A Comparison and Mapping of Data Distribution Service and High-Level Architecture

Rajive Joshi, Ph.D.

Gerardo-Pardo Castellote, Ph.D.

Real-Time Innovations, Inc.

3975 Freedom Circle, Santa Clara, CA 95054

+1-408-200-4700, info@rti.com

ABSTRACT: *The OMG Data-Distribution Service (DDS) is an emerging specification for publish-subscribe data-distribution systems. The purpose of the specification is to provide a common application-level interface that clearly defines the data-distribution service. The specification describes the service using UML, thus providing a platform-independent model that can then be mapped into a variety of concrete platforms and programming languages.*

DDS attempts to unify the common practice of several existing implementations enumerating and providing formal definitions for the QoS (Quality of Service) settings that can be used to configure the service.

In this paper we provide a comparative overview of the data distribution service with respect to high-level architecture. We describe the equivalent terminology and concepts, and highlight the key similarities and differences in the areas of declaration management, object management, data distribution management, ownership management, federation management, and time management.

We explore the architectural mapping between HLA and DDS. We develop an outline for translating from one model to the other, and examine the needed supporting transformations and assumptions. We conclude with remarks and observations on building applications that can utilize both HLA and DDS technologies.

1 Introduction

Data Distribution Service (DDS) is a newly adopted specification from the Object Management Group (OMG), an approximately 800 member consortium, to create vendor independent software standards. DDS is aimed at a diverse community of users requiring **data-centric publish-subscribe** communications. These include applications in aerospace and defense, distributed simulation, industrial automation, distributed control, robotics, telecom, and networked consumer electronics.

The goal of DDS (see [1], [2]) is to facilitate the efficient distribution of data in a distributed system. DDS is similar to HLA in some regards: it has a publish-subscribe communication architecture, supports object modeling and the notion disseminating updates to object instances, provides support for content based subscriptions which may be likened to DDM regions, and standardizes on the API specification for portability. It differs from HLA in some regards: in its support for object modeling, and ownership management. DDS addresses some new aspects not addressed by HLA, such as: a rich set Quality of Service (QoS) policies, a strongly typed data model, and support for state propagation including coherent and ordered data distribution; while leaving out some other aspects addressed by HLA, such as: time management and federation management.

DDS targets real-time systems; the API and Quality of Service (QoS) are chosen to balance predictable behavior and implementation efficiency/performance. The DDS specification describes two levels of interfaces:

- A lower Data-Centric Publish-Subscribe (DCPS) level that is targeted towards the efficient delivery of the proper information to the proper recipients
- An optional higher Data-Local Reconstruction Layer (DLRL) level, which allows for a simpler integration into the application layer.

The DDS specification [2] is developed under the OMG MDA process [3], and describes the service using Unified Modeling language (UML) and Interface Definition

Language (IDL). The specification provides a platform-independent model (PIM) that can then be mapped into a variety of concrete platform specific models (PSMs) and programming languages.

DDS draws upon common practice in existing publish-subscribe architectures including HLA ([5], [6]) OMG event notification service [7], Java Messaging Service (JMS) [8], and experience with Real-Time Innovations' (RTI's) RTI Data Distribution Service (formerly NDDS) product [8]. DDS departs from previous approaches in two primary aspects: (1) enumerating and providing formal definitions for the QoS (Quality of Service) settings that can be used to configure the service, and (2) the tight binding of a "topic" to a data-type, thus making it more than just a "routing" label. The coupling of "topic" with a data-type, along-with the additional QoS settings enables implementation optimizations such as pre-allocating the resources needed to send or receive a "topic".

2 DDS Synopsis

The publish-subscribe model connects anonymous information producers (publishers) with information consumers (subscribers). The overall distributed application (the PS system) is composed of processes, each running in a separate address space possibly on different computers. We will call each of these processes a "**participant**". A participant may simultaneously publish and subscribe to information.

Figure 1 illustrates the overall DCPS model, which consists of the following entities: ***DomainParticipant***, ***DataWriter***, ***DataReader***, ***Publisher***, ***Subscriber***, and ***Topic***. All these classes extend ***DCPSEntity***, representing their ability to be configured through QoS policies, be notified of events via listener objects, and support conditions that can be waited upon by the application. Each specialization of the ***DCPSEntity*** base class has a corresponding specialized listener and a set of ***QoSPolicy*** values that are suitable to it.

Publisher represents the objects responsible for data issuance. A ***Publisher*** may publish data of different data types. A ***DataWriter*** is a typed facade to a publisher; participants use ***DataWriter(s)*** to communicate the value of and changes to data of a given type. Once new data values have been communicated to the publisher, it is the ***Publisher***'s responsibility to determine when it is appropriate to issue the corresponding message and to actually perform the issuance (the Publisher will do this according to its QoS, or the QoS attached to the corresponding ***DataWriter***, and/or its internal state).

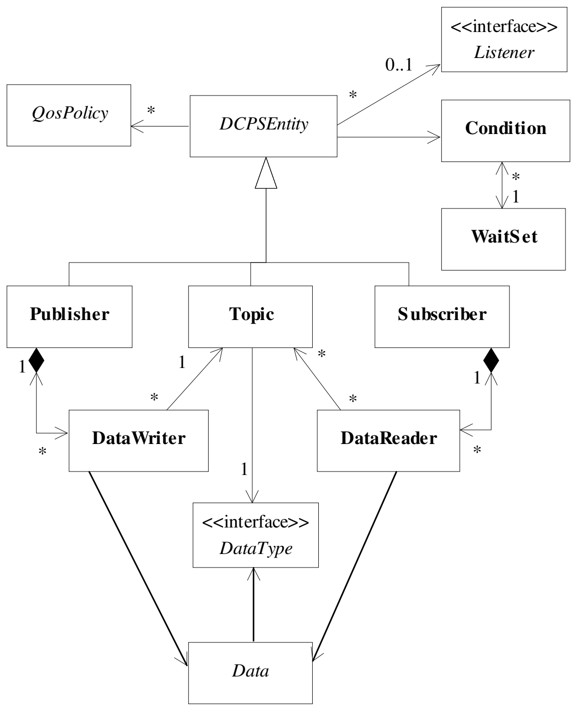

Figure 1 UML diagram of the DCPS model

A ***Subscriber*** receives published data and makes it available to the participant. A ***Subscriber*** may receive and dispatch data of different specified types. To access the received data, the participant must use a typed ***DataReader*** attached to the subscriber.

The association of a ***DataWriter*** object (representing a publication) with ***DataReader*** objects (representing the subscriptions) is done by means of the ***Topic***. A ***Topic*** associates a name (unique in the system), a data type, and QoS related to the data itself. The type definition provides enough information for the service to manipulate the data (for example, serialize it into a network-format for transmission). The definition can be done by means of a textual language (e.g. something like "float x; float y;") or by means of an operational "plugin" that provides the necessary methods.

A technical overview of DDS can be found in [1] with details in the OMG submission [2]. A good high-level overview of the background, benefits, and applications can be found in [11].

The rest of the paper will focus on a comparison of DDS with HLA (Section 0) and outline a mapping of HLA to

DDS (Section 5) from an end user/distributed application developer's perspective.

3 HLA/DDS Equivalents

A map of key HLA concepts and terminology and the DDS equivalents is summarized below. Additional details can be found in Section 5.

HLA	DDS
HLA-OMT	DDS-DLRL
HLA-RTI (IFSpec)	DDS-DCPS
HLA-Rules	-
Federation	Domain
Federate	Participant / Application
RTIAmbassador	***DomainParticipant, Publisher, DataWriter, Subscriber, DataReader***
FederateAmbassador	***Listener classes***
Object class	***Keyed Topic***
Interaction class	***Topic*** (no keys)
Update attribute	Write "keyed" instance
Reflect attribute	Read/Take "keyed" data samples
Send interaction	Write "non-keyed" instance
Receive interaction	Read/Take "non-keyed" samples

Table 1 HLA/DDS equivalents

4 Comparison of DDS with HLA

In this section we discuss DDS from the perspective of a distributed application developer familiar with HLA.

4.1 Similarities with HLA

4.1.1 Publish-subscribe architecture

Like HLA, DDS-DCPS offers a publish-subscribe communication model. Data dissemination between producers and consumers may be from one-to-one, one-to-many, many-to-one, or many-to-many. The communication model is decentralized: publishers and subscribers are decoupled i.e. have no knowledge of each other. ***Publishers*** and ***Subscribers*** can join/leave dynamically. This model enables dynamically scalable application architectures.

4.1.2 Support of diverse dissemination semantics

The information transferred by data-centric communications can be classified into: signals, streams, and states. **Signals** represent data that is continuously changing (such as the readings of a sensor). Signals can often be sent best-efforts. **Streams** represent snapshots of the value of a data-object that must be interpreted in the context of previous snapshots. Streams often need to be sent reliably. **States** represent the state of a set of objects (or systems) codified as the most current value of a set of data attributes (or data structures). The state of an object does not necessarily change with any fixed period. Fast changes may be followed by long intervals without change. Consumers of "state data" are typically interested in the most current state. However, as the state may not change for a long time, the middleware may need to ensure that the most current state is delivered reliably. In other words, if a value is missed, then it is not always acceptable to wait until the value changes again.

Both HLA and DDS support dissemination of signal and stream data. In HLA the "transportation" property of an object class attribute can be specified as "best-effort" or "reliable". In DDS the ***RELIABILITY QoSPolicy*** can set to ***BEST_EFFORT*** or ***RELIABLE***. DDS provides additional QoS policies to support the dissemination of state data, as described in Section 4.3.2. In HLA, the time-management aspect can be used to support aspects of state data dissemination, but comes with additional ordering semantics.

4.1.3 Object classes and per instance updates

Like HLA, DDS offers support for modeling object classes and updating specific object instances. In DDS, a ***Topic*** represents an object class. Topics can be associated with a ***Key*** to uniquely identify object instances. The representation and format of the key depends on the data type. However, since a ***Topic*** is bound to a unique type, the service can always interpret the key properly given the ***Topic*** and the value of a data object.

The combination of a fixed-type ***Topic*** and a ***Key*** (henceforth ***Keyed Topic***) is sensible for data-centric systems because the ***Topic*** represents either a unique data object (e.g. a temperature sensor) in the case where there are no keys, or a set or related data-objects that are treated uniformly (e.g. track information of aircraft as generated by a radar system), where each individual aircraft can be distinguished by a key. DDS delegates the interpretation of the key to the data-type, so that it is possible for the key to be a single value within the data-object (e.g. a serial number field) or a combination of fields (e.g. airline-name and flight-number).

4.1.4 Application portability via a standardized API specification

Like HLA, the DDS specification specifies a standardized API or interface specification. The DDS model is described in UML, while the DDS APIs are described in standard OMG IDL. The standardized IDL to language (C, C++, Java, Ada, etc.) mapping rules are applied to derive a precisely defined language specific DDS API. Implementation level details are not addressed by the specification.

4.2 Differences from HLA

4.2.1 Object modeling supports composition and inheritance

HLA Object Model Template (HLA-OMT) provides a common framework for object model documentation, thereby fostering interoperability and reuse of simulations and their components. HLA-OMT provides data modeling constructs that allows objects to be described in terms of their attributes and inheritance relationships.

The DDS-DLRL supports a richer set of data modeling constructs: it allows objects to be described not only in terms of their attributes and inheritance relationships, but also in terms of methods and aggregation relationships with other objects. The DDS-DLRL layer provides more direct access to the exchanged data, seamlessly integrated with the native-language data-accessing constructs.

4.2.2 Content based subscriptions vs. regions

HLA offers support for associating normalized "regions" with a publication to specify the range of data produced, and with a subscription to specify the range of data that can be consumed. Implementations generally exploit the region information to minimize data transfer and optimize routing.

DDS-DCPS supports content-based subscriptions by means of a filter that allows a ***DataReader*** to filter the data received from a given ***Topic*** based on the contents of the data itself. A filter is specified in terms of an expression and parameters, which can be interpreted by a ***Subscriber*** and/or a ***Publisher***. Thus, implementations can exploit this information to optimize information exchange.

In addition, DDS also has a notion of a ***PARTITION QoSPolicy***, which can be used to introduce a logical partition among the topics visible by a ***Publisher*** and a ***Subscriber***. A ***DataWriter*** within a ***Publisher*** communicates with a ***DataReader*** in a ***Subscriber*** only if (in addition to matching the Topic and having compatible QoS) the ***Publisher*** and ***Subscriber*** have a common partition name string.

4.2.3 Ownership semantics and granularity

HLA supports the notion of the attributes of an object instance being owned by federates. HLA requires exclusive ownership of an attribute----at most one federate can own it at any given time; however ownership of an attribute can be divested and/or acquired. DDS supports the notion of ownership, but with differences. DDS attaches an ***OWNERSHIP QoSPolicy*** to a ***Topic***, and can be either ***EXCLUSIVE*** or ***SHARED***. The ***EXCLUSIVE*** ownership of an instance is somewhat similar to the HLA ownership: at most one ***DataWriter*** can be the owner of an instance at any given time. To arbitrate ownership among multiple ***DataWriters,*** each is associated with an ***OWNERSHIP_STRENGTH QoSPolicy***. The highest ***OWNERSHIP_STRENGTH DataWriter*** is considered the owner. Thus, unlike HLA, the owner can change dynamically based on the ***OWNERSHIP_STRENGTH***, and does not require a divesture/acquire protocol. Unlike HLA, DDS also allows ***SHARED*** ownership of an instance (the default). Multiple writers are allowed to update an instance, and all the updates are made available to the readers.

Yet another difference is in the granularity of ownership. HLA allows different attributes of an instance to be owned by different federates. DDS allows ownership only at the level of instances. However, equivalent effect can be achieved in DDS by treating each separately "own"-able attribute as ***Keyed Topic*** and using the same key for all such attributes. This is described in Section 5.2.1.

4.3 New in DDS

4.3.1 Strongly typed data model

In DDS, participants can 'read' and 'write' data efficiently and naturally with a typed interface. Underneath, the DDS middleware will distribute the data so that each reading participant can access the 'most-current' values.

This is a significant departure from HLA, where the data elements themselves are un-typed and un-marshaled. In HLA, the data elements are plain sequences of octets; the data marshalling/demarshalling is left to the application developer.

4.3.2 State propagation semantics

An important use case for data-centric publish subscribe systems is the propagation of state information. Here we use the word "state" in the classic meaning of system theory and state-machines. That is, the state of the system is the information needed to determine future responses without reference to the past history of inputs and outputs. In general, the state of a system is described by the combined values of a set of data objects that dynamic systems call the “state variables”.

State propagation is important because it provides a compact way for an application to model a remote system as well as allowing a late-joining participant to behave as if it had seen the complete history of the system.

For data-centric systems, if so desired by the application, the DCPS service can ensures that:

(a) The states reconstructed by the subscribing participant should be restricted to states that actually existed in the publishing participant.

(b) The order in which the states are reconstructed on the subscribing participant should preserve the order in which the states happened in the publishing participant.

(c) If the state on the publishing side settles (i.e. does not change for “a while”) the state seen by the subscribing participant should match that of the publishing participant.

4.3.2.1 Coherent updates

Sometimes multiple state variables must be updated together for the state to transition to the next coherent (or valid) state. Imagine for example that the latitude, longitude, velocity vector, and altitude of an aircraft are kept as three separate state variables A=(latitude, longitude), B=(velocity_vector) and C=(altitude). The DDS interface must provide the participant the means to update A, B, and C “atomically” in the sense that the receiving participant should not be allowed to see a new value of A without simultaneously seeing the new value for B and C as well. Otherwise they may erroneously infer the aircraft is on a collision course.

DCPS allows a participant to request that a set of changes be propagated in such a way that they are interpreted at the receiver's side as a consistent set of modifications. This functionality is provided by a ***Publisher*** via two operations, namely *`begin_coherent_changes()`* to start a coherent set and *`end_coherent_changes()`* to terminate it.

4.3.2.2 Ordered delivery

DDS also defines a ***DESTINATION_ORDER QoSPolicy*** that can be associated with a ***Topic*** or a ***DataReader***. The destination order can be ***BY_RECEPTION_TIMESTAMP***, which is like the “Receive ordered” (RO) delivery in HLA, i.e. data is ordered based on the reception time in each ***Subscriber***. Thus, there is no guarantee that changes will be seen in the same order at different ***Subscribers***.

The destination order can be ***BY_SOURCE_TIMESTAMP*** which can means data is ordered based on the time-stamped placed at the source either by the service or the application. Assuming that a global logical time-stamp can be maintained by some alternate means (say by a helper “logical clock” utility), DDS can effectively support “time-stamped order” (TSO) delivery in the HLA-sense.

4.3.2.3 Access units

In large systems, it may not be practical to model all the state variables as defining a single monolithic state. It may also not be practical to insist that all changes to state variables made by a participant are propagated in order without introducing delays. For this reason, the application may partition the state into separate independent units, each composed of several variables. DCPS refers to each of these units as an “access unit”.

DCPS offers several ways for the application to define the access units: (1) by means of grouping ***DataReader (DataWriter)*** objects under ***Publisher (Subscriber)*** objects, and also (2) by means of the ***PRESENTATION QoSPolicy*** on a ***Publisher (Subscriber).*** This policy defines the ***access_scope***---the largest scope spanning the object instances for which order and coherency of updates can be preserved within a ***Publisher (Subscriber)***, and weather or not the application is interested in preserving coherency and/or ordering within that scope.

4.3.3 Rich QoS policies to capture communication semantics

A distinguishing aspect of DDS is the clean separation of the *syntactical* communication model specified via UML diagrams, from the communication *semantics* captured by associating ***QoSPolicies*** with various ***DCPSEntity*** defined in the UML model. This approach is extensible to support future needs as new semantic requirements emerge, without requiring a change in the syntactical structure of the communication model.

DDS enumerates a list of ***QoSPolicies***, defines their interpretation, specifies the possible values, specifies the DCPS entities to which they are applicable, and specifies weather they are dynamically changeable during the course of operation. The ***QoSPolicies*** supported by DDS can be categorized into those:

- Relating to what information is disseminated: ***PARTITION, DURABILITY;***
- Relating to the grouping, coherency, and ordering of information: ***PRESENTATION, DESTINATION_ORDER;***
- Relating to the priority of information disseminated: ***OWNERSHIP, OWNERSHIP_STRENGTH;***
- Relating to the validity of information: ***LIVELINESS, DEADLINE, TIME_BASED_FILTER;***
- Relating to the resources dedicated to manage information: ***RELIABILITY, HISTORY, RESOURCE_LIMITS;***
- Relating to optimizing information exchange: ***LATENCY_BUDGET;***
- Relating to user-defined information semantics (eg. authentication): ***USER_DATA.***

Some of these are elaborated here. Additional details can be found in [2].

4.3.4 Access to meta-data

Meta-data refers to the information about events happening in a domain, such as participants joining a domain, creation of or topics, data readers and/or data writers in a domain. A distinguishing aspect of DDS is the access to *meta-data* via **built-in topics** via a ***DomainParticipant's*** `get_builtin_subscriber()` method. DDS specifies the following built-in topics:

- DCPSParticipant
- DCPSTopic
- DCPSPublication
- DCPSSubscription

Using these, an application can get information on the events happening in the domain and/or middleware.

4.4 In HLA, but not in DDS

4.4.1 No APIs specific to federation save/restore and synchronization points

Unlike HLA, DDS does not provide standardized APIs specifically for domain save/restore, or for defining domain-wide synchronization points. However, DDS does not preclude one from building an HLA-like federation management services using the standardized DDS APIs and QoS policies (see Section 5.2.5).

4.4.2 No APIs specific to time management

Unlike HLA, DDS does not provide standardized APIs specifically for time-management. However, DDS does not preclude one from building an HLA-like time management service using the standardized DDS APIs and QoS policies (see Section 5.2.6).

4.5 Beyond the scope of DDS and HLA

4.5.1 Implementation details

Like HLA, DDS leaves the implementation details up to the middleware provider/vendor. DDS standardizes only the on the APIs, the QoS policies, and their semantics.

4.5.2 Wire protocol

Like HLA, DDS does not specify an underlying wire-protocol (as it is considered to be an implementation detail). However, for networked distributed applications, this is a highly significant (and often controversial) choice, as it influences the interoperability with other distributed applications. OMG CORBA middleware uses the IIOP standard wire-protocol for distributed applications. An efficient wire protocol suited for data-centric publish subscribe is the Real-Time Publish-Subscribe (RTPS) wire protocol [10] developed by RTI, and adopted as an industry standard by the IDA Group. The prevalent choice will likely be determined by a combination of market forces, data delivery performance, backwards compatibility, future extensibility, and ease of portability of the underlying wire protocol.

4.5.3 Topic "key" specification and mapping

The DDS specification defines APIs for handling ***Keyed Topics***, and implicitly defines a mapping of "keys" to ***instance_handles***. Thus, an implementation provider can chose the manner in which the "keys" are specified and mapped to instance handles.

4.5.4 Tools and process for distributed application programming and debugging

Like HLA, the tools and process for distributed application programming and debugging using DDS are considered outside the scope of the specification. Vendors can chose to provide the best value driven by market forces.

5 Mapping HLA to DDS

In this section we briefly examine each major aspect of the HLA specification and relate it the DDS specification, from the perspective of a distributed application developer. The HLA/DDS equivalents are summarized in Table 1.

5.1 Object modeling

The DDS equivalent of the HLA-OMT is the DDS-DLRL layer. It defines support for modeling object class specialization and aggregation hierarchies. The DDS-

DLRL layer defines a UML meta-model that specifies the object relationships that can be described using DLRL. The DDS equivalent of developing a HLA FOM and/or SOM would be to describe the simulation entities in DDS-DLRL.

5.2 Interface specification

The DDS equivalent of the HLA-RTI interface specification is the DDS-DCPS communication model and APIs specified using UML and and IDL. The HLA-RTI API specifies two primary interfaces: the ***RTIAmbassador***, which defines the APIs that an application can use to access the middleware facilities, and the ***FederateAmbassador***, which defines the *callback* APIs that the middleware can use to notify an application of updates and events. The DDS equivalent of the ***RTIAmbassador*** APIs is embodied in the ***DomainParticipant, Publisher, DataWriter, Subscriber, DataReader*** classes; while the callback equivalent of the ***FederateAmbassador*** APIs is embodied in the ***Listener*** classes.

In addition, DDS-DCPS APIs also support a "wait-based" data access mechanism. The wait-based approach provides a set of conditions that threads inside the participant can use to block while waiting for specific sets of changes. When any of the changes of interest occur, the thread is unblocked and can access the data directly in its own context.

Lets briefly examine how each of the key HLA-RTI interface specification areas relate to DDS-DCPS APIs.

5.2.1 Declaration management

The DDS equivalent of an HLA "object class" is a ***Keyed Topic***, while that of an HLA "interaction class" is a non-keyed ***Topic***. However, the mechanism for updating a subset of an object instance attributes in DDS is different than in HLA. In DDS, a ***Keyed Topic*** must be created for each subset of object instance attributes that is to be updated as a unit. Each such "attribute-subset" ***Keyed Topic*** should use the same key fields so that they can refer to a common object instance. ***DataWriters*** (***DataReaders***) are bound to these "attribute-subset" ***Keyed Topics*** and attached to an "object-class" ***Publisher*** (***Subscriber***). The ***PRESENTATION QoSPolicy*** on the "object-class" ***Publisher*** (***Subscriber***) is set to ***GROUP*** to achieve coherent and ordered data dissemination for "attribute-subsets" of an object class. The grouping, coherency, and ordering semantics may be adjusted to achieve a range of behaviors.

HLA provides ***FederateAmbassador*** callbacks to let a producer know if there are consumers for a data item via the `startRegistrationForObjectClass()`, the `stopRegistrationForObjectClass()`, the `turnInteractionsOn()`, *and the* `turnInteractionsOff()` methods. In DDS, the equivalent effect is achieved by using the built-in subscriber (Section 4.3.4) to listen for equivalent events.

5.2.2 Object management

DDS-DCPS supports a ***DURABILITY QoSPolicy*** that can be set to allow late joining ***DataReaders*** to "discover" previously published object instances. DDS-DLRL provides support for modeling complex object relationships, including support for object specialization (which is the only kind of relationship modeled by HLA).

DDS provides additional control over the semantics of data distribution by providing hooks to tweak the grouping, coherency, and ordering of updates.

5.2.3 Data distribution management

HLA provides relevance and scope advisory switches for attributes and interactions; callback methods such as `turnUpdatesOnForObjectInstance()` provide hints for optimizing data distribution performance. In addition, publications and subscriptions can be tagged with regions, to optimize data routing.

DDS has the operations `suspend_publications()` and `resume_publications()` that provide a hint to the middleware that multiple data-objects within the ***Publisher*** are about to be written, and thus allow the middleware to use bandwidth more efficiently by batching the distribution of a set of writes. An implementation could disable the dissemination of messages and accumulate changes until `resume_publications()` is called.

DDS content-based subscriptions can be used to achieve some of the same goals catered to by HLA regions, as discussed in Section 4.2.2. DDS has ***TIME_BASED_FILTER QoSPolicy*** that can be used to specify a minimum separation between periodic updates delivered to a ***DataReader***, and thus cut down on the network bandwidth used.

5.2.4 Ownership management

DDS provides support for ownership management, but with differences, as discussed in Section 4.2.3. The DDS ownership management model is significantly different with regards to not forcing a "centralized" implementation. Additional handshaking and divesture/acquisition coordination semantics can be added on top of it to effectively support an HLA-like ownership model.

5.2.5 Federation management

Unlike HLA, DDS does not define standardized APIs for participant join/resign events, saving/restoring distributed application state, or for defining arbitrary synchronization points. Some of these requirements are specific to the simulation application domain, and DDS is aimed at a broader distributed application community.

However, DDS can effectively support an HLA-like federation management using the standard APIs and QoS policies. For example, an application can specifically create a ***SaveRestoreTopic*** or a ***SynchronizationPointTopic***, and require all participants subscribe to them. The QoS policies on these topics can be set to be ensure the correct operational behavior semantics, e.g. reliable and ordered.

5.2.6 Time management

Unlike HLA, DDS does not define standardized APIs for time management. These are specific to the synchronous simulation application domain, while DDS is aimed at a broader audience.

However, as hinted in Sections 4.3.2.2 and 4.4.2, DDS can effectively support an HLA-like time management model using the standard APIs and QoS policies. A DDS ***DataWriter*** allows the application to specify the time-stamp when it writes an update to an instance. A "logical clock" service can be built on top of the DDS APIs that maintains a monotonically increasing logical time within a simulation, via the use of a ***LogicalTimeTopic*** to which each participant subscribes. This logical time stamp can be use to write an object instance update. A consumer can specify a ***DESTINATION_ORDER QoSPolicy*** of ***BY_ SOURCE_TIMESTAMP***, thus effectively receiving the data in time-stamped order (TSO) delivery. Additional HLA-like time management APIs can be provided at the application level. The DDS QoS policies provide the flexibility to tailor the time management behavior to simulation needs.

5.3 Rules

HLA specifies ten rules that provide policy guidance on setting up HLA federations. The DDS specification does not specify any rules or policy guidance, but neither does it preclude the use of established policies and procedures in setting up a distributed application. Thus, one could build a distributed simulation using DDS that adhere to the equivalent HLA rules.

6 Conclusions

DDS is suitable for a large subset of the class of problems targeted by HLA, and builds on the experience of prior publish-subscribe architectures. It is novel in the use of strongly typed topics, support for state propagation, and the specification of QoS semantics. It addresses the key HLA areas except for simulation specific requirements such as time management and federation synchronization---leaving them up to the simulation domain level services that can be built on top of the standardized APIs and QoS policies. This is consistent with the broader scope of DDS, aimed at addressing a wide variety of data centric communication needs.

Possible directions for utilizing DDS within the HLA community range from: (1) building a "bridge" layer for mapping HLA and DDS, (2) implementing HLA on top of DDS, and (3) implementing DDS on top of HLA. New product categories may emerge in these areas, driven by market forces.

The RTI Data Distribution Service product line [9] includes a commercially supported implementation of DDS, along with a supporting set of distributed application programming and debugging tools.

7 References

[1] Gerardo Pardo-Castellote: "OMG Data Distribution Service: Architectural overview", IEEE International Conference on Distributed Computing Systems, 2003.

[2] OMG: "Data Distribution Service for Real-Time Systems RFP", Document orbos/2003-03-15, http://www.omg.org, March 2003.

[3] Shawn Parr, Russell Keith-Magee: "The Next Step - Applying the Model Driven Architecture to HLA", Paper Id 03S-SIW-123, 2003 Spring Simulation Interoperability Workshop. Spring 2003.

[4] Andrew Tolk, "Avoiding another Green Elephant -- A Proposal for the Next Generation HLA based on the Model Driven Architecture", Paper Id 02F-SIW-004, 2002 Fall Simulation Interoperability Workshop, Fall 2002.

[5] SISO, IEEE: "High-Level Architecture (HLA)", http://www.sisostds.org/ 2000.

[6] OMG: "Distributed Simulation Systems V1.1", Document **formal/2000-12-01,** http://ww.omg.org, 2000.

[7] OMG: "The CORBA Notification Service", Document orbos/98-11-01, http://www.omg.org, 1998.

[8] Sun Microsystems: "The Java Messaging Service (JMS) specification", http://java.sun.com/products/jms/, 2001.

[9] Gerardo Pardo-Castellote, Stan Schneider, Mark Hamilton: "NDDS: The Real-Time Publish-Subscribe Network", White Paper, Real-Time Innovations, Inc., http://www.rti.com, 1999.

[10] Interface for Distributed Automation (IDA) Group: "Real-Time Publish-Subscribe (RTPS) Wire Protocol", www.ida-group.org 2001.

[11] Gerardo Pardo-Castellote, Brett Murphy: "New Networking Standard from OMG Will Simplify Distributed Simulation Development", Paper Id 03F-SIW-65, 2003 Fall Simulation Interoperability Workshop, Fall 2003.

Author Biographies

RAJIVE JOSHI, Ph.D. is the Director of Research at Real-Time Innovations, Inc. Dr. Joshi specializes in object-oriented and component-based software architecture and design, and is currently leading the implementation of the emerging OMG's Data Distribution Service.

His professional experience includes serving as the product lead and architect of component-based graphical programming tools for distributed control applications, and serving as a consultant and developer for distributed systems projects in the areas of robotics and automation, including Schilling's Quest remotely operated vehicle, CrouseHinds's automated filament alignment system, and teleoperation experiments that aired on CNN.

Dr. Joshi is the author of several publications in the areas of software platforms, multisensor fusion, and computer vision, and evolutionary computing. He is the author of the book "Multisensor Fusion: A Minimal Representation Framework", a recipient of the best paper award at the 1996 Multisensor Fusion and Integration Conference, and a recipient of the Charles M. Close best doctoral thesis prize.

Dr. Joshi received his Ph.D and M.S. degrees in Computer and Systems Engineering from Rensselaer Polytechnic Institute, and a Bachelor of Technology in Electrical Engineering from the Indian Institute of Technology at Kanpur.

Dr. Joshi can be reached at 408-200-4754 or rajive.joshi@rti.com

GERARDO PARDO-CASTELLOTE, Ph.D. is the Chief Technology Officer at RTI. Dr. Pardo-Castellote specializes in Real-Time software architectures and networking.

His professional experience includes time-critical software for data acquisition and processing, real-time planning and scheduling software, control system software, and software-system design.

Dr. Pardo-Castellote actively participates in numerous conferences and standards-making organizations. He currently chairs the Data Distribution Group at the Object Management Group (OMG) and is the primary author of the DDS specification.

Dr. Pardo-Castellote received his Ph.D. in Electrical Engineering in 1995 from Stanford University. Dr. Pardo-Castellote also holds an MS in Computer Science and an MS in Electrical Engineering from Stanford University and a BS in Physics from the University of Granada (Spain).

Dr. Pardo-Castellote can be reached at 408-200-4751 or pardo@rti.com.