Efficient Communication between the Nodes Using Decentralized Data Distribution Service Communication Middleware

Harish.K¹, S.Rajeswari²

¹Student of M.Tech (Software Engineering), Department Of Information Science, New Horizon College of Engineering, India
²Assistant. Prof, Department Of Information Science, New Horizon College of Engineering, India

Abstract— In recent years there is a demand for the development of information management capabilities that ensure the right information is delivered to the right place at the right time to satisfy quality of service (QoS) requirements in heterogeneous environments. In general Middleware is computer software that acts as mediator between the existing applications to interact across the network. The Object Management Group’s DDS is emerged to address the data distribution requirements of critical mission specifically for Defence Community. Data Distributed Service (DDS) defines a service for efficiently distributing application data between participants in a distributed application. It provides real-time, manage connectivity required for device-to-device application. Unlike other technologies the DDS appreciates the phenomenon of decentralized broker-less architecture such that it enables seamless message or data sharing between producers and consumers. The idea of DDS is based on “Global Data Space” in which the producers write to data space and consumers read from data space [8]. The success is mainly due to the capacity of publish/subscribe to completely decouple communication participants since system applications would interact with data model not directly with each other. This paper shows how efficiently the messages can is shared between the publishers and subscribes without any delay brokers.

Keywords— Middleware, DDS, Publisher, Subscriber, QoS

I. INTRODUCTION

The Data Distribution Service (DDS) specification standardizes the software application programming interface (API) by which a distributed application can use “Data-Centric Publish-Subscribe” (DCPS) as a communication mechanism. Since DDS is implemented as an infrastructure solution, it can be added as the communication interface for any software application [3]. Publish-subscribe applications are typically distributed applications with endpoint nodes that communicate with each other by sending (publishing) data and receiving (subscribing) data anonymously.

Usually the only property a publisher needs in order to communicate with a subscriber is the name and definition of the data. The publisher does not need any information about the subscribers, and vice versa. The service is divided into two levels of interfaces: the Data-Centric Publish-Subscribe (DCPS) layer and an optional Data Local Reconstruction Layer (DLRL).

The DCPS layer transports data from the publishers to subscribers according to Quality of Service constraints associated with the data topic, publisher, and subscriber. The DLRL allows distributed data to be shared by local objects located remotely from each other as if the data were local. The DLRL is built on top of the DCPS layer.

II. MESSAGE-CENTRICITY AND DATA-CENTRICITY

There are several protocols emerged today for efficient communication such as Advanced Message Queuing Protocol (AMQP) model, Java message service (JMS) model and Message Queuing Telemetry Transport (AMQP) and Representational state transfer (REST).

All the above mentioned protocols come under message-centric technologies. In a message-centric approach the focus completely on deliver the message itself no matter what type of data payload it contains and the engineers would design the infrastructure such that message should reach the intended recipients.

In a data-centric approach the focus is completely on the user defined data or data model. The data value is the unit of exchange in this type of approach. The middleware understands the data content within the message and guarantees that all interested subscribers have a correct and consistent view of the data.

III. DDS GLOBAL DATA SPACE

The DDS of a strongly typed Global Data Space (GDS) where publisher and subscriber respectively write (produce) and read (consume) data. Subscriptions are dynamically matched and data flows from Publisher to Subscribers as it is shown in fig 1.

IV. DATA DISTRIBUTION SERVICE ENTITIES

A. Domain

The DDS applications can send and receive the data within the domain [2]. The domain is the fundamental partitioning unit within DCPS.
B. **DOMAIN PARTICIPANT**

A domain participant represents the participated applications in the communication module.

C. **TOPIC**

The topic is the fundamental unique key element and also considered as means of interaction between publishing and subscribing applications. Each topic determines the particular type of data flows from publisher to subscriber.

D. **DATA WRITER**

The *data writer* is used by the publishing application to enter the new values to the DDS. Each data writer writes the values always referenced to a particular topic.

E. **PUBLISHER**

Publisher is responsible for disperse the data of same or different data types. Fig 3 shows how the entities (Domain Participant, Topic, Data Writer, and Publisher) needed to publish data are connected.

![Publication Model](image)

E. **DATA READER**

The Data Reader object represents the subscriptions with reference to particular *Topic* and provides access to the data.

F. **SUBSCRIBER**

The Subscriber, the name itself suggests that it receives the published data and makes it available to domain participants. Fig 4 shows the entities associated with subscriptions. Once created and configured with the correct QoS, an application can be notified that data is available in one of three ways:

1) **Listener Call-back Routine:** In this method for accessing received data is to set up a listener call-back routine that DDS will run immediately when data is received.

2) **Polling the Data Reader:** In this method is to “poll” or query the Data Reader to determine if data is available.

3) **Conditions and Wait Sets:** In this the application waits until a specified condition is met and then accesses the data from the Data Reader.
V. QUALITY OF SERVICE (QOS) USED IN DDS

A. DURABILITY

The policy controls how data will be stored in Global Display Space and data availability with respect to late joiners. There are three variants in durability:

- **Volatile**: In this setting the past samples will not store.
- **Transient**: Only few samples are stored determined by history parameters.
- **Persistent**: All the samples are stored in non-volatile memory such as hard disks. These settings would allow the subscribers to join at any time.

B. DEADLINE

This policy indicates that the Data Writer must commit to send the data within the specified deadline period. Similarly the Data Reader commits to receive the data within the specified dead line period.

C. LATENCY_BUDGET

The policy determines the urgency associated with transmitted data. Its time starts from data written by the publisher until it is available for respective data readers.

D. OWNERSHIP

The policy controls which data writer should gain access to update the instance when there are multiple data-writers.

E. OWNERSHIP_STRENGTH

The policy is combined with OWNERSHIP policy. The data writer which has highest OWNER_STRENGTH value is owned to publish the data.

F. LIVELINESS

This policy controls the mechanism and parameters used by the Service to ensure that particular entities on the network are still “exists.”

G. PARTITION

This policy allows the introduction of a logical partition inside the ‘physical’ partition induced by a domain.
H. RELIABILITY

This policy indicates the level of reliability requested by a DataReader or offered by a DataWriter.

I. TRANSPORT_PRIORITY

The purpose of this QoS is to allow the application to take advantage of transports capable of sending messages with different priorities.

J. LIFESPAN

The policy determines expiry time beyond which the data writer should not deliver the data and the data readers will not read the values since they are too old.

K. HISTORY

The policy controls number of subsequent samples stored for readers as well as writers.

VI. SOFTWARE ARCHITECTURE

A. Application

Applications generally represent all heterogeneous applications built on different operating system and differ in architecture, network, Database, implementation languages, Operating System which are going to communicate by sending and receiving messages.

B. XML Transformation

Here the application is converted into XML messages. By converting native data to XML, messages are platform and language neutral which can be easily transported over network. System which receives the message can later on parse the XML format message to native data structure accordingly.
C. DDS Invocation

Application communicates by establishing call in following scenarios:

- From calling Application to the DDS and from DDS to Application requested in the request path.
- From requested Application Delegate to DDS and from DDS to requesting Application in the response path.

VII. IMPLEMENTATION AND PROOF OF CONCEPT

There are many types of software available but in this paper we will opt for Prism Tech’s OpenSpliceV5.410SS which offers exceptional intra-nodal data sharing performance using which middleware is developed ensuring there are no brokers between the publisher and subscriber.

Fig 6: Open splice started

Though the DDS is language independent the two nodes are implemented using VC++ (see fig 7) where left dialog box is Node-1 and Right dialog box is Node-2.

- The node-1 act as a publisher when it publishes the data and act as a subscriber when it receives the data, the sub menu consists of SERVER, TEST1, and TEST2. Here SERVER is a topic and remaining are two test cases TEST1 and TEST2 to publish the data.
- In the first test case i.e. selecting TEST1, the dialog box will displayed to enter personal information would be send to Global Display Space (GDS) (see fig 8).
- In the second case i.e. selecting TEST2, The dialog box displayed to send a file (Desert.jpg) to the Global Display Space (GDS).
- The Node 2 act as a subscriber when it receives the data and act as publisher when it publishes the data, the sub menu consists of SERVER, TEST1, and DISPLAY IMAGE. Here SERVER is a topic and remaining are two test cases (see fig 7).
- In the first test case i.e. selecting TEST1, the dialog box will displayed to enter personal information would be send to Global Display Space (GDS).
- In the second case i.e. selecting DISPLAY IMAGE, The dialog box displayed to check whether the image is received correctly from Global Display Space depending on intended topic.
A. Node-1 as publisher for Test-1

Once the data is published then immediately the data would not flow to subscribers (see fig 9) rather the subscriber read topics in the global data space for which a matching Subscription the topic, such as its type, its name, its key, its actual content. Once the Topic is matched then subscriber receives the data from the publisher and passes it to any relevant data readers that are connected to it.
Once the Topic (SERVER) is matched then subscriber receives the data from the publisher (here node 1) and passes it to any relevant data readers that are connected to it. The content XML file displayed in Node-2 (refer fig 10) ensures that the data is correctly received to intended subscriber and for reference another dialog is displayed to check whether the data sent by the publisher and received by the subscriber is the same or not.

Once the Topic (SERVER) is matched then subscriber receives the data from the publisher (here node 1) and passes it to any relevant data readers that are connected to it. The content XML file displayed in Node-2 (refer fig 10) ensures that the data is correctly received to intended subscriber and for reference another dialog is displayed to check whether the data sent by the publisher and received by the subscriber is the same or not.

B. Node-2 as subscriber for Test-1

C. NODE-1 as a Publisher for Test-2

- Select TEST2 option to publish the data (see fig 7).
- Browse the file (example: Desert.jpg) on send button to publish the data to GDS (refer fig 11).
D. **NODE-2 as a Subscriber which receive the file published by the NODE-1**

- The content XML file displayed in Node-2 (refer figure 12) ensures that the data is correctly received to intended subscriber.
- To cross check whether the data is received or not, select DISPLAY IMAGE (refer figure 13) test case in the NODE-2 so that a dialog box will be displayed, upon clicking **Display** button the image will be displayed suggesting that Node-2 has received the data properly.
Fig 13: Node-2 receiving Image file

E. NODE-2 as a Publisher

- Select TEST1 option from node-2 (refer fig 7) a dialog box will be displayed.
- Fill the personal details in the dialog and click on send button to publish the data in GDS (refer fig 14).
- As it is explained earlier the data directly will not flow unless the topic is matched. Once the Topic is matched Data flows from GDS to intended subscribers.
- The content XML file displayed in Node-1 ensures that the data is correctly received to intended subscriber (see fig 15).

Fig 14: Node-2 Publishing the data to GDS
F. NODE-1 as a Subscriber

G. NODE-1 as a Subscriber

VIII. CONCLUSION AND FUTURE WORKS

Publish-subscribe provide a data model that makes complex systems fundamentally simpler to model and understand. This design decouples the dataflow. If your fundamental problem is data distribution, the right solution is usually obviously DDS. It is designed specifically for high-performance data distribution, will greatly simplify your network design. As detailed above, the Data Distribution Model also provides high performance, fine QoS control, multicast when you need it, dynamic configuration, and connectivity with many transports and operating systems. If you need any or all of these things, you should strongly consider Data Distributed Service. Our future work will include (1) empirically evaluating a wider range of QoS configurations, e.g. durability, reliable vs. best-effort, and integration of durability, reliability and history depth, (2) designing mechanisms for migrating processing toward data sources, (3) measuring participant discovery time for various entities, (4) identifying scenarios that distinguish performance of QoS policies and features (e.g., collocation of applications), and (5) evaluating the suitability of DDS in heterogeneous dynamic environments, e.g., mobile ad hoc networks, where system resources are limited and dynamic topology and domain participant changes are common.
References


