International Journal of Scientific Research in Computer Science, Engineering and Information Technology
ISSN : 2456-3307 (www.ijsrcseit.com)
doi : https://doi.org/10.32628/CSEIT206432

DDS : A Solution to Network Centric Warfare

Isha Jakhar
INHS Asvini, Colaba, Mumbai, Maharashtra, India

Article Info
Volume 6, Issue 4
Page Number : 270-277
Publication Issue :
July-August-2020

Article History
Accepted : 25 July 2020
Published : 30 July 2020

ABSTRACT

The Open Management Group Data Distribution Service (OMG DDS) is a standard for publish-subscribe data distribution systems which is emerging as a specification for data exchange. It is a type of Message Oriented Middleware (MOM) that provides various functionalities such as portability and interoperability across many DDS implementations. In this, we create models which are platform independent. One platform can be mapped to others. These platforms can be in different programming languages. The existing issues with this technology include its inability to support request-reply services, file transfer and transaction processing. These issues can be considered as research work for future. Nowadays, there is always a need to exchange data among several communication machines instead of just a single machine. DDS caters this need by allowing data to be sent and received in a distributed environment. While doing so, the various security issues related to data integrity and loss of data are also taken into consideration. In this paper, Data Distribution Service (DDS) has been implemented to be used in network centric warfare, in three programming languages i.e Java, C and C++ in order to allow for cross language communication without any loss of data on a standalone mode .Data can also be exchanged between several machines in a distributed environment.

Keywords : DDS, MOM, data exchange, OMG

I. INTRODUCTION

OpenSplice is the most widely used implementation of the OMG DDS standard, both as Commercial as well as Open source, as far as completeness and advancement are concerned. OpenSplice makes it possible to share data and integrate it across different operating systems and different platforms. It provides a full implementation of both the OMG DDS latest rev1.4 (DCPS profiles) and the OMG-DDSI / RTPS v2.1 interoperable wire-protocol standards. It is designed to be used with server-class (desktops, racks etc.) platforms, real-time
embedded environments and operating systems.

OpenSplice DDS provides an infrastructure for real-time data distribution and offers middleware services to applications. The aims of real-time data distribution service are:

Supporting incremental development, providing a base infrastructure upon which fault-tolerant real-time systems can be often built, reducing the complexity of the real time systems, and deployment of systems.

DDS for Real-Time Systems is a type of Message Oriented Middleware (MOM) that supports a data-centric publish and subscribe style of communications. One of the major advantages of DDS is that it enables real-time, scalable, high performance, reliable and interoperable data exchanges between systems. DDS can be very effectively used for mission and business-critical applications such as traffic control, military command and control systems, SCADA & utilities, modelling and simulation, transportation management and other big data applications.

DDS is both language and Operating System (OS) independent. In order to ensure that DDS applications can be ported effectively among different vendor's implementations, there is a need to use standardized APIs.

1.1 DDS-DCPS OVERVIEW:

DDS provides a model for efficiently distributing useful data among several participants in a distributed scenario. This model consists of two levels of interfaces(layers): the Data-Centric Publish-Subscribe (DCPS) layer and a Data Local Reconstruction Layer (DLRL), which is optional. The DCPS layer is responsible for transporting data from publishers to the respective subscribers, efficiently taking into consideration the Quality of Service constraints associated with the data topic(on which the work is being done), publisher, and the subscriber. The DLRL allows this distributed data to be shared among local objects located remotely from each other as if that data were local. The DCPS layer is built under the DLRL.

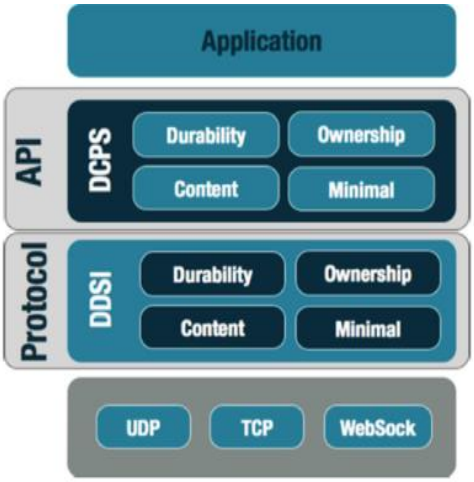

Figure 1 DDS-DCPS Overview

1.2 DDS vs CLIENT-SERVER ARCHITECTURE

CLIENT SERVER:

In client-server architectures, we have clients (the chatters) that connect to a server (the chat room) and identify themselves by giving their user name. Authentication is done by using passwords. After successful authentication, unlimited chat messages can be sent by the client. The chat room collects the chat messages of each client and will forward them to all other participating clients. New clients can request to join a chat room at any moment in time: they will then have to identify themselves to the server, and the server will make sure that all chat messages received from that moment on will also be forwarded to the newly added client.

The server is the single point of failure. If it fails, all chatter applications get disconnected.

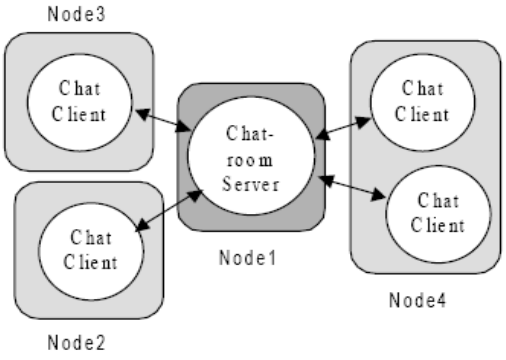

Figure 2 Client Server Approach

DDS-BASED APPROACH:

The idea here is to remove the chat room server altogether and let the chat applications directly communicate with each other. The architecture will then become less centralized. Now, all the applications are equal; there is no centralized point of failure. If one node crashes, all the Chatters associated with that node also die, but all the other nodes and chatters are up and can keep communicating normally with each other. Every chat message only has to be transmitted over the network once to deliver it to all the other interested Chatters. Adding more Chatter applications does not use any extra bandwidth, except for the messages sent by these newly added entities.

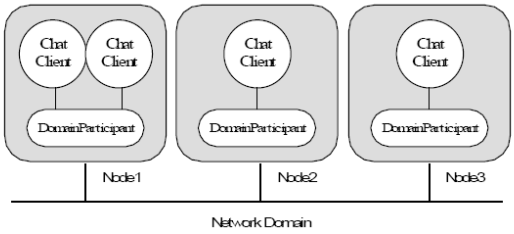

Figure 3 DDS Approach

1.3 DDS PARTICIPANTS:

In DCPS, applications must use APIs to create entities (objects) in order to establish publish-subscribe communication between each other. In object-oriented terms, *Entity* is the base class from which other DCPS classes Topic, DataWriter, DataReader,Publisher, Subscriber, Doma in Participants derive. The sending side uses objects called Publishers and DataWriters. The receiving side uses objects called Subscribers and DataReaders.

An application uses DataWriters to send data. A DataWriter is associated with a single Topic. You can have multiple DataWriters and Topics in a single application. A Publisher is the DCPS object responsible for the actual sending of data. Publishers own and manage DataWriters. A DataWriter can only be owned by a single Publisher while a Publisher can own many DataWriters. An application uses DataReaders to access data received over DCPS. A DataReader is associated with a single Topic. You can have multiple DataReaders and Topics in a single application. A Subscriber is the DCPS object responsible for the actual receipt of published data. Subscribers own and manage DataReaders. A DataReader can only be owned by a single Subscriber while a Subscriber can own many DataReaders.

DomainParticipants are the focal point for creating, destroying, and managing other DDS objects. A DDS domain is a logical network of applications: only applications that belong to the same DDS domain may communicate using DDS. A DDS domain is identified by a unique integer value known as a domain ID. In order to participate in a DDS domain, the application has to create a DomainParticipant first, for that domain ID.

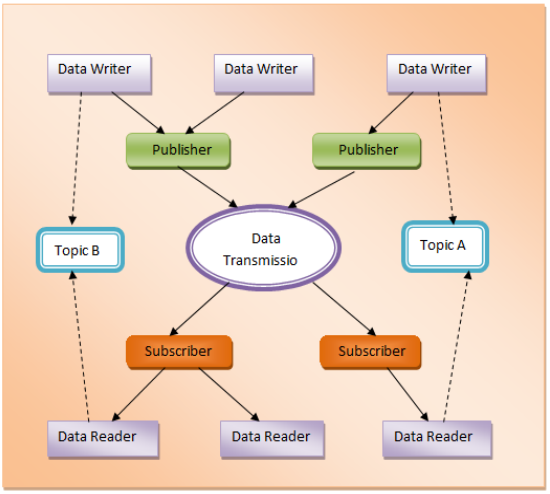

Figure 4 DDS Architecture

The main purpose of a DomainParticipantFactory is to create and destroy DomainParticipants. For a DataWriter and DataReader to communicate with each other, it is necessary for them to be working on the same Topic. A Topic includes a name and an association with a user data type that has been registered with DDS. Each topic has a unique topic

name. These Topic names are important as this is how different parts of the communication system locate or find each other.

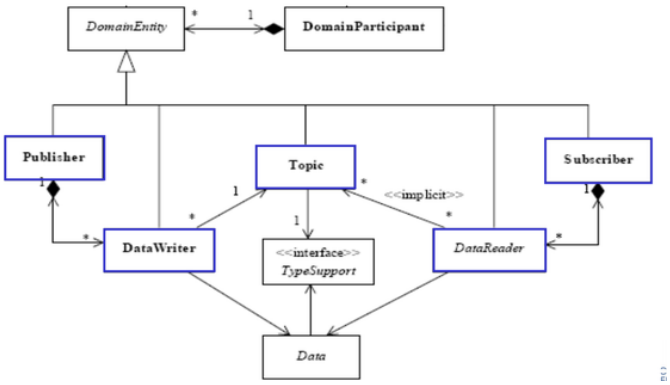

Figure 5 Flow Of Data In DDS

In this paper, such a model has been used to establish communication between various warfare entities in order to help them interact with each other, irrespective of the platform on which the system is working on. Current implementation is done on a standalone PC and can be further extended to distributed systems. This experiment was done in three languages C, C++ and JAVA .The objective behind using three languages is to enable cross language communication .For example, if one machine runs C and another machine runs JAVA, then they will be able to connect with each other and communicate, irrespective of their being a difference in platforms.

II. IMPLEMENTATION

A data type represents a structured data type, like an IDL **struct** with several members and a keylist. When we read or write topics, we actually read or write samples of a specific data type. The definition of each data type you will be using has to be written in (a subset of) OMG IDL. To invoke the IDL, Create a file named filename.idl. Insert the IDL definition into this file. Run the IDL pre-processor from the command line.

We start the software and start implementing the code in three languages. First, the interfaces (publishers and subscribers) are allowed to interact with each only on single platform(either both work on C or both work on C++ or both on Java). The output for java is attached.

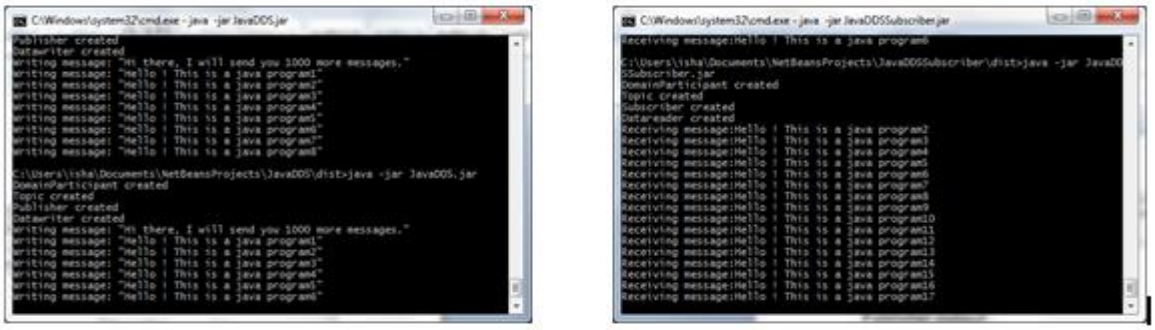

Figure 6

Then, one platform is allowed to interact with other subscriber platforms in order to check if the information is being communicated between systems operating on different platforms in a desired manner. After that, for ensuring privacy of the data, the message, before being transmitted over the network is encrypted using the desired encryption algorithm and then, decrypted on the receiver side after the message is successfully received.

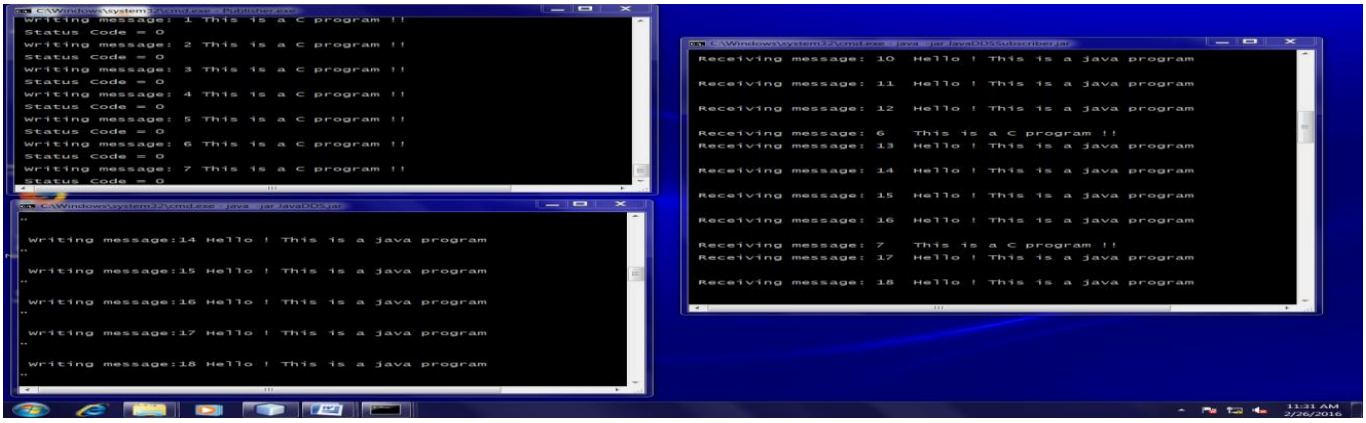

Figure 7

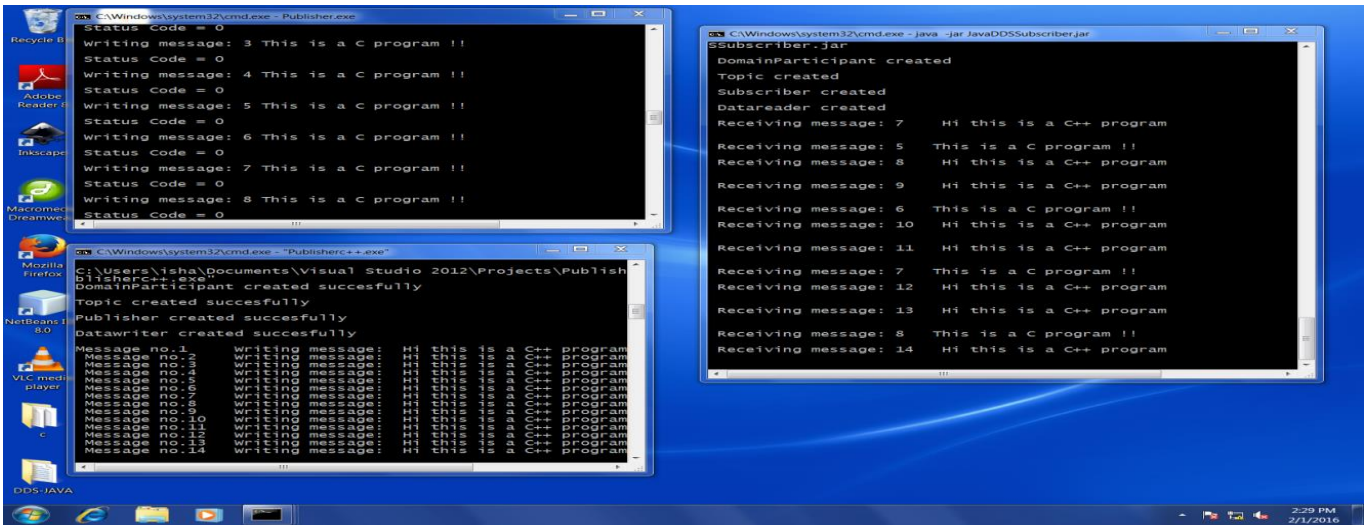

Figure 8

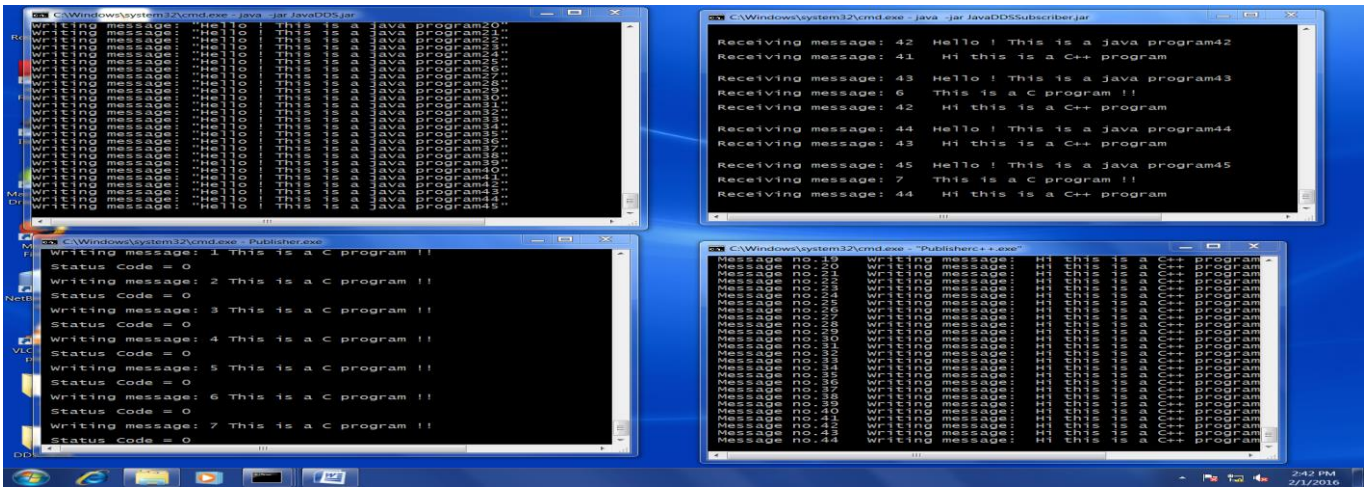

Figure 9

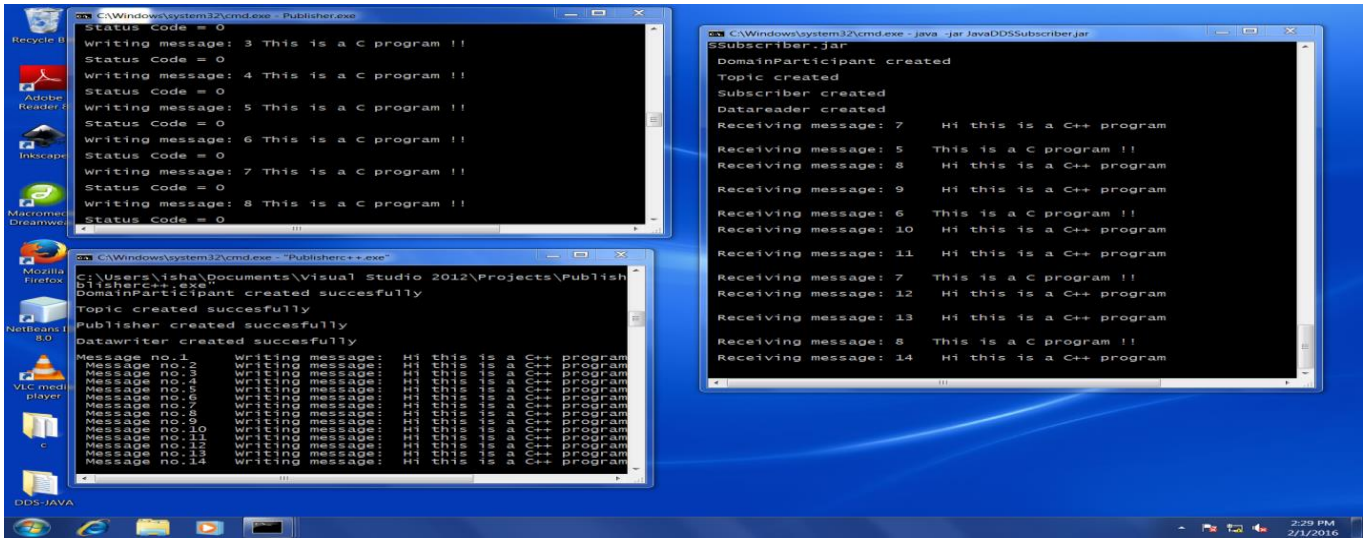

Figure 10

After successfully implementing the DDS code on command prompt, GUI was designed to make it more user friendly and appealing to the user working on it.
It has publish and subscribe buttons, fields to enter DomainID, content and topic name to identify the system over the network. Once these fields are entered, the user can write the message and send it over the network successfully.

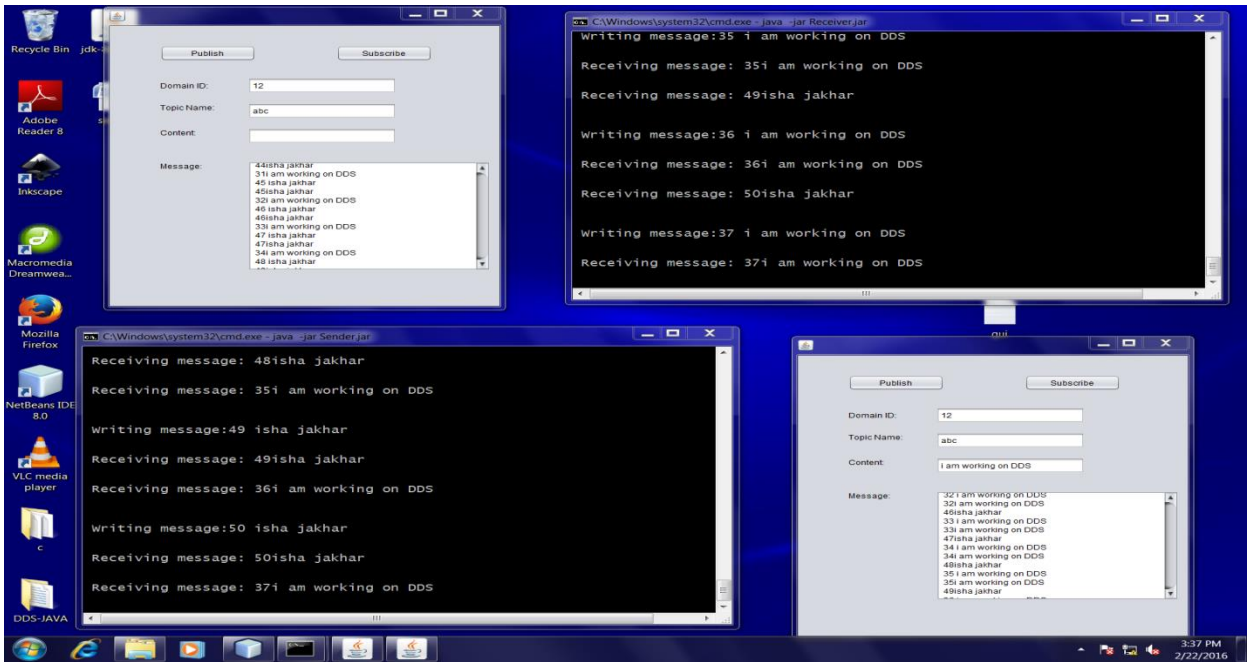

Figure 11

At last, encryption is performed to ensure privacy and security of the data.

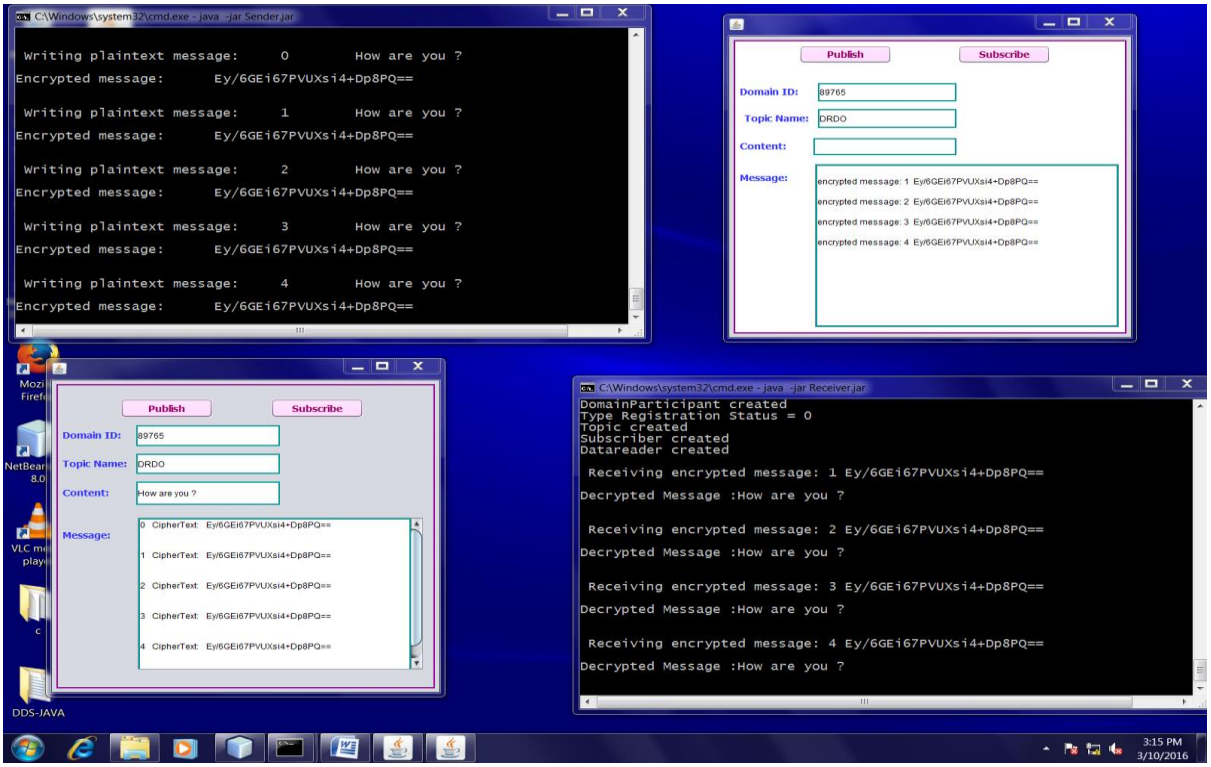

Figure 12

III. RESULTS

The project implemented worked correctly on a single platform as well as multiple platforms. No communication gap or defect was found on systems communicating on different platforms. For encryption, Blowfish algorithm was used which proved to be successful in encryption and decryption of the message. DDS has an added feature of automatic discovery. Irrespective of their locations, one system can locate any other system over the network very easily. Since, DDS has gained popularity over the past few years; soon it will be implemented on a full basis in the modern warfare.

IV. CONCLUSIONS AND FUTURE WORK

In this project, we have explored the various functionalities that DDS can provide. The DDS participants in DDS domains that communicate with each other, were studied in detail to understand more about the Data Distribution Service. Several comparisons were made between DDS and other technologies as to highlight the advantages of DDS over those technologies due to which we chose DDS to work with.

Future work will look into how to provide QoS guarantee in wide area networks using advanced infrastructure for Next Generation Network architecture that builds, uses and manages end-to-end QoS across different administrative domains and heterogeneous networks. For this, efficient work is being done on an interoperable DDS security specification which will be able to cater many security issues such as data integrity, confidentiality, availability and assurance.

V. REFERENCES

[1]. OMG Data-Distribution Service: Architectural Overview Gerardo Pardo-Castellote, Ph.D. Real-Time Innovations, Inc. gerardo@rti.com

[2]. IJCSNS International Journal of Computer Science and Network Security, VOL.7 No.1, January 2007 313 Manuscript received January 5, 2007. Manuscript revised January 25, 2007. Performance of a Publish/Subscribe Middleware for the Real Time Distributed Control systems, Mohamed Anis MASTOURI and Salem HASNAOUI SYSCOM Laboratory, National School of Engineering of Tunis TUNISIA.

[3]. www.rti.com/docs/Comparison-Mapping-DDS-HLA.pdf.

[4]. Addressing the Challenge of Distributed Interactive Simulation With Data Distribution Service Akram HAKIRI 1, 2, Pascal BERTHOU1, 2, Thierry GAYRAUD1,2 1 CNRS ; LAAS, 7, avenue du Colonel Roche, 31077 Toulouse, France 2 Université Toulouse; UPS, INSA, INP, ISAE; LAAS; F-31077 Toulouse, France Email: {Hakiri, Berthou, Gayraud}@laas.fr.

[5]. Mapping different communication traffic over DDS in industrial environments , Dept. of Comput. Languages & Syst., Univ. of the Basque Country, Vitoria-Gasteiz, Spain ; I. Calvo ; F. Pérez

[6]. DDSS: A Communication Middleware based on the DDS for Mobile and Pervasive Systems , K .J Kown , C.B Park , H.Choi.

[7]. IJCSNS International Journal of Computer Science and Network Security, VOL.7 No.1, January 2007, Performance of a Publish/Subscribe Middleware for the Real-Time Distributed Control systems, Mohamed Anis mastouri and Salem hasnaoui,syscom Laboratory, National School of Engineering of Tunis TUNISIA.

Cite this article as :

Isha Jakhar, "DDS : A Solution to Network Centric Warfare", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 6 Issue 4, pp. 270-277, July-August 2020. Available at doi : https://doi.org/10.32628/CSEIT206432
Journal URL : http://ijsrcseit.com/CSEIT206432