and the design is discussed in Chapter 7, and the distribution of information within the Conference Infrastructure and Local RTI Component is discussed in Chapter 8. A demonstration of the DDM services is provided in Chapter 9, along with test results on the reduction of regions processed by the Dynamic Region Reduction technique. The document closes with conclusions and recommendations for continued research in Chapter 10.

Throughout this document, an illustrative example of the RTI in use will be provided to help ground the concepts described in a tangible manner. A scenario, shown in figure 1.1 will be used, involving an airplane flying in a loop, which is communicating with one or more radio towers on the ground. This scenario is useful because the physical phenomenon of radio signal degradation can be represented by regions within the RTI, providing a boundary where the plane can and cannot successfully receive messages transmitted by the radio towers.

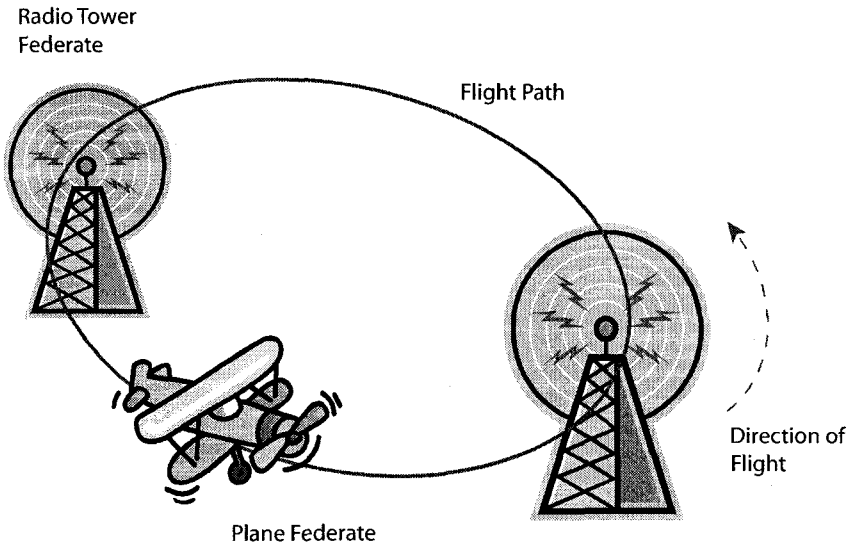


Figure 1.1: Example Plane Scenario

Chapter 2

Background

This chapter provides the reader with the necessary background information on relevant topics. The Publish-Subscribe architecture is introduced first, as it serves as the basis for communication in the RTI. The Java Messaging Service and OMG Data Distribution Services are introduced as they provide a similar method of message filtering to the HLA Data Distribution Management services. The history and evolution of the HLA specification from the DMSO 1.3NG version to the IEEE 1516 specification is covered, followed by a comparison of the Data Distribution Management services between the two HLA specification versions. A look at the two broad categories of matching algorithm research is provided. Finally, the SIP-RTI is described, as it is the starting point for this research, and a novel Region Matching algorithm is described, which was also used within this work.

2.1 The Publish-Subscribe Architecture

In large scale distributed applications, the use of the Publish/Subscribe architecture is employed because it facilitates a loose coupling between components that produce data ("publishers"), and those that consume it ("subscribers"). This loose coupling

is achieved through a messaging framework that allows components to declare what data they intend to provide to the system, as well as what they wish to receive. The focus of the communication rests on the data available, so application components need not concern themselves with which specific components require or produce the data.

Data transfer occurs when a publisher has a new value to report. The publisher will send the updated information to the supporting middleware, which will then broadcast this new data to all the subscribing components. The subscribing components receive the updates from the network, unaware of which component may have produced it. This method of data exchange allows for greater scalability of the application, and ease of implementation, because end nodes need not be aware of each others comings and goings, and they will not block on each other.

2.2 Java Message Service

The Java Message Service (JMS) [7] is an API for an asynchronous messaging service for the Java 2 Enterprise Edition (J2EE) platform. The JMS provides a publish-subscribe and peer-to-peer communication middleware that supports message filtering. While primarily concerned with sending asynchronous Messages, JMS also provides a method of creating a "durable" session to facilitate streams of data, such as video. The middleware supports user-created Sessions, in which Topics are created. Topics represent application-specific information, and JMS provides no special means of Topic hierarchy. Clients using the API publish or subscribe to Topics through TopicPublisher and TopicSubscribers. Figure 2.1 shows a Client using a TopicPublisher to send Messages to the Topic within a Session. Clients using the TopicSubscribers within the same Session can receive these messages.

Messages can be filtered by the Clients, to allow them to receive only specific

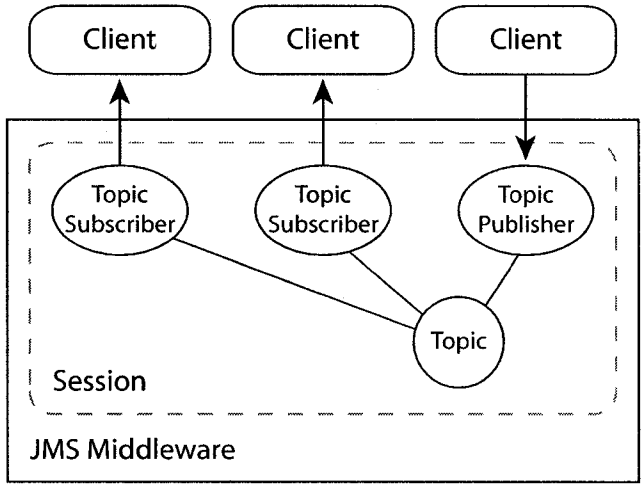


Figure 2.1: JMS Middleware Example

Messages. Filters are assigned to the TopicSubscriber when it is created for the Client with a Message Selector. The Message Selector contains a property and value pair, which define the criteria under which the JMS middleware should notify the Client of an update. When creating a message to send, the publishing Client can add a Selector to its Message. If the property and value of the Selector on the published Message match the Selector defined for the TopicSubscriber, the Message will be transmitted to the Client.

2.3 Object Management Group Data Distribution Service

The Object Management Group (OMG) [8] created the Data Distribution Service (DDS) [9] standard to provide a standard API and architecture for implementing a distributed publish-subscribe middleware. DDS also attempts to provide a quality of service (QoS) for the data exchange, and supports various forms of messaging;

streams, signals and state. Streams represent data that is provided as a snapshot and is expected to be received after the previous snapshot of data. Stream data should be sent reliably, and if it is not received correctly or in order errors could result in the application. Signals are data that are continuously changing, such as data being provided by a sensor. This type of data can be sent with a best effort level of support, because the data is constantly being modified and sent. State represents the current state of an object. State data does not have a predictable pattern of frequency of updates, and so must be sent reliably, to ensure that the state is propagated correctly to all subscribers.

The DDS specification provides two levels of API: the Data-Centric Publish-Subscribe (DCPS), and the Data-Local Reconstruction Layer (DLRL). The DCPS API provides the application with a set of services for publishing or subscribing to data. The DLRL provides a simpler version of the DCPS API, which is streamlined to be easier to use. The application has access to both levels of DDS API, as shown in figure 2.2.

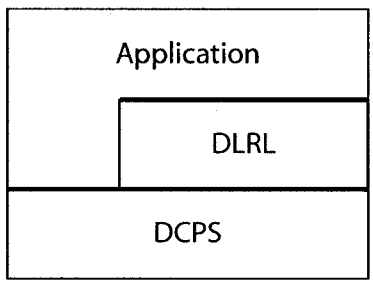


Figure 2.2: DDS API Layers

The DDS middleware provides a set of objects that are used by the application to either publish or subscribe data. A DataWriter provides a typed interface to a Publisher object, and the DataReader provides a typed interface to a Subscriber. The association between DataWriters and DataReaders, is made through a Topic.

The Topic can also provide a QoS policy to ensure timely communication between the Publisher and Subscribers. The Topic itself has a globally unique name within the distributed application. Figure 2.3 shows a Publisher using a DataWriter in association with a Topic. The Topic, for example, could be a radio tower signal, and the published values are messages being broadcast. The Subscribers are each using a DataReader to receive the updated value. The DataReader provides a typed interface for the Subscriber.

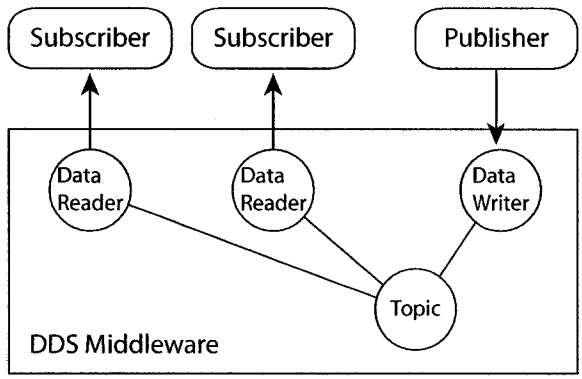


Figure 2.3: DDS Middleware Example

2.4 The High Level Architecture And The Run-Time Infrastructure

The High Level Architecture (HLA) was developed by the United States Department of Defense (DoD) in the early 1990s to provide an interoperability framework for their many simulation projects. Having found that reuse wasn't feasible with their simulation development practices at the time, they proposed the HLA to support simulation component ("federate") development, and the Run-Time Infrastructure (RTI) to act as the middleware providing services to the complete simulation ("federation").

There was a need for simulations in training, but the previous practices were to develop a simulation from scratch every time a new one was needed. This practice was both time consuming and expensive [1]. The simulations that were developed needed to be reusable to get greater benefit for the investment [1]. A common standard was required for this to happen.

The development of the RTI was integral to the acceptance and support of the HLA standard. The initial versions of the RTI, known as the 0.X series, were developed as quickly as possible to get simulation developers using it as soon as possible [1]. These early RTIs did not support the full HLA specification. Once the HLA was being used, the new versions of the RTI were developed, the 1.0 series. The initial release, in 1996, supported all of the HLA services, except Data Distribution Management (DDM) services, and some of the Federation Management services [1]. The DDM services were added later, in 1997 [1].

The first full RTI, the RTI 1.3NG, was released shortly thereafter, along with a revision of the HLA specification. At that point, DMSO determined that it could no longer maintain sole development of the RTI, and left the door open for private companies to develop RTI software. In the year 2000 the HLA became an official IEEE standard, the IEEE 1516.

The HLA specification introduced a software architecture to facilitate federate interoperability and reusability, and is currently comprised of three documents. The HLA Framework and Rules [10] describes the rules that govern the responsibilities of the federates and the federation, to ensure correct and consistent implementation of federations. The Federate Interface Specification outlines the API for the RTI [11], and the seven service groups the RTI provides the federation: Federation Management, Declaration Management (DM), Object Management, Ownership Management, Time Management, Data Distribution Management (DDM), and the Support

Services. The Object Model Template (OMT) [12] document describes the contents and structure of the Federation Object Model (FOM) file. The FOM defines the simulated objects that are shared by federates during the federation execution.

The Federation Management services provide services to create, modify, and delete a federation execution. A federation execution is created based on its object model data within the provided FOM file.

Declaration Management services support a federate's declaration of intent to publish or subscribe to object class attributes or interaction classes during the federation execution. A federate declaring it's intent to publish does not necessarily guarantee that it will, rather it informs the RTI that at some point during the federation execution it may publish values.

The Object Management services allow federates to register new instances of Object Class Attributes so that they can publish attribute updates for subscribing federates, or sending and receiving Interactions. These services also provide the means for federate notification when instances are registered, or if attribute instances come into or out of scope with subscribing federates, as well as delivery of Attribute updates.

The Ownership Management services provide federates with a means of exchanging attribute instance ownership with each other. A federate can request the ownership to an owned instance, which may or may not be released by the original owner.

Time Management services provides federates within the federation execution with a means of ordering the delivery of Attribute and Interaction updates. Federates using these services can ensure that messages sent between federates will be delivered in the same ordering.

The Data Distribution Management services provide federates with a means of refining their interest in Attributes or Interactions. Regions are defined, which are then used to filter the published updates to only federates that meet the filtering

criteria.

Finally, the Support Services provide the federates with services to access or modify advisory switches, access object handles, manipulating region ranges, and RTI start up and shut down.

The RTI compares to the OMG DDS services in it's similar approach to sharing data between distributed objects, or the DM services in the HLA world. The DDS Topics act similarly to Object Class Attributes defined within the FOM, and instances of topics can be created by assigning Keys to Topics. The RTI provides more services than data exchange, and has been designed to support large-scale distributed simulations. Rather than specific Data Reader or Writer objects interacting with the Topics within the system, the RTI uses a generic RTI Ambassador, that provides the federate with all of the RTI's functionality.

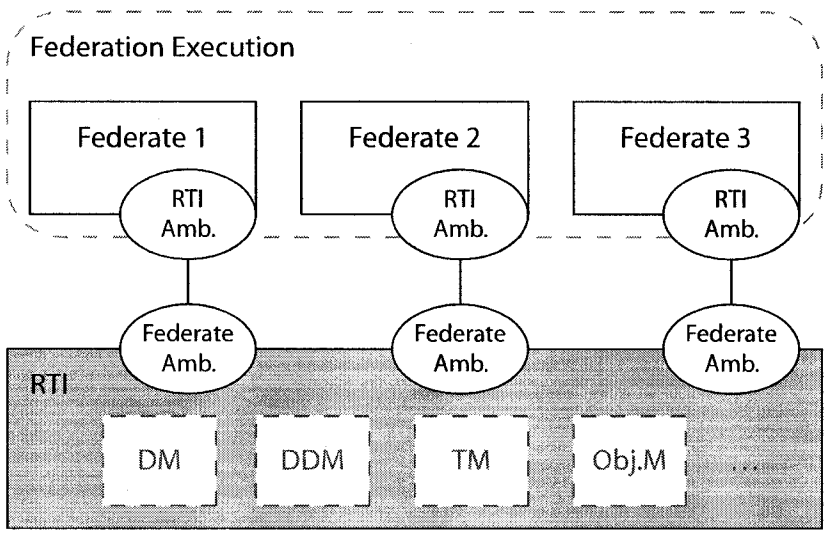


Figure 2.4: Federation/RTI Interaction

The RTI and federates interact with each other using Ambassadors, as shown in figure 2.4. The Federate Ambassador defines an interface to the federate, so the RTI

can make certain function calls to it, and the RTI Ambassador allows the federate to communicate with the RTI. The Federate Ambassador guarantees that a federate will have callback methods that the RTI would need to use, such as notifying a federate with the updated value of an attribute. The RTI Ambassador provides an API to the federate for all of the services that are implemented within the RTI.

2.5 The Data Distribution Management Services

The Data Distribution Management services provide more control over messages sent between federates than the DM services alone. These services offer support for defining specific filters that are applied to an Object Class Attribute or Interaction when a federate declares its intent to publish or subscribe. These filters restrict the messages sent to subscribers only when the filter from the publisher and subscriber coincide. This allows the RTI to reduce the amount of network traffic that is sent between federates by simply not sending messages where they are unneeded. This also allows the federates to be designed in such as way that they can assume the updates they are receiving are relevant.

The HLA DDM services provide better filtering functionality than the OMG DDS services. The DDS services provide a means of differentiating published objects via Topics only, which is analogous to the DM services. DDM furthers message filtering by allowing federates to create filters within an Object Class, which would be similar to creating a filter within a DDS Topic. The JMS Message Selectors provide a similar feature as regions, however are limited in that the Selector is a single value, rather than a range.

Filters are defined in the RTI by regions. A region is created by the federate, and can be associated to an Attribute Instance or Interaction Class for publishing, or an Object Class Attribute for subscribing. Attribute Instances are used for pub-

lishing, since a federate needs to own an instance to be able to publish. Subscribers are not concerned about specific instances of the Object Class, and so will subscribe to the Object Class Attribute which encapsulates all the Attribute Instances. Subscribing regions can overlap with multiple Attribute Instance publishing regions at the same time. Instances of Interaction Classes are not created within the RTI, so federates associate regions with the Interaction Class itself for both publication and subscription.

Regions are defined by ranges on one or more dimensions. A dimension is a linear range defined in the FOM, which has some meaning relevant to the federation execution. An example of a dimension is the latitude or longitude within a simulated environment. Dimensions begin at zero, and extend to the upper bound found in the Dimension Table of the FOM. Ranges are created by the federate and consist of a lower and an upper bound. The values of the bounds must be between zero and the upper bound of the dimension the range is created for. The ranges on the dimensions specify the size of the region, and the dimensions are the criteria that the filters are based on. A set of dimensions is referred to as a Routing Space. The Routing Space has a different interpretation in the DMSO 1.3NG and IEEE 1516 HLA specifications, and will be explained in the following sections.

Federates cannot publish or subscribe with regions to just any available Object or Interaction Class. The Object Class Attribute and Interaction Class definitions within the FOM must support the use of DDM, which is denoted by the association of dimensions with the Object Class Attribute or Interaction Class definition. This restricts which dimensions can be used when creating regions for an Object Class or Interaction, and prevents federates from creating regions on the same Object that do not share any dimensions. A region being created for use with an Object Class Attribute, for example, cannot use dimensions that are not associated with that

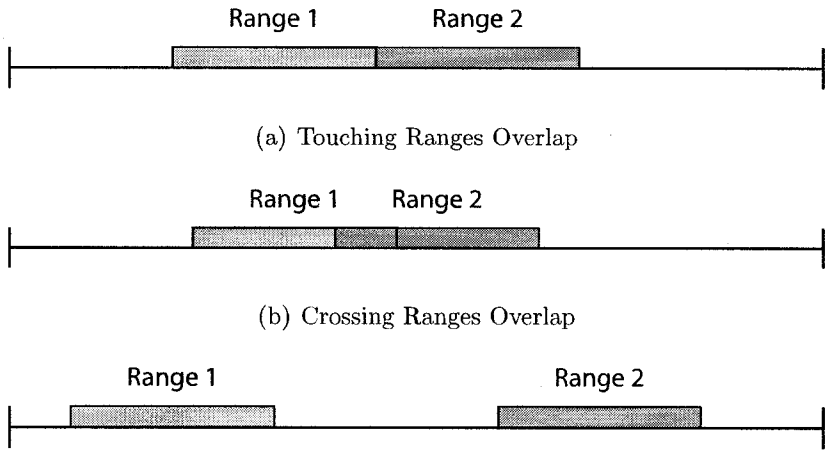
Attribute in its FOM definition.

While the associated dimensions of an Object Class or Interaction Class dictate which dimensions can be used, it does not imply that all of those dimensions must be used. A region can be associated with an Object Class Attribute or Interaction Class using as few as one of the available dimensions for that Object. The RTI will assume that dimensions that do not have ranges specified for them use the Default Range. The Default Range is added implicitly by the RTI, and will extend from zero to the dimension's upper bound.

Attribute value updates or Interactions are sent to subscribing federates when their subscription regions overlap with publishing regions. Regions are in overlap with each other when ranges on all of the dimensions used by both regions overlap with each other. Examples of ranges overlapping or not overlapping are shown in figure 2.5. Range overlap on a dimension means that the bounds of each of the ranges touch or cross into each other, as shown in figures 2.5(a) and 2.5(b). When ranges are not touching, such as in figure 2.5(c), they do not overlap. Region overlap is achieved when there is range overlap on each dimension in both regions.

The RTI also implicitly creates a region that consists of all of the dimensions within Dimension Table whose ranges extend from zero to the upper bound of each dimension. This region is known as the Default Region, and is not accessible directly by any federates. The Default Region is used by the RTI to handle the interaction between the DM and DDM services.

The DM and DDM services are related since they both operate on the same set of Object Class Attributes and Interactions. The services interact with each other via their own perspectives of the Object Class Attributes, Instances and Interactions. From the perspective of the DM services, there is no difference in use of these objects when a federate is using DDM. The Publisher still must register an Object Instance,



(a) Touching Ranges Overlap

(b) Crossing Ranges Overlap

(c) Distant Ranges Do Not Overlap

Figure 2.5: Range Overlap Examples

and Attribute values are updates from that federate. From the DDM perspective, DM publications and subscriptions are made with the Default Region. Attribute updates made from DM publishers will overlap with all DDM subscribing regions, and all DM subscribers will overlap with all publishing regions.

Figure 2.6 shows an example of how DDM filtering is achieved. A generic view of the objects is presented to keep this example from being specific to either DMSO 1.3NG or IEEE 1516 specification. Examples showing the differences are provided in the following sections. The example in figure 2.6 shows the subscription regions of two plane federates ("Plane 1" and "Plane 2"), flying over a simulated landscape, with a radio tower ("Tower") publishing region they can overlap with. The two dimensions being used represent the Latitude and Longitude of the simulated environment, and the upper bounds of these dimensions define its size. The regions in figure 2.6 are created within the Routing Space that is composed of these two dimensions.

A radio tower is broadcasting a message, however the planes can only receive the messages when they are within radio range. The radio ranges for the three

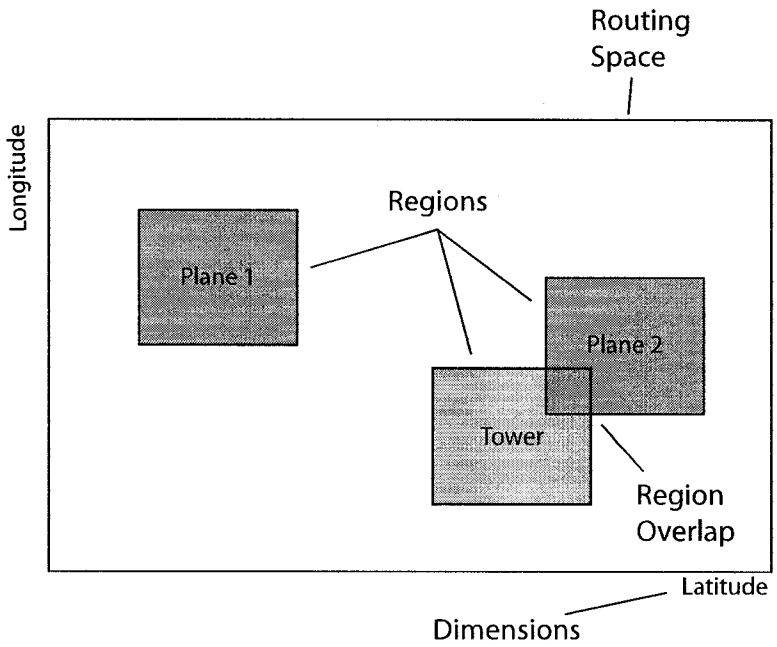


Figure 2.6: DDM Objects Example

simulated objects are defined by regions, with ranges on both Latitude and Longitude dimensions. These regions are shown as shaded boxes within the Routing Space in figure 2.6. Plane 2's radio range overlaps with the Tower's, so it can receive the messages, while Plane 1 is not overlapping.

Though the concept remains the same, there are variations in the definitions and resulting implementations of the DDM objects between the DMSO 1.3NG and IEEE 1516 DDM specifications.

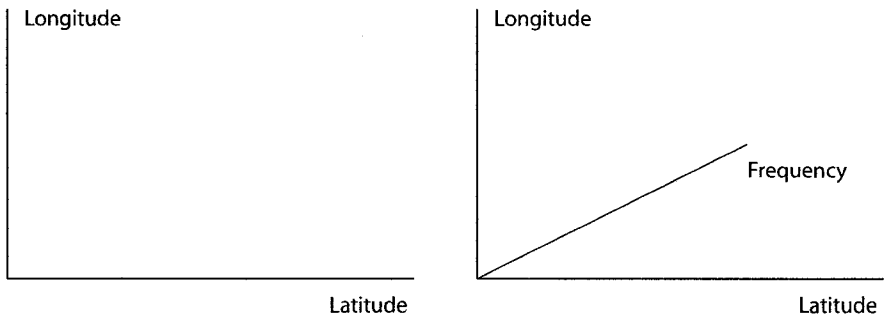
2.5.1 DMSO 1.3NG DDM

The DMSO 1.3NG version of the DDM services relied on the use of Routing Spaces when creating regions and associating them with Object Class Attributes and Interactions. Routing Spaces were a separate set of definitions within the FOM that

were uniquely named, and contained a subset of the available dimensions from the Dimension Table in the FOM. These Routing Spaces contained all the dimensions that a federate could use for the Object Class Attribute or Interaction. Regions were required to have ranges on each of the dimensions within the Routing Space.

The Routing Space concept proved difficult to use, as it made the federates more difficult to modify if a Routing Space was changed. Regions were required to create ranges on each dimension within the Routing Space, so any time a Routing Space was modified in the FOM, federates needed to be updated to handle the new dimensions. Routing Spaces could not overlap each other, so regions associated with different Routing Spaces on the same attribute could never overlap, regardless of the number of dimensions in common between them.

Figure 2.7 shows two Routing Spaces using similar dimensions for different purposes. The first Routing Space in 2.7(a) is made up of only Longitude and Latitude dimensions, and can be used to define regions based on visual or audio limitations. The second Routing Space in figure 2.7(b) uses both the Longitude and Latitude dimensions, and also a Frequency dimension. Regions can be made within this Routing Space to represent radios of various transmission strengths and frequency ranges.



(a) Routing Space for Visual Range

(b) Routing Space for Radio Range and Frequency

Figure 2.7: Routing Space Examples

Regions are defined by a set of extents. An extent is a range on one of the dimensions within the routing space. The extents define a sub-space within the Routing Space, that created the area of interest for the federate. A region can have several sets of extents, in effect giving it several ranges on each dimension within the Routing Space. Figure 2.8 shows a region ("Region 1") with two extents, "Extent 1" and "Extent 2". Each extent has a range on both dimensions in the Routing Space from figure 2.7(a), which are "Latitude" and "Longitude" in this example.

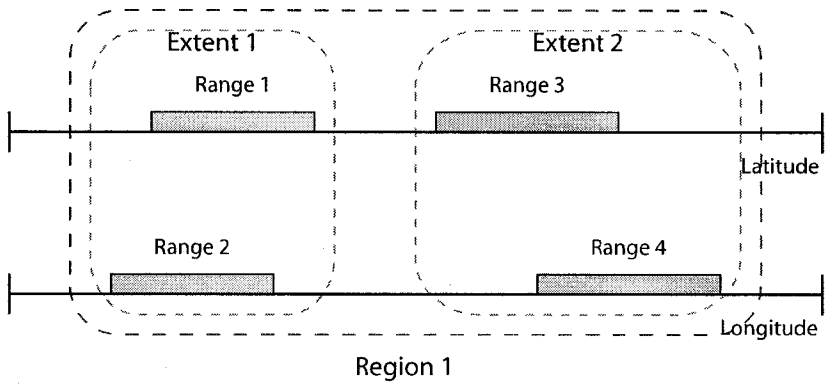


Figure 2.8: A Region With Two Extents

Region overlap is determined by extents overlapping each other on the dimensions within the Routing Space. Extent overlap is analogous to range overlap, described above. All of the extents within the same extent set must overlap with all the extents from the same extent set of another region. Not all of the extent sets need to overlap with each other, only all of the extents within a set.

Figure 2.9 shows two overlapping extents. Region 1 is defined by extents "Extent 1" and "Extent 2", and Region 2 is defined by "Extent 3". Extents 2 and 3 overlap each other because their ranges overlap on both dimensions. Region 1 and Region 2 overlap because both regions have a set of extents that overlap each other.

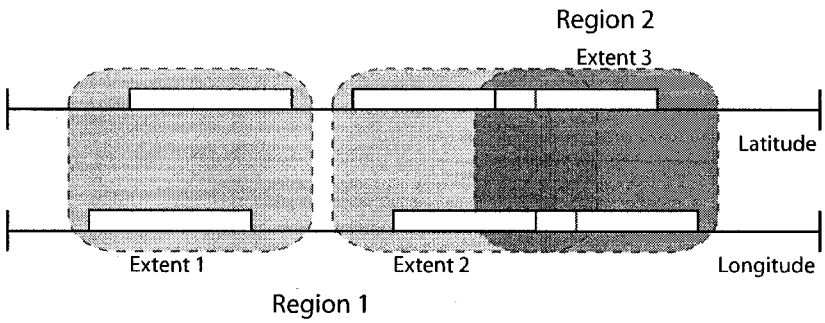


Figure 2.9: Overlapping Extents

2.5.2 IEEE 1516 DDM

The revision of the HLA services for the IEEE 1516 specification brought several changes, with two of the most notable being the removal of user-defined Routing Spaces from the FOM, and the removal of multiple extent sets from regions in the DDM services.

Rather than defining routing spaces separately within the FOM, the IEEE 1516 DDM allows the set of dimensions to be specified directly within the definition of the Object Class Attribute or Interaction Class within a single, global coordinate space. Attributes now have a set of dimensions added to them directly, and a single dimension can be added or removed without requiring a change to a Routing Space definition. Furthermore, regions are no longer required to specify ranges on all of the dimensions that are associated with an attribute, with the RTI in effect ignoring the dimensions that are not common to all regions being used in the overlap calculation.

Figure 2.10 shows an example of the single, global coordinate space available in the IEEE 1516 HLA specification. Regions can be made to represent both visual or audio ranges, such as those created within a smaller routing space shown in figure 2.7(a) as well as regions using the Frequency dimension. Object Class Attributes and Interaction Classes can specify the dimensions that can be used, and regions can use

as many or a few of them as required.

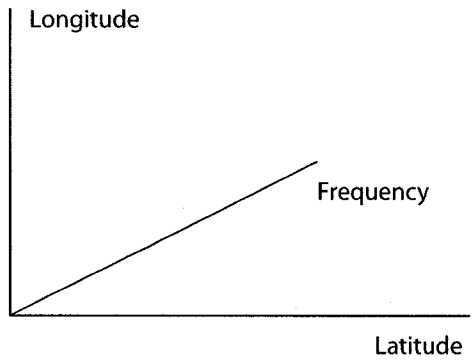


Figure 2.10: Global Coordinate Space in IEEE 1516 HLA

Another important change was the removal of extents within regions. Regions are no longer created with a set of extents. In effect, the number of extents has been fixed at one, where only a single set of ranges can be specified within a region. Regions themselves are now referred to as Region Specifications. When a federate creates a region, the handle of this abstract Region Specification is returned to it. When a region is associated for publication or subscription, the RTI creates an internal instance of that Region Specification, called a Region Realization. A Region Specification can have multiple Realizations within the RTI. When a federate modifies the ranges within that Region Specification, the changes will be applied to all derived Realizations.

An example of two regions created on the global coordinate space is shown in figure 2.11. Region 1 has ranges on only two of the dimensions, "Latitude" and "Longitude". Region 2 uses all three available dimensions, "Latitude", "Longitude" and "Frequency". Both of these regions can be used on the same Attribute or Interaction of the FOM has associated all three dimensions with it. In an overlap calculation, only the Latitude and Longitude dimensions would be used, and the RTI would assume

an overlap on Frequency dimension by associating a default range to the Frequency dimension for Region 1.

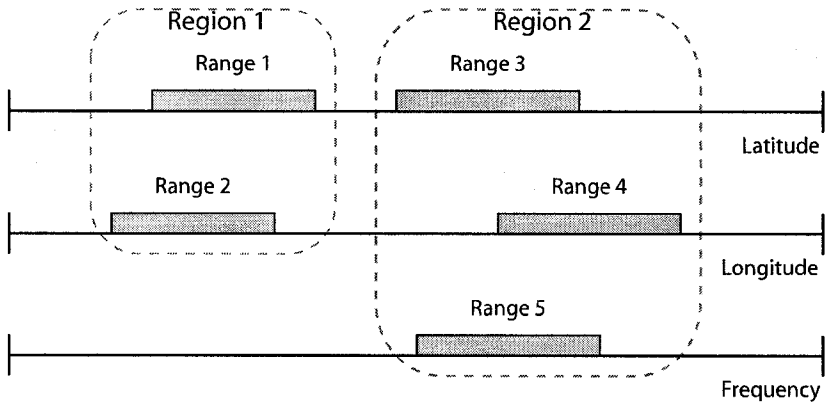


Figure 2.11: Two Regions on the Global Coordinate Space

Overlap between regions is handled similarly in the IEEE 1516 DDM, and is simpler than the DMSO 1.3NG version. A region is in overlap when all of it's ranges overlap pairwise on each dimension in common between the two regions. Since regions do not have multiple sets of extents, there is only one set of ranges that can be used for determining overlap, and no confusion regarding overlap arising from extents from different sets overlapping, but not all of them from one set.

2.6 Matching Algorithm Research

An important aspect of any implementation of the DDM services is the functionality to determine region overlap. Much research has gone into implementing efficient and scalable matching algorithms. Research can be roughly categorized into two schools of thought; grid-based implementations, which focus on overlaying a grid onto the routing space to help narrow the set of regions to calculate, or region-based, which attempts to determine overlap using all the regions. Current research in both of these

areas will be described in some detail below. Since most of the current research is done using the DMSO 1.3NG version of the HLA specification, some of the terminology and concepts discussed will be related to that version, rather than the IEEE 1516 DDM.

2.6.1 Grid-Based Matching Algorithms

Grid-Based matching algorithms rely on the RTI nodes dividing up the Routing Space into a grid, or overlaying a grid on top of two dimensions of a possibly multi-dimensional Routing Space. The cells within the grid become the basis for performing the region overlap calculations. Grid-based research has evolved over the years, from the initial fixed-grid approaches [13] to more effective dynamic grid-based approaches [14, 15].

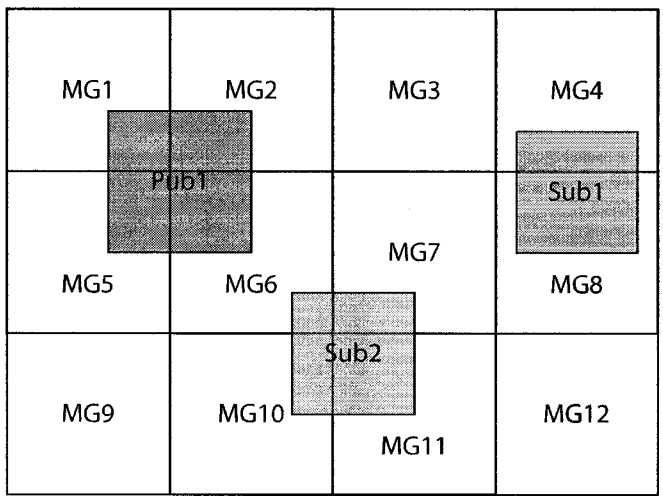


Figure 2.12: Fixed-Grid Matching Algorithm Example

In the fixed-grid algorithms, each cell in the grid is assigned a multicast group when the RTI is being initialized. Figure 2.12 shows a grid overlaid over a routing space. Each of the cells within the grid has already been assigned a multicast group,

"MG1", etc. When a region overlaps with a cell in the grid, such as the publishing region "Pub1" in cells 1, 2, 5 and 6 in figure 2.12, it is added to each of those cell's multicast groups, "MG1", "MG2", "MG5", and "MG6" respectively. All the regions that overlap a cell will belong to it's multicast group, and are assumed to be overlapping each of the other regions overlapping that cell. Subscribing region "Sub2" is part of multicast group "MG6" as well. When a publisher is updating its instance attribute, the update is sent to all the multicast groups that the region is associated with. Subscribers in each of those multicast groups will receive the updated attribute value. An update sent by Pub1 would be sent to multicast groups MG1, MG2, MG5 and MG6. The update would be received by subscriber Sub2 since it is also part of multicast group MG6.

This is a fairly simplistic solution, and is also prone to errors. The matching is performed simply by regions being in contact with the same cell in the grid. Messages can be incorrectly sent to subscribing regions if they are within the same cell, but still not actually overlapping the publishing region, requiring receiver-side filtering, possibly performed by the federate itself. These kinds of errors can be reduced by making the cells in the grid smaller, ensuring that regions sharing the same cell are much closer to each other. This however would lead to a greater number of multicast groups being created, increasing the overhead of the grid cells in the RTI nodes.

Dynamic grid-based matching algorithms build on the fixed grid approaches by incorporating region-based matching to improve their accuracy, and reduce the number of multicast groups that are created by not assigning a multicast group to a cell unless there are regions in the cell [14, 15, 16]. Multicast groups are created when there is at least one publisher and subscriber within the cell. Following the example in figure 2.12, the only multicast group that would be created is "MG6". This prevents the creation of unnecessary multicast groups when there is only one subscribing

or publishing region, or several regions of the same type, where no messages would need to be sent. As well as creating the multicast group, the DDM services will run a matching algorithm on the regions in the cell to determine if the publisher and subscriber actually overlap. If there is no overlap, none of the regions are added to the multicast group [14, 15, 16].

2.6.2 Region-Based Matching Algorithms

Region-Based matching algorithms [13] offer a straight forward approach to managing the overlapping regions and performing a matching by maintaining a multicast group for each publishing region [17]. All the regions associated with a particular Object Class Attribute would be compared together in an arbitrary order while the matching algorithm is being performed. Figure 2.13 shows an example Routing Space with a publishing region "Pub1" and two subscribing regions "Sub1" and "Sub2". When performing the matching for Pub1, the brute-force approach would be to select a subscribing region arbitrarily, and compare the ranges on all of the dimensions in the routing space. Once the matching is complete, the matching algorithm selects the next subscription region and performs the same matching.

The brute-force approach can lead to some unnecessary work because regions which are far from the publishing regions will still be included in the overlap calculation. The full Routing Space in figure 2.13, where all the regions are situated provides no provisions for excluding regions based on their location. To combat this wasted work, the dynamic grid-based matching algorithms presented earlier attempt to combine the accuracy of region-based algorithms with the inherent region trimming provided by the grid.

Further research on region-based algorithms seek to overcome the unnecessary work by designing the matching algorithm to handle larger loads of regions efficiently

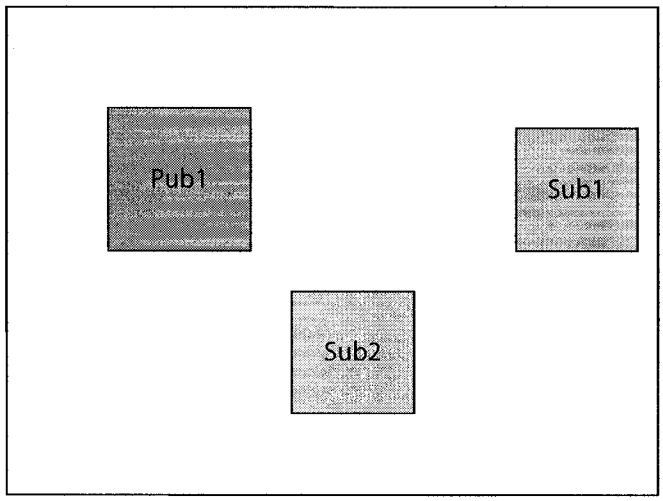


Figure 2.13: Region-Based Matching Algorithm Example

[18, 17, 19]. The P-Pruning algorithm [17] creates multicast groups only when it determines that there is an overlap on a dimension, but then will remove multicast groups as it checks other dimensions for overlap. The algorithm performs three separate steps to create the multicast groups and then prune them. The first step involves collecting the regions, and placing an entry into a table, which has a column for each point in the dimension, where the start and end points for the are. When all the regions have been added, the second step is performed; the table is scanned, and multicast groups are created for each publisher that has at least one subscriber overlapping with it. Finally, the pruning step is performed. The algorithm will examine the ranges on the next dimension for all the regions in a multicast group, and any subscriber that does not overlap will be removed from the multicast group. If no subscribers are left, then the multicast group is removed.

Another region-based matching algorithm is the Sort-Based Matching Algorithm. This algorithm is used in the DDM services implemented in this research, and is explained in detail in Chapter 3.

2.7 SIP-RTI

While there are several RTIs commercially available, using one requires that all components in the federation use the same RTI. While the interface may be the same, the implementation of the RTI is proprietary, making RTIs from different vendors unable to communicate with each other. This makes interoperability extremely difficult at the RTI level.

The SIP-RTI was developed to provide a solution for interoperability between different RTIs, as well as to facilitate RTI and non-RTI communication [4, 20, 21, 22]. It was developed on an open conference model architecture, which maps the object class and attributes described in the FOM as a hierarchy of conferences. The SIP-RTI also makes use of the SIP protocol for initial communication and set-up of the SIP-RTI nodes.

The SIP-RTI is developed as a two-tiered application, comprised of the Conferencing Infrastructure (CI) and RTI layers. The CI provides a conferencing structure, which provides the open layer for intercommunication between applications using the CI. The conferences provide a forum for applications to exchange data. This in effect provides a publish-subscribe architecture middleware. From the federation perspective, the SIP-RTI appears as in figure 2.14. The federation execution operates using the RTI. The RTI in turn is using the CI below it. The view in figure 2.14 is a high-level view of the SIP-RTI, where each layer views the layer below it as a black-box containing desirable functionality.

Looking deeper, each layer consists of one or more components, which can be running on separate nodes within a network. The deeper view of the SIP-RTI is shown in figure 2.15. Within the federation layer, distributed federates use the RTI below it for communication and data sharing to execute the federation. The RTI layer is comprised of distributed RTI nodes, called Local RTI Components (LRCs). The

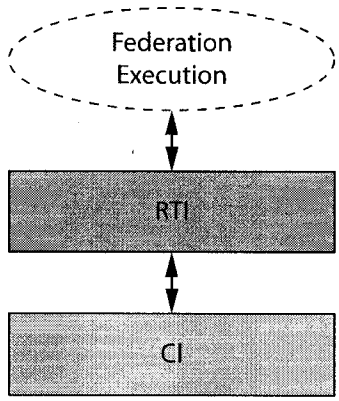


Figure 2.14: SIP-RTI Layers

RTI nodes use the CI below it to transfer data and maintain some of the application's state. Each of the LRCs manages one or more local federates. Finally, the CI layer is comprised of Local Conferencing Components (LCCs). The LCCs implement the CI functionality, and contain a local conference structure used by the application. On any node, the LCC will support one LRC instance. The LRC is designed to handle multiple federates per node.

Applications participate in conferences by sending conference Attendees. The Attendees represent applications using the LCC at a particular node. Attendees can come in one of two forms: a Speaker Attendee, or a Listener Attendee. As their names indicate, the Listener Attendees can only receive announcements from the conference, while the Speakers can make announcements to the conference. Speakers and Listeners are analogous to publishers and subscribers, respectively.

The state of Listener and Speaker Attendees within a conference is shared among LCC nodes. A minimal set of information is used so that each Attendee is not duplicated in each node. The presence of a Speaker or Listener attendee in a conference is represented by a Speaker or Listener Indicator in a remote LCC. Indicators are placed or removed from remote LCCs on a first-in, last-out principle. The Indicators

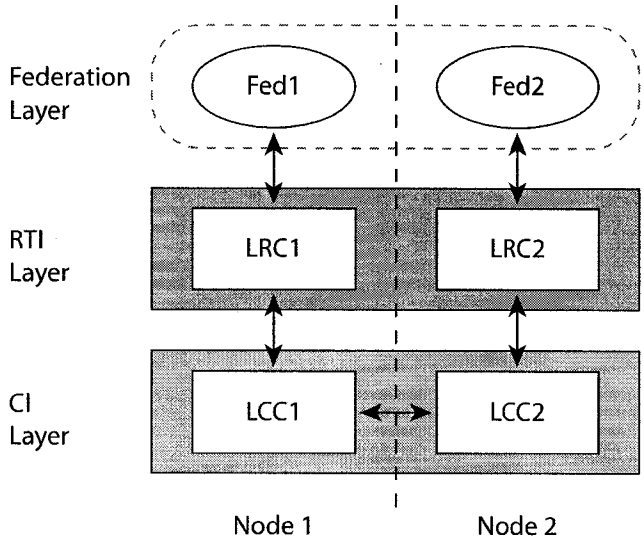


Figure 2.15: SIP-RTI Layer Components

will signify that there is at least one Attendee of that type on the remote node. A Listener Indicator will be placed in connected LCCs when an application first places a Listener Attendee within a conference. If more Listeners are placed at that node, no further Indicators are required. An LCC, such as LCC1 from figure 2.15, may have many Listeners sent to a conference by the application using LCC1. Other connected LCCs will maintain a single LCC1 Listener Indicator in their local conference rather than Indicators for each of the application Listeners in LCC1. When the last Listener Attendee is removed from a conference, then the Listener Indicator is removed from remote LCC nodes. Indicators for Speaker Attendees is shared in the same way.

For the SIP-RTI, the conference structure is created to resemble the object class structure of the federation execution as defined in the FOM. The LRC implements a model to show how it makes use of the LCC for RTI purposes, called the cRTI Model [4]. An example of the model in use is shown in figure 2.16. The model assumes that the initial, root conference created by the CI represents the federation

execution. This necessarily contains all the object classes that are specified in the FOM document. The Object Class RT Conference for a radio tower is created within the federation conference in figure 2.16. The conference structure is intended to map to the Object Class hierarchy, so Attributes and sub-classes will be created as sub-conferences within an Object Class conference, such as the Transmitter Attribute in the RT.Trans conference. The conference class, shown in figure 2.16 as Object Class RT Conference, contains an open floor, and a set of Speaker and Listener Attendees. Speakers are placed in Object Class conference by the RTI when there is also a Speaker in an Attribute sub-conference to announce to all Listeners when new floors are created. Listeners are placed in the Object Class conference when there is a Listener placed in any of the Attribute sub-conferences. The Listener in the Object Class conference allows them to receive announcements regarding new floors.

The Object Class Attributes are represented by sub-conferences within the Object Class conferences, as shown by the RT.Trans conference in figure 2.16. Within each Attribute conference, there is a set of Attendees, which are placed by federates that either publish or subscribe to that Attribute. When the federate is using the DM services, Speaker Attendees are placed within the Attribute conference when a federate declares its intent to publish that Attribute. The floors are created when the federate registers a new Object Class Instance. One set of floors is placed in each Attribute conference whenever a new instance is created. The Object Class Instance is highlighted by the dashed box around the two floors in 2.16. Ownership of a floor is obtained by a Speaker Attendee when the federate registers the Object Instance, or acquires ownership of an Instance from another federate. The Ownership Management services have been discussed in detail in [4] and are omitted from further discussion. Listener attendees are placed in an Attribute conference when a federate subscribes to the Object Class Attribute that the conference represents.

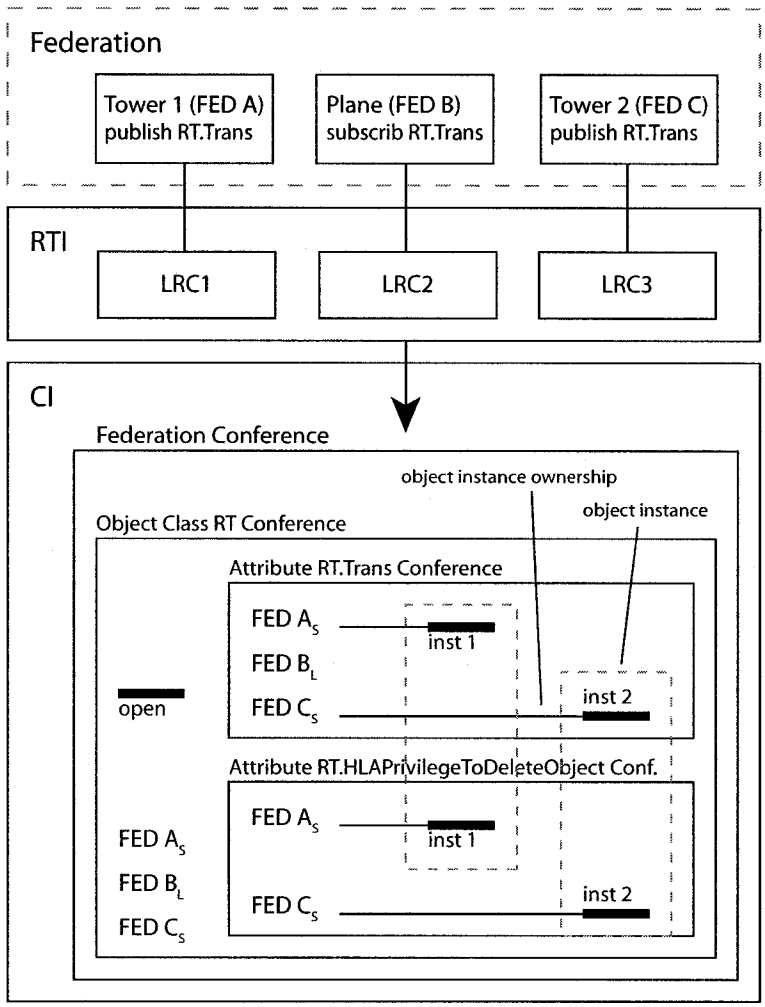


Figure 2.16: The cRTI Model

The LRC component implements the HLA RTI services, and uses the LCC following the cRTI Model. The LRC gives the RTI support for the federation execution. Prior to this research, the LRC implemented a subset of the Declaration Management, Object Management and Ownership Management services. The LRC uses the LCC for communication, and to maintain the state of the connection among publishers and subscribers. This creates a tighter coupling between the LRC and the LCC, as the LCC is used for more than just communication. The LRC does not duplicate information that can be maintained within the LCC. This design follows the idea that the LCC is a publish-subscribe middleware component for applications that use it. The application, in this case the LRC, is relieved from maintaining knowledge of other LRCs.

Chapter 3

State of the Art

A review of the current state of the art that was influential in this research is provided in this chapter. Three topics are discussed: a unique matching algorithm that organizes region data prior to performing the overlap calculation to achieve efficiency; an extension to the cRTI model to accommodate the filtering requirements of the DDM services; and tagging, which is a popular method of user-defined filtering on the Internet.

While there has been research going on from various sources into improving matching algorithms, perhaps the most promising research is done by a group at the University of Singapore on designing a region-based matching algorithm [18, 19]. This algorithm involves sorting the region ranges on each dimension, and letting the order help in determining whether or not the extent bounds overlap. This algorithm is used in the DDM services implemented in this project. The details of the algorithm are discussed in this chapter, while Chapter 4 identifies some areas where it can be improved.

The SIP-RTI provides an interesting first step in the development of an RTI which is open to communication with other applications. The initial implementation did not include the DDM services, however an extension was proposed to allow for them

[23]. The state of the publisher and subscribers used for the DM services is handled by conference attendees within the Conferencing Infrastructure (CI). The CI does not provide any filtering for announcements made within a conference, so the proposed extension identifies a means of adding this functionality for use by DDM services.

A fairly new form of user-defined filtering has become popular on community web sites such as blogs. Users are able to give a series of keywords, or tags, that define what their post, images or videos are about. These tags allow other users to find content, or find related material by other users. Tagging also can be used for filtering data associated with the tags.

3.1 The Sort-Based Matching Algorithm

The Sort-Based DDM Matching Algorithm [18, 19] was designed to provide an efficient method of calculating DDM region overlap. Developed for the DMSO 1.3NG HLA specification of DDM, the algorithm deals with region extents within a named Routing Space. The terminology used within this discussion, and some of the objects used will therefore follow the DMSO 1.3NG standard. The algorithm can operate on any number of publishing and subscribing regions at the same time. All the region extents will be compared one dimension at a time for each dimension in the Routing Space. Extent lower and upper bounds are sorted in increasing order, and then processed linearly. The extents that have been fully encountered (where both lower and upper bounds have been reached) and those not yet encountered (the lower bound has not yet been reached) are kept track of in separate lists. When a publishing region's extent has been fully encountered, the subscribing region's extents that are partially encountered between the publisher's lower and upper bounds overlap on that dimension.

To explain the Sort-Based matching algorithm, the example shown in figure 3.1

is used. This example scenario shows four planes ("Plane 1" to "Plane 4") flying in a circular loop. At either end of the flight path, identified by the dashed arrow, there is a Radio Tower ("Tower 1" and "Tower 2") transmitting messages. The planes will fly into, and then out of the radio range of the two Radio Towers as they traverse the path. This example illustrates the need for DDM in this scenario, since physical radio signals diminish with distance, the radio transmissions can only be received when the Planes are within the range of the Radio Tower.

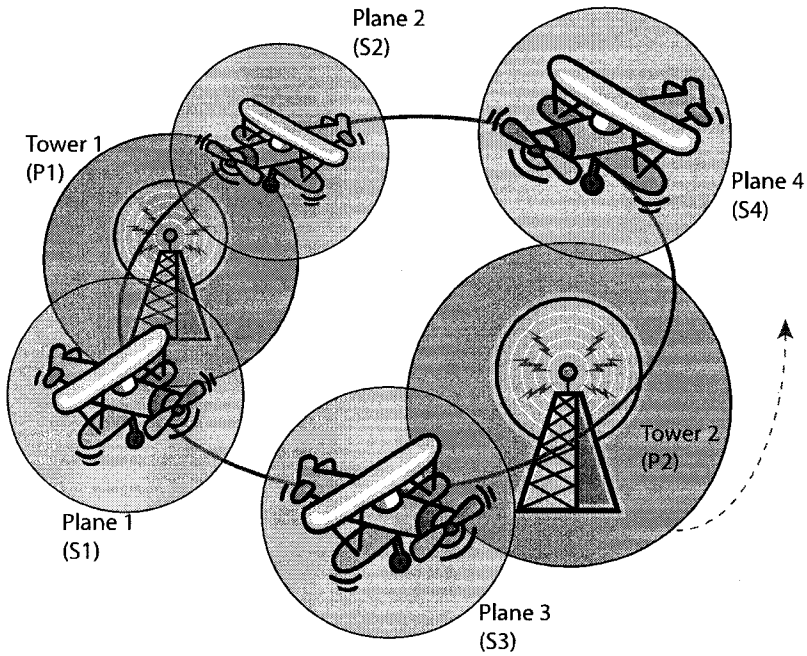


Figure 3.1: Matching Algorithm Example Scenario

In the RTI, the Routing Space for this example has two dimensions "x" and "y", representing the latitude (x) and longitude (y) of the simulated flight area. The radio towers, with a signal range of approximately 600 meters, have publishing regions six units in length on both dimensions. The planes have a smaller radio range of 400 meters, and subscribing regions are four units in length on both dimensions. The

regions within the Routing Space are shown in figure 3.2.

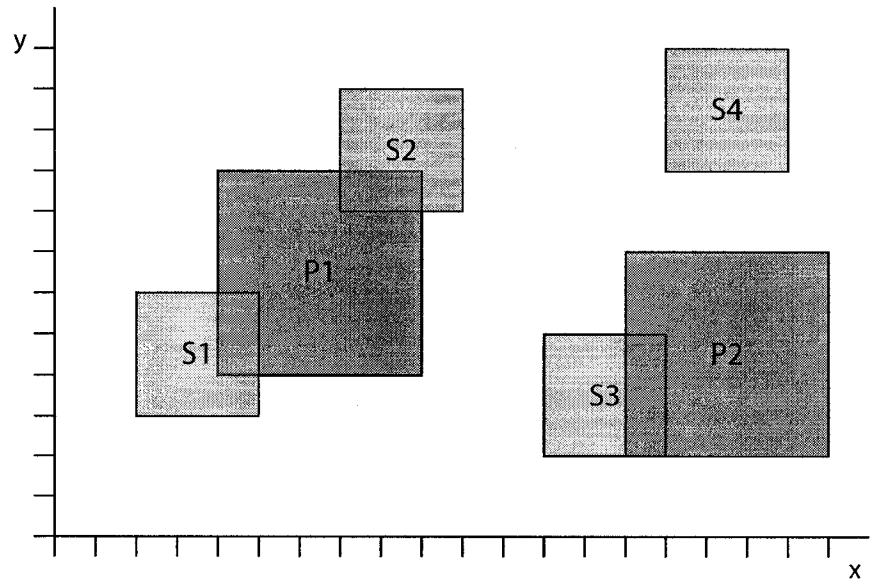


Figure 3.2: Example Scenario Regions in Routing Space

The pseudo-code [18] of the Sort-Based matching algorithm is shown in figure 3.3 to indicate the steps being performed by the algorithm. Before the algorithm begins, the region data is massaged into a form that can be used by the algorithm efficiently. Each extent is mapped to a bit in a bit array. This mapping allows each extent used in the matching algorithm to be represented by a single bit in the array, and is shown in figure 3.4. Representing each extent by a single bit allows for a full set of extents to be processed with simple binary operations such as AND or OR.

Each of the publishing extents will keep track of which subscription extents they do not overlap with for each dimension in the Routing Space. A multi-dimensional bit array is used to store this information. The bit array has a bit for each extent, and as many dimensions as there are in the Routing Space. Figure 3.5 shows an empty bit array that both P1 and P2 would have at the beginning of the matching process. All of the bits are set to 0, since no extents have been processed. As the algorithm runs

```
(1)  for each extent R_i in the routing space
(2)  {
(3)    insert lower bound point of R_i into list L
(4)    insert upper bound point of R_i into list L
(5)  }
(6)  sort list L
(7)  SubscriptionSetBefore = 0
(8)  insert all subscription extents into SubscriptionSetAfter
(9)  for all point P_i in the sorted list L
(10) {
(11)   R_i = Extent Id of P_i
(12)   if (R_i is a subscription extent) {
(13)     if (Pi is the lower bound point of R_i)
(14)       remove R_i from SubscriptionSetAfter
(15)     else  // P_i is the upper bound point of R_i
(16)       insert R_i into SubscriptionSetBefore
(17)   } else {  // R_i is an update extent
(18)     if (P_i is the lower bound point of R_i)
(19)       all extents in SubscriptionSetBefore do not overlap with R_i
(20)     else
(21)       all extents in SubscriptionSetAfter do not overlap with R_i
(22)   }
(23) }
```

Figure 3.3: Sort-Based Matching Algorithm Pseudo code

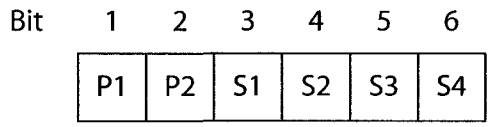


Figure 3.4: Mapping of Extents to Bits in the Sort-Based Matching Algorithm Example

its course, the bits will be set based on which extents are in `SubscriptionSetBefore` when a publishing extent's lower bound is reached, and `SubscriptionSetAfter` at the publishing extent's upper bound. After all dimensions have been processed, all the dimensions' bit arrays are ANDed together, and the results are flipped to determine the extents in overlap.

Dimension \ Extent	P1	P2	S1	S2	S3	S4
x	0	0	0	0	0	0
y	0	0	0	0	0	0

Figure 3.5: Publishing Extent's Two-Dimensional Bit Array for Overlapping Extents

Dimension x from the routing space in figure 3.2 is selected first, and the matching process begins with lines 1 to 5 in figure 3.3. The lower and upper bounds are placed into a list, `L`. The algorithm will iterate through this list in steps 9 through 23, performing a task at each bound according to the type of region it is from. Line 6 will sort the list of extent bounds, so that they will appear in increasing order. Figure 3.6 shows the extents along the x dimension, and the ordering of the bounds.

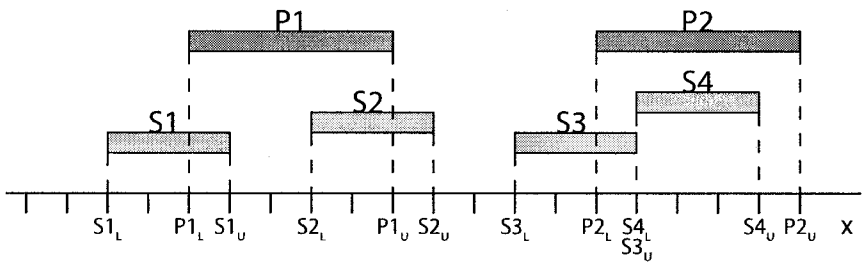


Figure 3.6: Extent bounds on the X Dimension

The two sets, `SubscriptionSetBefore` and `SubscriptionSetAfter` are used to

keep track of the extents that have or have not been encountered. An extent is said to be "fully encountered" when both lower and upper bounds have been reached while the algorithm iterates through the list of bounds. These are separate lists from the list of bounds, and are only used to keep track of the extents. The sizes of these bit arrays is the same as the extent mapping initially done before the algorithm started, and each bit will identify one of the extents participating in the matching. The initial state of these two lists are shown in figure 3.7. The appropriate bit for an extent is set in the `SubscriptionSetBefore` to indicate that an extent has been fully encountered, and initially set to 0 since none have been fully encountered at the beginning of the algorithm iteration. All of the bits are set to 1 in `SubscriptionSetAfter`, as the iteration begins at the lowest bound in the sorted list, since none of the extents have been encountered.

	P1	P2	S1	S2	S3	S4
SubscriptionSetBefore	0	0	0	0	0	0
SubscriptionSetAfter	1	1	1	1	1	1

Figure 3.7: Initial State of SubscriptionSetBefore and SubscriptionSetAfter

The algorithm iteration for the selected dimension begins on line 9 of the pseudo-code in figure 3.3, where the sorted bound list `L` will be iterated through. The first bound in the list is the lower bound for subscribing extent S1, "$S1_L$" as identified in figure 3.8. The bound's mapped extent bit is stored temporarily in line 11. Line 12 will check if this extent is a subscribing or publishing extent. This bound belongs to a subscription extent, so lines 13 is performed next. The bound is also the lower bound of this subscription extent, so line 14 is reached. The extent is removed from `SubscriptionSetAfter` by setting the extent's bit to 0 in that array. The extent

S1 has only partially been encountered, so it is not yet set in `SubscriptionSetBefore`. The mapped extent bit for the subscription extent S1 is now set to 0 in both `SubscriptionSetBefore` and `SubscriptionSetAfter`. The resulting state of the two lists are visible in figure 3.9. This iteration on the bound list is complete.

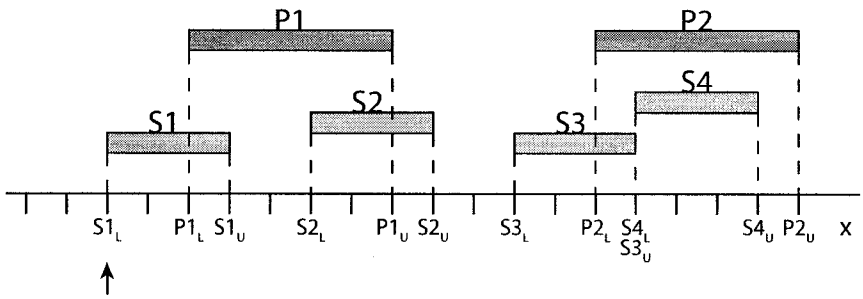


Figure 3.8: The First Bound Encountered

	P1	P2	S1	S2	S3	S4
SubscriptionSetBefore	0	0	0	0	0	0
SubscriptionSetAfter	1	1	0	1	1	1

Figure 3.9: SubscriptionSetBefore and SubscriptionSetAfter after line 14

The next bound in L is examined, and is the lower bound of the extent of the publishing region P1, "$P1_L$", as shown in figure 3.10. This is a publishing extent, and a lower bound, so line 19 is performed. All of the extents ids that are set in `SubscriptionSetBefore` are subscribing extents that have been fully encountered. These extents cannot overlap with this publishing extent because they have already passed their upper bound. Figure 3.10 shows that at the current bound in the list, no subscription extents have been fully encountered. A copy of this bit array is stored with P1. Following the example, the state of `SubscriptionSetBefore` and

`SubscriptionSetAfter` is the same as figure 3.9 at this point, and the list of non-overlapping extents for P1 remains the same as in figure 3.5.

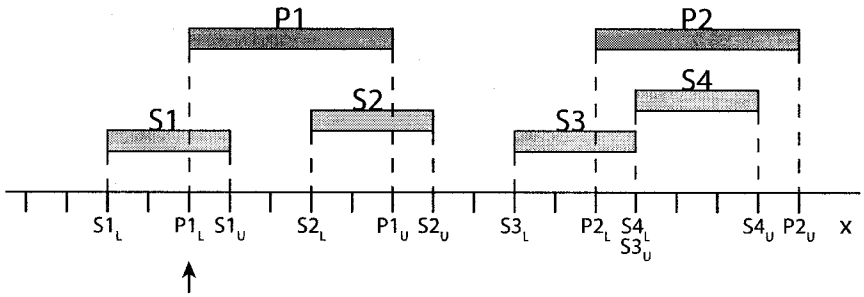


Figure 3.10: The Second Bound Encountered

The next iteration on the bound list begins, and finds the upper bound of S1, "$S1_U$", as shown in figure 3.11. This is now an upper bound, so line 16 is reached. The subscription extent has now been fully encountered, since both lower and upper bounds have been reached. The extent bit is set in `SubscriptionSetBefore`, so that all future publication extents will be aware that this subscription extent has been fully encountered. The state of the two lists is shown in figure 3.12.

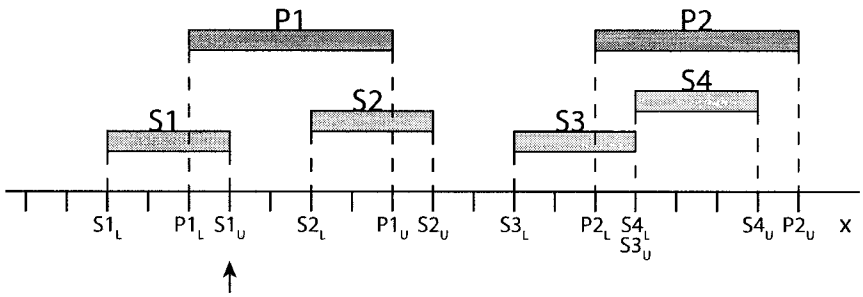


Figure 3.11: The Third Bound Encountered

Moving forward a few iterations, the upper bound of P1, "$P1_U$", is now encountered in the sorted list L and shown in figure 3.13. The lower bound of subscription

	P1	P2	S1	S2	S3	S4
SubscriptionSetBefore	0	0	1	0	0	0
SubscriptionSetAfter	1	1	0	1	1	1

Figure 3.12: SubscriptionSetBefore and SubscriptionSetAfter after line 16

extent S2 has been reached, so it has been set to 0 in `SubscriptionSetAfter`. The upper bound of S2 hasn't been reached, so it is not yet set in `SubscriptionSetBefore`. The remaining extents in `SubscriptionSetAfter` are extents S3 and S4, as shown in figure 3.13. The state of the two lists is shown in figure 3.14.

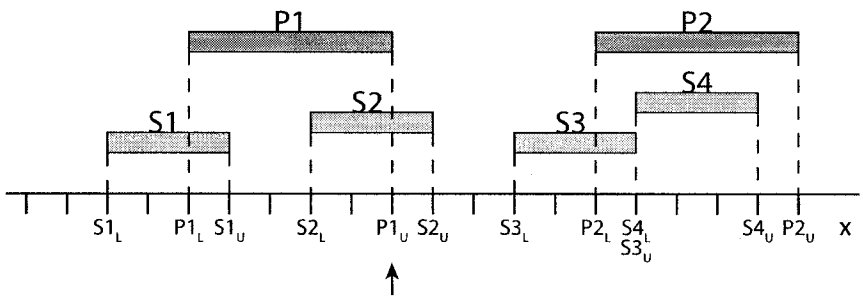


Figure 3.13: P1's Upper Bound Encountered

	P1	P2	S1	S2	S3	S4
SubscriptionSetBefore	0	0	1	0	0	0
SubscriptionSetAfter	1	1	0	0	1	1

Figure 3.14: SubscriptionSetBefore and SubscriptionSetAfter after line 21

Line 21 of the algorithm in figure 3.3 is performed. The extents in `Subscription-`

`SetAfter` do not overlap with P1, since they have not yet been reached. The bit array saved with the extent previously is now ORd with `SubscriptionSetAfter`. This combined bit array will now store all of the mapped extent bits of subscription extents that do not overlap with the publishing extent. The resulting bit array in the extent P1 is shown in figure 3.15. When the algorithm completes the iteration of the bound list for this dimension, publishing region P1 does not overlap with subscriber S4, and publishing region P2 does not overlaps with subscribers S1 and S2.

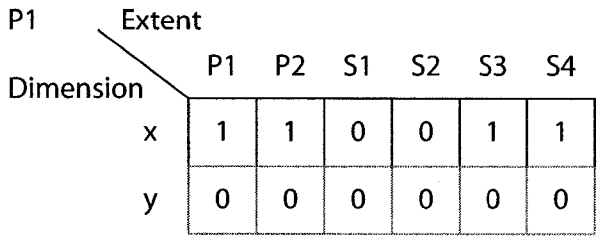


Figure 3.15: P1's Non-Overlapping Extent Bit Arrays

All of the steps are now performed again with the bounds of the extents on the y dimension. The bounds on the y dimension are shown in figure 3.16. The publishing extents will keep a separate bit array of extent ids for each dimension in the routing space. The results for this dimension when the algorithm completes its iteration are P1 not overlapping with S3 and S4, and P2 not overlapping with S2. The bit array from P1 after the y dimension has been processed is shown in figure 3.17.

When all dimensions have been iterated over, the arrays are ANDed together to give only the extents that did not overlap on all of the dimensions, as shown in the first bit array in figure 3.18. This result set is flipped, and the mapped extent bits that are set indicate subscribers that overlap. The flipped bits for publisher P1 are shown in the second bit array in figure 3.18. The final result of the matching algorithm is publisher P1 overlapping with S1 and S2, and P2 overlapping with S3.

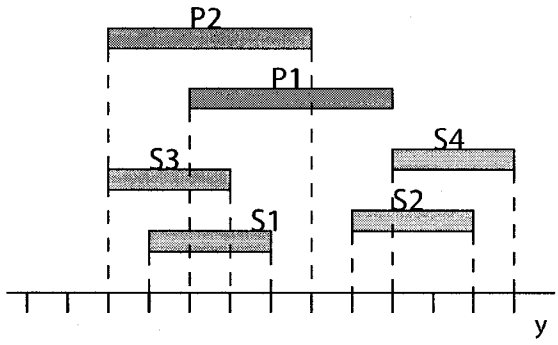


Figure 3.16: Extent bounds on the Y Dimension

P1 Extent

Dimension	P1	P2	S1	S2	S3	S4
x	1	1	0	0	1	1
y	1	1	1	1	1	1

Figure 3.17: P1's Non-Overlapping Extent Bit Arrays

The Sort-Based Matching Algorithm is composed of several steps, each with its own performance characteristics [18]. A theoretical analysis of the algorithm's performance is made by estimating the overall complexity of the algorithm's parts. Assuming there are n extents to be matched, the number of extents remains constant and does not need to be considered further. The original research [18] divides the algorithm into four major steps: sorting the extent bounds and initializing the bit arrays; processing the subscription extents and moving the mapped extent bits from `SubscriptionSetAfter` to `SubscriptionSetBefore`; processing the bounds of the publishing extents, and storing the overlap information; and finally combining the results to determine the actual overlap results. Overall, the complexity of the algorithm is determined to be $O(n^2)$ [18].

	P1	P2	S1	S2	S3	S4
ANDed Results	1	1	0	0	1	1
Flipped Results	0	0	1	1	0	0

Figure 3.18: P1's ANDed and Flipped Results

The Sort-Based Matching Algorithm provides a matching algorithm that is efficient and can be implemented to run with a set of regions on a single node. This algorithm is used for region matching within the DDM Services implemented in the SIP-RTI for this work. Further enhancements were made to the algorithm, which are described in Chapter 5.

3.2 cRTI Model and DDM Extensions

A common trait amongst open source or academic RTI implementations is the lack of the DDM services. The DDM services themselves require a significant amount of work, and are often left to be completed later, likely because of the extra complexity they add to the RTI, and because other RTI services, such as Time Management are seen to have greater importance than DDM. The necessary services to have a basic RTI functioning are the Federation Management, Declaration Management and Object Management services. These three are arguably the essential service groups and should be implemented first. This however leaves the DDM services to be picked up later, and attempted to be built into what is essentially a completed application.

The SIP-RTI was designed similarly. The goal of the project was not only to build an RTI, but to build it using a layered architecture, which would allow external applications to interact with the federates using the SIP-RTI. As described earlier

in Chapter 2, the lower conferencing infrastructure is used to transfer data between LRCs, and ultimately the federates. The LRC uses the conferencing infrastructure to create a set of conferences which follow the Object Class and Attribute hierarchy defined in the FOM. Basic DM communication operates by Speaker Attendees within the conference making announcements, and Listener Attendees in the conference hearing these announcements.

To add the DDM Services to the SIP-RTI, the services must not only be implemented within the LRC layer, but care must be taken when sending messages within the conference. Since attribute value updates are only sent to federates whose subscription regions overlap with the publishers' regions, filtering is required. Filtering cannot be done within the LRC layer because the update messages cannot be selectively sent to Listener Attendees within the conference. A filtering solution is required within the conferencing component as well.

Modifications to the CI and the cRTI model were proposed previously [23]. The extended model is shown in figure 3.19. The model makes use of new objects within the conference. The floor that is associated with a publishing Object Instance with regions is wrapped in a Sidebar, which is similar to a sub-conference. Overlapping Attendees are let into the Sidebar via the Doorman. The Doorman will also allow DM listeners, since they need to be able to hear the messages sent by DDM publishers. Listeners that do not overlap with the publisher of the Sidebar are placed in another sub-conference, called the Wait Zone.

The sample conference in figure 3.19 shows a conference representing the RadioTower Object Class, an Attribute sub-conference; the RTTransmitter Attribute. The HLAPrivelegeToDeleteObject Attribute is also part of the RadioTower Object Class, but it is not used in DDM, and no modifications are required within an Attribute conference for it. It has been omitted from the diagram for clarity. The

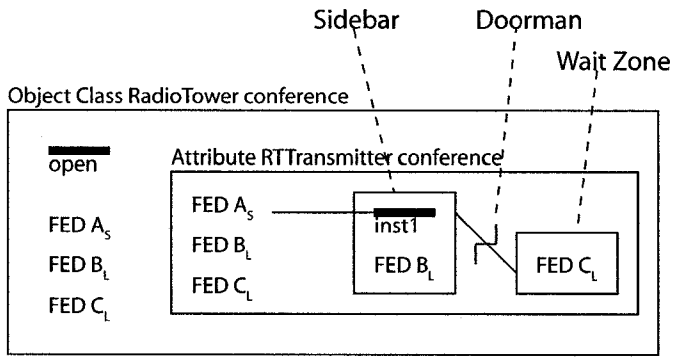


Figure 3.19: Proposed Extended cRTI Model

interest in this example lies within RTTransmitter attribute conference. The publishing federate, "FED A_S", is the speaker within this conference, and it owns a floor "inst1". The federate has associated a publishing region to this attribute instance, so the floor is within a sidebar. The two subscribing federates, FED B and FED C, are subscribed to this attribute with regions, and have listener attendees within the conference, "FED B_L" and "FED C_L" respectively.

FED B's subscription region overlaps with the publishing region, so the Doorman allowed a Listener for FED B (FED B_L) to be created within the Sidebar. FED C, however, does not overlap with the publishing region, and so it's Listener (FED C_L) must be placed within the Wait Zone. When FED A updates the RTTransmitter Attribute, it will make a conference announcement within the Sidebar where the Attribute Instance floor is. Only the Listeners within the Sidebar will hear the announcement, so filtering the non-overlapping subscribers is achieved.

The proposed extensions have been used as a basis for the modifications made within this research. An analysis regarding their suitability is covered in Chapter 4.

3.3 Tagging

Metadata, or "data about data" is a method of cataloguing, or adding structured data for the purposes of organizing books, articles, or data. Traditionally, this sort of activity is found in classification schemes such as the Dewey Decimal System [24] and the Library of Congress Classification Scheme [25], and are based on an elaborate set of rules [26]. These schemes are often of high quality, and contain various taxonomies that accurately describe the content they are referencing. While this is an acceptable solution within an institution such as a library, the volume of data being created every day on the internet makes classification by an organization impossible. Applying a well known classification scheme to user-created content online is also a difficult task, as many of these schemes require highly specialized training to be used correctly [26].

Various websites based on user communities and content, such as Flickr [27], Del.icio.us [28], YouTube [29], and any number of blogging websites have implemented a form of cataloguing scheme that is both functional and easy to use for the average user. These websites allow users to assign keywords to the content they create or upload, called Tags. These tags do not follow any strict conventions, hierarchy, or have any prescribed meanings associated with them. These tagging systems allow users to use plain-english keywords to describe the content they have added, be it images, videos or blog posts. Keywords aren't limited to picture websites, however. Academic sites such as CiteSeer [30] use keywords to sort and identify published research by keywords within the paper, author, or subject area.

The incentive to use tags lies in the social networking aspect of these sites. Tags are used as keywords in searches, allowing users to find similar or related content provided by other users. Users can easily browse the network for similar items, and even find new, related material based on another users' tags [31]. These user-defined classification systems have become known as folksonomies [32], a term coined by

information architect Thomas Vander Wal. The term is based on the words "folk" and "taxonomy".

Tagging provides a method of creating filtering for content by users of the application without having to establish a formal taxonomy beforehand. This is advantageous in an application that requires filters that may change over time, such as the filtering required by the DDM services.

Chapter 4

The Thesis

Current research revolving around the HLA DDM services is heavily focused on designing an efficient matching algorithm. The research is narrowly focused on the matching algorithm's results and efficiencies, yet rarely delves into the surrounding and supporting DDM services implementation. One can only assume that each research group is implementing the services in their own way. This presents a problem when attempting to understand the implementations described in the research, as there is no common method of organizing and sharing region data. Furthermore, each research group attempting to implement other algorithms for comparison will need to adjust them to fit their implementations of the supporting DDM services.

Another downside to DDM research is the use of the DMSO 1.3 version of the RTI specification. Many researchers are using the DMSO version of DDM to implement their matching algorithms. While this version of DDM, and the HLA Specification, has the advantage of being more familiar to researchers, the revised DDM services in the IEEE 1516 specification benefit from some reorganization and removal of the limiting Routing Spaces [33]. Other than proprietary RTI software packages supporting the full IEEE 1516 specification [34, 35], there are yet to be any open source solutions implementing the IEEE 1516 version of DDM available to the research community.

4.1 Limitations of DDM Research

Implementing the DDM Services requires greater consideration than providing an efficient matching algorithm. Based on the specification, the most important aspect of the DDM services is the region information. Regions are associated to both Attribute Instances for publication and Object Class Attributes for subscription, and provide the information to drive the matching algorithm. Published DDM research rarely addresses region management and usage within the DDM services.

Ignoring region information makes comparing matching algorithms difficult, as one implementation of the DDM services may be designed to accommodate the requirements a specific matching algorithm. Researchers are left with little choice, since there is no reliable and open implementation of the DDM services that can be used. Comparisons between matching algorithms is limited because there is no baseline implementation of the services supporting the matching algorithm. An open source implementation of the DDM service would provide the research community with a means of comparing their algorithms on a common platform.

The DDM services are not a completely separate entity within the RTI. Using regions for publication and subscription affects other service groups, such as the Declaration Management (DM) services. The Default Region within the DDM services encompasses the full Routing Space, yet is inaccessible to federates. From the perspective of the DDM services, other federates publishing Attribute Instances or subscribing to Object Class Attributes without regions, are assumed to be using DDM and be associated with the Default Region. The overlap between these service groups presents a challenge when determining which federates receive messages, when an Attribute with a region associated with it is updated. The DDM services also present a difficult concept to grasp, and an available implementation would be a valuable learning tool.

Using the DMSO 1.3 specification is a disadvantage in DDM research since improvements in the DDM services are available. The use of routing spaces was inflexible and made modifications to dimensions difficult for federate developers. Regions having multiple sets of extents associated with them was a source of confusion for users, and the relationship between sets of extents and a region needs to be addressed by the DDM services and matching algorithm. Continuing to use this version of the specification is holding back the DDM research community.

4.2 DDM within the SIP-RTI

As described in Chapter 3, above, a DDM extension to the cRTI Model was proposed in [23]. This solution proposed that sub-conferences, called Sidebars are added to the Attribute conferences in order to filter the DDM messages from other subscribers that did not overlap with a particular Attribute Instance publisher. Subscribers with Attendees in the conference that did not overlap, but were using regions, were placed into Wait Zones, another container of sorts, which would not receive messages sent by the overlap sub-conferences. A Doorman would control access to the Sidebar and Wait Zones. The Doorman would allow subscribers using the DM services to pass into the Sidebars because of the overlap of the DM and DDM services.

While this extension appears to cover all the bases in terms of the additional filtering, this proposal does appear to take certain issues for granted. The LCC itself does not know about DM or DDM, and the Attendee objects contain no application specific state themselves. The LCC therefore cannot send Attendees into either Wait Zones or Sidebars without communication with the LRC. The Doorman is assumed to provide this functionality, though it is unclear whether the Doorman would operate through application calls to the LCC, or if the LCC would use the Doorman to request information from the application. Furthermore, by adding these Sidebars

and Wait Zones, the Attendee objects that are placed in them will essentially be duplicated. The more object instances using DDM, and in turn conference floors, the more Sidebar and Wait Zone objects will be required. The paper also only deals with the local conference structure. Each of the Sidebars and Wait Zones will need to be duplicated on the remote notes, with their Attendees being adjusted and Indicators being updated for them as well. While seemingly a straight forward solution, there are several difficulties surrounding the implementation of conference filtering in this manner.

4.3 Limitations of the Conferencing Infrastructure

The Conference Infrastructure provides a means of communication between applications through Speaker and Listener Attendees within conferences. The conferences can be used to give applications a broad means of filtering messages, which can be based on application specific needs. Conferences can be nested to create a hierarchy of filterable objects. When the LRC parses the FOM, the Object Classes and Attributes are created using conferences and sub-conferences to mimic the Object Class structure from the FOM.

The filtering provided by the conference does not allow any announcement messages to be received by Attendees outside of a conference. While this kind of filtering is necessary for Object Class Attributes in the RTI, the rigidity makes using the CI difficult for DDM publishing and subscribing. The overlap between DM and DDM requires that some of the subscribed federates can receive Attribute updates while others do not. Providing this kind of dynamic and selective filtering using sub-conferences requires continual maintenance of the Attendees by the LRC.

Tagging was developed to allow users to create filters on data at run-time, and without prior knowledge of other tags or data within the system. Tagging Attendees

provides the kind of dynamic and selective filtering required by the DDM services, without the application needing to create and maintain knowledge of additional Attendees or sub-conferences.

4.4 Statement of Thesis

The thesis of this research is that DDM research embodies more than just Matching Algorithm research. The thesis is comprised of the following three claims:

- Region data organization can be used to improve the Sort-Based matching algorithm by changing the focus of the matching algorithm, and by dynamically adjusting the regions being matched to eliminate regions that do not overlap as early as possible.
- Adding tags to Attendees within the CI provides finer grained filtering for messaging, without requiring changes to the conference structure at run time.
- DDM and Matching Algorithm research can be extended to focus on the IEEE 1516 Specification, and DDM research can be shown to include the broader aspect of the DDM services, including a matching algorithm as one component.

4.5 Contributions

The claims of the thesis are realized by the following three significant contributions to the body of knowledge surrounding both the RTI and the DDM services. The three contributions correspond to each of the three claims made.

4.5.1 Dynamic Region Reduction in the Sort-Based Matching Algorithm

The Sort-Based Matching Algorithm was modified to reduce the amount of work it performs overall, by utilizing a property of the overlap requirements mentioned in the RTI specification to dynamically reduce the set of regions to compare after each dimension is checked. This technique, dubbed Dynamic Region Reduction, will allow the matching algorithm to work more efficiently by always dealing with a reducing set of regions that still have the potential to overlap. The region data is also organized to aide the algorithm by only having one region to use as the test control to maximize the efficiency of the Dynamic Region Reduction. The Dynamic Region Reduction technique is explained in Chapter 5. Tests are performed to compare the improved algorithm with the original, and the results are discussed in Chapter 9.

4.5.2 Tag-Based Filtering in the Conference Infrastructure

A tag-based filtering system was implemented in the LCC to support the DDM overlap state and filtering requirements. Tagging is application independent, and can be used by other non-LRC applications using the Conferencing Infrastructure without constraining them to using it in the same manner as the LRC. This filtering mechanism is a great benefit to the Conferencing Infrastructure by adding some versatility to message filtering without having to create a large hierarchy of conferences. A discussion on the use of sub-conferences versus tagging is given in Chapter 6, along with the tagging implementation details.

4.5.3 An IEEE 1516 DDM Implementation

The DDM services were implemented according to the IEEE 1516 HLA RTI specification. Most research on DDM focuses on the matching algorithm aspect, which is only one part of the requirements for a functional set of DDM services. Moreover, the current research is based on the DMSO RTI, which has since been improved upon in the IEEE version of the HLA. A functional DDM implementation using the IEEE 1516 DDM specification provides knowledge on how the DDM services can be implemented, with region information shared amongst RTI nodes. The overlap between the DM and DDM services is an important part of a DDM services implementation. An implementation of the DDM services also allows different matching algorithms to be compared fairly, because organization of the region data, which will ultimately be used to feed the matching algorithm, will be available in a consistent form for all matching algorithm designers. The Sort-Based matching algorithm is implemented with the DRR technique to provide the matching algorithm for the DDM services. The implementation of the DDM services is described in Chapter 7. Since the RTI operates in a distributed environment, data distribution details are discussed in Chapter 8, and a test federate created to ensure the services were functional is described in Chapter 9.

4.6 Scope

As with any large software project, concessions were made to focus the essential goals of the project and to identify ancillary objectives that would be nice to have, while maintaining a manageable amount of work to be completed. Implementation was scoped to what would be required to implement the DDM services within the LRC and support the overlap between DM and DDM service groups, and to modify the

LCC to support tagging.

There were some limitations that were imposed on federate data exchange from the implementation of the SIP-RTI from the exclusion of Interactions. Interactions were determined to be a simplified form of sending messages between federates, and so were left out in favor of implementing the conference structure to allow DM and OM functionality. This remains true still with the use of Interactions with regions. Similar to how Interactions are an extension of DM publish/subscribe, Interactions with regions is an extension of using regions, so they can be implemented easily at a later time for completeness. The methods in the DDM API that were concerned with using Interactions were omitted from the implementation. The methods that were implemented are summarized briefly in Appendix A.

While the SIP-RTI only implemented the DM and OM services, this work only focuses on the DDM services. There are other service groups within the RTI specification yet to be added to the SIP-RTI, but these were not considered. The overlap between the DM and DDM services was examined, however. This is a necessary situation to consider, given the similarities in the services, and that the specification described the interaction between the two.

The Conference Infrastructure layer was largely unmodified, except where necessary to provide the tagging functionality. Tagging is implemented for Listener Attendees and Listener Indicators only, since tagging is applied from the LRC where messages send to the conference are created. The LRC will be aware of which tags are necessary when making conference announcements. Where necessary, the CI API was modified to support tags as a parameter or to provide additional methods to interact with tags. The methods added or modified in the CI are summarized briefly in Appendix B.

Chapter 5

Dynamic Region Reduction in the Matching Algorithm

This chapter introduces and discusses the Dynamic Region Reduction (DRR) technique which has been added to the Sort-Based matching algorithm [18, 19], and further improvements made to generalize the Sort-Based matching algorithm. This technique was developed to help reduce the amount of work that the matching algorithm needs to perform as it iterates through the dimensions while processing a set of regions. DRR will remove regions from successive iterations of the matching algorithm as soon as it is determined that they will not overlap. The DRR technique is discussed in detail, and examples are provided to illustrate the concept.

Another generalization of the matching algorithm that goes hand-in-hand with the DRR technique is the organization of the region data being sent to the matching algorithm. The DRR technique is most effective when the regions being compared form a one-to-many relationship, since there will only be one region used to determine overlap. The matching algorithm can be modified to use either the publishing or subscription region as the basis for comparing regions.

Since the SIP-RTI is based on the IEEE 1516 HLA specification, modifications

are necessary to implement the algorithm for region objects that follow the IEEE specification.

5.1 Dynamic Region Reduction

The Dynamic Region Reduction (DRR) concept is derived from a property of the overlap calculation as outlined in the RTI specification [11]. The specification indicates that two regions can be in overlap with each other only if the ranges on all supplied common dimensions overlap pair-wise. When the regions are being compared, a subscribing region can be determined to be out of overlap with the publishing region as soon as one pair of ranges do not overlap. The publishing region is the control region, since the algorithm uses it as the basis for matching subscribing (test) regions. Once a region is known to be out of overlap, there is no further reason to continue checking the remaining dimensions of that region.

In the best case, all the regions used do not have an overlapping set of ranges on the first tested dimension. The overlap calculation would be finished, since the result is known - there is no overlap. The worst case would be finding ranges that do not overlap on the last tested dimension. In this case, there would be no benefit from DRR, since all of the dimensions will have been checked regardless. Two examples are provided to solidify the concept; a simple example to show how DRR works with one publishing region and several subscribing regions, and a more complicated example with more than one publisher. The second example also highlights the need for enforcing the one-to-many relationship between the control and test regions being matched.

5.1.1 A Simple Example

To help illustrate the DRR technique in practice, a simple example will be discussed, featuring a set of four Planes flying in a circular pattern near a Radio Tower. Figure 5.1 shows the Planes ("Plane 1" to "Plane 4") flying around their flight path, and the Tower ("Tower 1") they are communicating with. The circles behind the Planes and Radio Tower show the radio ranges of the objects; the Radio Tower has a radio range of 600 meters, and the Planes each have a range of 400 meters. As the planes fly around the flight path, they will fly into and out of the radio range of the tower. This example will focus on the reduction of the subscription regions, using the DRR technique.

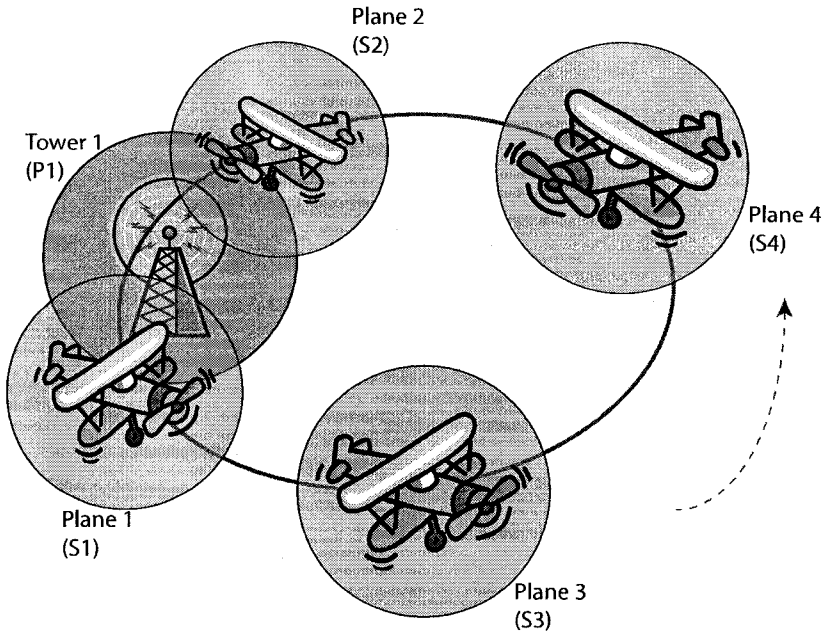


Figure 5.1: Simple Example Scenario

The objects in figure 5.1 can be translated into a simple two dimensional Routing Space, with dimensions for Latitude and Longitude as done earlier in Chapter 3.

Regions for both the Planes and the Radio Tower are made up of a range on both of these dimensions. A view of the Routing Space of the example scenario is shown in figure 5.2. The Radio Tower, which is broadcasting a message is represented by the publishing region labeled "P1". The planes each have their own subscription regions, labeled "S1" through "S4". The dimension for Longitude is labeled "x" and Latitude is labeled "y".

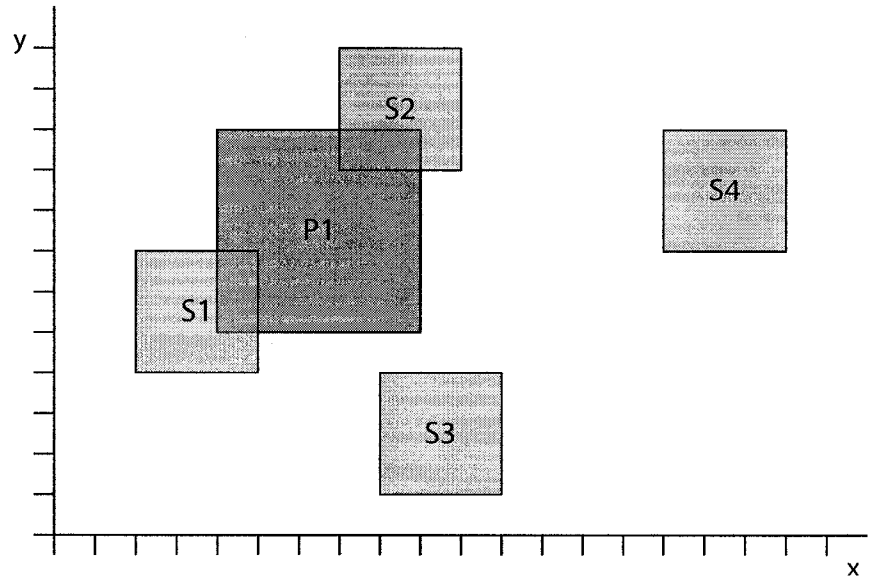


Figure 5.2: Simple Example Routing Space

The matching algorithm is run to determine which subscription regions overlap with the publishing region P1. Following the Sort-Based algorithm described in Chapter 3, one dimension is selected, and the ranges for the regions in that dimension are sorted in increasing order. Figure 5.3 shows the regions sorted on the x dimension. The matching algorithm iterates through the extent bounds, finding subscription regions S1, S2 and S3 open within the bounds of P1. S4 has not yet been encountered when the upper bound of P1 is reached, so it does not overlap with P1.

The second dimension is processed next. Before the region bounds are sorted, the

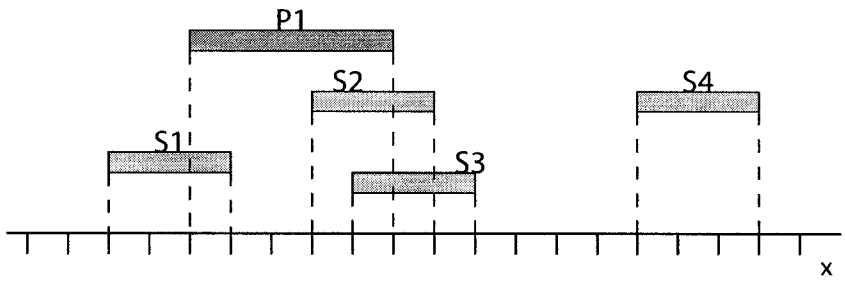


Figure 5.3: All Regions on the X Dimension

DRR-enhanced algorithm will check for regions that did not overlap in the previous iteration on the x dimension. S4 did not overlap, so no ranges will be used from that region. The ranges for regions P1, and S1, S2 and S3 are remaining, and are sorted. Figure 5.4 shows the remaining region bounds on the y dimension. Subscribing region S3 is fully encountered before publishing region P1. S3 does not overlap on this dimension, and therefore does not overlap with P1.

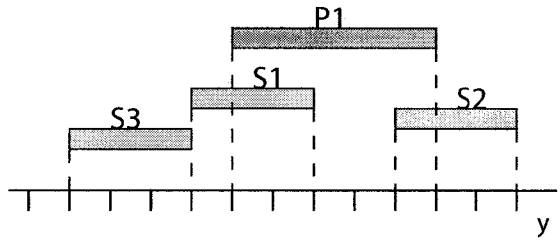


Figure 5.4: Remaining Regions on the Y Dimension

The DRR technique allowed the reduction of one subscription region in this example. Without it, S4 would have been checked on both dimensions, even though the final result would have been the same; that only regions S1 and S2 overlapped with the publishing region.

5.1.2 A More Involved Example

The previous example illustrated a simple case with only one publisher. This is an ideal case, because regions can be removed as soon as they do not overlap. In an actual federation execution, there can be more than one publishing region associated with an attribute. The scenario in figure 5.5 expands the simple example from figure 5.1 by adding a second Radio Tower, "Tower 2". As the Planes fly along the flight path, they will pass through two separate radio signal ranges. This example shows that even in the presence of multiple publishing regions the DRR technique can still be used.

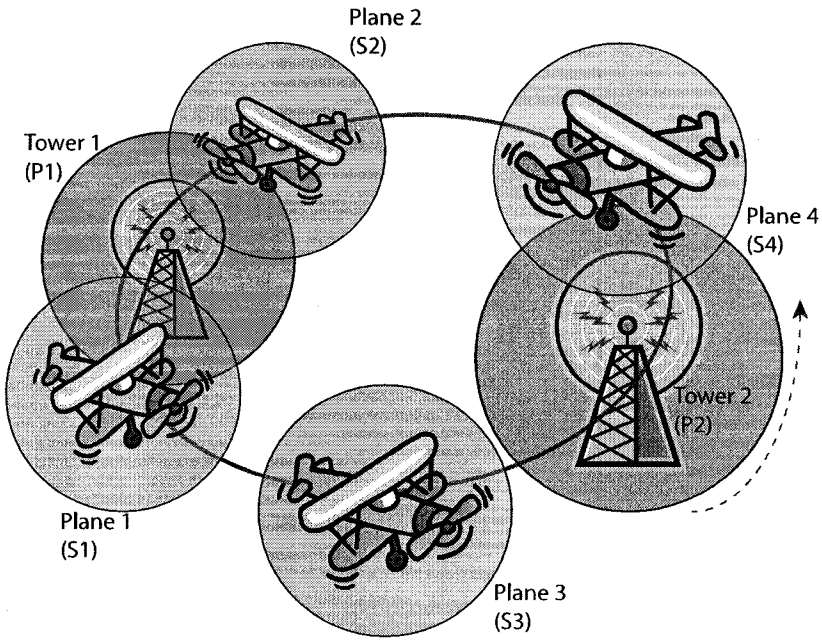


Figure 5.5: More Involved Example Scenario

The objects from figure 5.5 are mapped to regions within a two dimensional routing space, shown in figure 5.6. The x dimensions represents the Longitude, and the y dimension is the Latitude of the simulated environment. Tower 2 is represented by

the publishing region "P2". The planes are represented by the subscription regions S1 through S4.

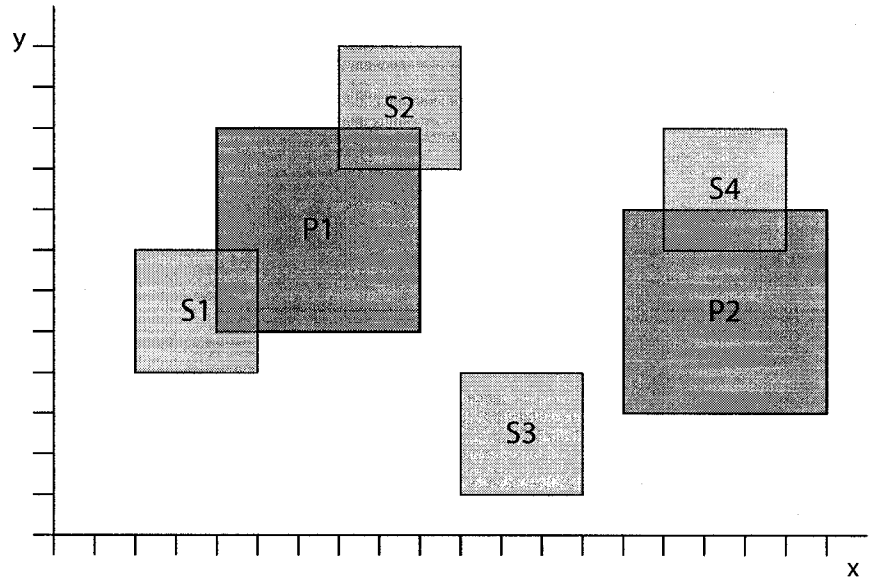


Figure 5.6: More Involved Example Region Space

The matching algorithm begins with the x dimension. All of the regions are used, and the extents bounds are sorted along the x dimension, as shown in figure 5.7. The bounds of all the regions are processed on this dimension, and S3 is the only region that is not encountered while a publishing region was open.

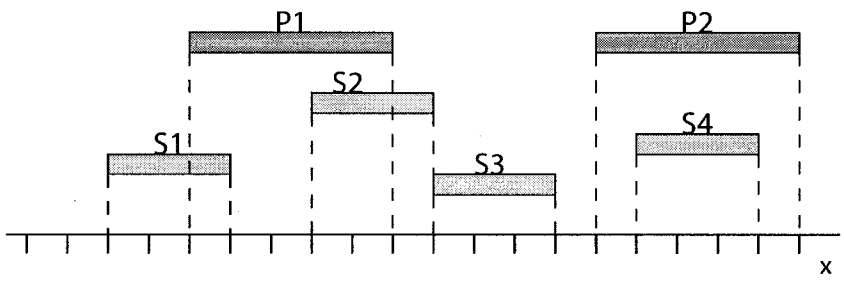


Figure 5.7: Regions on the X Dimension

The ranges are then organized for the y dimension, as seen in figure 5.8. Since the region S3 did not overlap with both publishing regions on the x dimension, it is left out of this iteration. If the region had overlapped with one of the publishers in the previous dimension, it would have to be included in this iteration. In the presence of multiple publishing regions, such as in this example, the DRR technique can only safely remove a subscribing region from future iterations once it has been determined that it does not overlap with all of the publishing regions. If a region does not overlap with all publishers on one dimension, such as the case of S3 on the x dimension in figure 5.6, then the region must remain for successive iterations until a dimension is reached where the subscribing region does not overlap with all of the publishing regions.

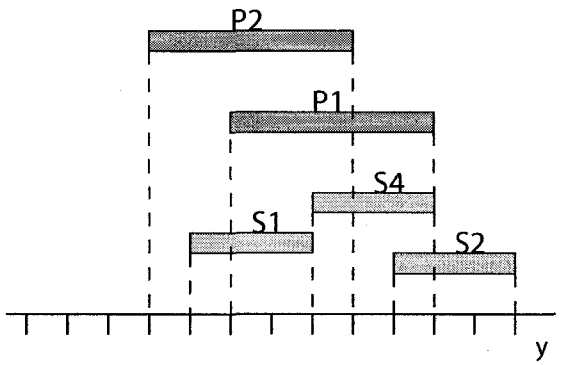


Figure 5.8: Regions on the Y Dimension

After the matching algorithm has processed the ranges on the y dimension, the final overlap results are that publishing region P1 overlaps with subscribers S1 and S2, and publishing region P2 overlaps with subscription region S4.

5.2 Generalizing the Matching Algorithm

The inclusion of the DRR technique in the Sort-Based matching algorithm provides motivation for further improvements to the organization of region data and design of the matching algorithm. The one-to-many relationship of control to test regions provides the ideal scenario for the DRR technique, since reduced regions only need to be out of overlap with one control region. When the matching algorithm is required to run, the control region is determined by the calling federate. The algorithm must be flexible to allow the support the control region being either a publishing or subscription region.

The SIP-RTI follows the IEEE 1516 HLA specification, while the original Sort-Based matching algorithm follows the DMSO 1.3NG specification. The algorithm can be updated to follow the IEEE 1516 specification with minimal modification.

5.2.1 Region Organization

While the more involved example showed that the DRR technique could be used in the presence of multiple publishing regions, it was not able to operate as freely as in the first example in figure 5.1. The reason for this was because there were multiple publishing regions, as the control regions. The Sort-Based Matching Algorithm, was designed to match publishing regions against subscribing regions. The publishing region is the region that controls the communication, so it is an acceptable assumption to design the algorithm around the publishing regions.

The Sort-Based matching algorithm was designed with publishing regions as the control regions to be compared against. The algorithm stores overlapping subscription region information with the publishing regions used in the matching. This design focuses the matching to always use the publishing regions as the basis for matches. This bias towards publishing regions can be illustrated by the example scenario pre-

sented in figure 5.9. The plane ("Plane 1") in this scenario is flying past three radio towers ("Tower 1", "Tower 2", and "Tower 3"), and in this case not overlapping with any of them. Being stationary objects, the radio towers will not be updating their regions, so matching is performed whenever the plane moves and updates its subscribing region.

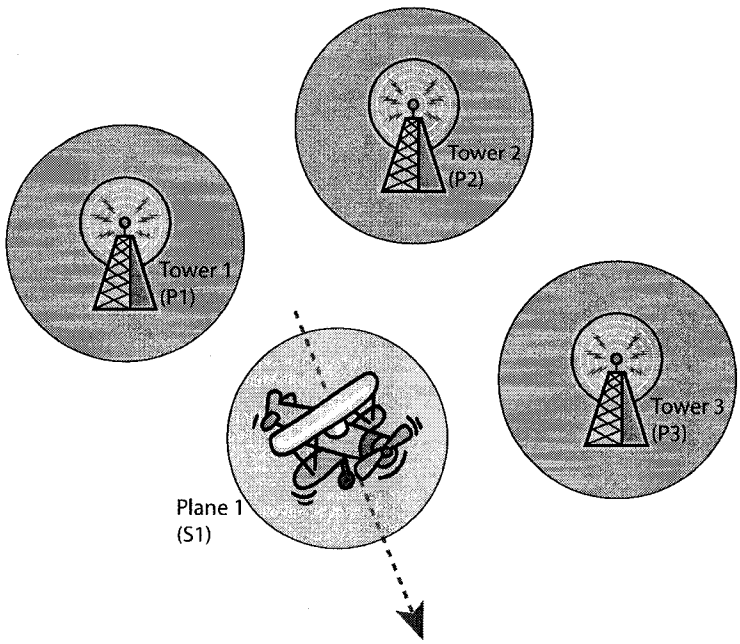


Figure 5.9: Publishing Region Bias Example Scenario

The regions of the radio tower and planes within the Routing Space are shown in figure 5.10. The publishing regions for the three towers are identified by "P1" through "P3", and the plane's subscription region is labeled "S1". The plane in this case is flying between all of the radio towers, and has modified its subscribing region, moving it one unit to the right. It is out of overlap with all three of the towers.

When the matching is performed, the x dimension is examined first. Figure 5.11 shows the region bounds on the x dimension. The matching algorithm will need to examine all the boundaries, as this is the first iteration of the algorithm. The

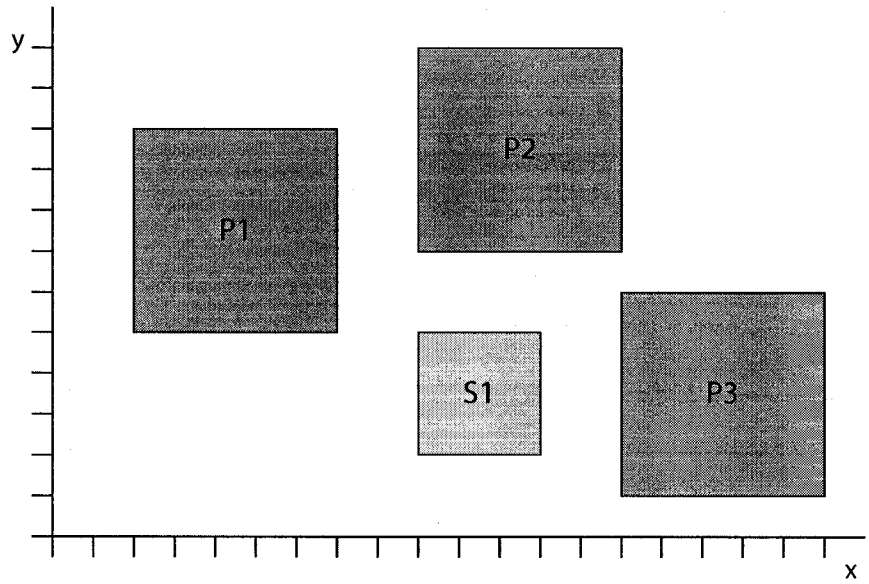


Figure 5.10: Publishing Region Bias Example Region Space

subscribing region does not overlap with the bounds of publishing regions P1 and P3, but the subscribing region cannot be reduced, because on this dimension, it overlaps with publishing P2. For the next iteration on the y dimension, all four regions will be used again.

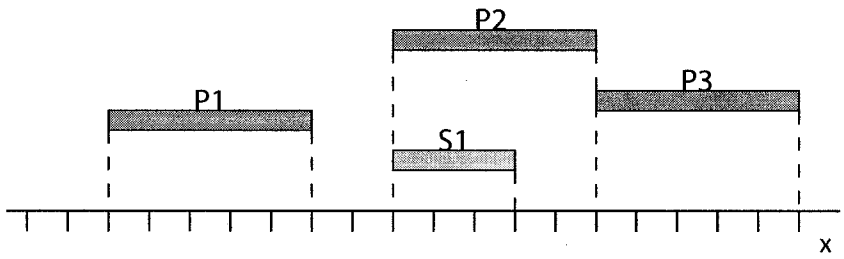


Figure 5.11: Regions on the X Dimension

All three publishing regions shown in figure 5.12 are used when performing the matching on the y dimension, even though S1 was already determined not to overlap

with publishers P1 and P3. The algorithm is using the publishing regions as the basis for the matching, and is keeping track of which subscription region is overlapping with each of the publishing regions. The DRR technique provides no gain in this case because the subscription region can only be removed if it does not overlap with all three of the publishing regions. The many-to-one relationship between the control regions and the test region hinders region reduction.

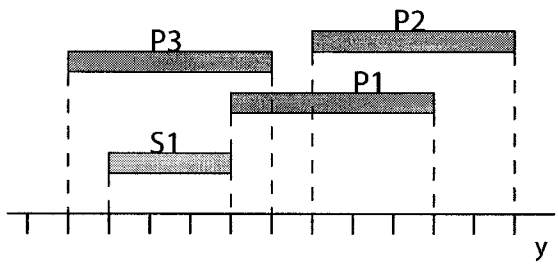


Figure 5.12: Regions on the Y Dimension

If the perspective of the algorithm was altered in this case to check from the point of view of the subscribing region, a one-to-many relationship between the control and the test regions is achieved. The DRR technique can be used during the matching process, and non-overlapping publishing regions can be reduced from successive iterations. Using the subscribing region as the control, figure 5.13 shows the bounds arranged on the x dimension. Since this is the first iteration, all four regions are present. Since the publishing regions P1 and P3 are found not to overlap with S1 during this iteration, they can be reduced out of the next iteration.

Figure 5.14 shows the regions that will be processed for the y dimension. P1 and P3 are not present since they were reduced out in the x dimension iteration. The matching algorithm now only has one test region to process, reducing the number of bounds to be processed by two-thirds. Region P2 does not overlap after this iteration, and the result for subscribing region S1 is that it does not overlap with any of the

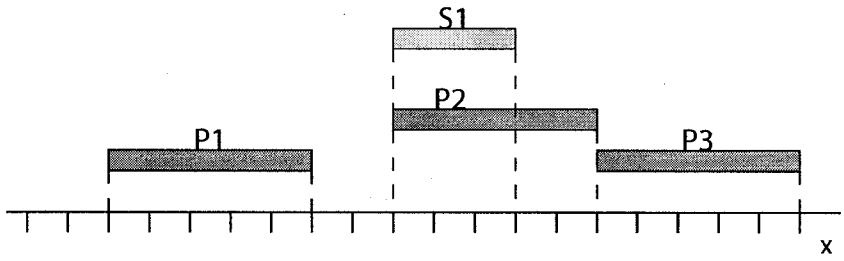


Figure 5.13: Test Regions on the X Dimension

publishing regions.

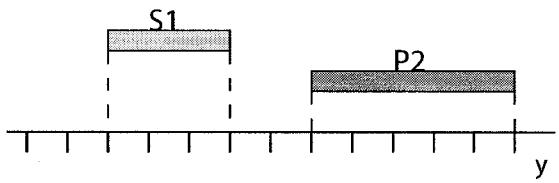


Figure 5.14: Reduced Test Regions on the Y Dimension

The one-to-many relationship helps the matching algorithm take advantage of the DRR technique by ensuring that only one region is the control, and any regions not found overlapping during an iteration, can be reduced from successive iterations.

5.2.2 DMSO 1.3NG DDM vs. IEEE 1516 DDM

With the DRR technique and the control region focus in mind, changes to the Sort-Based Matching Algorithm were required. The algorithm was designed with the DMSO HLA DDM services in mind, while the SIP-RTI was designed to follow the IEEE HLA specification. The changes in DDM services between the DMSO HLA and the IEEE HLA bring differences to the DDM services from the federate's perspective, and from the implementation of the region objects. For the matching algorithm itself, fewer changes are required. Rather than getting the set of dimensions to use from a

named Routing Space, the common set of dimensions on all regions being matched can be used instead. Extents are merely the range bounds of a region within a Routing Space, so the ranges associated with a dimension within a Region Realization can be used to the same effect, and with the simplified understanding that an extent will only map to one Region Realization.

5.3 Updating the Matching Algorithm

The DRR technique is added to the algorithm via a few additional lines to the pseudo-code. The effects of the one-to-many region relationship are seen through the data and when regions are loaded to use by the matching code. The determination of which region type is the control is left out of the matching algorithm itself. The updated pseudo-code can be seen in figure 5.15. The modifications, either changes or additions, from the original pseudo-code shown in figure 3.3 are highlighted in bold, and are discussed below. See the discussion of the original algorithm in Chapter 3 for details regarding the un-highlighted steps.

The first modification made to the algorithm appears on line 1 in figure 5.15. The set of extents that will be used for the algorithm iterations is no longer static, as the DRR technique will remove the extents that do not overlap. The set of remaining extents is calculated at the end of an iteration of the loop. Initially, the list starts out with all of the extents.

The two lists that will store the extents encountered before and after the control extent have been renamed in lines 7 and 8. Originally, they were called `SubscriptionSetBefore` and `SubscriptionSetAfter`. Functionally they are the same lists, however a generic name is more appropriate since the algorithm is now ambidextrous regarding which extent is being used as the control.

A set is added to the algorithm: the `CurrentOverlapExtents` set. This is also a

```
(1)  for each extent R_i in the set of remaining extents
(2)  {
(3)    insert lower bound point of R_i into list L
(4)    insert upper bound point of R_i into list L
(5)  }
(6)  sort list L
(7)  TestSetBefore = 0
(8)  insert all subscription extents into TestSetAfter
(9)  CurrentOverlapExtents = 0
(10) for all point P_i in the sorted list L
(11) {
(12)   R_i = Extent Id of P_i
(13)   if (R_i is a test extent) {
(14)     if (Pi is the lower bound point of R_i)
(15)       remove R_i from TestSetAfter
(16)     else  // P_i is the upper bound point of R_i
(17)       insert R_i into TestSetBefore
(18)   } else {  // R_i is a control extent
(19)     if (P_i is the lower bound point of R_i)
(20)       all extents in TestSetBefore do not overlap with R_i
(21)     else
(22)       all extents in TestSetAfter do not overlap with R_i
(23)       all overlapping extents updated in CurrentOverlapExtents
(24)   }
(25) }
(26) remaining extents to calculate list updated based on CurrentOverlapExtents
```

Figure 5.15: Modified Sort-Based Matching Algorithm Pseudo code

bit array, like the `TestSetBefore`. This set will contain the mapped extent bits, of the test extents that have been found to be in overlap with the control extent during the current loop iteration. It is initially empty, but the bits are set within it when the final set of overlapping regions is determined on line 22.

Line 13 in figure 5.15 has been slightly modified to check for a test region, rather than a subscription region, as was originally specified in figure 3.3. Similarly, line 18 now handles a control extent, rather than a publishing extent.

Line 23 is where the extents that are overlapping the control extent are determined. After that, the set of overlapping extents in `CurrentOverlapExtents` is modified to reflect the extents that are in overlap for control `Ri`. This action is performed by ORing the final overlap set for that dimension, with the results already in `CurrentOverlapExtents`. The bits in this set remaining 0 at the end of this run of the calculation will be the extents that do not overlap with any control extents.

Line 26 in 5.15 updates the set of extents to be used for the next iteration of the matching algorithm. This is performed by ANDing the `CurrentOverlapExtents` set with another, which is used in line 1 to determine which extents will place bounds in the list `L`. This operation will leave only the extents that have a value of 1 in both bit sets. Where the `CurrentOverlapExtents` set has a 0, that extent will be removed from the set, and the reduction is performed.

The DRR technique provides a reduction in the amount of work that needs to be done to determine region overlap which is highly dependent on the region data being used. If none of the test regions overlap with the control on the first dimension tested, then the algorithm can complete it's execution at that time. In the worst case, all of the regions being tested will overlap with the control, and all the dimensions will need to be fully processed. In that case, the theoretical performance of the DRR-enhanced Sort-Based matching algorithm is be same as the original Sort-

Based matching algorithm described in Chapter 3. The worst-case performance of the algorithm is $O(n^2)$.

5.4 Implementing the Matching Algorithm

With the modifications made to the matching algorithm as outlined above, the pseudocode was translated into functional Java code. The classes that were created are shown in figure 5.16. The main class, `HeapSortMatchingAlgorithm` contains the implementation of the improved Sort-Based matching algorithm. The `Extent` class contains the region data, such as the set of dimensions and range bounds, the region ID, and the owner ID. The `Bound` class will be used to store either the upper or lower bound of an extent. The `Bounds` are what are placed into the sorted list L in lines 1 to 6 in the algorithm in figure 5.15.

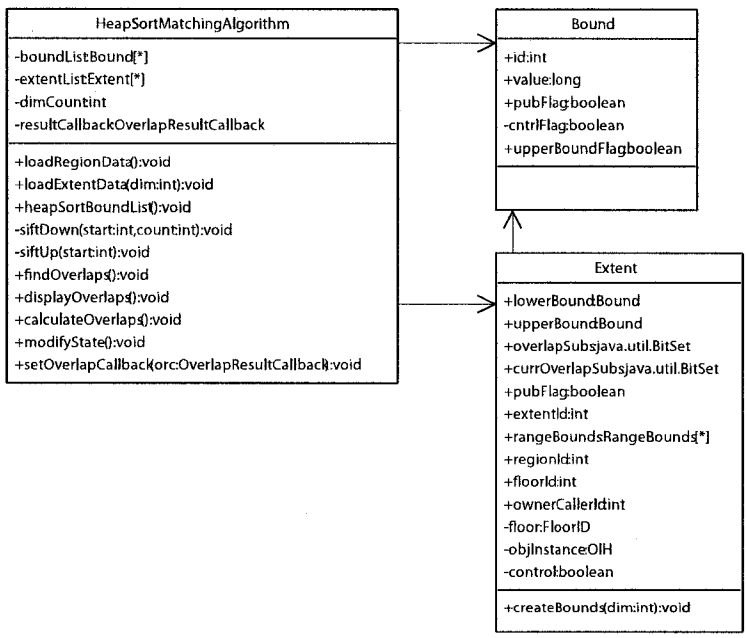


Figure 5.16: Sort-Based Matching Algorithm Java Classes

The work of the algorithm is split up between three methods within the `HeapSortMatchingAlgorithm` class: the `loadExtentData`, `heapSortBoundList`, and `calculateOverlaps` methods. The `loadExtentData` method will perform lines 1 through 5 from the algorithm in figure 5.15. Line 6 is performed by the `heapSortBoundList` method, which will use the Heap Sort algorithm to sort the list of `Bound` objects for the extents being used for the current iteration of matching algorithm. The rest of the algorithm, lines 7 through 26, is performed in the `calculateOverlaps` method. The `loadRegionData` method is first called when the set of regions are given to the Match Manager (which is discussed in Chapter 7), and will create the extent objects used by the algorithm.

Both the `Extent` and `Bound` classes have a flag that indicates whether or not they are the control. This is important in both classes for different parts of the algorithm. When the extent data is being loaded for the next iteration, the `loadExtentData` will create the `Bounds` for the upper and lower bounds of the extent for the next dimension being examined. If it is found that only the control is being loaded, then the algorithm can stop, since there are no test extents left to check. The control flag on the `Bound` class is needed so the algorithm can decide what to do with it as it is encountered in the extent set, in line 13 of the improved algorithm.

The implementation of the Sort-Based matching algorithm provides a key element in the DDM services, and will support the matching requirements of the services. The DRR technique will help the matching algorithm to run more efficiently, and complete quicker, allowing the subscription effects of the DDM services to be applied to the conference Attendees sooner. The matching algorithm provides part of the solution to implementing the DDM services in the SIP-RTI. After the matching algorithm is run, the overlap results need to be saved somewhere. This is achieved through the modification to conference Attendees by adding tags indicating overlap state, which

is described in Chapter 6.

Chapter 6

Tag-Based Filtering in the Conference Infrastructure

The Conference Infrastructure (CI) provides applications with a means of sharing data by using a topic-based filtering approach. The CI is implemented as a distributed application by connected Local Conferencing Components (LCC) nodes. The conferences themselves can represent any class of data that the application requires, and participants in that conference are expected to publish or subscribe to that data.

When used by an application such as the RTI, conference-based filtering provides a static means of sharing data from different Object Class Attributes. Conference-based filtering does not easily facilitate the kind of fine-grained filtering within a conference that DDM requires. Conferences do not support any form of grouping of Attendees within the conference. A workaround for filtering Attendees within an Attribute conference can be achieved by creating more sub-conferences for each group, however Attendees need to be added by the RTI to each new sub-conference.

This chapter presents the modifications made to the LCC, namely the inclusion of Attendee filtering by user-definable tags. These tags allow the application using the CI to create a dynamic filtering scheme that does not require the creation of any new

conferences or Attendees. A problem scenario where using sub-conferences are an ineffectual solution is presented first to make a case for tagging. The tagging concept is then presented, followed by a discussion regarding its implementation.

6.1 Sub-Conferences and Conference-Based Filtering

The conferences within the CI provide separate channels for communication between Speaker Attendees and Listener Attendees. Conferences are distinct from each other, and there is no overlap between conferences, or parent and sub-conferences. The conferences can be created in a hierarchy to suit the application using the CI, and can be used to provide coarse-grained filtering of messages.

Messages are not filtered within a conference. Any announcement a Speaker makes is heard by all of the Listeners. Figure 6.1 shows an example scenario of four planes ("Plane 1" to "Plane 4") flying in a loop, with two radio towers ("Tower 1" and "Tower 2") sending messages. A satellite ("Satellite 1") is used to send messages to all four planes. This example is mapped to objects within a conference, representing the radio transmissions in figure 6.2. Within an RadioComm Object Class, the RadioComm.Trans Attribute conference contains Attendees for all of the simulated objects. The Speakers "S1" through "S3" represent the two radio towers (S1 and S2) and the satellite (S3). Each plane has a Listener Attendee, called "L1" through "L4".

A problem with the example in figure 6.2 is that there is no way, at the conference level, to send messages to a subset of the available subscribers in the conference. Messages will need to be filtered by the receiving application, and discarded if they are not needed. Figure 6.3 updates this scenario with radio signal ranges for each of the simulated objects. The rings around each of the planes shows the distance it can

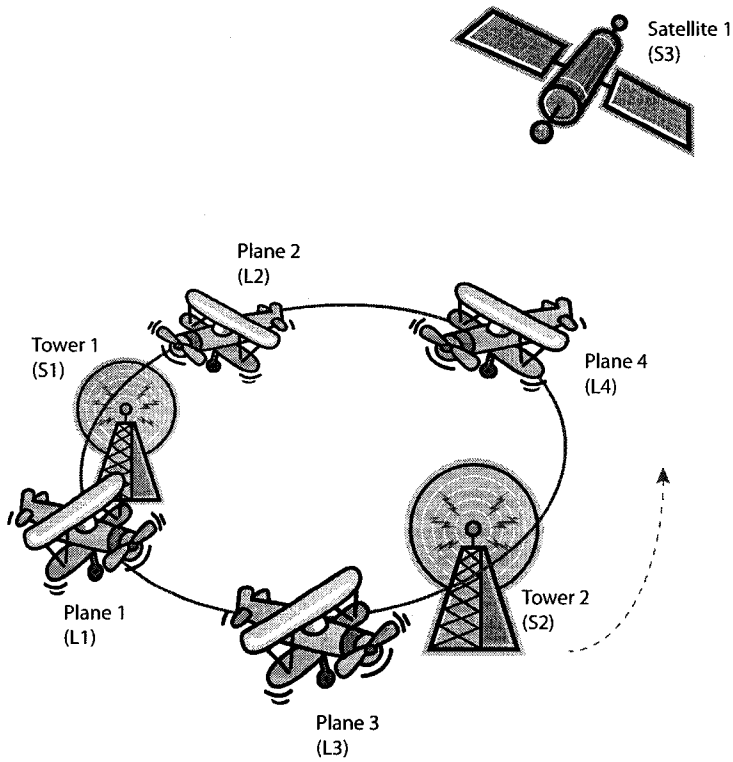


Figure 6.1: Conference-Based Filtering Scenario

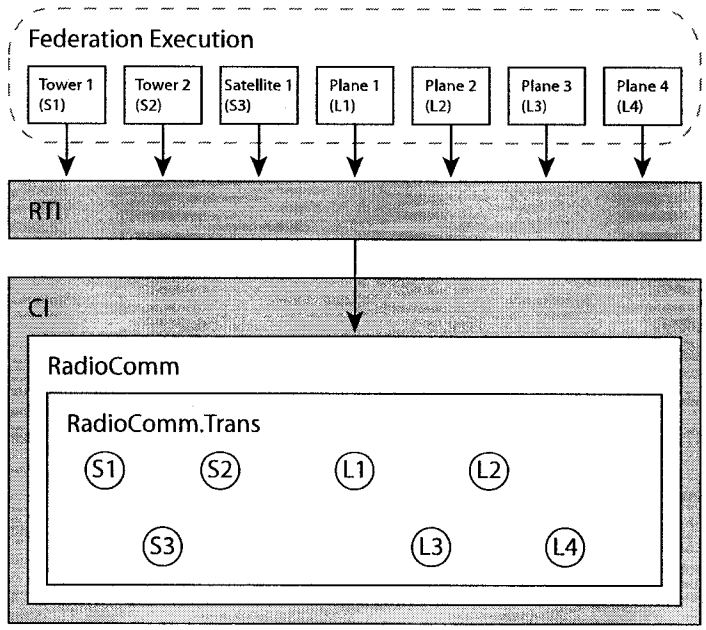


Figure 6.2: RadioComm.Trans Conference with No Sub-Conferences

receive signals within in, and the rings around the towers show how far the sent signals can travel before degrading. The satellite does not have any transmissions restrictions, and its messages can be received anywhere within the simulated environment. This radio signal range scenario highlights the need for a finer grain of filtering within a conference.

To facilitate some kind of filtering, sub-conferences can be employed to create sub-groups where messages are only received by a select group of Listeners. Speakers can then send a message out to that sub-conference without worrying about out of scope Listeners receiving the message. Figure 6.4 shows the updated conference, with a sub-conference used to separate each of the radio towers transmission ranges, and a third for the satellite. Since Planes 1 and 4 overlap with the radio range of Tower 1, listeners L1 and L4 are placed in the sub-conference with Speaker S1. Similarly, Planes 2 and 3 overlap with the radio range of Tower 2, so listeners L2 and L3 are

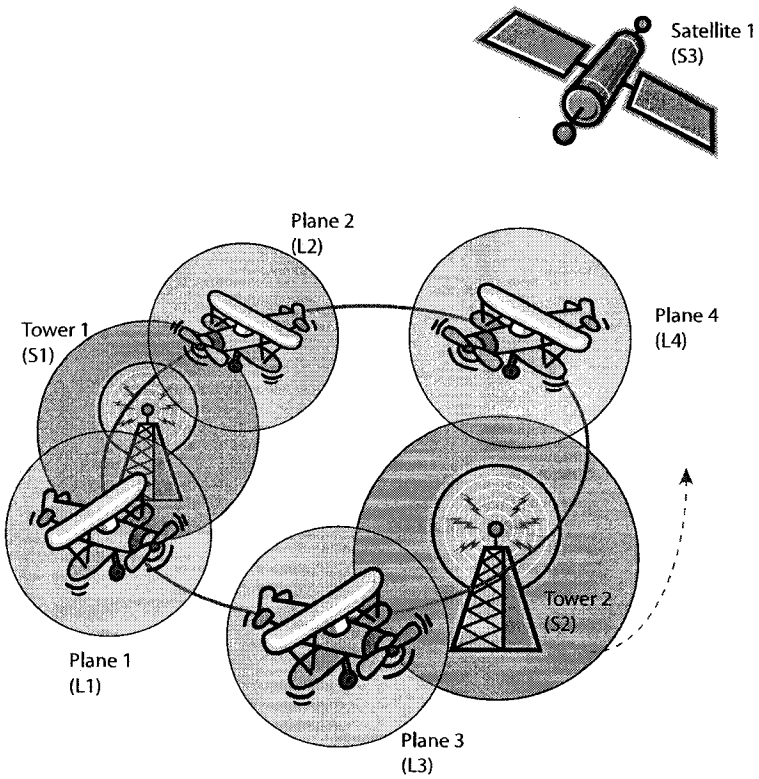


Figure 6.3: Conference-Based Filtering Scenario with Radio Signal Ranges

placed in a sub-conference with S2. The announcements made by speaker S3 for the satellite need to be received by all of the planes, so a sub-conference with S3 and all four listeners is used.

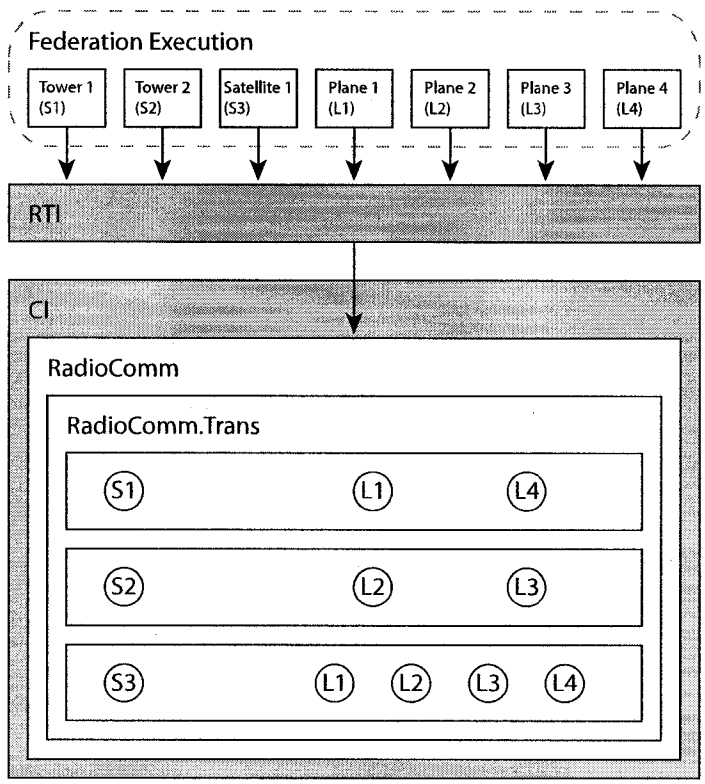


Figure 6.4: RadioComm.Trans Conference with Three Sub-Conferences

Using only conferences for coarse-grained filtering allows the application to separate Listeners and Speakers into groups of common interest. The CI does not need to do any processing when a message is sent, because all of the Listeners within the conference are interested in receiving it. For the RTI, this facilitates DM communication. DDM communication, however, requires a finer level of filtering, and the overlap between the DM and DDM services necessitates communication between filtered groups of Listeners. Using only sub-conferences, this filtering can be achieved,

however not without duplicating many of the conference Attendees.

Separate sub-conferences for each speaker shown in figure 6.4 illustrates how filtering can be achieved for DDM publication regions. The satellite Speaker S3 in the third sub-conference is using only DM to publish Object Class Attribute updates, so it needs to have all of the plane Listeners within it's conference. The onus is placed on the RTI to manage more than one Listener Attendee within the RadioComm.Trans sub-conferences. This configuration still does not address the need for a publisher with regions to make an announcement within a conference to DDM subscribers that are not overlapping with it. This special case comes into play when discussing sharing region information in Chapter 7.

The overlap between the DM and DDM services introduces conditions regarding where attendees can be placed within the attribute conference to ensure correct communication. An attribute can be used for both DM and DDM publications, so non-overlapping subscribers still receive updates from those publishers. Subscribing federates that do not use DDM need to receive all of the attribute updates, regardless of the publishing region. The conference attendee for that subscriber can either place a single Attendee within the Attribute conference, one have to place a Listener within each sub-conference. The latter requires the RTI hosting the subscribing federate to continually manage adding or removing extra Listener Attendees on behalf of the subscribing federate. When a publisher is using the Attribute without DDM, all of the Listeners, using DM or DDM, will need to receive the Attribute update announcements.

6.2 The Tag-Based Filtering Solution

Tagging provides a solution to the issues mentioned above, while also reducing the number of duplicated Attendees required within the conference, and the number of

sub-conferences that the application will need to manage. With tagging, all of the Attendees remain within the original Attribute conference, and messages are filtered by the CI based on the tags used when the Speaker makes an announcement. Sub-conferences for each publishing region is unnecessary, because a region-related tags can be added to Attendees that overlap with that region. DM and DDM subscribers can be differentiated by tags as well, indicating which type of subscription service they are using.

Figure 6.5 shows the example conference from figure 6.4 with tagged Attendees instead of sub-conferences. The Speaker Attendees, S1, S2 and S3 are all tagged with a name to identify the publishing region they are using. Listeners L1 and L4 are tagged with "tower1", since Plane 1 and 4 overlap with the radio range of that Tower 1, and listeners L2 and L3 are tagged with "tower2". All of the Listener Attendees are tagged with "satellite1" since the Satellite's signal in the example scenario can reach all the planes. The speakers make announcements using their tags, and only Listeners that have the same tag will hear them. Speaker S1 or S2 will only make announcement heard by two of the four Listeners in the conference. Announcements made by S3 will be heard by all of the Listener Attendees.

The Attribute conference supports sending messages for DM users, DDM users, as well as a combination of both types. Messages sent via the conference can be attribute updates or region updates. The filtering is achieved within the conference without requiring additional sub-conferences or Attendees. The application can add or remove tags from attendees to change the filtering easily. For the filtering requirements imposed by the addition of the DDM services, tagging provides a streamlined solution with minor impact on the conference structure at run-time.

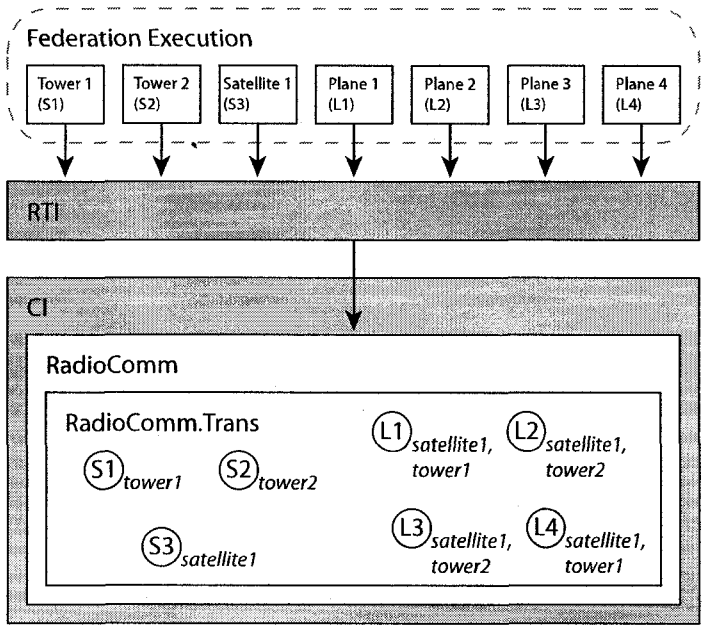


Figure 6.5: RadioComm.Trans Conference with Tagged Attendees

6.3 Conference Infrastructure Modification Details

Adding tags to the conference Attendees requires some modifications to the design of the local components that implement the CI. The LCC was initially designed with the use of sub-conferenes as the only means of filtering Attendees. Sharing conference state information, which is described in greater detail in Chapter 8, is extended to include tag information. The use of tags is described from the point of view of the local LCC node, describing remote node information where required.

The tags are placed on the local Attendees, and filtering is achieved when a conference announcement is made with a given subset of tags. Conference announcements can be used for either region information updates, or Attribute value updates. The Listener Attendees that have the matching tags will receive the announcement. Those that do not, will be filtered out and will not receive notification of an announcement.

The tags themselves are not implemented as Java classes. Tags are basic String objects, kept in a list within the classes that can be tagged. The application does not need to maintain handles for tags in use.

The Attendee class required only a few minor changes. Figure 6.6 shows the Attendee class, and only the parameters and methods added for tagging. The `listOfTags` list was added to the class, to keep track of the tags that have been given to that Attendee. Several accessor methods have been provided to add, remove or return the set of tags that the attendee is associated with.

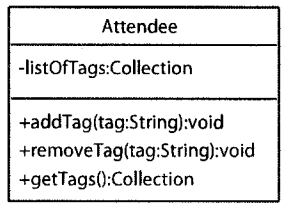


Figure 6.6: The Attendee Class

The Conference class was modified in a few areas. Figure 6.7 shows the additions and changes made to the methods of the Conference class. Only the methods modified or added are shown in this diagram. The method for adding an attendee was extended to allow a set of tags as a parameter, so they can be assigned to the attendee when it is created. Tags can be added to the conference for later use with the `addTag` method, or removed from the set of available tags via `removeTag`. Removing a tag that is in use will untag all objects that were associated with that tag. The `tagAttendee` and `untagAttendee` methods are used to add or remove a set of tags from an attendee object. `untagAllAttendees` will remove the given tags from all attendees that are associated with it.

The `makeAnnouncement` method is also extended to include a parameter for an array of tags. When the application is sending an announcement to the conference, the

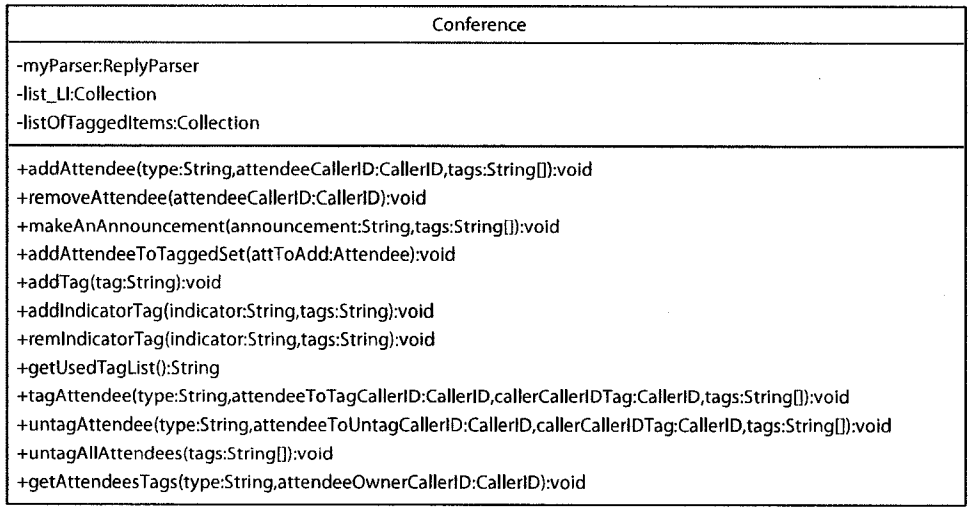


Figure 6.7: The Conference Class

tags that are supplied will be used to determine which listeners or remote nodes will receive the message. The tags are used to get the appropriate `TaggedItems` objects (figure 6.9) from the conferences `listOfTaggedItems` list. The listener attendees within the `TaggedItems` classes are sent the announcement. Remote LCC indicators are tagged as well, and are filtered the same way as attendees.

When messages are received from remote LCC nodes, the Reply Parser class examines the message, and then takes the appropriate action. To support tagging, two LCC command messages were added (and described in Chapter 8). Two new methods are added to the Reply Parser class as shown in figure 6.8. The `takeTAGAdd-Action` method handles a `TagAdd` command, and calls the `addIndicatorTag` method of the Conference. Conversely the `takeTAGRemAction` handles a `TagRem` command and calls the `remIndicatorTag` method in the Conference.

In order to organize and keep track of the tags and conference objects (local Attendees and Remote Indicators) that are associated with each other, a new object was created to manage these references. The Tagged Items class, shown in figure

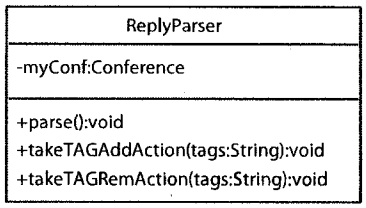


Figure 6.8: The ReplyParser Class

6.9 also reduces the number of lists required, by maintaining both attendees and indicator lists in the same object. A list of tagged items, the `listOfTaggedItems` list is maintained in the Conference within a Java hash map, where the key into the set is the tag. Methods for adding and removing Attendees and Indicators are provided, as well as methods to check of the existence of tagged objects.

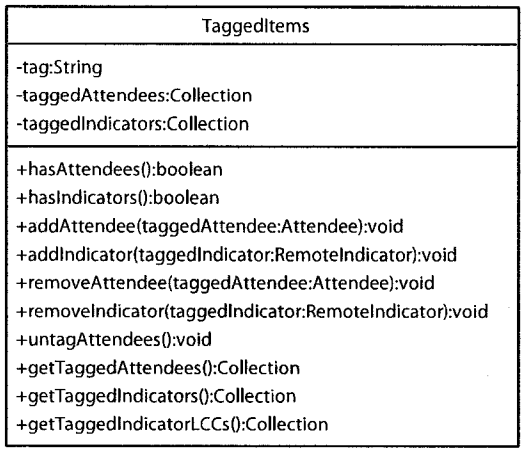


Figure 6.9: The TaggedItems Class

To maintain consistent state between remote LCC nodes, tags must be applied to remote indicators. Recall from Chapter 2 that indicators are used to denote attendees within remote LCC nodes. Tags are applied to Listener Indicators in a similar way, to express tag information on the remote Listener Attendee objects. More detail on sharing tags is given in Chapter 8.

When adding tags to Remote Indicators, a larger change was required. Originally, Remote Indicators were simply a set of Strings with remote LCC node names. The nodes were assumed to have one or more local Attendees within that conference. A small object was created to maintain some additional information with the node name, which is shown in figure 6.10. The `RemoteIndicator` has a `listOfTags`, much like an Attendee. The Remote Indicator will contain a set of tags, allowing it to hear announcements made with any of the tags used by Listeners at the node it represents.

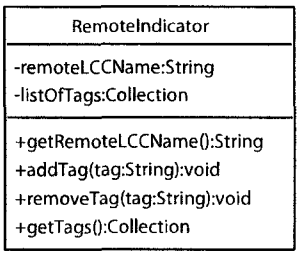


Figure 6.10: The RemoteIndicator Class

Tags have only been added to Listener Indicators in this implementation. Given the time constraints, only the implementation of tag sharing for Listener Indicators was possible. Applying tags to Speaker Attendees and Speaker Indicators would be an ideal future use of Attendee tagging. Tags could be used implicitly by the LCC, so when a Speaker Attendee sends a conference announcement, the tags associated with the Speaker are applied to the announcement.

Tags could have been applied to Floors in a similar manner, to assign Object Instance tags to Floors. Tags could be used to indicate floor ownership as well. Changes to the Floors and even conference objects was outside of the scope of this project, however, it would have opened new possibilities for implementing message filtering.

The classes discussed comprise the changes made to the LCC in order realize

tagging. The Conference Infrastructure can support filtering within a conference, which is ideal for an application with filtering requirements such as an RTI with DDM. Previously, the LCC could only provide filtering by the conference structure, and further filtering would require creating more sub-conferences. This method is still available, as tagging is not necessary, but it does provide a streamlined filtering solution for the DDM services, without the overhead of additional sub-conferences and attendees.

Chapter 7

DDM Services Design and Implementation

This chapter discusses the design and implementation of the DDM services within the SIP-RTI. The design adds a subset of the functionality of the DDM service group of the IEEE HLA specification as outlined in the project scope in Chapter 4, while having minimal impact on the SIP-RTI code already in place. Furthermore, the services provided by the LCC, in particular the attendee tagging, were utilized to support the filtering requirements imposed by the region overlaps.

The two-tiered design that was developed allows the LRC to provide the implemented functionality of the DDM services, while reducing the amount of information it needs to keep track of, and using the inherent publish-subscribe structure already provided by the LCC. The LCC already keeps track of nodes that are interested in subscribing or publishing within an Object Class, and the DDM services operate within the same objects as well. The LCC uses tags to maintain which nodes are subscribed using DDM as well as those using DM only. The LRC primarily manages regions for its local federates, though it will also manage remote federates' publishing regions as well, when relevant.

Within the DDM services, there are several region objects used to represent region information. The Region Specification (RS) and the Region Realizations (RR). Region Specifications are created for all local federates that create and use regions. The Region Realizations are the instances of the Specifications on Attributes within the RTI. These can be used for either publication or subscription. Region Realizations for remote publishing federates are created on local LRC nodes, if they contain a subscriber to the same Attribute. This sharing of region information is described in Chapter 8.

The DDM services in the LRC uses three main region objects: the Local Region Specification (LRS); a Local Region Realization (LRR), and a Remote Region Realization (RRR). Local Specifications and Realizations are owned by the federate using that LRC, and can be used for either publishing and subscribing. Remote Realizations are region realizations from remote publishing federates. The regions are copied at LRCs that have federates subscribing to the Attribute class the publishing region is associated with. These three region objects are the basis for using the DDM services and performing the overlap calculations.

The LCC plays an important role in the implementation of the DDM services. Attendee tagging, described in Chapter 6, is used to filter the updates for federates using DDM. Tags are applied to Listener Attendees within the conference after the matching algorithm is run. Updates are sent as conference announcements with a set of tags, and listener attendees with matching tags will receive the updates.

The implementation details of the DDM services are discussed by first addressing the changes to the cRTI model and the Conference Infrastructure. The Java classes developed to implement the services are then discussed. The DDM Services implementation consisted of approximately five thousand (commented) lines of code. High-level class diagrams showing only the class names are provided to illustrate the

relationship between the implemented Java classes.

7.1 Conference Infrastructure and Model Modifications

As detailed in Chapter 6 above, the Conferencing Infrastructure was modified to provide tagging to the local Attendee and Remote Listener Indicator objects within a conference. The tags are used by the LRC for filtering messages within a conference, and are the means of applying the required filtering to selectively send Attribute updates to overlapping subscribers. The tags applied to Attendees within the CI will be used to maintain and share region overlap state for the LRC.

7.1.1 The cRTI Model

Originally the cRTI model was created to provide a means of describing how the conferences would be used by the SIP-RTI application to share application specific information. While the conferences themselves are designed to be independent of any application, the cRTI model shows how they are used in conjunction with the SIP-RTI. The model is updated here to show how tagging is used by the SIP-RTI to help implement the DDM services.

Using tags with Attendees provides an easy solution to all of the issues (raised in Chapter 4) encountered with the current implementation of the CI regarding the filtering required by the DDM services. Tag-based filtering can achieve the required filtering for the DDM services, allow for the overlap between DM and DDM subscription effects, and also provide an application-independent filtering mechanism built within the CI's set of services.

The model presented in figure 7.1 shows the updated cRTI model with tagged

Attendees. The model is similar to that of figure 2.16 from Chapter 2. Another plane federate ("Plane 2") has been added to help illustrate how the overlap between the DM and DDM services is handled with tags. The Object Class RT Conference represents a Radio Tower and has two Attributes: RT.Trans and RT.HLAPrivilegeToDeleteObject. The federates for Tower 1 (FED A) and Plane 1 (FED B) are using DDM, and have Region Realizations within their local RTI nodes. Tower 1 has a region associated for publication, labeled "RRa", and Plane 1 has a subscription region labeled "RRb".

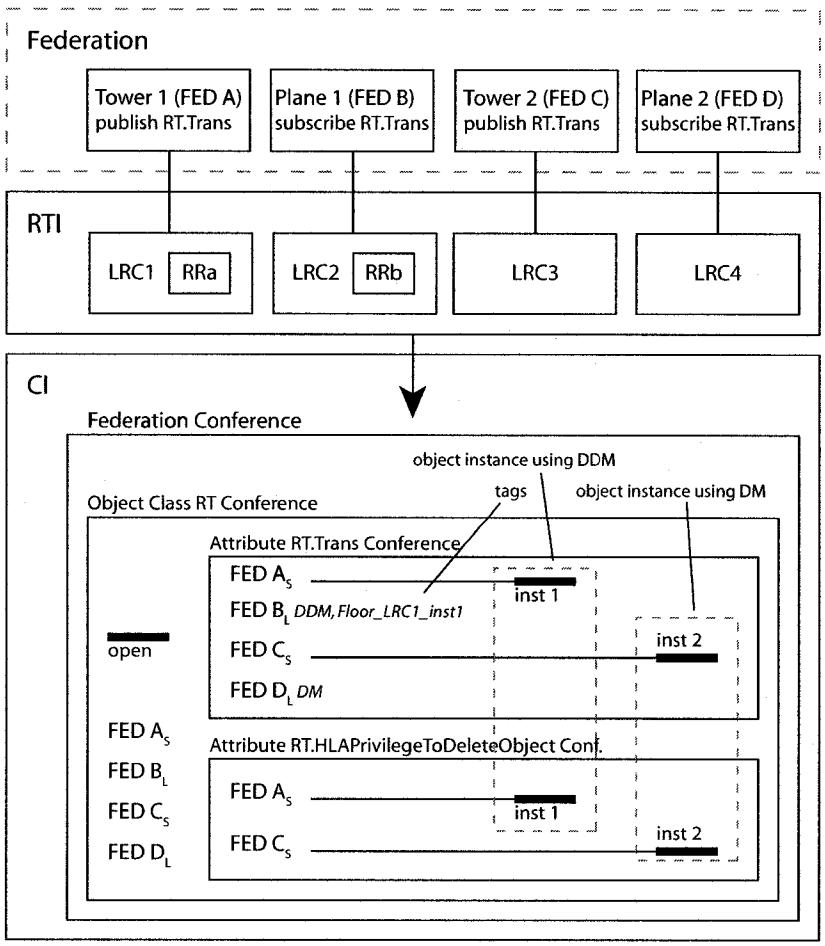


Figure 7.1: The cRTI Model, with Attendee Tagging

The subscribing federates in figure 7.1 have tags on each of their Listener Attendees

in the RT.Trans Attribute conference. In this example, Federates A and B are using DDM, while the Tower 2 (FED C) and Plane 2 (FED D) federates are only using the DM services for publishing and subscribing, respectively. The Attendee for FED B is given the "DDM" tag, and FED D is tagged with "DM". The Attendees in the HLAPrivilegeToDelete conference are left untagged, because there is no need for filtering in this conference. FED B is overlapping with the publishing region RRa, so it can receive DDM Attribute updates from FED A. FED B's Listener Attendee is tagged with "Floor_LRC1_inst1". The region overlap is determined in the LRC by the DDM matching algorithm.

The tags used for indicating region overlap are based on the LRC name and the ID of the Floor the Attribute Instance the publishing region is associated with. The LRC name is used to identify the node where the publishing region was created. The floor IDs are unique locally on their LCCs, but not within the CI. Combining them together ensures that the tag will be uniquely named within the CI.

Recall from background in Chapter 2 regarding the overlap between the DM and DDM service groups, that the RTI assumes the Default Region is used by DM publishers and subscribers. The Default Region covers the entire Routing Space, having a range from zero to the maximum value on each dimension. The Default Region will overlap with all publication and subscription regions created by federates. The DM and DDM tags allow Attribute updates to be sent using the Default Region, without accidentally sending updates to non-overlapping subscribers.

When an Attribute is updated using DM, such as RT.Trans in figure 7.1 by FED C, the RTI will send a message to the conference Attendees tagged with "DM, DDM". The use of these two tags will send the announcement to Listeners using both DM and DDM services for Object Class Attribute subscription. From the DDM perspective of FED B, an Attribute update using the Default Region is sent. From the DM

subscriber perspective of FED D, an Attribute Instance update was made using the DM services.

To send an Attribute update from a publisher using DDM, the publishing region will use the "DM" tag, as well as the tag that corresponds to its publishing region. FED A in figure 7.1 will use the tags "DM, Floor_LRC1_inst1". The announcement will be heard by Listener Attendees that have at least one of these two tags. In figure 7.1, Attendees FED B and FED D would receive the message. If FED B did not overlap with the publishing region RRa, it would only have the DDM tag, and would not receive the Attribute value update. Using the DM tag when sending an Attribute update from a federate using DDM addresses the overlap with the DM services, by including all the subscribers that are using the assumed Default Region.

7.2 DDM Services

The DDM Services were designed as a set of services supporting the LRC, which provided functionality for region manipulation and overlap calculation. The implementation of the DDM objects don't necessarily follow the API of the DDM services themselves. The implemented objects provide an API to the LRC which provides the necessary functionality to use the DDM Services. While a federate using DDM uses the RTI API through the RTI Ambassador as shown in figure 7.2, and the LRC uses the DDM services.

The functionality required for the DDM services shown in figure 7.2 is separated into smaller service groups: Regions Services, Dimension Services, and the Matching Services. The goal was the provide the required functionality in a way that would also be easy to extend and modify. This is especially true for the matching algorithm, which is implemented in a way to be easily modified without affecting the rest of the DDM service classes. The API for the DDM Services class extends those provided

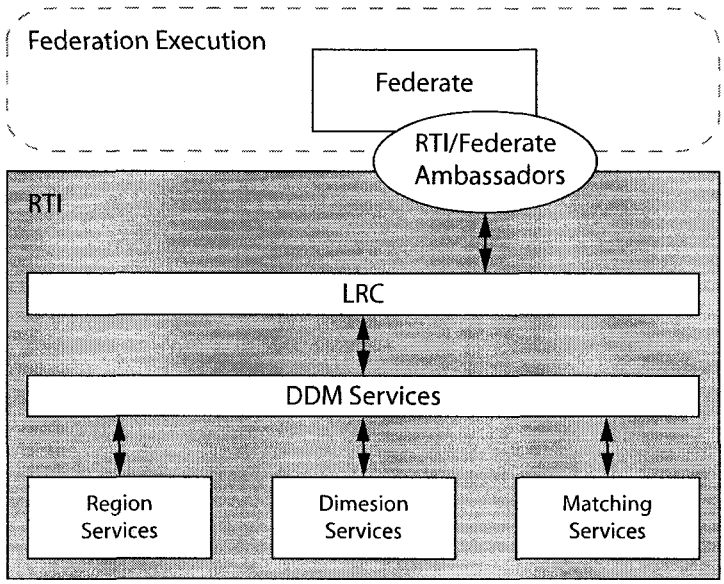


Figure 7.2: The DDM Services

by the Region Services and Dimension Services, and is the sole interface between the rest of the RTI and the DDM service functionality.

Each of the functional groups from figure 7.2 is implemented as a separate set of classes which are described below. Each group interacts with the `DDMServices` class through a "manager" class: the `RegionManager`, `DimensionManager`, and `MatchManager`. The Managers handle different aspects of the DDM functionality. The `DDMServices` object is the main entry point for all of the DDM service related functionality.

7.2.1 The Region Services

The Region Services are responsible for the bulk of the work in the DDM services module, because the largest amount of functionality comes from the use of Regions. The main Java class that manages the Region Services functionality is the `Region-`

Manager. Regions must be created, associated with Attributes, and modified when range bounds change. Figure 7.2 shows the Region Services has three types of regions associated with it: Local Region Specifications, Local Region Realizations, and Remote Region Realizations. The Local Region Specifications and Realizations are regions created and used by local federates using the RTI on the same node. Remote Region Realizations are regions used for publishing on remote RTI nodes. Local copies are made of publishing regions to allow the matching algorithm to run locally. The RegionManager keeps track of the region objects and associations with three lists: the localRegionSpecifications list, which stores all the locally created Region Specifications, the remoteRegionRealizations list, which stores all the remotely created publishing regions that are relevant to federates on this node, and the attributeRegionsCollection list, maintains the set of attribute to region associations. Locally created Region Realizations are stored with their parent Region Specifications, so a separate list is unnecessary.

As shown in figure 7.3, the main objects that the Region Manager creates and uses are the RegionSpecification, RegionRealization, and the AttributeRegionCollection (ARC) classes. The RegionSpecification is key in the RTI because that is the object created for the federate when the call is made to create a region. The federate is returned a RegionHandle to a RegionSpecification object within the SIP-RTI. When associating a region for use, a RegionRealization is created. The RegionRealization is the instance of the Specification within and Object Class Attribute or Attribute Instance. Federates can only access the RegionSpecification objects they have created using the RegionHandle.

The RegionSpecification object keeps track of the Federate that owns it, the dimensions (which are handled by the Dimension Manager, and discussed later) associated with the region, and the ranges on each of the available dimensions. The

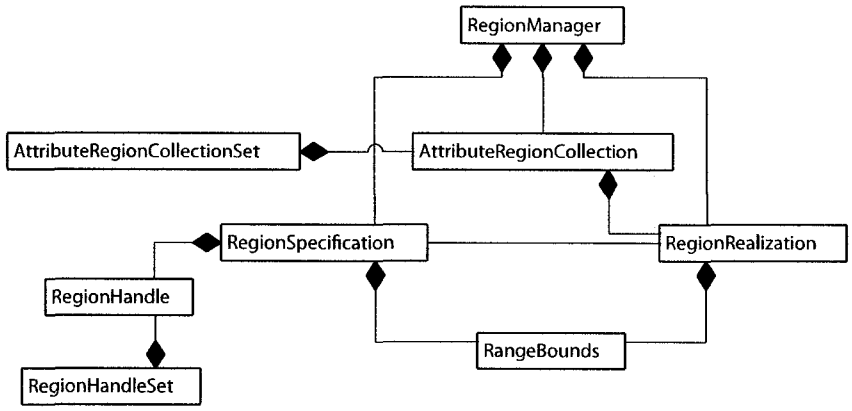


Figure 7.3: The Region Services Classes

`RegionSpecification` also contains a set of all of the Realizations derived from it. Methods in the `RegionSpecification` are provided for convenience as well, such as preparing a string containing the dimensions and ranges, which is required when sending publishing region information to remote subscribing LRCs. Other methods, such as those for modifying the range bounds, are provided to support required functionality of the RTI specification. The commit region method (see Appendix A for implemented RTI method details) will percolate any changes made to the ranges in the `RegionSpecification` down to all of its derived Realizations.

The `RegionRealization` object serves a dual purpose in the Region Services. It is used to realize instances of local `RegionSpecification` objects, and also to create local copies of remote publishing regions. The remote realization requires more information than a locally created realization. The conference ID, LRC node name, and floor ID the remote Speaker Attendee owns are required to correctly tag overlapping Listener Attendees. The `RegionRealization` object contains a set of parameters to handle both cases, where the extra remote parameters remain null when the Realization is used for a local region.

The local `RegionRealization` keeps a copy of the dimensions and ranges of the

RegionSpecification. The ranges are only changed in the realization once the region commit method has been called by the federate, so there is a requirement that the Realizations have their own set of ranges. RegionRealization objects also contain references to their parent Specifications, and a reference to the ARC they have been placed in. The AttributeRegionCollection object reference allows the realization to flag the ARC that it has been modified when the commit region method has been called by the federate.

The ARC object is used to represent the Object Class Attributes that have dimensions associated with them in the FOM. The ARC objects are stored in a Java hash map in the RegionManager, with the Attribute name as the key. This object contains the RegionRealizations that have been associated to that Attribute, for both subscription and publication. The ARC contains four sets of RegionRealization objects, two for publishing regions and two for subscription regions. The publishingRegions and modifiedPubList sets keep track of publishing regions, and the subscribingRegions and modifiedSubList sets keep track of the subscription regions.

The lists are used to store Realization objects, but also to identify to the ARC that they have been recently added or modified. When a commit region (RTI) method is called, the Realization calls the modifiedRegion method, to place itself into the modified list for its type. The modified lists are used to indicate to the matching algorithm which regions have been modified recently, which determines which regions to use in the overlap calculation. Methods are included in this object to add or remove regions, flag modified regions, and get sets of regions. The implementation of this class is not complex but it is an important component in the functionality of the DDM services.

The remaining objects in figure 7.3 support the functionality of the Region Services

objects addressed. The `AttributeRegionCollectionSet` object is used to return several ARCs to the `DDMServices` object for queueing in the Matching Services. The `RangeBounds`, `RegionHandle` and `RegionHandleSet` objects were defined by the IEEE 1516 Specification [11].

7.2.2 The Dimension Services

The Dimension Services classes are shown in figure 7.4 and handle accessing the dimensions that were loaded from the FOM. When the federate requests a dimension handle, the `DimensionManager` responds by providing the handle. The Dimension Services check whether or not dimensions exist when calls are made to the RTI. The dimensions themselves are stored as individual `Dimension` objects in a list within the `DimensionManager`. The dimension properties associated with each dimension in the FOM are parameters in the `Dimension` class.

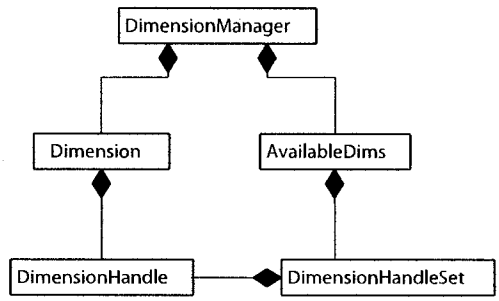


Figure 7.4: The Dimension Services Classes

Dimensions are associated with Object Class Attributes in the FOM, and that relationship is maintained within the Dimension Services as well. Figure 7.4 shows the `AvailableDims` object that maintains a `DimensionHandleSet` containing the dimensions associated with an Object Class Attribute. The `DimensionHandleSet` is used in the `AvailableDims` object to accommodate the RTI API, which has a method that

returns a `DimensionHandleSet` for a given Object Class Attribute. The `AvailableDims` objects are stored in a Java hash map, with the Attribute name as the key.

7.2.3 The Matching Services

The Matching Services handle the implementation of the matching algorithm and supporting classes. A simple class diagram of the Matching Services classes is shown in figure 7.5. The main class is the `MatchManager`, which provides the API for the Matching Services. The main components within the Matching Service, aside from the matching algorithm itself, are the `MatchQueue` and `MatchingThread` classes. The `MatchQueue` queues `AttributeRegionCollection` objects that have modified regions associated with them. The `MatchingThread` is locked on the queue until a new set of regions are made available for the thread to work on. The thread will use the matching algorithm to determine which regions in the set are overlapping each other. The minimum set of regions to calculate is determined by which region type was modified in the ARC. The control region type and the affects on the matching were discussed in Chapter 5. The `MatchManager` provides an entry point for the matching thread to make the callback to the LRC when the results have been found.

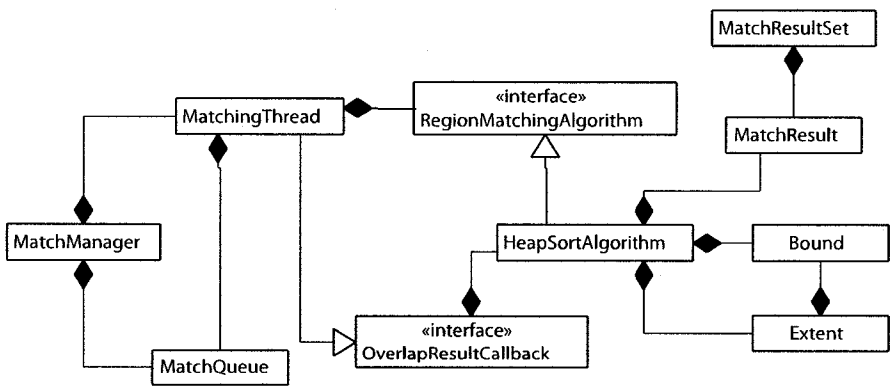


Figure 7.5: The Matching Services Classes

The MatchManager queues ARCs for overlap calculation within the MatchQueue, and returns overlap results to the LRC where the conference objects are modified. The MatchManager class contains two objects, a thread which blocks on a queue, and a blocking queue which contains collections of regions. A blocking queue is used so that new ARCs can be added while the thread is processing a set of regions. The matching thread allows the region overlap calculation to be performed without blocking the execution of the application.

The two interfaces are used to facilitate the ease of switching matching algorithms in the DDM Services. The RegionMatchingAlgorithm interface provides a set of methods to allow the matching thread to load region data and run the matching algorithm without knowing about the specific matching algorithm implementation. Methods listed in the interface provide an API for the matching algorithm to load region data from an ARC, calculate the overlap, and give the matching algorithm a reference to the matching thread. The OverlapResultCallback interface is implemented by the MatchingThread class, provides a method for the matching algorithm to make a callback to the rest of the RTI. This interface allows the matching algorithm to make a callback to the DDM Services once the matching algorithm has been run and the results are available. The results are returned in a MatchResultSet of MatchResult objects. This allows the results to be used by the RTI without exposing algorithm-specific objects outside of the algorithm's implementation.

The MatchResult object keeps track of the data needed to identify listener attendees, and creates the appropriate tag for the publishing region. Each instance of the object relates to a publishing region, and contains the caller IDs of both the listeners that are currently in and out of overlap with that publishing region.

The HeapSortMatchingAlgorithm, Bound, and Extent objects are explained above in the discussion of the Region Matching Algorithm in Chapter 5.

Chapter 8

Data Distribution

The SIP-RTI facilitates communication among distributed federates. Recall from Chapter 2 that the SIP-RTI is comprised of two layers, the RTI layer and the Conference Infrastructure (CI) layer. The CI provides a framework for the RTI to operate on, and handles the state of the publisher and subscriber relationships for the RTI. Each layer is seen as a black box to the layer above it. The RTI layer consists of Local RTI Components (LRCs) communicating with the CI. The CI in turn is comprised of Local Conference Components (LCCs) which are connected to other LCC nodes. Both the RTI and CI maintain some state for the federation execution using the SIP-RTI, and each requires sharing some of that state information.

The sharing of state information within the RTI and CI layers are discussed in this chapter. Figure 8.1 shows the layers and an example of the data shared at each layer. Using the familiar example of a Plane federate flying around a simulated space and a Radio Tower sending out a message, the two federates and their publication and subscription intentions are shown running on separate nodes in figure 8.1.

The RTI layer maintains regions that have been created by locally hosted federates. This example shows only the Region Realizations, and the Region Specifications have been left out intentionally. When region information is shared, the Region Re-

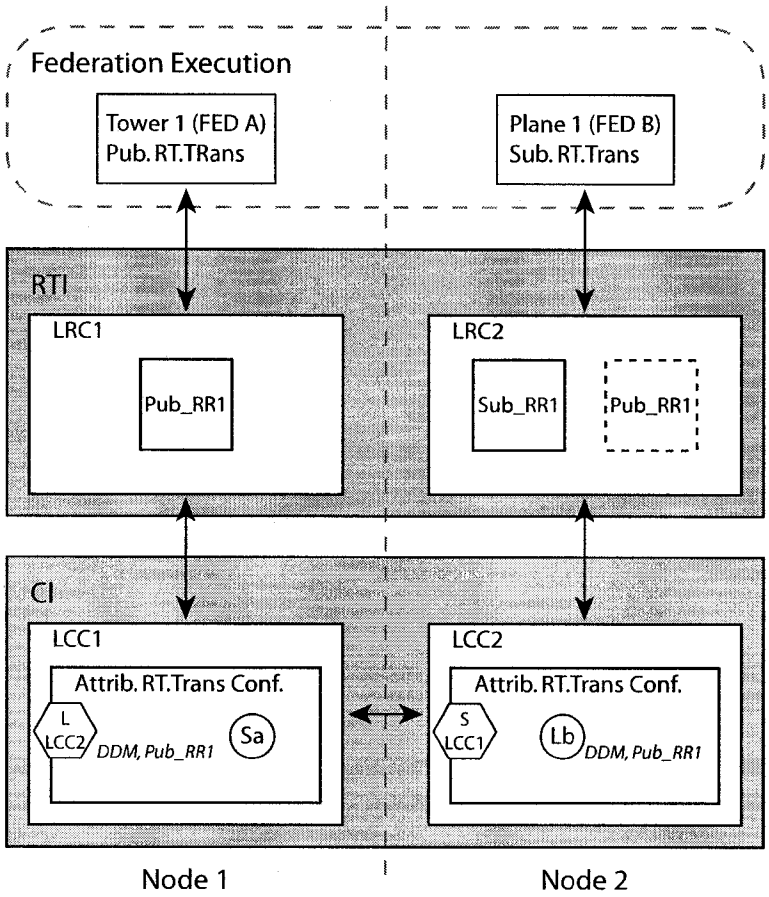


Figure 8.1: SIP-RTI Layers and Shared Data

alization is used, and recall from Chapter 7, that the Region Manager keeps track of Remote Region Realizations only. Subscribing regions are not shared with other LRC nodes, but Publishing regions are. The publishing region "PubRR1" in figure 8.1 is also created in LRC2, whereas the subscription region "Sub_RR1" is not shared. The Remote region in LRC2 appears as a box with dashed lines, while local Region Realizations (RR) have solid lines. This is done to provide an immediate visual aide for shared regions, and is used in the examples throughout this chapter.

In the CI layer, local Attendee state is shared through Remote Indicators, as mentioned in Chapter 2. For clarity, figure 8.1 is only showing the RT.Trans attribute conference, though there may be other conferences within the hierarchy. The LCC on Node 1 in figure 8.1 has a Speaker Attendee in the RT.Trans Attribute conference, and also the Listener Indicator representing all the Listener Attendees from LCC2. The Remote Indicators are shown in a hexagon shape hugging the side of the Conference box, and local Attendees as a circle. This is done to visually separate the two conference object types, and is used in all the examples in this chapter. Attendee tags, as seen associated with the Listener Attendee in LCC2 in figure 8.1, are also applied to the Remote Indicator in LCC1. Tags are shown in italics next to the object they are associated with. Tags indicating overlap are identified by the RR name of the publishing regions. Floors and floor ownership have been omitted form the diagrams for clarity and tags indicating overlap use the region name of the publishing region. Similarly, the LCC on Node 2 has an indicator showing there is at least one Speaker Attendee at LCC1.

The design objective for sharing data within both layers was to share only what is necessary. Within the RTI layer, only publishing information is required by the nodes with subscribing federates, so the matching algorithm can be run and the overlaps calculated. In the CI, Indicators inform remote LCC nodes about Listener Attendees

within a conference, without requiring a local copy of each Listener. Tagged Indicators allow the conference to be aware of all of the tags used at an LCC node, providing a broad view of Listener interest from that remote conference.

While the example in figure 8.1 showed both regions in the RTI layer and Attendees and Indicators in the CI layer, the discussions of each layer is presented separately in this Chapter. Region sharing is presented first by discussing why publishing regions are shared. Region information is shared through new messages sent to LRCs as conference announcements. These messages contain all of the information needed to create a Region Realization object at the remote LRC, or to remove a region that is no longer being used. Tag sharing within the CI layer will be introduced and discussed. Tags are added or removed from Listener Indicators by two new LCC command messages. Illustrative examples are provided to show the messages change the state of the nodes.

8.1 Sharing Regions

Region sharing within the RTI layer of the SIP-RTI is incorporated into the design of the DDM services to always allow the matching algorithm to run on nodes that have subscription regions. The requirement for running the matching algorithm on subscriber nodes stems from the design of the CI layer below the RTI, and its use in maintaining DDM overlap state. The RTI uses the CI to create Speaker and Listener Attendees to publish and subscribe to Object Class Attributes and Attribute Instances for federates using the RTI on the same node, and tags the Attendees to indicate which service group they are using and when a subscription region is overlapping a publishing region as described in Chapter 7.

Publishing regions are shared among LRCs that have subscription regions associated to the same Attribute class as the publishing region. Regions are not shared

amongst nodes that are not using the same Object Class Attribute, because the region information would not be useful to any federates at that node. The subscription regions are not shared because of the implementation of tagging. As mentioned in Chapter 6, tags are applied to Listener Attendees, so they are only used to identify subscription regions that are in overlap with publishing regions. Sharing publishing regions only with LRCs that have subscribing federates using the same Attribute class allows all of the necessary region information to be available where it is needed with minimal overhead.

LRCs that have local copies of publishing regions can tag the Listeners that they are responsible for with greater ease than if subscription regions were shared with publishers. Consider allowing the publishing LRC to determine whether subscription regions are in overlap. This would result in more communication overhead among LRCs. Figure 8.2 shows a set of LRCs where the subscription region from Plane 1 is shared with the publishing LRCs. This example is extended with a third node, LRC3. A second Tower federate (FED C) has joined the federation, and has also created a publishing region, "PubRRc". When the plane moves and updates its subscription region, the updated region information has to be sent to both publishing LRC nodes. After the matching algorithm is run at LRC1 and LRC3, the results are returned to LRC2, where the tag for "Pub_RRa" is added to the Listener Attendee.

In the case where the publishing regions are being updated and the subscribing region remains unchanging, there is still more LRC communication when sharing subscribing regions. Tags can only be added to local Listener Attendees, so after the matching algorithm is run on the publishing node, the results need to be sent to all subscribing nodes, where local Listener Attendees are tagged or untagged.

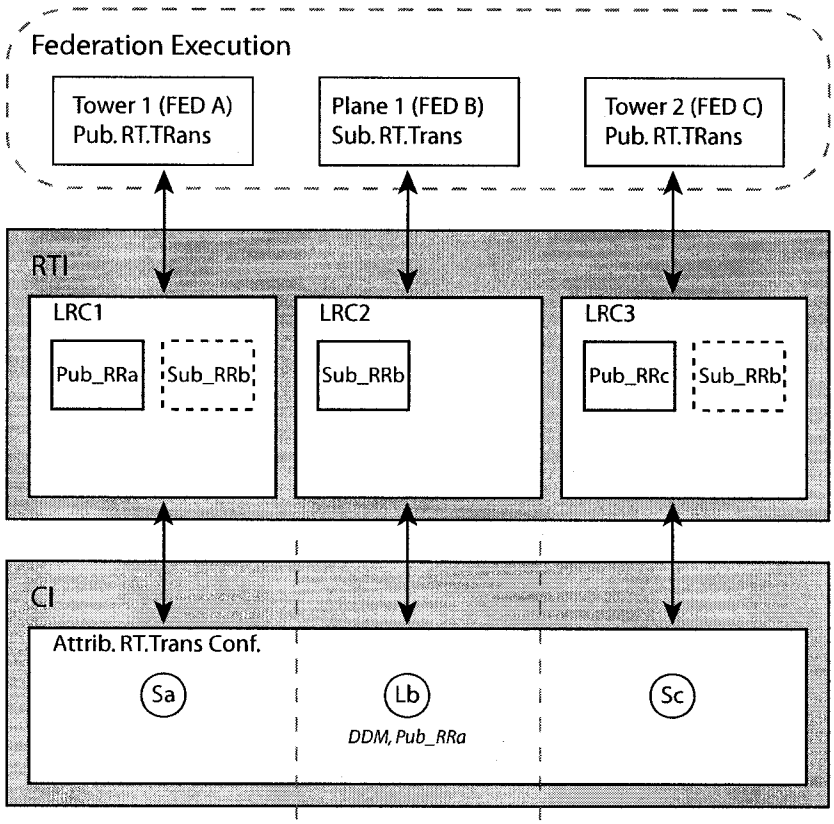


Figure 8.2: LRCs Sharing Subscription Region Information

8.1.1 Creating and Updating Remote Regions

When a federate associates a region for publication, the region information must be sent to the remote nodes that have Listeners within the Attribute conference. This allows the subscribing nodes to determine if they are in overlap, so they can apply tags to their Listener Attendees as necessary. The region needs to be transferred to the subscribing LRC so the subscribing node's DDM services to have access to the publishing region. This is done by making a createRegion announcement in the attribute conference that the region is associated with. This region information is parsed at the receiving LRC and a new local Region Realization object is created and added to the appropriate ARC.

The elements that make up the createRegion message are shown in figure 8.3. The message is passed as a text string within a conference announcement. The message is received and interpreted by the LRC through listeners placed within the conference. The first two parameters after the command name identify the LRC and LCC node where this message is coming from. The third parameter is the full name of Object Class Attribute whose conference this message is being sent in. The full name is used by the LRC to find the local Object Class Java objects for this attribute.

createRegion *LRC Node Name* *LCC Node Name* *Attribute Name* :
Local Region Id : *Dim1* ; *Dim2* ; ...

(a) createRegion Parameters

Dim#: *Dim ID, Lower Bound, Upper Bound*

(b) Dimension Parameters

Figure 8.3: createRegion Message Parameters

The portion of the message in figure 8.3 that starts with the Local Region ID is

the publishing region information. The local region ID of the region is provided so the Remote Realization object that is created in the subscribing LRC has a unique ID for this region. The ID is also required to help the LRC find this region when an updateRegion or removeRegion message is received. The dimension information is shown in 8.3(a) as "Dim1", etc.. The dimension parameters are shown in figure 8.3(b), and composed of the the dimension ID, the range lower bound, and the range upper bound. Each dimension is separated by a semi-colon. The ellipses show that there is an unspecified number of dimensions since not every region will use the same number of dimensions. The tag associated with overlapping subscribers is not needed in the createRegion message because it can be generated following the convention stated in Chapter 7.

To illustrate the use of the createRegion message and example federation is used that has two Radio Tower federates ("Tower 1" and "Tower 2"), with a Plane ("Plane 1") flying in a loop between the two of them. Figure 8.4 shows the initial state of three LRC nodes. The state in figure 8.4 shows that Tower 1 has created an instance of the RT.Trans Attribute and is publishing using the region "Pub_RRa". The plane federate is subscribing to the RT.Trans Attribute class with the region "Sub_RRb". The subscription region overlaps with the publishing region from Tower 1, and the Plane's Listener Attendee in the CI is tagged with "Pub_RRa". Tower 2 will eventually broadcast a message, but hasn't yet registered an Instance of the Object Class Attribute.

When Tower 2 registers a new Object Instance with regions, a Region Realization is created within LRC3. Since this is a publishing region, LRC3 will send a createRegion announcement into the RT.Trans conference. Figure 8.5 shows the updated LRC state. Recall from Chapter 7 that the announcement is made using the "DDM" tag, so the listener added by LRC2 receives the announcement, and uses the data provided

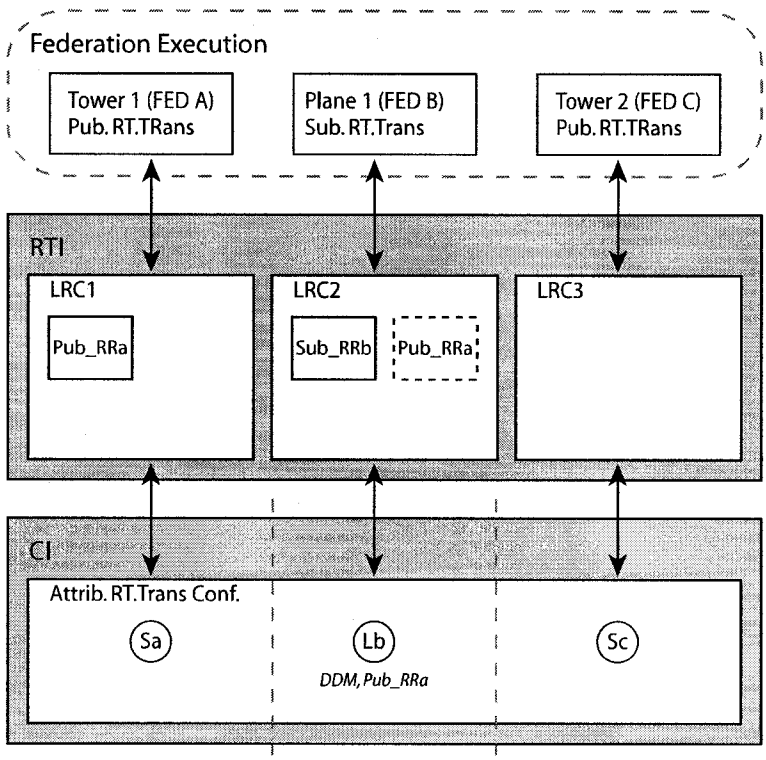


Figure 8.4: Region Sharing Example Initial State

to create the local copy of the publishing region. The matching algorithm is run on LRC2, however Sub_RRb does not overlap with Pub_RRc, so no additional tags are added to the Listener Attendee.

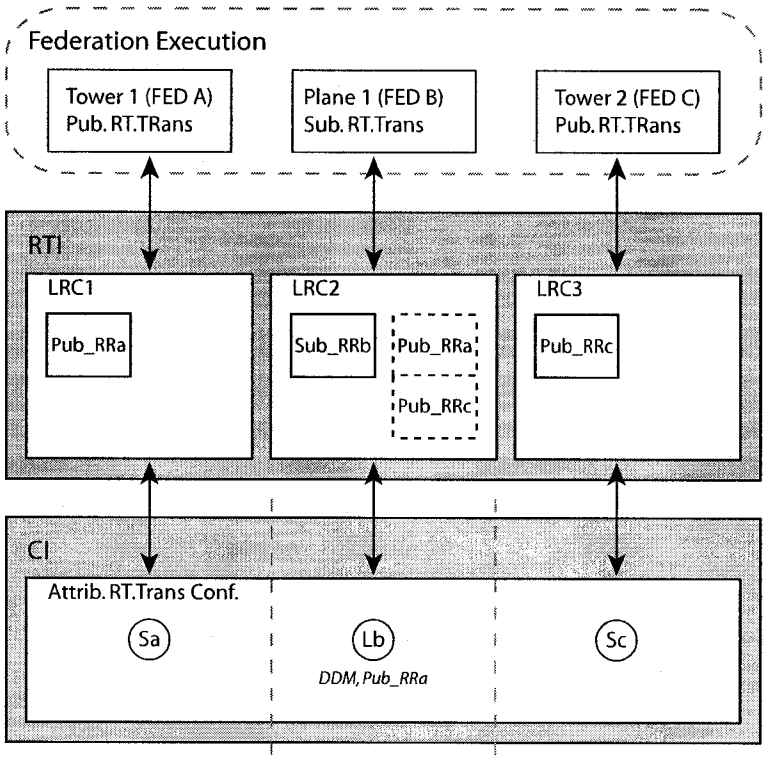


Figure 8.5: Region Sharing Example - Creating A Remote Region

A federate updates a publishing region by modifying dimension ranges and calling the commit region (RTI) method. The publishing regions then send an updateRegion announcement to Listeners in the associated Attribute conference to provide them the new set of dimension ranges. This allows the subscribers to update their local copies of the publishing Region Realization before running the matching algorithm. The format and parameters of the updateRegion message are nearly identical to the createRegion message, except the command identifying the message as updateRegion. The update message contains the full set of dimensions and ranges for the locally

updated publishing region, regardless of how many dimensions ranges were changed. The receiving LRC simply replaces them all, and then runs the matching algorithm with that updated region against the local subscribing regions to determine the new overlap status.

Suppose Tower 1 modified the size of its publication region to simulate reducing the signal power, thereby shrinking the range its messages can be received in. Figure 8.6 shows the effects of the associated updateRegion announcement. The announcement is sent to the attribute conference with the "DDM" tag, and the listener placed by LRC2 receives it. The local copy of "Pub_RRa" at LRC2 is updated, and the matching algorithm is run. In this scenario, the matching algorithm determines that the subscription region no longer overlaps "Pub_RRa", so the tag is removed from its Listener Attendee.

8.1.2 Removing Remote Regions

When a federate disassociates a region from an Object Class Attribute or Attribute Instance, the Region Realization is removed from the LRC. The removeRegion message is used by publishing regions to inform subscribing LRCs that the publishing region is no longer being used. The removeRegion announcement is sent in the associated Attribute conference using the "DDM" tag. When received by a subscribing LRC, the Remote Region Realization will be deleted, and all relevant Listener Attendees will have the region's tag removed.

The parameters of the removeRegion message are shown in figure 8.7. The remove region command will contain the region ID of the region to delete. The second part is the conference ID and tag to remove. These are associated with the region that will be removed. The tag provided is the tag of the publishing region that will be deleted. The conference ID provided is the ID of the Attribute conference where this

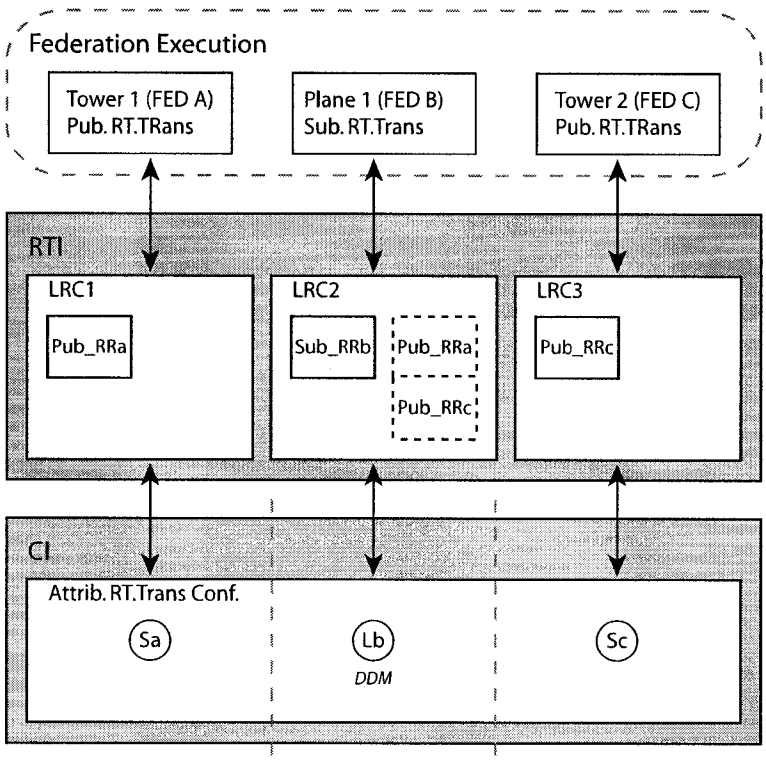


Figure 8.6: Region Sharing Example - Updating A Remote Region

region was associated. The tag is provided for convenience to untag all the Listener Attendees. The LRC will make a call to the conference to remove the association of that tag from all local listener attendees.

removeRegion *LRC Node Name* *LCC Node Name*
Local Region Id : Conference Id : Tag

Figure 8.7: removeRegion Message Paremeters

8.2 Sharing Tags

The implementation of tagging Listener Attendees and the reasons for doing so were discussed in Chapter 6. Tags are used by the SIP-RTI to differentiate Listeners within an Attribute conference by which service group they are subscribed with, and any regions their owners overlap. Tags to identify that the DM or DDM services are used when a Listener Attendee is created by "DM" and "DDM" tags, respectively. When a federate's subscription region overlaps with a publishing region, a tag is added to the Listener Attendee as well. Two LCC command messages have been added to the LCC to support adding and removing tags from Remote Indicators: TagAdd and TagRem.

The background discussion on the SIP-RTI in Chapter 2 introduced how Attendee state is shared among connected LCC nodes. Remote Indicators are used to signify that a remote LCC has Attendees within a conference. Listener Indicators provide a conference a means of identify which connected LCCs have Listener Attendees in their local conference. Indicators allow other LCC nodes to be aware of local Attendees without sharing each Attendee with all the other nodes.

Tag information is shared following the same principle as Indicators. When a tag

is associated for the first time to a Listener Attendee within a conference, the LCC sends a command message to add the tag to its Listener Indicators on remote LCC nodes. Similar to how the Listener Indicator informs remote conferences regarding Listeners at that node, tags associated to a Listener Indicator identify all the tags being used by listeners at the indicated LCC node.

Changes in tag state result from changes in region overlap in the LRC. When the matching algorithm is run and overlap changes, tags will be modified in the LCC. The example in figure 8.8 shows the state of three LCC nodes sharing tag information. There are five federates in this example: two Radio Towers ("Tower 1" and "Tower 2"), and three Planes ("Plane 1" to "Plane 3"). The Towers are each located on their own RTI node, publishing the RT.Trans Attribute using regions. A Speaker Attendee is placed in the local LCC on each of those nodes (LCC1 and LCC3). A Speaker Indicator is placed in the other two conferences in figure 8.8, so all three are aware of the Speaker Attendees in LCC1 and LCC3. The middle node with LRC2 has three Plane federates. Two of the Planes, Plane 1 and Plane 3, are subscribing to RT.Trans using regions, and the Listener Attendees for these federates, "Lb" and "Le" respectively, are tagged with "DDM". Plane 1's subscription region ("Sub_RRb") overlaps with Tower 1's publishing region, and so its Listener Attendee is tagged with "Pub_RRa". Plane 3 does not overlap with either publishing region. Plane 2 is subscribed to RT.Trans using the DM services, so its Listener Attendee, "Ld", in LCC2 is tagged with "DM".

The Listener Indicator placed in the attribute conference in LCC1 and LCC3 in figure 8.8 is tagged with all three tags used by Listener Attendees within LCC2 - DM, DDM and Pub_RRa. The indicator is tagged with all three tags because it represents all of the Listeners at that LCC.

Suppose, the subscription region for Plane 3, "Sub_RRe" is updated and now

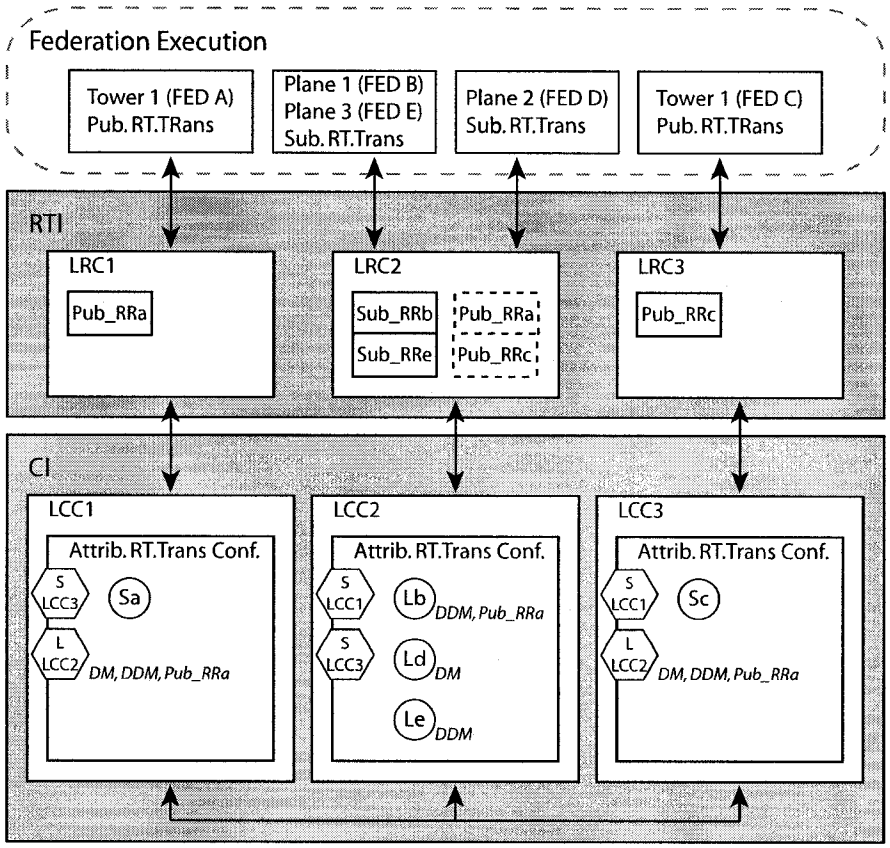


Figure 8.8: Tag Sharing Example Initial State

overlaps with the publication region for Tower 2, "Pub_RRc". Since this is the first time the tag "Pub_RRc" is used by any of the Listener Attendees in LCC2, the tag needs to be added to the Listener Indicators at the remote LCCs. The TagAdd LCC command message is sent to LCC1 and LCC3 to add the tag to the Listener Indicator for LCC2. Note that since the Listener Indicator for LCC2 is already present at both LCC1 and LCC3, it is not necessary to add another Listener Indicator and the Pub_RRc tag is added to the existing Listeners.

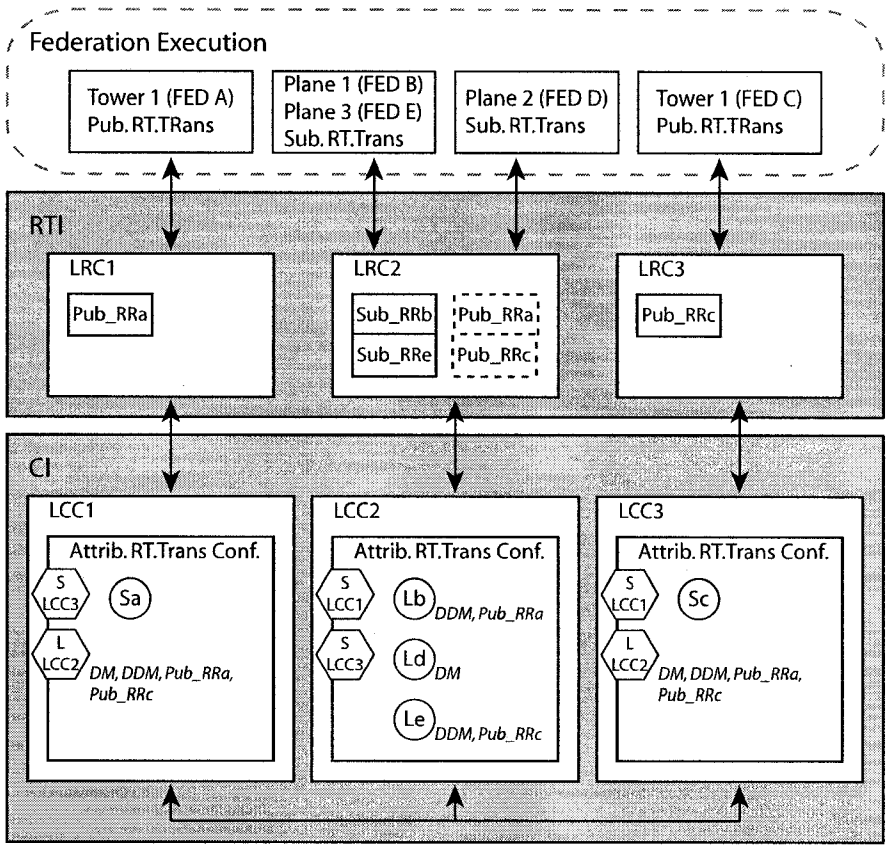


Figure 8.9: Tag Sharing Example - Adding a New Tag

Tags are removed from Listener Attendees due to regions in the LRC layer moving out of overlap. Consider the subscription region Sub_RRb being updated. Plane 1

moved, and updated its subscription region. The matching algorithm is run and determines that Sub_RRb is not overlapping Pub_RRa. The publishing region tag is removed from the Listener Attendee in LCC2 as shown in figure 8.10. Lb was the only Listener Attendee to have this tag, and since it is no longer being used by any Listeners at LCC2, remote Listener Indicators must be updated. The TagRem command is sent with the tag "Pub_RRa" to both LCC1 and LCC3, and the tag is removed from the LCC2 Listener Indicators on those nodes.

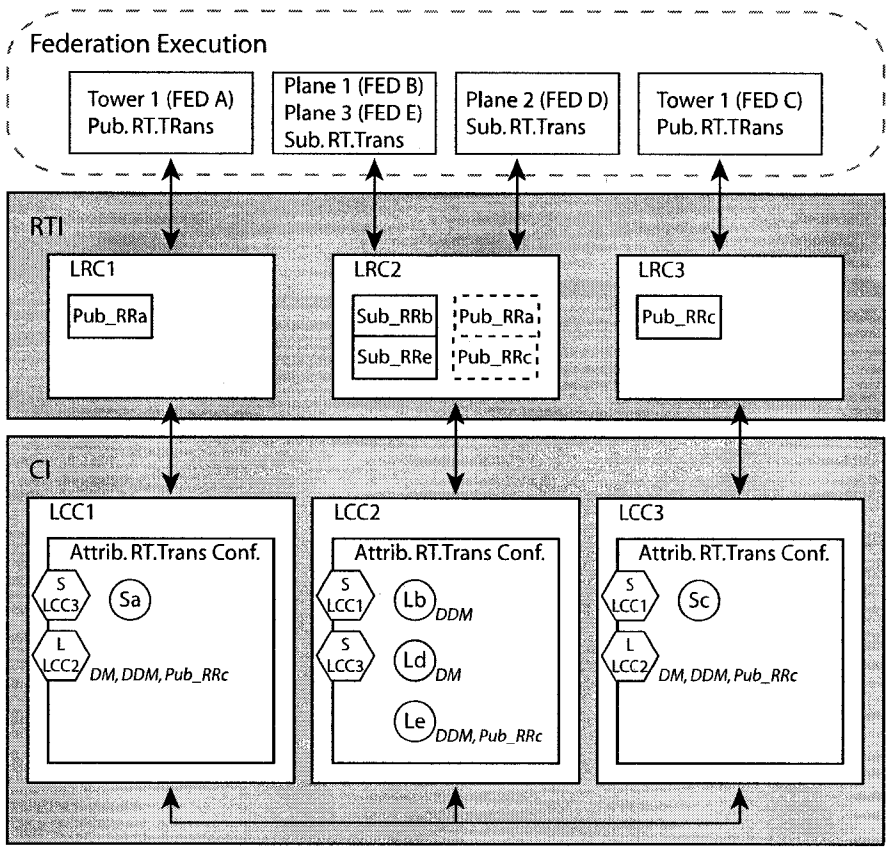


Figure 8.10: Tag Sharing Example - Removing a Tag

The distribution of data within the LRC and the LCC is of chief importance in maintaining the state of the whole system, even though the actual knowledge of

attendees and regions is kept on local nodes for the most part. The information that is shared has been selected because it can be used to give an accurate representation of the remote nodes for local use, without having to send an overabundance of messages to transfer all local state changes to all nodes.

8.3 Tag Notification

The implementation of the DDM services provided a local copy of a remote LRCs publishing region to LRCs hosting subscribing federates. Using a local copy of the publishing region, the LRC can determine if local subscribing federates are in overlap with the publishing region without requiring communication between LRCs. Furthermore, since the overlap state is stored on the Remote Indicator tags, as far as the publisher-side is concerned, the local LRC would need to receive the response and tag the Attendee in order for the Remote Indicator to be properly tagged.

It was discovered late in the development process that an issue would arise when a subscriber was created after a publishing region. Since the publishing region would send the message to create a remote region when it was created, the late subscriber does not receive notification, and would have to assume that there were no publishing regions associated with that Attribute. Since it has the DDM tag, which the publisher will use to send updateRegion messages, it can receive an update, but it can only act on it if there is a region already created locally, and then, it will only be sent if the publisher updates its region.

A solution for this issue sees the publisher resending region information when new subscribers are created. Further modifications made to the CI to provide a new call-back method to the application when a new Listener Indicator is placed within a conference where the local application has a speaker attendee. The RTI, receiving this notification, can determine if any of the speakers within that conference are using

region. If so, a new createRegion announcement can be made to that conference. The listener at the newly added indicator's node will receive the region details, and create the Remote Region Realization. Nodes that already have this region will compare the region ID's with the Remote Region Realizations they have already. If it is the same, then the redundant createRegion message can be discarded.

Chapter 9

Demonstration

Two of the major contributions made by this research involve the design and implementation of the Dynamic Region Reduction (DRR) technique into the Sort-Based matching algorithm, and the design and implementation of the DDM services surrounding the matching algorithm. Both of these contributions were tested to prove the validity of their claims, and to ensure that they were working as intended. This chapter discusses the tests performed on both contributions: the DRR technique and the DDM services.

The DRR technique is tested in a stand-alone environment, where an artificial set of regions is created and used. The DRR-enchanced algorithm is tested against the original Sort-Based matching algorithm, which was used as the basis for the matching algorithm in this research and introduced in Chapter 3. Comparative tests show the reduction in regions used while the matching algorithm iterates, and the savings in processing required to determine region overlap.

The DDM services as a functional whole are tested using a small federation with several federates publishing and subscribing using DDM. The federation consists of three federates, two publishing with DDM, and a third subscribing using DDM. The subscriber is moving around the simulated environment, updating its subscription

region and moving into and out of overlap with the publishing regions.

9.1 Dynamic Region Reduction Demonstration

Chapter 5 described how the Dynamic Region Reduction was implemented into the Sort-Based matching algorithm. The goal of the DRR technique is to reduce the number of regions processed before each iteration of the matching algorithm by eliminating regions that do not overlap from successive iterations. The HLA specification states that regions must overlap on all dimensions in common, so once a region does not overlap on a dimension, it cannot be considered to overlap at all.

A demonstration application was made to compare the performance, in numbers of regions processed between an implementation of the original Sort-Based matching algorithm as described in [18], and the Sort-Based matching algorithm with DRR. The test application creates a set of regions, and then runs the original Sort-Based matching algorithm with them, and then the DRR Sort-Based matching algorithm. The results in number of regions available for processing before each algorithm iteration are saved, as well as the actual number of regions found in overlap at the end.

These tests were performed separately, and not within the SIP-RTI to isolate the DRR technique. The Sort-Based matching algorithm was tested within an RTI implementation and found to perform between 30% and 99% better than the brute-force and hybrid grid-based matching algorithms [18, 19]. The objective with the tests conducted here is to observe the effects of the DRR technique in comparison to the Sort-Based matching algorithm without it.

The regions used for testing do not reflect any real-world simulation so removing them from the RTI allows the tests to be performed within a controlled environment. With previous matching algorithm research [14, 15, 18, 19], algorithm performance

was tested with large sets of several hundred regions or dimensions. Test scenarios such as these are designed solely to assess the processing speed of the algorithm under extreme loads. Each researcher utilizes their own method of testing the performance of their algorithm in relation to other algorithms.

Results on the number of regions available for processing are collected, and the average is taken at the end of the execution. Each set of regions is matched using both matching algorithms, and a given set of parameters is tested with one thousand sets of regions. Regions are randomly created, with ranges of a fixed size randomly distributed within each of the dimensions. None of the regions were hardcoded to overlap with the control region, so the test results show a large number of regions not overlapping. This scenario is ideal, since the DRR technique was designed to facilitate handling regions when they are not overlapping. If the all the regions overlapped, then the DRR technique would show no advantage, as no regions would be reduced.

A number of factors affect the performance of the matching algorithm, in both real-world use of DDM and in the test application used in this research. These factors are the number and size of dimensions, the number of regions used, and the size of the ranges. Each of these factors can affect the performance of the matching algorithm, and the advantages offered by the DRR technique.

The number of dimensions drives the number of iterations of the Sort-Based matching algorithm. The greater the number of dimensions, the more iterations are required to fully test all of the regions for overlap. With the DRR technique, the greater number of regions can increase the likelihood that a region is removed, since each dimension offers an opportunity for a test region's range to not overlap with the control region's range on that dimension. The requirements for a region to overlap narrow because an overlapping region must be within range of the control on all dimensions.

The number of regions also affects the performance, since it determines how much work is performed during each iteration. Each region's range has both an upper and lower bound, which are sorted and iterated through by the Sort-Based matching algorithm. Increasing the number of regions increases the time spent processing each dimension. The DRR technique seeks to counteract this by removing a number of regions after each iteration.

Range size also has an affect on the performance of the matching algorithm, and is related to the size of the dimensions. As the size of ranges increase, they increase their chances of overlapping with the control regions' ranges. If the ranges are large in proportion to the size of the dimension, the ranges will be more likely to overlap with each other, and the DRR technique will find fewer regions to reduce.

The test application creates a set of regions randomly, based on a number of parameters such as the size of the region ranges, the size of the dimension, and the number of dimensions the regions use. For these tests, one parameter at a time was adjusted, so the effect of the changing parameter could be observed without being influenced by other parameters. Two test cases were run; the first changing the size of the region ranges, and the second changing the number of regions used. The test cases and parameters adjusted are discussed along with their results in the next section.

9.1.1 DRR Test Results

The first test that was performed sought to examine how well the DRR technique would perform when the size of region ranges was increased. Increasing the size of the ranges increases the chances they will overlap, because there is less "open space" for them on the dimension. The test was run one thousand times, with a different set of regions each time. The results were collected and the average number of regions processed was calculated. Five dimensions were used, each being one hundred units

in length.

Each dimension was given the same size to give the regions the same random distribution of ranges on each dimension. In each test, fifty regions were used. The sizes of the ranges are adjusted in percentage of the dimension size, to give a more generalized description of the region size effects. The sizes of the ranges was increased from 10% of the dimension size, to 60%. Figure 9.1 shows the results of these tests, displaying the number of regions processed per iteration of the matching algorithm.

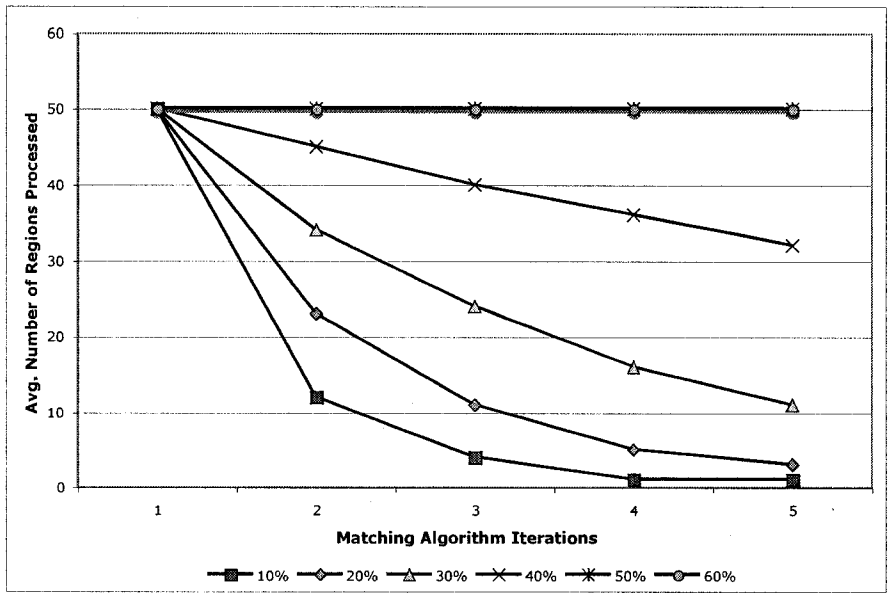


Figure 9.1: The Average Number of Regions Processed as the size of the Regions is increased

The results in figure 9.1 indicate that the size of the region ranges has an impact on the number of regions that overlap, and on how effective the DRR technique is. When the ranges are small (10% of the dimension size), the ranges are widely dispersed along the dimension. The number of ranges that overlap with the control region are relatively small in comparison to the number available. Looking at the second iteration on the 10% curve shows that the number of regions still overlapping

with the control after the first iteration is just over 10, compared to the initial 50 tested in the first iteration. In the original Sort-Based algorithm, all 50 regions would have been tested in the second iteration, even though almost 80% of them cannot overlap.

As the size of the ranges increased, the number of regions processed per iteration increases. Figure 9.1 shows increase in regions processed per iteration. Once the size of the ranges approaches 50% of the dimension size, the all of the regions' ranges overlap on all dimensions. The DRR technique has no effect on the matching algorithm in these cases, since every test region overlaps the control.

The second test sought to examine the effectiveness of the DRR technique under an increasing number of regions. Increasing the number of regions slows down the matching algorithm because more ranges need to be examined for every dimension processed. The DRR technique helps in these circumstances because it will actively reduce the number of regions being processed per matching algorithm iteration, counter-acting the larger set it began with. The original Sort-Based matching algorithm examines every region on each dimension.

This test was performed one thousand times with five dimensions one hundred units in length. The size of region ranges is fixed along all dimensions at 25% of the dimension size. This range size was selected so there would be less chance of regions overlapping simply because there is no place for them to be. The number of regions used was increased from 50 to 350, with 50 regions being added each test. Figure 9.2 shows the average number of regions processed for each iteration of the matching algorithm.

The graph in figure 9.2 shows the decrease in number of regions processed per iteration of the matching algorithm. While increasing the number of regions increases the number of regions overlapping with the control, there is still a sharp decline in the

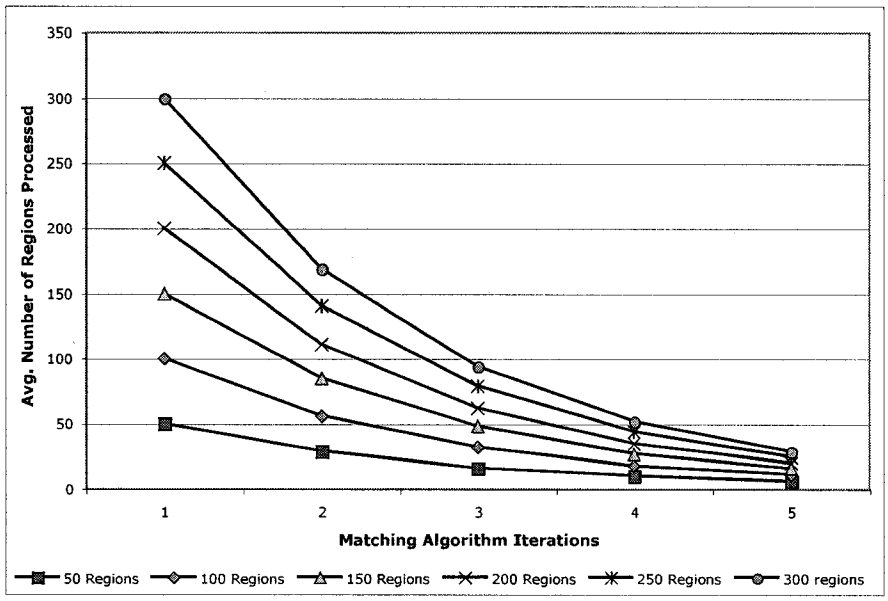


Figure 9.2: The Average Number of Regions Processed as the number of Regions is increased

number of regions processed after the first iteration of the algorithm. The number of regions processed does not decrease linearly, because each successive iteration is based on the results of the previous reduction. The total number of regions to process can change each iteration of the algorithm, and the regions that remain after each iteration are those with an increasing chance of overlapping with the control, providing diminishing returns on the number of non-overlapping regions. In the original Sort-Based matching algorithm, the lines would remain at the initial level for each iteraion. Figure 9.3 shows the number of regions reduced as a percentage of the total processed.

The results in figure 9.3 start with the second iteration of the algorithm, since there are no reductions before the first iteration. A common trend is immediately visible in the results. In the average case, regardless of the number of regions used, the DRR technique reduces the number of regions processed after the first iteration by about 40%. The DRR technique provides consistent results in the reduction of

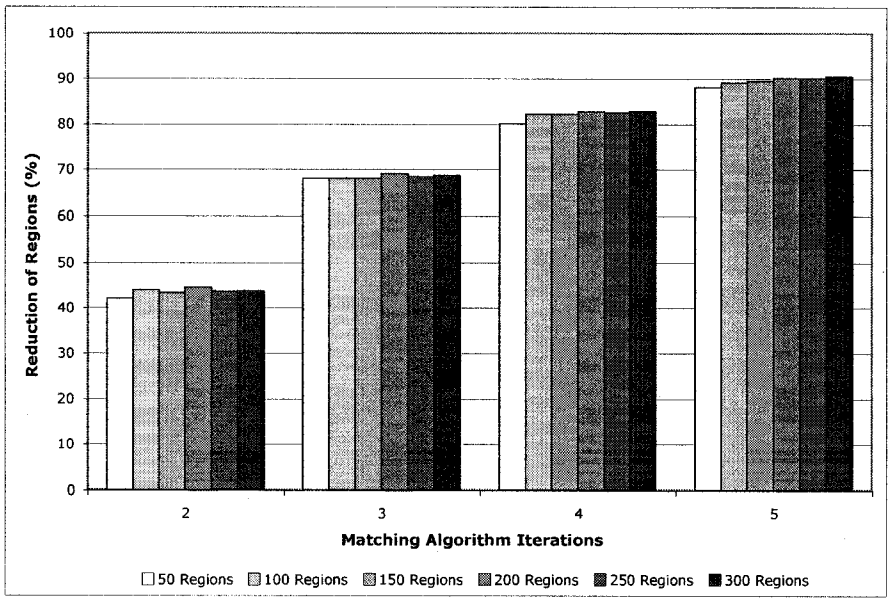


Figure 9.3: The Average Reduction of Regions Processed as the number of Regions is increased

regions based on the size of the region ranges. With an increasing load of regions to process, the DRR technique offers a predictable reduction in the amount of work performed over the standard Sort-Based matching algorithm.

The tests show that the Dynamic Region Reduction technique can reduce the amount of work that needs to be performed by the Sort-Based matching algorithm. In the test cases where the regions were not densely populated, with a high chance of overlapping each other, the DRR technique offers a predictable level of work reduction. When the number of regions overlapping the control is high, the DRR technique is less visible in the results, however that is to be expected. The DRR technique is designed to reduce work when regions do not overlap, and it has no effect on the matching process when they do.

9.2 DDM Services Demonstration

A demonstrative federation was developed to test that the DDM services gave the expected results once implementation was completed. The federation uses the DDM services for publishing and subscribing to an Object Class Attribute. The federation consists of a Plane federate, and two Radio Tower federates. The Radio Towers are both broadcasting messages, and the Plane flies in a loop between the two towers. To take advantage of the DDM services, the radio range of the Tower and Planes will be simulated using DDM regions. An example of the scenario being simulated is shown in figure 9.4.

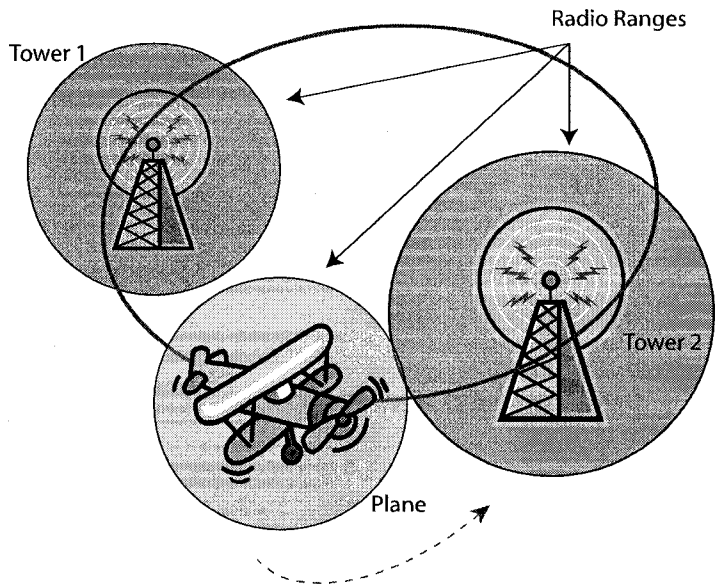


Figure 9.4: Demonstration Federation Scenario

The federates in the test federation use the DDM services to model the radio ranges shown in figure 9.4. The Radio Towers ("Tower 1" and "Tower 2") have a transmission range of approximately 600 meters, and the Plane's radio can transmit and receive signals in a 400 meter range. The Tower federates each create a region

for publication, and the Plane federate creates a region for subscription. The test federation's FOM defines two dimensions, RangeX and RangeY. The federates create a region using both dimensions to model the transmission range of the radios along the X and Y axes of the simulated environment.

The FOM defines two Object classes, the RadioTower and Controller classes. The RadioTower object class has one attribute, RTTransmitter. This attribute is an integer, and is used by the Tower federates to broadcast messages. The RTTransmitter attribute has the RangeX and RangeY dimensions associated with it, so regions can be used to publish and subscribe to this attribute. The Controller class has an EndSim attribute, which is used by the Plane federate to inform the Tower federates that the federation execution is complete and it will be shutting down.

There are numerous "moving parts" being tested by this federation. During the set-up, regions are created by all three federates. Since the two radio towers have publishing regions, their region information must be sent to the subscribing LRC. Every time the plane updates its position, it updates its region ranges to reflect the radio range where it can receive signals. This activates the matching algorithm, which adds or removes an overlap tag from the local Listener Attendee. Changing tags in the conference then reflects changes in the Remote Indicator of that LCC node. When the Indicator has been updated, the messages from the Radio Tower will be sent to the subscribing federate.

The publishing federates, the two Radio Towers, must make several DDM services API calls in order to successfully publish with regions. The full API methods can be seen in Appendix A. Figure 9.5 shows a high level pseudocode for the operation of the publishing federates. The steps highlighted in bold are those specific to the DDM services. After joining the federation in step one, the publishing federate must acquire dimension handles for the two dimensions it will be using, shown in step two. In the

third step the publisher creates a region, using the dimension handles obtained from the RTI in step two. Range Bounds are set for each of the dimensions in step four. When the publisher registers a new instance of the object class, the region handle for the region created in step two will be used to associate that region with the attribute instance. The publisher uses the standard DM method to update the attribute value for the federation. The LRC will ensure the region is used and that the update is filtered correctly.

```
(1)  Join Federation
(2)  Get Dimension Handles for RangeX and RangeY
(3)  Create Region
(4)  Set Range Bounds on Region
(5)  Create Instance of Object Class
(6)  Register Instance With Region
(7)  Loop
(8)  {
(9)    Update Attribute Value (locally)
(10)   Publish Object Attribute Update
(11) }
(12) End
```

Figure 9.5: Publishing Federate Pseudocode

Subscribing federates follow a similar process to the Publishers for using the DDM services. The pseudocode for the Plane federate is shown in figure 9.6. Steps one through four use the same methods as in the publishing federates. In step five, the subscriber uses the region created in step three to subscribe to the object class attribute. The RTI now handles the overlap calculation to determine whether or not updates should be received by this federate. The federate assumes that all updates the RTI notifies it about will fall within this region. While the Plane is flying along its path, position updates are performed to represent the plane moving within the simulated environment as shown in step eight. To remain synchronized with the publishing federate and to ensure that it correctly receives Attribute updates, the subscriber updates the range bounds of its subscription region, as shown in step nine.

To apply the changes to the Region Realization associated with the Object Class, the subscriber uses the commit region modification method in step ten.

```
(1)  Join Federation
(2)  Get Dimension Handles for RangeX and RangeY
(3)  Create Region
(4)  Set Range Bounds on Region
(5)  Subscribe to Object Class Attribute With Region
(6)  Loop
(7)  {
(8)    Update Position (locally)
(9)    Update Range Bounds
(10)   Commit Region Modification
(11) }
(12) End
```

Figure 9.6: Subscribing Federate Pseudocode

Using the DDM services is not a transparent process, as the steps indicated in the two federates show. Federates must be designed with the use of the DDM services in mind, since there is extra local information to manage, such as the current range bounds. When the federate updates its position, as the Plane did, additional steps are required to ensure the bounds on the subscription region are still valid. This is a tradeoff however versus using only the DM services, and handling unnecessary Attribute updates within the federate itself. In large federations, the amount of filtering within the federate may increase, as well the network will need to handle larger load to transfer these Attribute updates that may be discarded.

9.2.1 Results

The demonstration federate was run across three machines, two of which were Radio Tower federates, sending out simple messages at different intervals, and the third being the Plane federate, flying in a loop of predefined coordinates. The loop takes the Plane in and out of overlap with both Radio Towers. Figure 9.7 shows the initial

position of the Plane in the test execution. It starts out of overlap with either Radio Tower at the bottom of the loop, and will proceed around the loop in a counter-clockwise direction. The coordinates the Plane moves to are numbered in figure 9.7. The Plane federate is in overlap with Tower 1, on the right, when it is at positions 2 through 6. The Plane is in overlap with Tower 2 on the left when in positions 8 through 12. When at positions 1 and 7, the Plane does not overlap either Radio Tower.

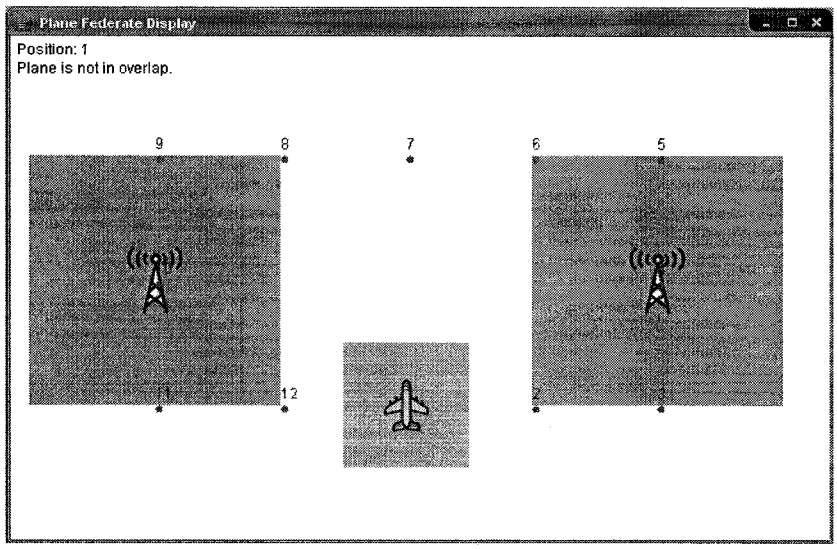


Figure 9.7: The initial position of the Plane, out of overlap with any Radio Towers.

Figure 9.8 shows the Plane at a later stage of its execution. In the plane's position in figure 9.8, it is partially overlapping the publishing region for the right Radio Tower (Tower 2). The display shows the overlap status of the plane, which is updated when the LRC uses the `AttributeInScope` callback method. The current message received from the remote node is updated as well. The Plane receiving the message when it is in this position indicates that the matching algorithm correctly determined that there was an overlap, and the LRC successfully added a tag to its Listener Attendee in the local conference. Furthermore, messages received at the local LCC node shows

that the remote Listener Indicator was successfully tagged as well, since the message is being sent to it from the publisher's LCC.

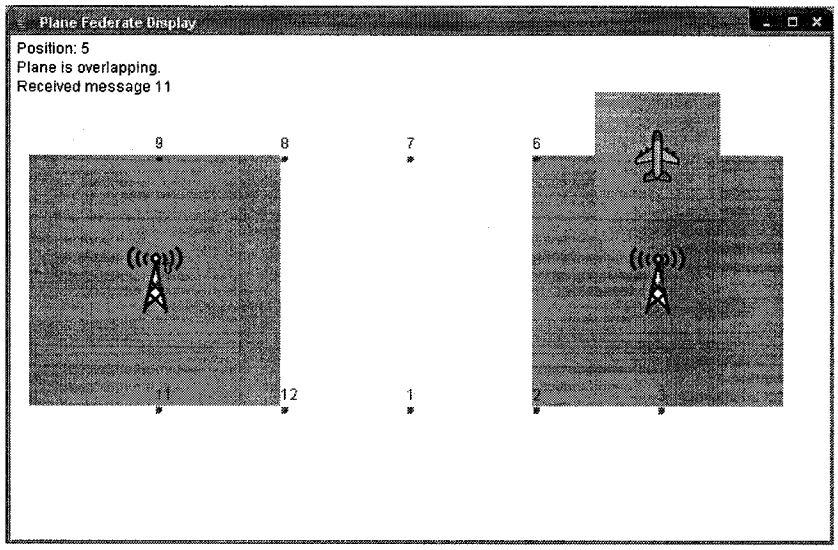


Figure 9.8: The demonstration during execution, showing the Plane in overlap with a Radio Tower.

The functionality of the DDM services and Conference Infrastructure is functioning correctly, as the test federation shows the messages being sent from the publishing federates, the Radio Towers, are only received by the Plane when it is overlapping the publishing regions. Messages sent between LCC nodes were displayed in the Java console, and were used to verify that the nodes were communicating correctly. This demonstrates the correct operation of the matching algorithm, the conference tagging, and sending the publishing region information to the subscribing node.

Using DDM in this federation allowed the number of messages sent between federates to be minimized and reduced the complexity of the federates. Checking where the Plane is in relation to the Radio Towers, and whether or not it should accept a received message from a Tower was unnecessary within the Plane federate, because it only received attribute updates when it was close to either one of the Towers.

The federation used for testing the DDM services focuses on a small number of federates, only using the DDM services for exchanging Object Class updates. The test is intentionally limited in scope to ensure that there is no communication interference from other federates. In order to ensure that the correct messages are being exchanged, increasing the volume of the messages being sent increases the difficulty of observing the results. The objective is to observe that filtering is occurring within the CI, and messages are not being received when the Plane does not overlap with either Tower.

A standardized test federation would have provided comparative results, however there is no such suite available. The academic or open source RTIs are limited in scope and functionality, and any testing done during development is not available. Given the limited implementation of the DDM services, there is no test federation to be used. As mentioned previously, each researcher uses artificial tests designed solely to stress the matching algorithm, rather than highlight the uses of the DDM services or validity of their implementation. A test suite for generalized DDM services testing was outside the scope of this research, though would be a valuable contribution for future work.

Chapter 10

Conclusions and Future Work

The thesis of this document makes three claims: that the Sort-Based matching algorithm could be further improved, that tagging Attendees within the CI provides finer grained message filtering within a conference, and that DDM research should have a broader focus than just the matching algorithm. The claims made in the thesis were realized by the software implementation of the DDM services and modifications to SIP-RTI Local RTI Component and Conferencing Infrastructure. This chapter summarizes the results of the research, outlines the contributions, and offers some suggestions for future work with the SIP-RTI software.

Realizing the claims made in the thesis involved research and development surrounding the Sort-Based Matching algorithm and the DRR technique. To support the DDM services, some modifications to the LCC were required. User definable tagging was introduced and added to the LCC to provide a filtering mechanism within conferences. Given that published research does not focus on the implementation of the DDM services or matching algorithm, this work provides the community with an open source DDM implementation that can be used both as a basis for developing new matching algorithms and as a learning tool for implementing the DDM services.

The success of this implementation was demonstrated in Chapter 9 by two sets

of tests. The first test on the DRR technique added to the Sort-Based Matching algorithm showed that there was a reduction in work performed by the algorithm when subscription regions did not overlap. The second test involved a federate that updated its subscription region, and correctly received or did not received the updates sent by the publishing federates. The resulting implementation of the DDM services nearly completes the IEEE 1516 Specification outlined for the DDM services, and addresses the relationship between the DM and DDM Services.

10.1 Contributions

The major contributions were made to the body of knowledge surrounding the RTI are summarized below.

10.1.1 Dynamic Region Reduction

This techniques was developed for the Sort-Based Matching Algorithm in order to improve the performance of the algorithm. This technique dynamically reduces the number of regions that need to be checked in each iteration of the algorithm by removing regions that do not overlap in the previous iterations. This technique has been tested by comparing the number of iterations with the original algorithm. The results in Chapter 9 indicate that the DRR technique reduces the number of regions being processed by at least 40% after the first iteration (when the regions ranges are 25% of the dimension size).

10.1.2 User-definable filtering in the LCC through Attendee Tagging

Tagging within the Conference Infrastructure was added to facilitate message filtering within a conference, rather than relying solely on the conference structure to filter messages. This also brings a user-definable filtering mechanism to the LCC, which will in turn make it a more robust platform for application connectivity. This is used by the LRC to facilitate the storage of DDM state information, and provides the filtering necessary to selectively send messages to conference Attendees. Attendee tagging is described in Chapter 6. The use of tags for filtering Attendees for the DDM services reduces the overhead of modifying the conference structure during the federation execution, and provides a means for applications to follow the changes in region overlap through the tags being used within a conference.

10.1.3 Implementation of IEEE 1516 DDM Services

Prior and current research is focused on implementing the DMSO 1.3 version of DDM. While it is still in use today, research has not yet advanced to implementing the newer version of DDM. The implementation of the DDM services in this project followed the IEEE 1516 specification, providing the research community with an open source implementation of the DDM services, not just a matching algorithm. The overlap between the DM and DDM services has been explored, and the discussion regarding the design of the services is presented in Chapter 7.

10.2 Future Work

While the presented work completed the objectives of this research, there are numerous ways in which it can be continued. Further insight into the use of the Data Distri-

bution Management services or the implementations and improvements to matching algorithm research can still be found. Several suggestions for addition work are outlined below.

- Apply the DRR technique to other matching algorithms. While it provided an improvement in the Sort-Based matching algorithm, it begs the questions regarding its universal appeal. Adding DRR to other algorithms could allow the technique to be a generalized improvement for matching algorithm efficiency.

- Add application call-backs from the CI to inform the application when Indicators and tags are added to conferences. This would solve the late-subcriber issue discovered late in the development process, as discussed in Chapter 8. Providing methods to turn on or off notification of these events would allow the application to decide whether or not they are necessary, and use them accordingly.

- Increase the use of tagging within the LCC. For this project, tags were only added to Listener Attendees and Listener Indicators. An ideal implementation would see all objects within the LCC tagged. Speaker Attendees and Indicators could be tagged, as well as conferences. Tagging speakers allows the conference to send messages automatically using the tags associated with the Speaker, so filtering could be handle transparently within the LCC layer.

- Following the previous suggestion, selectively sending tags to remote LCC nodes would reduce the number of unnecessary tags sent between LCC nodes. If a Listener within a conference is tagged, it should not update its Remote Indicators at LCC nodes that do not have any Speakers using that tag. No new messages will get sent to the Listening conference. With Speaker Attendees and Speaker Indicators tagged, the LCC can check if the Speaker Indicator has the same tag as the Listener, if so, then the Listener Indicator on that node is updated.

- Removing floor objects. While floors were introduced in the original implementation of the conference infrastructure to support object ownership management, their functionality could be implemented with tags. Floors are used to denote ownership within a conference, and in order to make an announcement, a Speaker has to own a floor. This functionality can be replaced with an owner tag. Ownership transfer can be accomplished by removing the tag from one Speaker, and associating it with another.

- Distribute the DDM services. A future implementation could allow the DDM services to function as a stand alone service, handling the region management for several remote LRCs. This way, all region operations could happen within one location, and messages could be sent back to the original LRC to adjust tags.

- Develop a set of tests that can be used to test RTI service groups individually as well as all the services together. A test suite with a number of test cases can benefit not only future development of the SIP-RTI, but can provide a set of results that can be used to compare RTI implementations.

References

[1] F. Kuhl, R. Weatherly, and J. Dahmann. *Creating Computer Simulations An Introduction to the High Level Architecture.* Prentice Hall, Englewood Cliffs, N.J., 1999.

[2] Henning Schulzrinne and Mark Handley. Session Initiation Protocol. http://www.cs.columbia.edu/sip/.

[3] T. Pearce and N. Farid. If RTI's Have a Standard API, Why Don't They Interoperate? In *Proceedings of the Simulation Interoperability Standards Organization Fall Simulation Interoperability Workshop*, 2004.

[4] Claude Van Ham. SIP-RTI: A High Level Architecture, Runtime Infrastructure built on a SIP-enabled Conferencing Mechanism. Master's thesis, Carleton University, July 2006.

[5] FDK - Federated Simulations Development Kit. http://www-static.cc.gatech.edu/computing/pads/fdk/.

[6] Dominique Canazzi. yaRTI Yet Another RTI. http://perso.orange.fr/dominique.canazzi/dominique.htm.

[7] Sun Microsystems, Inc., Santa Clara, California. *Java Message Service*, 2002.

[8] Object Management Group. http://www.omg.org/.

[9] Object Management Group. *Data Distribution Service (DDS) for Real-Time Systems v1.2*, 2007.

[10] The Institute of Electrical and Electronics Engineers, Inc., New York. *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules, IEEE Std 1516-2000*, 2000.

[11] The Institute of Electrical and Electronics Engineers, New York. *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification, IEEE Std 1516.1-2000*, 2000.

[12] The Institute of Electrical and Electronics Engineers, New York. *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template, IEEE Std 1516.2-2000*, 2000.

[13] D. J. Van Hook, S. J. Rak, and J. O. Calvin. Approaches to RTI Implementation of HLA Data Distribution Management Services. In *Fifteenth Workshop on Standards for the Ineroperability of Distributed Simulations*, pages 16–20, September 1996.

[14] A. Boukerche and A. J. Roy. Dynamic Grid-Based Approach to Data Distribution Management. In *Journal of Parallel and Distributed Computing*, volume 62, pages 366–392, 2002.

[15] A. Boukerche and K. Lu. Optimized Dynamic Grid-Based DDM Protocol for Large Scale Distributed Simulation Systms. In *Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS'05)*, 2005.

[16] Azzedine Boukerche and Caron Dzermajko. Dynamic Grid-Based vs. Region-Based Data Distribution Management Strategies for Large-Scale Distributed Systems. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)*, 2003.

[17] Pankaj Gupta and Ratan K. Guha. Design, Analysis, and Performance Evaluation of an Efficient Algorithm for Data Distribution Management in High Level Architecture. Technical report, School of Computer Science University of Central Florida, December 2005.

[18] C. Raczy, G. Tan, and J.Yu. A Sort-Based DDM Matching Algorithm for HLA. In *ACM Transaction on Modeling and Computer Simulation*, volume 15, pages 14–38, January 2005.

[19] C. Raczy, G. Tan, and J.Yu. Evaluation of a sort-based matching algorithm for the DDM. In *Proceedings of the 16th Workshop on Parellel and Distributed Simulation*, pages 68–75, Washington, DC, May 2002.

[20] Claude Van Ham. SIP-RTI: SIP Local Conferencing Component Data Distribution Design. Technical Report SCE–06–12, Carleton University, 2006.

[21] Claude Van Ham. SIP-RTI: SIP Local Conferencing Component Software Implementation. Technical Report SCE–06–10, Carleton University, 2006.

[22] Claude Van Ham. SIP-RTI: SIP Local RTI Component Software Implementation. Technical Report SCE–06–11, Carleton University, 2006.

[23] Claude Van Ham and Trevor Pearce. Implementing RTI Data Distribution Management Using SIP Services. In *Proceedings of the Simulation Interoperability Standards Organization European Simulation Interoperability Workshop*, 2005.

[24] OCLC Online Computer Library Center. Dewey Decimal Classification System. http://www.oclc.org/dewey/, Last Checked 2007-09-16.

[25] The Library of Congress. The Library of Congress Classification System. http://www.loc.gov/aba/cataloging/classification/, Last Checked 2007-09-16.

[26] Adam Mathes. Folksonomies - Cooperateive Classification and Communication Through Shared Metadata. Technical report, Graduate School of Library and Information Science, University of Illinois Urbana-Champaign, December 2004.

[27] Yahoo! Inc. Flickr - Photo Sharing. http://www.flickr.com/, Last Checked 2007-04-10.

[28] Yahoo! Inc. Del.icio.us. http://del.icio.us/, Last Checked 2007-04-10.

[29] Google Inc. YouTube. http://www.youtube.com/, Last Checked 2007-04-25.

[30] CiteSeer. http://citeseer.ist.psu.edu/, Last Checked 2007-04-25.

[31] Ellyssa Kroski. The Hive Mind: Folksonomies and User-Based Tagging. http://infotangle.blogsome.com/2005/12/07/the-hive-mind-folksonomies-and-user-based-tagging/, 2005, Last Checked 2007-09-16.

[32] Thomas Vander Wal. vanderwal.net - Folksonomy Coinage and Definition. http://vanderwal.net/folksonomy.html, Last Checked 2007-09-16.

[33] Katherine L. Morse. Data Distribution Management Migration from DoD 1.3 to IEEE 1516. Technical report, IEEE, 2001.

[34] Mak Technologies. Mak RTI. http://www.mak.com/.

[35] Pitch Technologies. Pitch pRTI 1516. http://www.pitch.se/.

Appendix A

HLA RTI Methods Implemented

This appendix outlines the HLA methods that were added to the SIP-RTI, in both the RTI and Federate Ambassadors.

A.1 RTI Ambassador Methods

A brief description of each of the RTI Ambassador methods implemented is provided. The methods are presented following the numbering scheme used in the specification. For full details, please refer to the IEEE 1516 HLA Specification [10].

9.2 Create Region

This method is used by the Federate to create a new region specification. The method returns a region handle for the newly created region.

9.2 Commit Region Modification

This method will commit the range bound changes made to a region specification object by the federate. The bound changes will be applied to all the used region

realizations based off of this region specification.

9.4 Delete Region

This method is used to remove a Region Specification. This method can only be called once a federate has unsubscribed with regions and unassociated all publishing regions based on this region specification.

9.5 Register Object Instance With Regions

This method is used to register a new object instance, and associate a region specification for use with publishing in one autonomous step, rather than registering a new object instance (using method 6.4 Register Object Instance) and then associating a region for publishing.

9.6 Associate Regions for Updates

This method creates a region realization based on the given region specification for use with the given attribute instance.

9.7 Unassociate Regions for Updates

This method removes the region realization association with the given object attribute instance.

9.8 Subscribe Object Class Attributes With Regions

This method creates a region realization that will be used for subscribing to an object class attribute.

9.9 Unsubscribe Object Class Attributes With Regions

This method removes the region realization associated with subscription for this object class attribute.

10.12 Get Dimension Handle

This method returns the dimension handle for the dimension with the supplied name.

10.13 Get Dimension Name

This method returns the name of the dimension represented by the given handle.

10.14 Get Dimension Upper Bound

This method returns the upper bound of the given dimension.

10.15 Get Available Dimensions For Class Attribute

This method returns a DimensionHandleSet containing dimension handles for all of the dimensions associated with the given object class attribute.

10.30 Get Dimension Handle Set

This method returns a dimension handle set of all the handles for the given region specification.

10.31 Get Range Bounds

This method returns the range bounds for the given dimension on the specified region specification.

10.32 Set Range Bounds

This method sets the bounds for the given ranges on the specified region specification.

A.2 Federate Ambassador Methods

A brief description of each of the Federate Ambassador methods implemented is provided. The methods are presented following the numbering scheme used in the specification. For full details, please refer to the HLA Specification [10].

6.15 Attributes In Scope

This method is called by the RTI to inform a federate that a subscription region it owns has come into overlap with a publishing region. This can be either because the subscription region was changed, or the publishing region was.

6.16 Attributes Out Of Scope

This method is called by the RTI to inform a federate that a subscription region it owns has left overlap with a publishing region. This can be either because the subscription region was changed, or the publishing region was.

Appendix B

LCC API Methods Implemented

This appendix summarizes the methods that were newly implemented or modified for this project. In some cases, methods were modified for the addition of tags. The methods summarized are those that change the public interface of the LCC, and can be used by applications using the LCC.

B.1 LCC

The methods added or modified to the main LCC class are noted below.

Add Attendee

The add attendee method places an attendee of the desired type into the selected conference. A set of tags can be provided to be added to the attendee when it is added to the conference. This method was originally part of the LCC, but was modified to support the tag argument.

Add Tag

This method is used to add a set of tags to a conference's list of available tags. This does not result in any messages being sent between any connected LCC nodes, as this is only a local modification. This method will create the tag in the conference, but it will not associate the tag with any attendees. This is used to add the DM and DDM tags to a conference when the FOM is being read.

Untag Attendees

The untag attendee method removes the given tag from all of the listener attendees in the conference. This is used by the LRC after a publishing region has been removed, and all of the tags associated with that region are no longer required.

Tag Attendee

This method tags a listener attendee in a conference associated with a supplied caller ID. This method is used when the an overlap is found and the attendee can be tagged with that publishing region's tag.

Untag Attendee

This method will untag a single listener attendee in a conference. This method is used when the region associated with this caller's federate has left overlap with a publisher.

Make An Announcement

This method sends a message to all the listeners within a conference. A parameter was added so a set of tags could be associated with this message. The tags filter

the local conference listener attendees and remote listener indicators, sending the announcement to only those that have are tagged with any of the tags provided. This method was originally part of the LCC, but it was modified to support the tag argument.