DATA SHARING AND ACCESS
WITH A
CORBA DATA DISTRIBUTION SERVICE IMPLEMENTATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

MUSTAFA DURSUN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2006

Approval of the Graduate School of Natural and Applied Sciences.

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. İsmet Erkmen
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof. Dr. Semih Bilgen
Supervisor

Examining Committee Members

Prof. Dr. Hasan Güran (Chairman) (METU, EEE) ____________________

Prof. Dr. Semih Bilgen (METU, EEE) ____________________

Asst. Prof. Dr. Cüneyt Bazlamaçcı (METU, EEE) ____________________

Dr. Ece Schmidt (METU, EEE) ____________________

Ali Özzeybek (M.S.) (ASELSAN Inc.) ____________________

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Mustafa Dursun

Signature:

ABSTRACT

DATA SHARING AND ACCESS WITH A CORBA DATA DISTRIBUTION SERVICE IMPLEMENTATION

DURSUN Mustafa,
M.Sc., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Semih BİLGEN

September 2006, 54 pages

Data Distribution Service (DDS) specification defines an API for Data-Centric Publish-Subscribe (DCPS) model to achieve efficient data distribution in distributed computing environments. Lack of definition of interoperability architecture in DDS specification obstructs data distribution between different and heterogeneous DDS implementations. In this thesis, DDS is implemented as a CORBA service to achieve interoperability and a QoS policy is proposed for faster data distribution with CORBA features.

Keywords: Middleware, Data Distribution Service (DDS)**,** CORBA, interoperability

ÖZ

BİR CORBA VERİ DAĞITIM HİZMETİ GERÇEKLEŞTİRİMİYLE VERİ PAYLAŞIMI VE ULAŞIMI

DURSUN Mustafa
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Danışmanı: Prof. Dr. Semih BİLGEN

Eylül 2006, 54 sayfa

Veri Dağıtım Servisi (DDS) Spesifikasyonu dağıtımlı bilgi işleme uygulamaları için Veri-Merkezli Yayıncı-Abone (DCPS) modeli kapsamında bir API tanımlamaktadır. DDS Spesifikasyonu'nda birlikte çalışabilirlik mimarisinin tanımlanmamış olması değişik ve heterojen DDS uygulamaları arasındaki veri dağıtımına engel oluşturmaktadır. Bu tezde, birlikte çalışabilirliği sağlamak için DDS CORBA servisi olarak uygulanmış ve CORBA araçları ile veri dağıtımını daha hızlı yapabilmek için bir QoS politikası önerilmiştir.

Anahtar Kelimeler: Arakatman, Veri Dağıtım Servisi (DDS), CORBA, birlikte çalışabilirlik

To my mother…

ACKNOWLEDGMENTS

I would like to express my gratitude to Prof. Dr. Semih Bilgen for his understanding, patience and supervision throughout this thesis. This thesis would not have been completed without his guidance.

I would like to thank ASELSAN Inc. for understanding and support for academic studies. I also want to thank my colleagues at ASELSAN Inc. for their valuable support throughout this study.

Finally, I wish to thank my mother for everything she does.

TABLE OF CONTENTS

ABSTRACT iv
ÖZ v
ACKNOWLEDGMENTS vii
TABLE OF CONTENTS viii
LIST OF TABLES x
LIST OF FIGURES xi
LIST OF ACRONYMS xiii
CHAPTERS
1. INTRODUCTION 1
2. LITERATURE SURVEY 4
2.1 Middleware Architecture Models 4
2.1.1 Client-Server Model 5
2.1.2 Publish-Subscriber Model 6
2.2 Data Distribution Service (DDS) 8
2.2.1 Overall Conceptual Design [8] 9
2.2.2 Information Flow 11
2.2.3 DDS QoS Policies 16
2.2.4 DDS Profiles 16
2.2.5 Open-Source DDS Applications 17
3. CORBA DDS (CDDS) IMPLEMENTATION 21
3.1 CDDS Participants 22
3.1.1 cDomainParticipantFactory 23
3.1.2 cDomainParticipant 23
3.1.3 cPublisher 23
3.1.4 cDataWriter 23

3.1.5 cSubscriber 23
3.1.6 cDataReader 24
3.1.7 cSampleInfo 24
3.1.8 cTopic 24
3.1.9 cDataReaderListener 24
3.1.10 cMediator 24
3.1.11 cDataHandler 26
3.2 CDDS Deployment 27
3.3 CDDS Data Dissemination Mechanism 28
3.3.1 Creating Factory Participants 29
3.3.2 Subscription 31
3.3.3 Publication 32
3.3.4 Data Notification 32
3.3.5 Data Reception 33
3.4 A New QoS Policy: BLOCK 34
3.5 CDDS Type-Specific Classes 35
4. EVALUATION 36
4.1 Experiment Overview 37
4.1.1 Experiment 1 37
4.1.2 Experiment 2 40
4.1.3 Experiment 3 42
4.1.4 Experiment 4 47
5. DISCUSSION AND CONCLUSION 50
REFERENCES 52

LIST OF TABLES

Table 1 Middleware Architecture Model Comparison 4
Table 2 Experiment 1 Testbed Properties 37
Table 3 Roundtrip time of CDDS and JacORB DDS (μs) 38
Table 4 Experiment 2 Testbed Properties 40
Table 5 Roundtrip time of CDDS in heterogeneous distributed environment (μs) .. 41
Table 6 Experiment 3.a Testbed Properties 43
Table 7 Roundtrip time of CDDS for different number of producers (μs) 44
Table 8 Experiment 3.b Testbed Properties 45
Table 9 Roundtrip time of CDDS for different number of producers (μs) 46
Table 10 Roundtrip time of CDDS for different number of blocksize (μs) 47

LIST OF FIGURES

Figure 1 Conceptual Design of DCPS Layer [8] 10
Figure 2 Information Flow Overview of DDS [8] 12
Figure 3 DDS Publication Sequence Diagram [8] 13
Figure 4 DDS Subscription with Listener Sequence Diagram [8] 14
Figure 5 DDS Subscription with Condition Sequence Diagram [8] 15
Figure 6 JacORB 2.2.3 Event Service Model [10] 19
Figure 7 JacORB 2.2.3 DDS Implementation Event Service Mechanism [20] 19
Figure 8 CDDS DCPS Entities Model Diagram 22
Figure 9 Mediator Interface Specification 25
Figure 10 DataHandler Interface Specification 26
Figure 11 CDDS Participants Deployment Diagram 28
Figure 12 Creating cPublisher Sequence Diagram 29
Figure 13 Creating cSubscriber Sequence Diagram 30
Figure 14 Creating cTopic Sequence Diagram 30
Figure 15 Subscription Sequence Diagram 31
Figure 16 Publication Sequence diagram 32
Figure 17 Data Notification Sequence diagram 33
Figure 18 Data Reception Sequence diagram 34
Figure 19 Testbed Configuration 36
Figure 20 Experiment 1 Testbed Configuration 38
Figure 21 JacORB DDS versus CDDS performance 39
Figure 22 Throughput of CDDS 40
Figure 23 Experiment 2 Testbed Configuration 41
Figure 24 Performance of CDDS in heterogeneous distributed environment 42
Figure 25 Experiment 3.a Testbed Configuration 43
Figure 26 Performance of CDDS for different number of consumers 44

Figure 27 Experiment 3.b Testbed Configuration 45
Figure 28 Performance of CDDS for different number of producers........................ 46
Figure 29 Performance of CDDS for different data volumes.................................... 48
Figure 30 Performance of CDDS for different blocksize values............................... 49

LIST OF ACRONYMS

API: Application Programming Interface

CDDS: CORBA Data Distribution Service

CDR: Common Data Representation

CORBA: Common Object Request Broker Architecture

CPU: Central Processor Unit

DCOM: Distributed Component Object Model

DCPS: Data-Centric Publish-Subscribe

DDS: Data Distribution Service

DLRL: Data Local Reconstruction Layer

DOC: Distributed Object Computing

GIOP: General Inter ORB Protocol

HTTP: Hypertext Transfer Protocol Overview

IDL: Interface Definition Language

IIOP: Internet Inter ORB Protocol

JMS: Java Message Service

OBV: Object-By-Value

OMG: Object Management Group

ORB: Object Request Broker

OS: Operating System

RPC: Remote Procedure Call

TAO: The ACE ORB

CHAPTER I

INTRODUCTION

Performance and usage requirements force information systems developers to use distributed systems. This means that information processing applications are distributed on separate systems over a network. Separate systems may use different types of hardware platforms and operating systems, while applications may be implemented in different programming languages over a heterogeneous network. The variety of systems requires a distributed system to use software, called middleware [1], to implement applications independent of platform, operating system, and programming language. Middleware provides simplicity and uniformity for the development.

For different distributed system requirements, different approaches are developed. These approaches, broadly classified as service-oriented [2] and message-oriented [1], are shaped according to the need for distributing service accessibility and availability on one hand, and distributing messages, on the other. In service-oriented approach middleware provides remote procedure call (RPC) or distributing object references. In message-oriented approach middleware focuses on distributing messages. Approaches for middleware are tied to middleware architectures in the phase of implementation. Client-Server and publish-subscribe architecture models are the main middleware architecture models.

Common Object Request Broker Architecture (CORBA) [3] is an Object Management Group (OMG) [4] standard for distributed object computing (DOC). CORBA is structured to allow integration of a wide variety of object systems [3]. CORBA provides predictability of real-time systems, but it causes a high overhead for data distribution [5]. CORBA meets interoperability requirement by specifying General Inter ORB Protocol (GIOP) with the Common Data Representation (CDR)

transfer syntax, and Internet Inter ORB Protocol (IIOP). OMG published CORBA Event Service [6] and CORBA Notification Service [7] CORBA service specifications to enhance CORBA with publish-subscribe model properties.

A new specification, Data Distribution Service (DDS) for Real-Time Systems Specification [8], published by OMG standardizes the software API for publish-subscribe model while adding data-centric property to publish-subscribe model. This specification defines interface classes for API, but does not specify any marshalling and protocol for interoperability between different DDS implementations. DDS inherits decoupling properties of publish-subscribe model and achieves a predictable data-centric distributed system by controlling Quality of Service (QoS) of each data object in the system. To achieve a predictable data-centric distributed system, DDS presents a new middleware architecture model: Data-Centric Publish-Subscriber (DCPS) model. DDS aims to minimize overhead of data in a distributed system while increasing scalability.

The objective of this work is to implement DDS specification by using CORBA features. The aim of using CORBA features is to transport interoperability to DDS to work in a heterogeneous distributed environment. On the other hand, implementing DDS by using CORBA features transports decoupling and data-centric properties of DDS to CORBA. Minimizing CORBA overhead in data distribution by applying a new QoS policy that we call BLOCK QoS policy, is another aim of this work.

The scope of this study consists of implementing the DDS Specification as a CORBA service that achieves interoperability while enabling high scalability. As a CORBA service, implementation aims to be more efficient and compatible with DDS specification data distribution from JacORB DDS implementation. Implementation is working towards the Minimum Profile, but provides architecture to implement Ownership and Persistence Profiles

Along with the details of the proposed implementation itself, its evaluation is also presented in the thesis. In order to evaluate the implementation an experiment environment is prepared. Performance, interoperability and scalability properties of the implementation are examined within the framework of four experiments. All experiments are carried out on an Ethernet infrastructure with no background traffic. This is expected to reflect the typical practical environments in which such applications are usually implemented. In Experiment 1 performance of implementation is examined with increasing data volume in a homogenous distributed environment. In Experiment 2 interoperability of implementation is examined in a heterogeneous distributed environment. In Experiment 3 scalability of implementation is examined with increasing number of producer and consumer applications in a homogenous distributed environment. Finally in Experiment 4 effect of BLOCK QoS policy in performance is examined with increasing data volume in a homogenous distributed environment.

The remainder of the thesis is organized as follows:
In Chapter 2, a survey of literature on distributed software system approaches and middleware architectures is presented. Specifically, CORBA and DDS implementations reported in the literature are reviewed.

Chapter 3 describes the features of the DDS implementation realized in this study. First, description of implemented DDS participants and their relationship are presented. Then, data dissemination mechanism of implementation is explained with the help of message sequence diagrams. Finally, BLOCK QoS policy is described in this chapter.

Chapter 4 presents an evaluation of the constructed software infrastructure. The evaluation focuses on performance, scalability, and interoperability analysis of the implementation.

Finally, Chapter 5 concludes the work by presenting the achievements and shortcomings of the study as well as suggestions for future work on the subject.

CHAPTER II

LITERATURE SURVEY

2.1 Middleware Architecture Models

We may classify middleware architecture models as client-server model, publish-subscribe model, and data-centric publish-subscribe model. Each of these models supports different network approaches, service-oriented approach and message-oriented approach. The cause of variety in network approaches is different system requirements. System requirements may be classified as performance, reliability, scalability, and interoperability [12]. Table 1 presents a comparison between architecture models.

Table 1 Middleware Architecture Model Comparison

Property	***Client – Server Model***	***Publish – Subscribe Model***
Network Approach	Service-Oriented	Message-Oriented
Data Distribution	Server to Client	Many to many nodes
Decoupling	Strongly Synchronized	Time, Space, Flow Decoupling
Data Units	Replies of Methods	Messages
Abstraction of Network	Connection-Oriented	Connectionless
Obligation of Applications	Data Dissemination and Process	Data Process

The client-server model is a reliable model because of its connection-oriented and strongly synchronized origin. On the other hand the connection-oriented and strongly synchronized origin of client-server model limits the scalability in data distribution [21]. Interoperability is provided by implementation of client-server model such as CORBA, and DCOM. Publish-subscribe model achieves reliability in spite of its connectionless origin. The most important property of publish-subscribe model is decoupling between publishers and subscribers in time, space, and flow [14]. Decoupling property of this model causes high scalability and fault-tolerance in a distributed system. Publish-subscribe model is far from meeting the interoperability needs, because marshalling is not defined by any publish-subscribe specification [1].

2.1.1 Client-Server Model

Client-Server model [22] supports the service-oriented approach. This model includes servers, managing services, and clients, requesting service. Server objects are invoked remotely by clients through stubs. In classical Client-Server model clients and servers are strongly synchronized. Synchronous communication in Client-Server model leads to leads to static distributed system.

Client-Server model achieves data distribution by requests and replies. Client-Server model works well for systems with centralized information, synchronous transactions, and large size replies. If multiple nodes are generating data in network, Client-Server model requires all data to be sent to the server. Such indirect client-to-client communication is inefficient [22].

Most Client-Server middleware designs present an Application Programmer Interface (API) that strives to make the remote node appear to be local. Successful Client-Server middleware designs include CORBA, used in developing object oriented distributed applications, DCOM, developed by IBM corporation for component based distributed applications and HTTP, used universally in web-based communications, and Enterprise Java Beans (EJB) [23].

CORBA is a standard published by OMG [4]. The heart of CORBA is Object Request Broker (ORB) that allows clients to invoke operations on distributed objects without concerns of object location, programming language, operating system, communication protocols, and hardware. CORBA specification enables interoperability between ORB implementations regardless of programming language by supporting CDR (Common Data Representation) and IIOP (Internet Inter ORB Protocol). CORBA specification defines a distributed object as an instance of IDL (Interface Definition Language). Objects are identified by object references. Clients possess object references identifying objects and make remote method calls. A standard CORBA client request results in the synchronous execution of an operation by an object in server. CORBA 3.0 supports both synchronous and asynchronous method calls. TAO [11] and JacORB [10] are among the most widely used open-source CORBA implementations.

2.1.2 Publish-Subscriber Model

Publish-subscribe model [22] supports message-oriented approach. Model includes publishers, publishing messages, subscribers, expressing an interest in messages (subscription), and message service, providing storage and management for subscriptions and efficient delivery of messages.

Publish-subscribe model enables a subscriber to express its interest in messages by different ways. Channel-Based subscription scheme is the most basic scheme. Communication by using Channel-Based subscription scheme is similar to notion of group communication [13]. Subscribers subscribe to a message channel, a message service, and receive all messages published to channel. Topic-Based subscription scheme is based on notion of subjects (topics). Subscribers subscribe to unique topics which are identified by keywords. Topic-based subscription scheme introduces an abstraction mapping of each individual topic to distinct message channels. Content-Based subscription scheme improves topic-based subscription scheme by introducing a subscription scheme based on content of messages. In other terms, messages are not classified according to some predefined keywords, but according to the properties of the messages themselves [13]. Type-

Based subscription scheme [14] usually regroup messages that present commonalities not only in content, but also in structure such as topic type that is defined in object oriented programming languages. In this scheme messages are considered as objects. [15]

Publish-subscribe model provides decoupling between publishers and subscribers in three dimensions; space, time, and flow. The interacting participants do not need to know each other, in other words they do not need to hold any references, do not need to be actively participating in the interaction at the same time, and data reception and data sending do not block participants. [14]

Publish-Subscribe model achieves data distribution by using messages. Publish – Subscribe model works well for systems that information is distributed on fault-tolerant and time-critical networks. While model decouples participants, it enables participants to send and receive messages instead of accessing object states. This means data units that form messages have to be built by applications.

Publish-Subscribe Model is deprived of a public API. Most known Publish-Subscribe implementations are CORBA Event Service [6], CORBA Notification Service [7], and Java Message Service (JMS) [17].

2.1.2.1 CORBA Event Service

In the scenarios that client and server must be more decoupled, standard CORBA becomes inadequate because of its client-server model. CORBA Event Service [6] decouples the communication between applications using CORBA by implementing Publish-Subscribe Model. CORBA Event Service defines Publisher as Supplier, Subscriber as Consumer, and Message Service as Event Channel. Event Channel is defined as standard CORBA objects and communication with an Event Channel is accomplished by using standard CORBA requests. CORBA Event Service transmits events by the help of Any feature of CORBA [3] .

2.1.2.2 CORBA Notification Service

CORBA Notification Service [7] extends the CORBA Event Service by adding new capabilities. CORBA Notification Service transmits events in the form of a well-defined data structure, in addition to Anys as supported by the existing Event Service. This makes Consumers to specify exactly which events they are interested in receiving.

2.1.2.3 Java Message Service

JMS [17] is the Java implementation of Publish-Subscribe model. JMS defines Publisher, and Subscriber as JMS Clients, and Message Service as Topics. Publishers and subscribers are active when the Java objects that represent them exist. JMS supports time decoupling.

2.2 Data Distribution Service (DDS)

DDS specification [8] standardizes the software API by which a distributed application can use publish-subscribe model as a middleware model, and extends publish-subscribe model to data-centric publish-subscribe model. DDS specification only defines interfaces between application and service. It does not address protocols and techniques for different actors implementing the service, does not address management of internal DDS resources, and does not support interoperability between DDS implementations.

Data-Centric Publish-Subscribe (DCPS) model [8] supports information-oriented approach. This model includes publisher writing data to data objects, subscriber subscribing to data objects, and reading data from data objects, and Global Data Space keeping data objects.

DCPS model uses high level data-models instead of exchanging object references or elementary data units like messages [8]. The data model defines the Global Data Space and specifies how Publishers and Subscribers refer to portions of this space. DCPS model provides the ability to configure QoS of system for each data object by using QoS polices.

NDDS by RTI and SPLICE by Thales are the most mature commercial products of DCPS model. These products have been the initiator of a public API of DCPS Model, DDS (Data Distribution Service) Specification [8] that is published by OMG in December 2005.

DDS specification describes two levels of interfaces; DCPS Level, and DLRL (Data Local Reconstruction Layer) Level. DCPS Level is targeted towards the efficient delivery of the proper information to the proper recipients. DLRL Level, which is optional, allows for a simple integration of the Service into the application layer.

2.2.1 Overall Conceptual Design [8]

DCPS entities, DomainEntity, DomainParticipant, Publisher, DataWriter, Subscriber, DataReader, and Topic, are the main communication objects. All these communication entities support QoS policy, accept a Listener and a StatusCondition. Relations of DCPS entities can be seen in Figure 1.

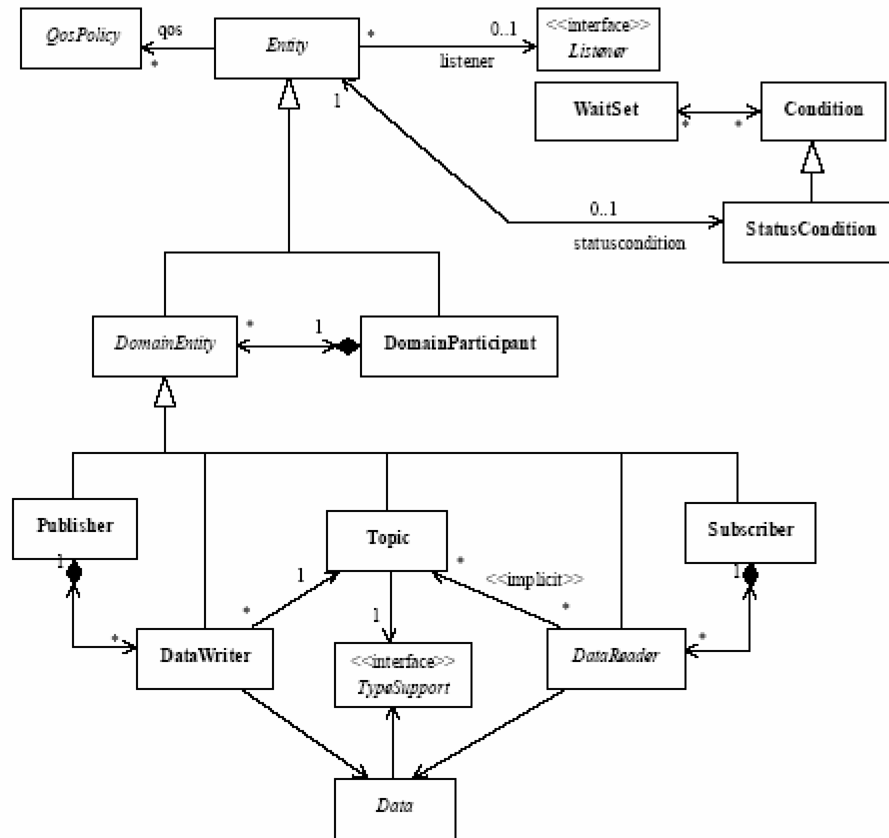

Figure 1 Conceptual Design of DCPS Layer [8]

DomainParticipant is the entry point for the service and represents the local membership of the application in a domain. A domain represents Global Data Space component of DCPS model. Domain is a distributed concept that links all the applications able to communicate with each other. Global Data Space in DCPS model is represented by a domain in DDS. DomainEntity is the abstract base class for all DCPS entities, except for the DomainParticipant.

A Publisher is the object responsible for the actual dissemination of publications. A publication is defined by the association of a DataWriter to a Publisher. A DataWriter is the object that allows the application to set the value of the data to be published under a given Topic.

A Subscriber is the object responsible for the actual reception of the data resulting from its subscriptions. It may receive and dispatch data of different specified types. To access the received data, the application must use a typed DataReader attached to the subscriber. A DataReader is the object that allows the application to make a subscription and to access the data received by the attached Subscriber.

Topic is the most basic description of the data to be published and subscribed. A Topic is identified by its name, which must be unique in the whole Domain. In addition it fully specifies the type of the data that can be communicated when publishing or subscribing to the Topic.

Each DCPS entity specifies its own QoS (Quality of Service). A QoS is a set of characteristics that controls some aspect of the behavior of the DDS Service by specifying the degree of coupling between the participants. QoS is comprised of individual QoS policies.

Listeners and StatusConditions are two alternative mechanisms that allow the application to be made aware of changes in the DCPS communication status, such as arrival of data corresponding to a subscription, and violation of a QoS setting.

2.2.2 Information Flow

Information flow [8] is achieved with the aid of Publisher, DataWriter, Subscriber, and DataReader entities as shown in Figure 2. Publisher object is responsible for data distribution, while Subscriber object is responsible for receiving distributed data. A data-object is described by Topic. Topic entity associates a name, a data-type, QoS related to a data-object.

Information flow behavior of publisher side is controlled by QoS of Topic defining the publishing data, QoS of DataWriter associated with Topic, and QoS of Publisher associated with DataWriter. Information flow behavior of subscriber side is controlled by QoS of Topic defining the subscribed data, QoS of DataReader

associated with Topic, and QoS of Subscriber associated with DataReader. In several cases, for information flow QoS policy on the publisher side must be compatible with QoS policy in the subscriber side. If the policies are incompatible the service does not establish communication between publisher and subscriber entities.

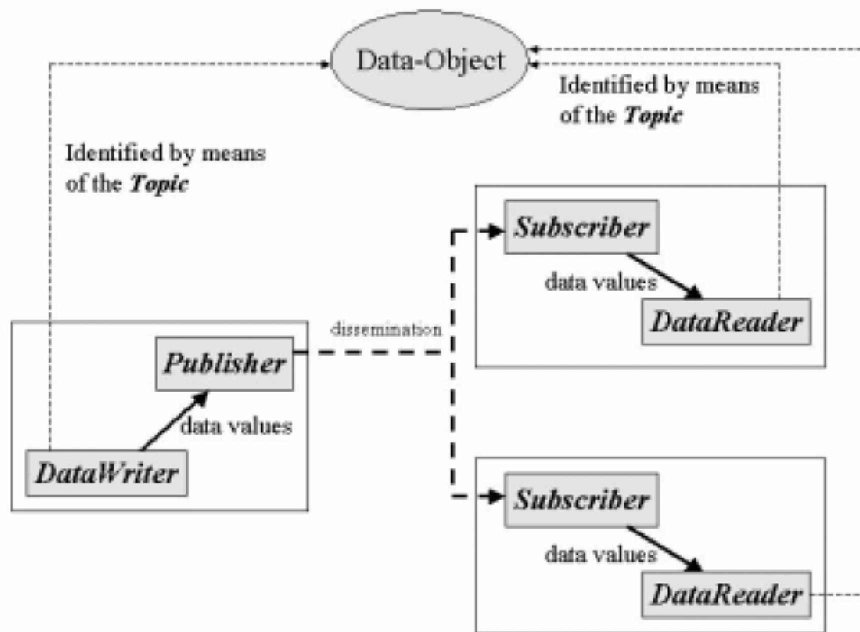

Figure 2 Information Flow Overview of DDS [8]

To publish data of a given type, an application must make a publication by creating a Publisher and a DataWriter with desired QoS policy, and modify data by using DataWriter as seen in Figure 3.

To receive data of a given type, an application must make a subscription by creating a Subscriber and a DataReader with desired QoS policy, and receive data by using DataReader by the help of Listeners as seen in Figure 4, and Conditions as seen in Figure 5.

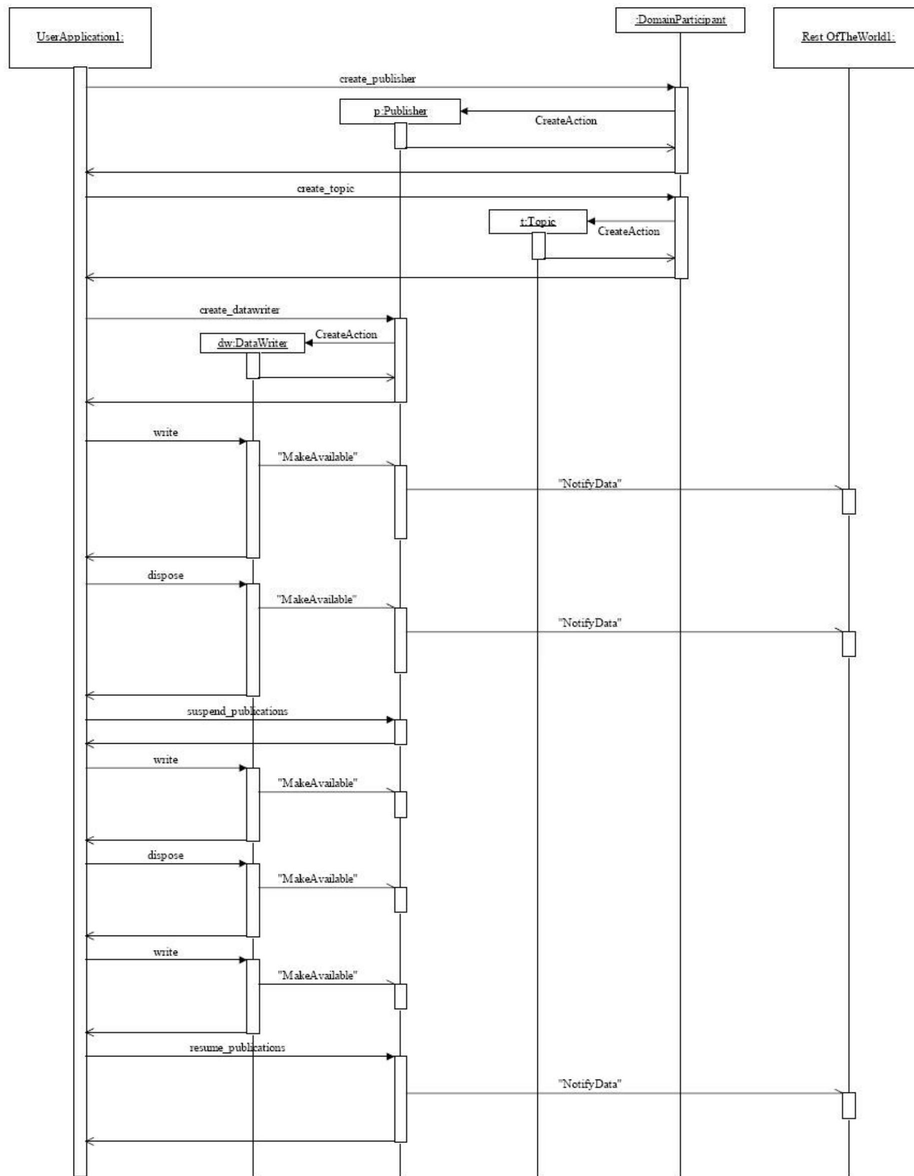

Figure 3 DDS Publication Sequence Diagram [8]

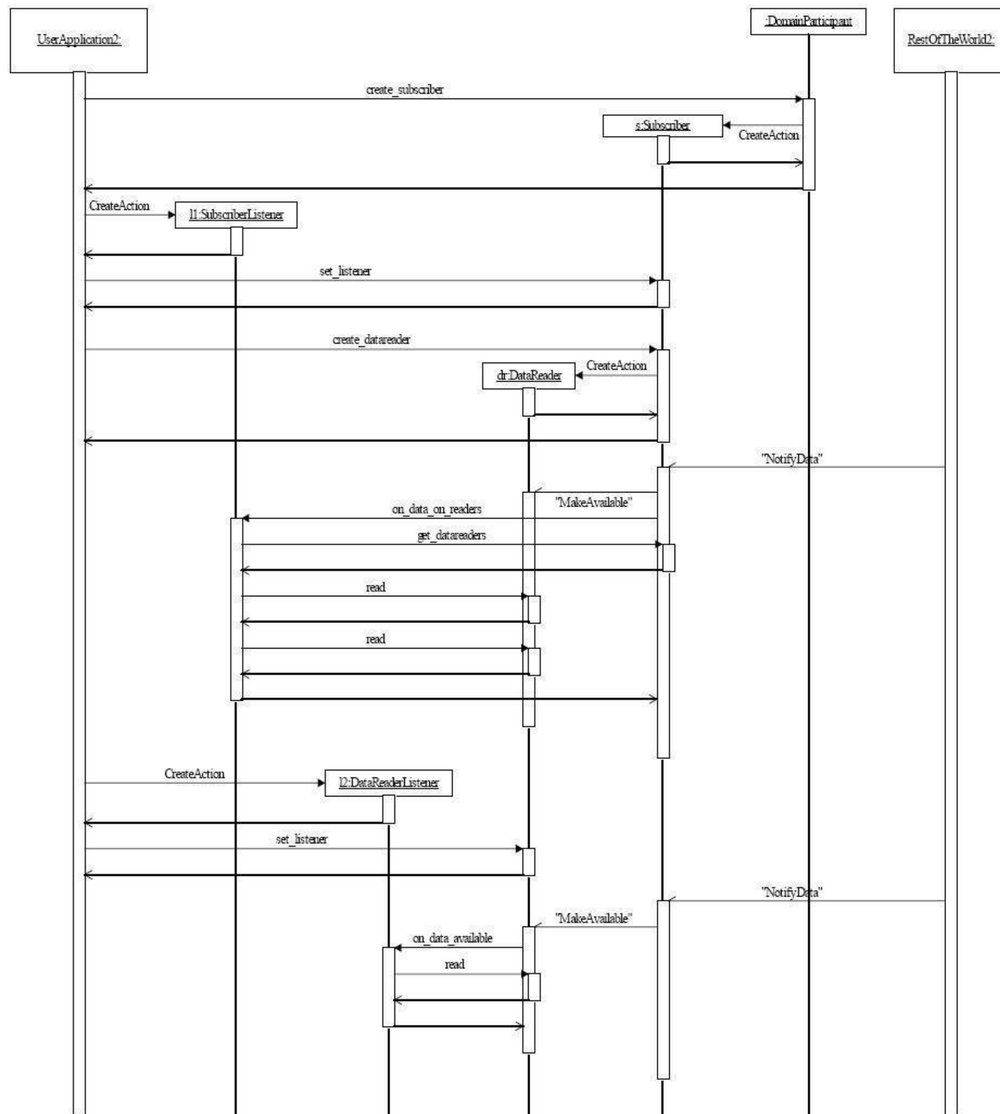

Figure 4 DDS Subscription with Listener Sequence Diagram [8]

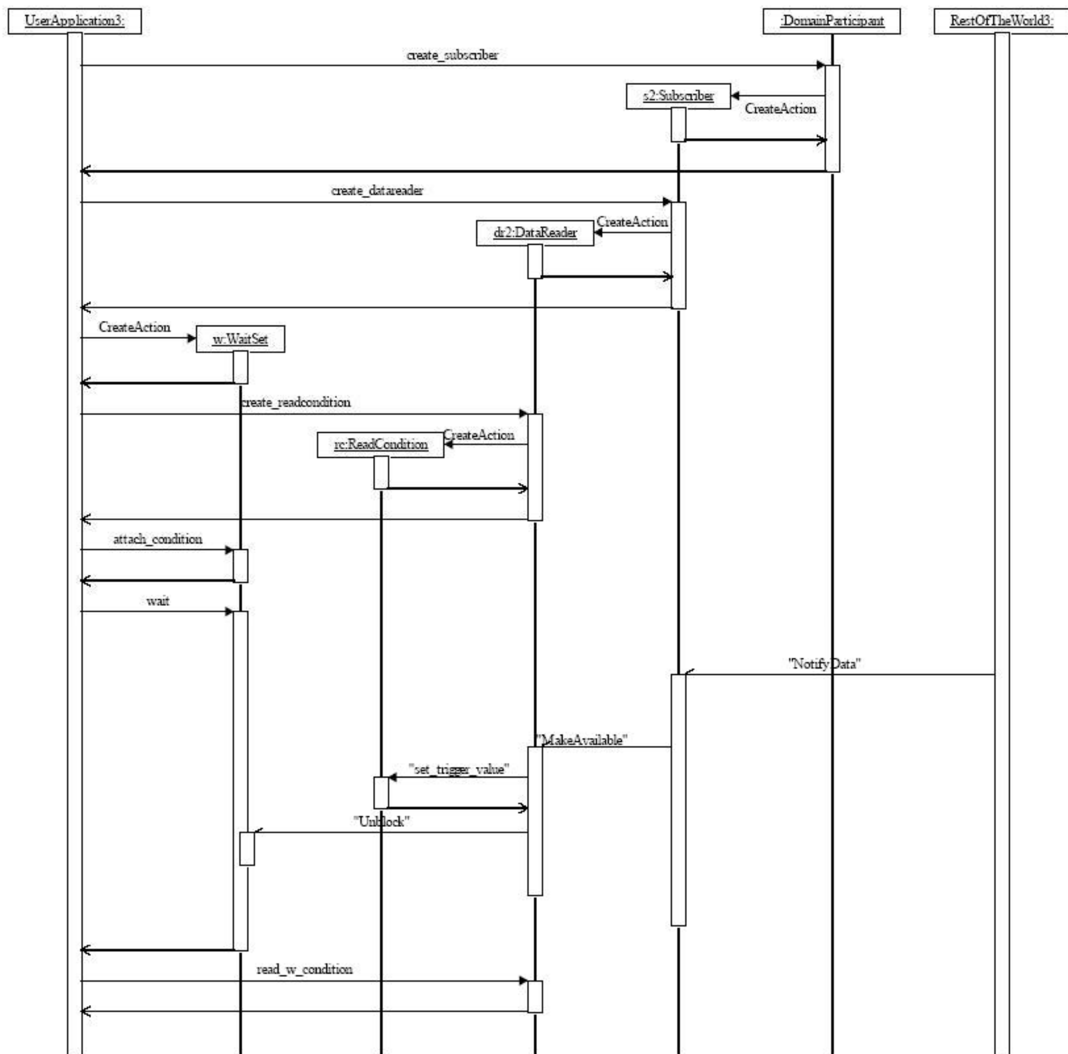

Figure 5 DDS Subscription with Condition Sequence Diagram [8]

2.2.3 DDS QoS Policies

The Data-Distribution Service (DDS) relies on the use of QoS [8]. The ability to specify different QoS policies for each data-object is a condition for data-centricity. DDS achieves this ability by associating QoS policies with Topic, DataReader, DataWriter, Subscriber, Publisher, and DomainParticipant entities.

DDS defines two kind of compatibility for QoS. First one is the compatibility between QoS policy values. In this case, some QoS policy values must be compatible with other ones. If a new QoS policy is inconsistent with previous QoS policies, change of QoS fails. Second one is the compatibility between QoS policies of entities. In this case a QoS Policy in publisher side must be compatible with QoS policy in the subscriber side. If there exist any inconsistency, service fails in communication.

DDS enables change in QoS policy values for some QoS policies after entity is enabled.

2.2.4 DDS Profiles

DDS specification includes 5 profiles [8]. Each profile designates different behavior and degree of coupling of service by specifying different QoS.

2.2.4.1 Minimum Profile

Minimum Profile contains just the mandatory features of the DCPS model. None of the optional features are included.

2.2.4.2 Content-Subscription Profile

Content-Subscription Profile enables DCPS model with content-based subscription. To achieve subscription by content, this model adds the optional ContentFilteredTopic, QueryCondition, MultiTopic classes.

2.2.4.3 Persistence Profile

Persistence Profile enables saving data transiently or permanently to achieve durability of the service. This profile adds the optional QoS policies DURABILITY_SERVICE, and DURABILITY with optional settings TRANSIENT, and PERSISTENCE.

2.2.4.4 Ownership Profile

Ownership Profile enables receiving the strength data [ref] from the replica publishers to achieve fault-tolerance. This profile adds the optional QoS policies OWNERSHIP_STRENGTH, and OWNERSHIP with optional settings EXCLUSIVE.

2.2.4.5 Object Model Profile

Object Model Profile includes the implementation of DLRL level to the service, and support PRESENTATION access_scope setting of GROUP.

2.2.5 Open-Source DDS Applications

NDDS and SPLICE are the commercial implementations of DDS specification. Except for these commercial implementations there exist open-source DDS implementations: TAO DDS implementation [18] and JacORB 2.2.3 DDS implementation [19]. The common property of open-source implementations is using CORBA Technology.

2.2.5.1 TAO DDS Implementation

TAO DDS Implementation [18] implements Minimum Profile of DDS. TAO DDS implementation defines CORBA server as Publisher, CORBA client as Subscriber, and DCPS Information Repository as Message Service. In TAO DDS implementation CORBA features are used to initialize and control service usage. ORB is used to register for Topic and QoS. The data transmission is not done with CORBA features. Instead of CORBA features, a TAO specific Pluggable Transport Layer is used. Pluggable Transport Level enables applications to use TCP, UDP, and their specific protocols. TAO DDS implementation uses a more efficient

variation of CDR of CORBA for marshaling. TAO DDS implementation defines Topics by using IDL. Implementation uses #pragma statement to identify the Topic type and Topic key in IDL.

TAO DDS implementation provides limited number of QoS policies, and does not support changing of QoS of an existing entity. The provided QoS policies are LIVELINESS, RELIABILITY, HISTORY, and RESOURCE_LIMITS.

2.2.5.2 JacORB 2.2.3 DDS Implementation

JacORB DDS implementation [19] is presented as a CORBA service in JacORB version 2.2.3. JacORB 2.2.3 DDS implementation defines CORBA server as Publisher, CORBA client as Subscriber and CORBA Event Service Event Channel implementation as Message Service. JacORB 2.2.3 DDS implementation implements DDS entities DomainParticipant, Publisher, Subscriber, Topic, and other classes to use CORBA Event Service Event Channel implementation as Message Service. DataWriter, DataReader, and DataReaderListener must be implemented by the developer of each Topic. JacORB 2.2.3 DDS uses CORBA Event Service for notification of subscribers.

Figure 6 shows the structure of JacORB 2.2.3 Event Service push-model as described in [10]. Proxy objects in Event Channel decouple the Suppliers and Consumers. An application that wants to push an event should create a supplier object and an application that wants to receive an event should create a consumer object.

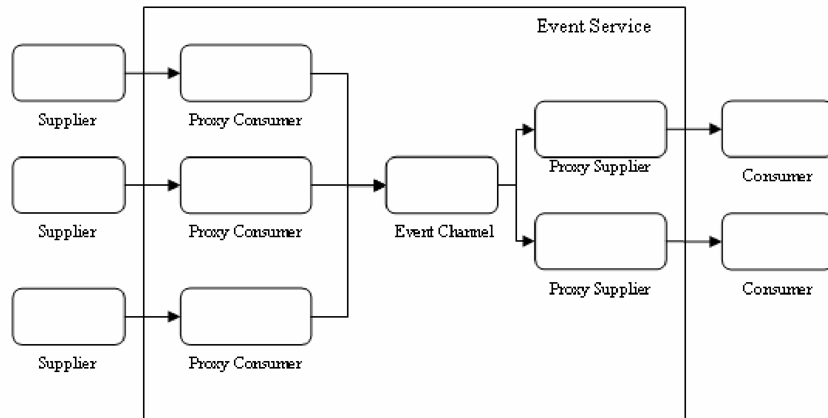

Figure 6 JacORB 2.2.3 Event Service Model [10]

JacORB 2.2.3 DDS implementation creates a supplier object for each one of DataWriter objects. On the other hand, only one consumer object is created as Super Consumer for service as in Figure 7. The super consumer has as a role to distribute the data collected to all Subscribers concerned. This causes notification of subscribers sequentially for all data-types in the service [20]. The super consumer object keeps the references of subscriber objects not only in one domain but also in the service. This causes communication between entities in different domains.

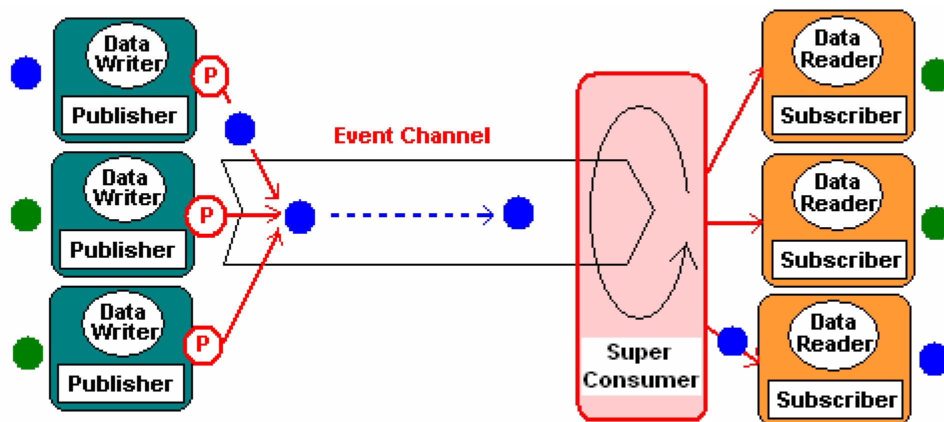

Figure 7 JacORB 2.2.3 DDS Implementation Event Service Mechanism [20]

The supplier objects are used by DataWriter objects, which mean data dissemination is under the control of DataWriter objects. This violates the responsibility of Publisher objects in data dissemination principle of service.

As TAO DDS implementation does, JacORB 2.2.3 DDS implementation defines Topics by using IDL, except for any statements to identify Topic type and key.

CHAPTER III

CORBA DDS (CDDS) IMPLEMENTATION

DDS specification defines standard interfaces and QoS policies to allow applications to use data-centric publish-subscribe model in communication. Definition of interfaces comprises definition of functions and interaction of DCPS entities. DDS specification does not describe implementation of functions and interaction of DCPS entities such as notifying data to subscribers and making data available between DCPS entities.

JacORB 2.2.3 DDS implements DDS specification as a CORBA Service in minimum profile. JacORB 2.2.3 DDS uses JacORB 2.2.3 Event Service to notify data to subscribers. The usage of Event Service makes JacORB 2.2.3 DDS incompatible with DDS specification in the points of responsibility of publisher in notification and domain definition. CDDS, developed and implemented within the scope of this study, essentially aims to overcome this incompatibility.

CDDS implements a subset of DCPS layer of DDS, whose boundary is determined by minimum profile of DDS. CDDS implements DDS as a CORBA Service like JacORB 2.2.3 DDS. CDDS is compatible with DDS specification in notification and domain definition. Publisher is responsible for notification of data in CDDS implementation, and any interaction is not possible between entities in different domains. Also CDDS implementation provides an infrastructure to implement different QoS policies and DDS profiles.

As a CORBA Service, CDDS implementation implements DCPS entities as CORBA servant objects. CDDS implementation uses JacORB 2.2.3 Naming Service to make applications get reference of CDDS entry point.

3.1 CDDS Participants

CDDS implements the DCPS entities DomainParticipantFactory, DomainParticipant, Publisher, Subscriber, Topic, SampleInfo, DataWriter, DataReader, and DataReaderListener as

- cDomainParticipantFactory,
- cDomainParticipant,
- cPublisher,
- cSubscriber,
- cTopic,
- cSampleInfo,
- cDataWriter,
- cDataReader,
- cDataReaderListener;

With two additional participants; cMediator, and cDataHandler. These participants are used to manage subscription, data distribution and notification in CDDS implementation. Figure 8 shows the relations between CDDS participants.

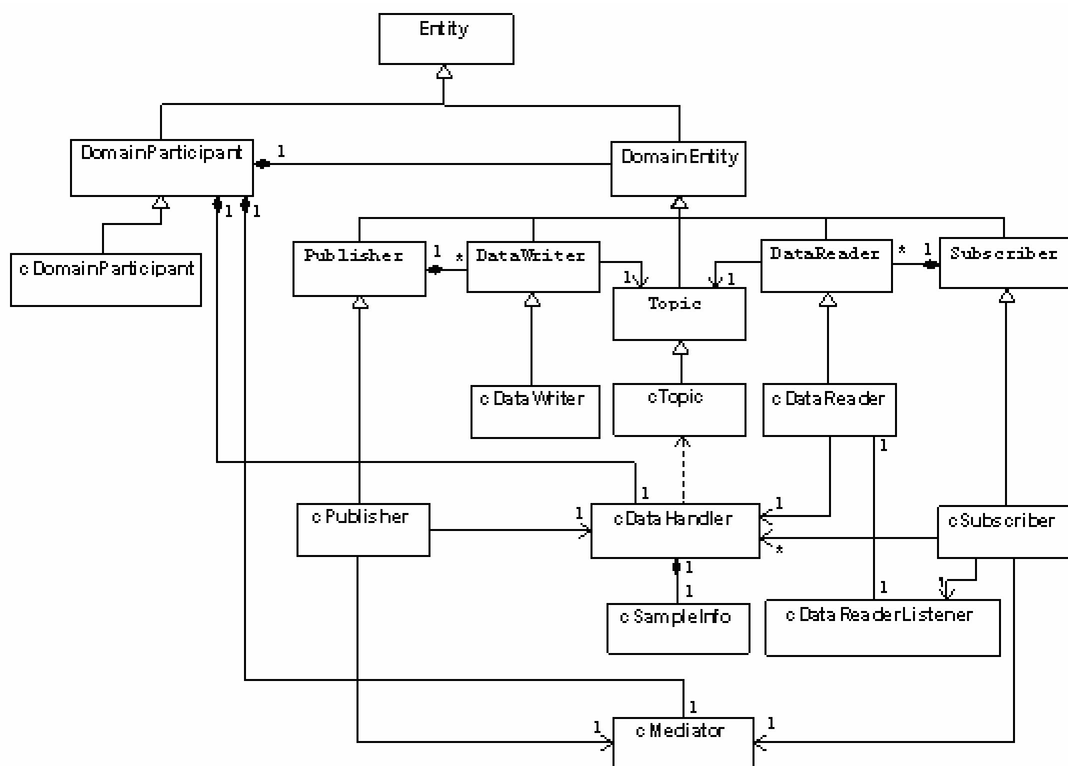

Figure 8 CDDS DCPS Entities Model Diagram

3.1.1 cDomainParticipantFactory

cDomainParticipantFactory class implements DomainParticipantFactory interface. The purpose of the cDomainParticipantFactory object is to create and destroy cDomainParticipant objects. cDomainParticipantFactory object is a singleton object. The applications get the reference of cDomainParticipantFactory object by using JacORB 2.2.3 Naming Service.

3.1.2 cDomainParticipant

cDomainParticipant class implements DomainParticipant interface. The purpose of cDomainParticipant object is to create a domain by acting as factory for cPublisher, cSubscriber, cMediator, cDataHandler, and cTopic objects. It is possible to create different domains by creating cDomainParticipant objects with different domain ID values. cDomainParticipant object is singleton for the same domain ID values.

3.1.3 cPublisher

cPublisher class implements Publisher interface. The purpose of cPublisher object is to notify cSubscriber objects of a change in a data object associated with one of its cDataWriter objects. cPublisher object makes a decision about notification by the help of QoS policy. cPublisher object acts as a factory for cDataWriter.

3.1.4 cDataWriter

cDataWriter class implements DataWriter interface. The purpose of cDataWriter object is to allow application to publish data. cDataWriter object achieves this by enabling application to set the value of data object that is associated with a cTopic object. cTopic object has priority in creating a cDataWriter object.

3.1.5 cSubscriber

cSubscriber class implements Subscriber interface. The purpose of cSubscriber object is to get value of data that application subscribes to. cSubscriber object acts as a factory for cDataReader objects.

3.1.6 cDataReader

cDataReader class implements DataReader interface. The purpose of cDataReader object is to allow application to make a subscription to a data object, and to access value of data that is received by cSubscriber object.

3.1.7 cSampleInfo

cSampleInfo class implements SampleInfo interface. cSampleInfo object creates a data sample with the data values. The purpose of cSampleInfo object is to keep the state values for a data sample. CDDS creates a cSampleInfo object for each one of cDataReader objects. cSampleInfo objects are created and kept by cDataHandler objects.

3.1.8 cTopic

cTopic class implements Topic interface. The purpose of cTopic object is to describe the data objects to publish and subscribe. A cTopic object is identified by its name in a domain. Topic name must be unique in the domain. cTopic object also specifies data type.

3.1.9 cDataReaderListener

cDataReader class implements DataReaderListener interface. The purpose of cDataReaderListener objects is to make DataReader object get data values, and to submit data values to application.

3.1.10 cMediator

cMediator class implements Mediator interface (See Figure 9) that is defined in Mediator design pattern [21].

IMediator		
no attribute		
Operations		
add_subscription		ReturnCode_t
	topic_name	String
	subscriber	Subscriber
get_subscribers		Subscriber []
	topic_name	String
set_datahandler		ReturnCode_t
	topic_name	String
	datahandler	DataHandler
get_datahandler		DataHandler
	topic_name	String

Figure 9 Mediator Interface Specification

3.1.10.1 add_subsription

This operation enables keeping a subscriber object reference for specified topic.

3.1.10.2 get_subscribers

This operation allows access to subscriber objects associates with specified topic.

3.1.10.3 set_datahandler

This operation enables keeping a DataHandler object reference for specified topic.

3.1.10.4 get_datahandler

This operation allows access to a DataHandler object associates with specified topic.

The purpose of cMediator object is to achieve decoupling between cPublisher and cSubscriber objects. At this concept it keeps DataHandler and Subscriber object references and allows accessing to DataHandler and Subscriber objects.

3.1.11 cDataHandler

cDataHandler implements DataHandler interface as in Figure 10.

DataHandler		
no attribute		
Operation		
set_data		ReturnCode_t
	data	Data
Get_data		Data
	datareader	DataReader
get_samplestate		sample_state
	datareader	DataReader
create_sampleinfo		ReturnCode_t
	datareader	DataReader

Figure 10 DataHandler Interface Specification

3.1.11.1 set_data

This operation enables keeping data object for receiving.

3.1.11.2 get_data

This operation allows accessing to data object.

3.1.11.3 get_samplestate

This operation allows accessing to sample_state associates with specified DataReader.

3.1.11.4 create_sampleinfo

This operation creates a SampleInfo for specified DataReader.

The purpose of cDataHandler object is to enable service to achieve data dissemination suitable with QoS of cTopic objects. CDDS implementation creates a cDataHandler object for data objects described by cTopic objects.

cDataHandler object creates a cSampleInfo object for each one of cDataReader objects related with specified cTopic object. When set_data operation is invoked, cDataHandler sets data value and changes sample state of all cSampleInfo objects to NOT_READ. When get_data operation is invoked, cDataHandler changes sample state of related cSampleInfo object to READ.

3.2 CDDS Deployment

The deployment of CDDS participant objects is depicted in Figure 11. All participant objects except for cDataReaderListener object are deployed in a server computer, CDDS Server. DCPS entity implementations are created as CORBA servant objects. This makes possible application to invoke operations on DCPS entity implementation objects by getting their CORBA object references. cDomainParticipantFactory is the first object created in CDDS server, entry point of CDDS. Applications get CORBA object reference of cDomainParticipantFactory object using JacORB Naming Service. cDataReaderListener objects are deployed in the computers where consumer applications run.

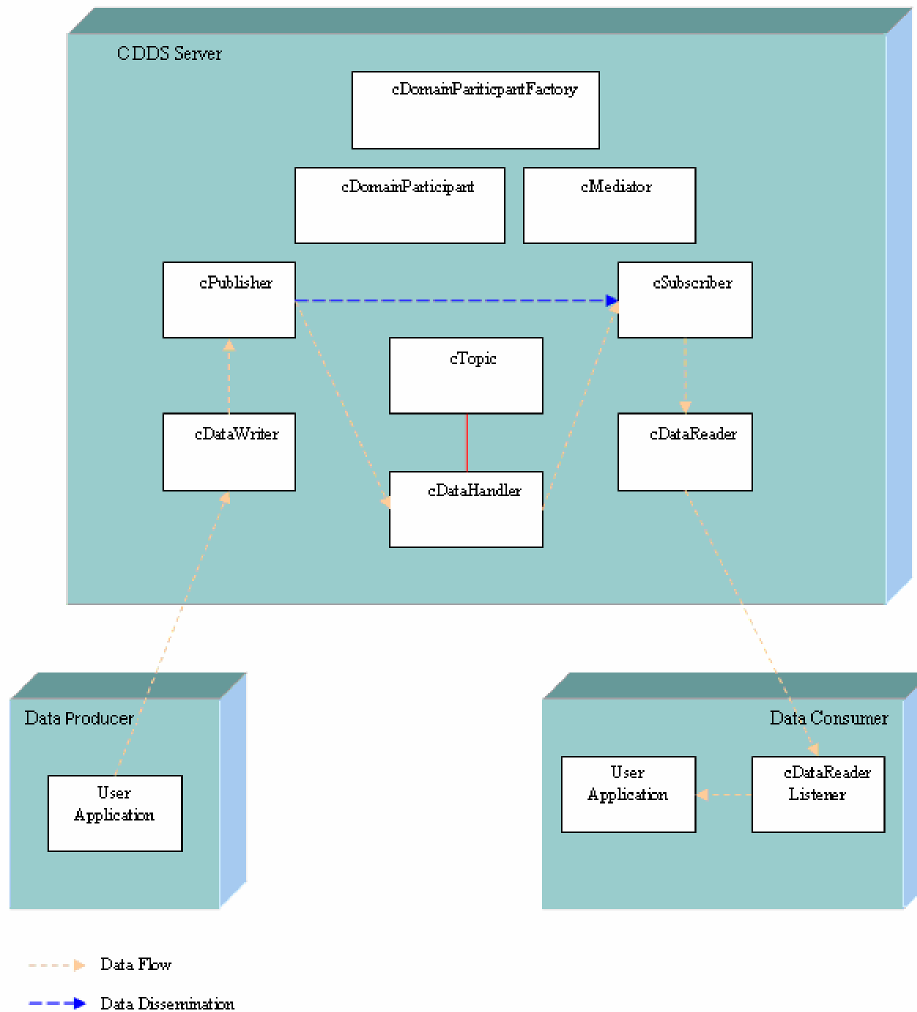

Figure 11 CDDS Participants Deployment Diagram

3.3 CDDS Data Dissemination Mechanism

DDS specification specifies that data dissemination is made from publisher to subscriber. This means publisher is responsible for data dissemination. A data supplier communicates with a publisher by a DataWriter to notify data. Subscriber receives published data and makes it available to the receiving application.

CDDS achieves data dissemination in four stages:

- Creating Factory Participants
- Publication

- Subscription
- Data Notification
- Data Reception

Factory participants mean participants, cDomainParticipant, cPublisher, cSubscriber, acting as factory for other participants, and cTopic.

3.3.1 Creating Factory Participants

An application must create a cPublisher object to publish data by invoking create_publisher operation of cDomainParticipant object. cDomainParticipant object creates a cPublisher object as CORBA servant and returns CORBA object reference of cPublisher object to application. Figure 12 shows sequence diagram of creating a cPublisher object.

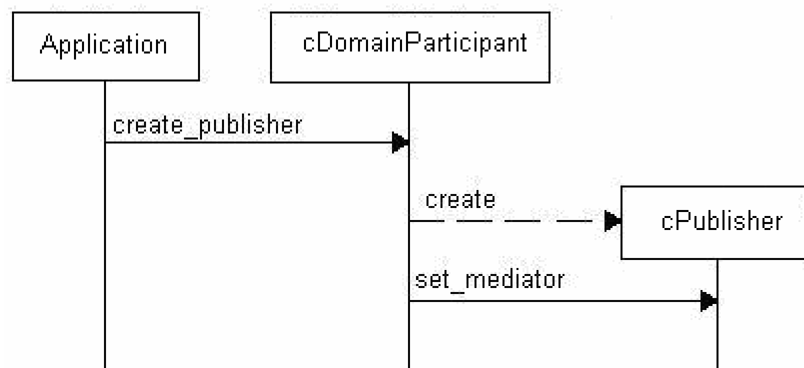

Figure 12 Creating cPublisher Sequence Diagram

An application must create a cSubscriber object to receive data by invoking create_subscriber operation of cDomainParticipant object. cDomainParticipant object creates a cSubscriber object as CORBA servant and returns CORBA object reference of cSubscriber object to application. Figure 13 shows sequence diagram of creating a cSubscriber object.

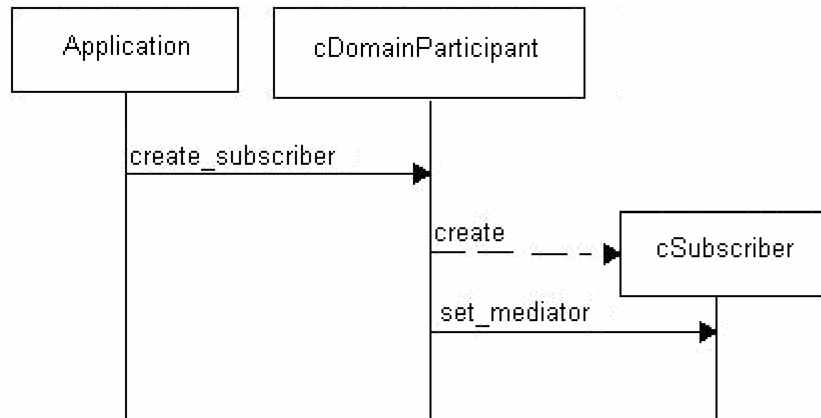

Figure 13 Creating cSubscriber Sequence Diagram

To publish or receive data as seen in Figure 14, a cTopic object must be created by any application. When an application invokes create topic operation from cDomainParticipant object, cDomainParticipant objects creates a cTopic object as CORBA servant, a cDataHandler object for specified type of cTopic objects, and returns CORBA object reference of cTopic object. After creating cDataHandler object, cDomainParticipant object registers cDataHandler object reference to cMediator object.

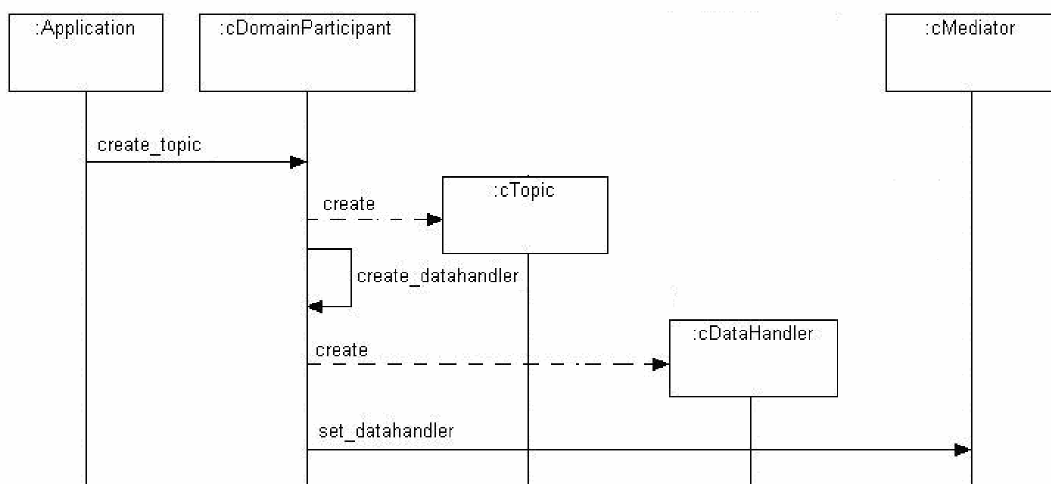

Figure 14 Creating cTopic Sequence Diagram

3.3.2 Subscription

In subscription stage cDataReader object, cSampleInfo object, and cDataReaderListener objects are created as in Figure 15. DDS Specification defines subscription by the association of a data-reader with a subscriber. An application invokes create_datareader operation on cSubscriber object to subscribe a topic. cSubscriber object creates a cDataReader object as a CORBA servant, and returns CORBA object reference of cDataReader object. After creating a cDataReader object, cSubscriber object registers this subscription to cMediator object. cSubscriber object creates a cSampleInfo object for created cDataReader object by invoking create_sampleinfo operation on cDataHandler object.

After getting CORBA object reference of cDataReader object, application must create a cDataReaderListener as a CORBA servant, and set CORBA object reference of cDataReaderListener object to cDataReader object.

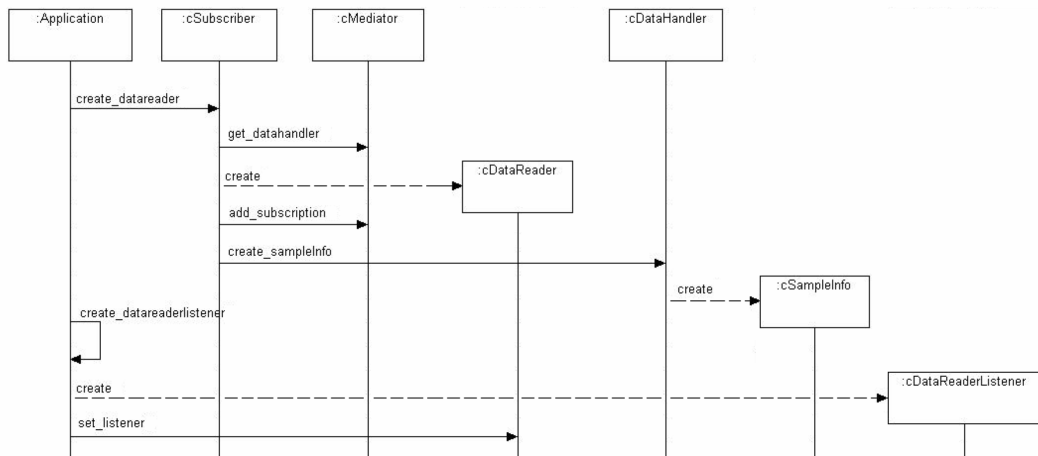

Figure 15 Subscription Sequence Diagram

3.3.3 Publication

DDS specification describes publication by the association of a cDataWriter to a Publisher. In publication stage cDataWriter object is created. An application creates a cDataWriter object for a topic by invoking create_datawriter on cPublisher object. cPublisher object creates cDataWriter object as a CORBA servant, and returns CORBA object reference of cDataWriter object. After creating cDataWriter object, cPublisher object gets cDataHandler object reference for the topic for which cDataWriter object created. Figure 16 shows sequence diagram of publication.

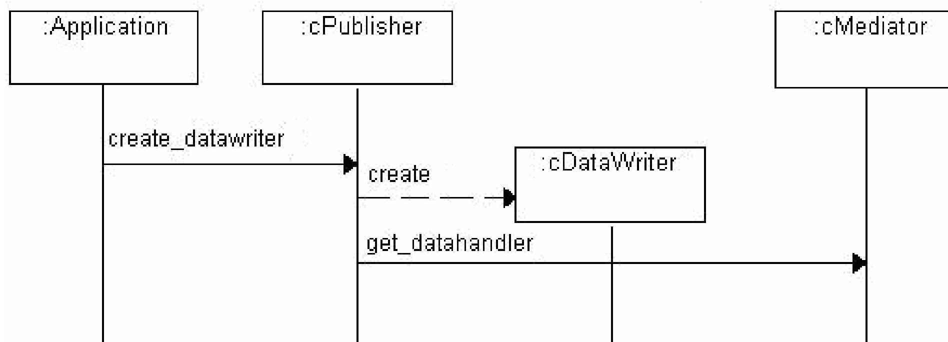

Figure 16 Publication Sequence diagram

3.3.4 Data Notification

When CDDS factory participants and publication is created, service is ready to notify data. An application invokes write operation on cDataWriter object to notify data. cDataWriter object invokes notify_readers operation on cPublisher object when write operation is invoked. At this point service gives responsibility of data notification to cPublisher object. Notification is shaped with the QoS policy of cPublisher object. To notify data, cPublisher object first get cSubscriber object references, which are subscribed to specified topic, from cMediator object, and set data to related cDataHandler object. When set_data operation is invoked on cDataHandler object, cDataHandler object sets sample state of all cDataReader

objects to NOT_READ. After setting data, cPublisher object invokes notify_datareaders operation on cSubscriber objects whose references are get from cMediator object. Figure 17 shows sequence diagram of Data Notification.

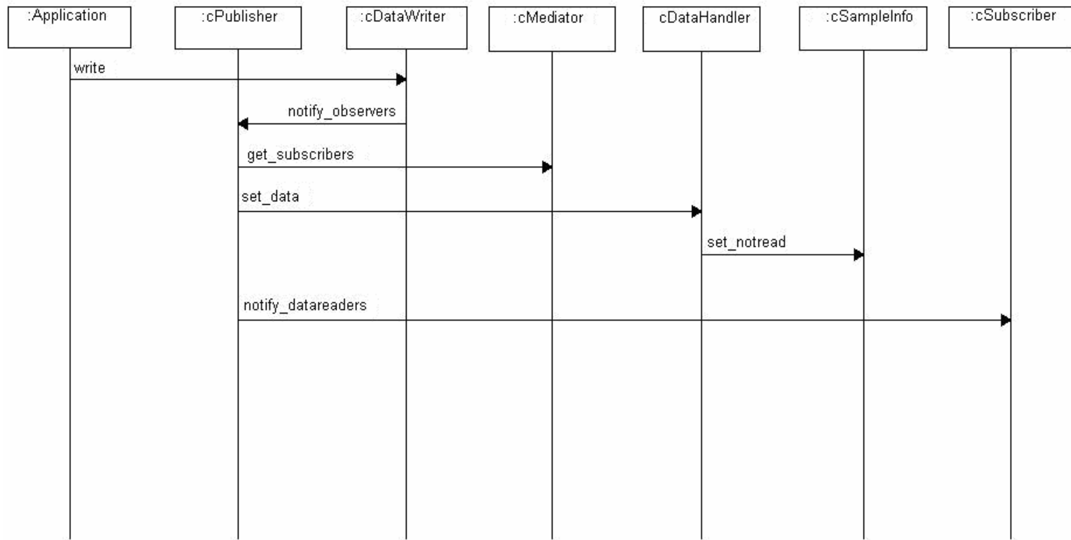

Figure 17 Data Notification Sequence diagram

3.3.5 Data Reception

Data reception is achieved by cSubscriber, cDataReader, and cDataReaderListener objects as shown in Figure 18. Data reception starts when notify_datareaders operation is invoked on cSubscriber object. cSubscriber invokes get_samplestate with parameter cDataReader reference on cDataHandler object. cDataHandler object returns cSampleInfo object sample state to cSubscriber object. If cSampleInfo object sample state is NOT_READ, cSubscriber gets cDataReaderListener CORBA object references from cDataReader objects, and invokes data_available operation on cDataReaderListener objects. When data_available operation is invoked, cDataReaderListener object invokes read operation on cDataReader object. cDataReader object gets data by invoking

get_data operation on cDataHandler object and return data to cDataReaderListener object. cDataHandler object sets sample state of cSampleInfo object of the related cDataReader object as READ, when get_data operation is invoked.

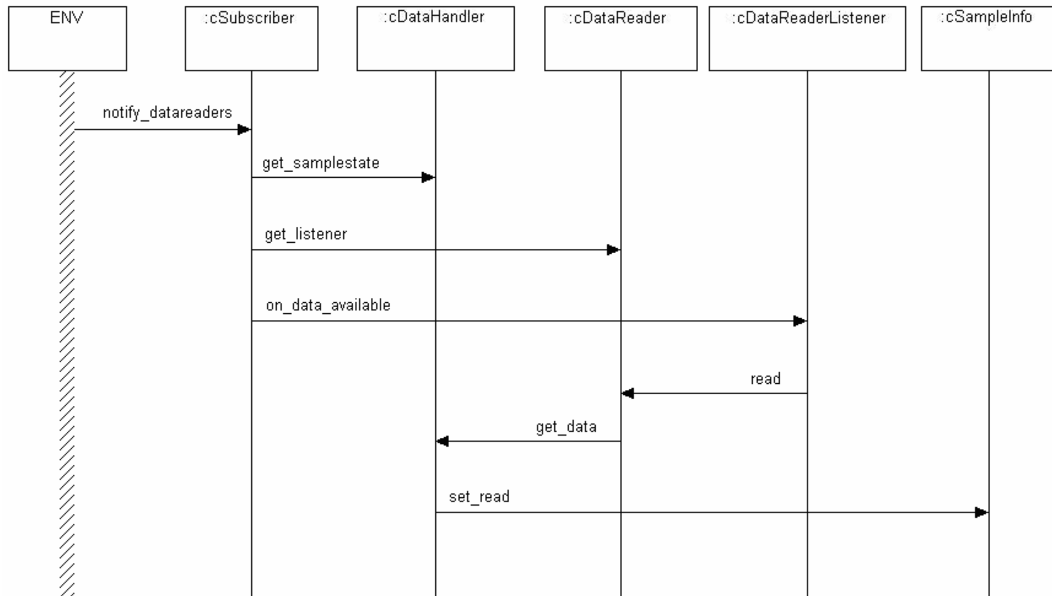

Figure 18 Data Reception Sequence diagram

3.4 A New QoS Policy: BLOCK

CORBA overhead is an important drawback from the aim of minimal overhead in data distribution with CDDS implementation. DDS specification [8] does not offer any QoS policy to minimize overhead. CDDS defines a new QoS policy, BLOCK policy, for data-objects that are periodically published. The purpose of BLOCK policy is to reduce CORBA overhead in read method invocation by increasing data payload size.

BLOCK policy has a value; blocksize. The default value of blocksize is 1. With this QoS policy, service keeps the published data-objects, and blocks notifying data to subscribers until the number of data-objects reaches blocksize value. When number of data-objects reaches blocksize, service notifies data to subscribers, and DataReaderListener object reads blocksize number of data-objects from DataReader. BLOCK policy concerns Topic and DataReader entities, and it is changeable.

Difference of BLOCK policy defined within the scope of this study, from HISTORY policy [8] is in the notification of data. With HISTORY policy, service notifies data to subscribers for each data publication. On the other hand, with BLOCK policy, service notifies data to subscribers when blocksize publications of data have been realized for the topic for which this QoS policy has been set.

3.5 CDDS Type-Specific Classes

Data Distribution Service requires to create a number of specialized classes for each data class defined by the application to facilitate the type-safe interaction of the application with the service. Specialized classes for data classes are DataWriter, DataReader, and Type Support classes. CDDS handles type-specific classes by the help of Java Reflection Technology.

CHAPTER IV

EVALUATION

This section investigates the performance of CDDS implementation with changing number of publishers and subscribers.

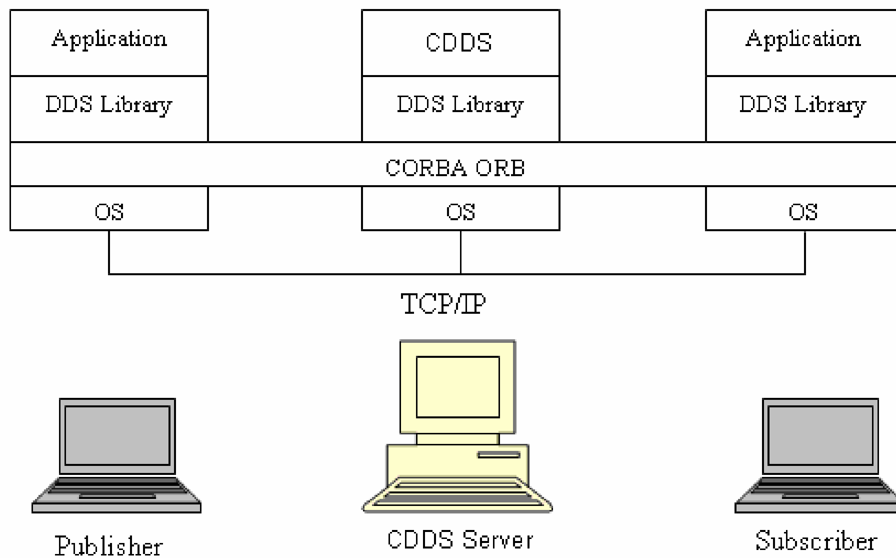

Figure 19 Testbed Configuration

Configuration of CDDS server, producer application, and consumer application in the testbed is illustrated in Figure 19. While CDDS implementation is running on CDDS Server machine, producer and consumer applications run on Producer and Consumer machines. Network connection between server, producer and consumer

machines is achieved by 100Mbps Ethernet. No background traffic is present, as these systems aim to support distributed and closed communication networks with no external traffic load.

4.1 Experiment Overview

The experiments presented here investigate performance, interoperability and scalability of CDDS implementation by measuring average latency of write operation of DataWriter object. Experiment 1 measures the performance of CDDS and JacORB DDS implementations in a homogeneous distributed environment. Experiment 2 investigates interoperability of CDDS by measuring the performance of CDDS in heterogeneous distributed environment. Experiment 3 investigates scalability of CDDS by measuring the performance of CDDS with increasing number of producers and consumers in a homogeneous distributed environment. Finally, Experiment 4 investigates effect of BLOCK QoS policy on the performance of CDDS.

4.1.1 Experiment 1

Experiment 1 investigates performance by measuring the variation of throughput and latency of write operation in CDDS and JacORB DDS with increasing data volume in data dissemination from one producer to one consumer. The testbed is illustrated in Table 2 and Figure 20.

Table 2 Experiment 1 Testbed Properties

	CPU	OS	ORB
CDDS Server	Pentium IV, 3 GHz	Windows XP	JacORB 2.2.3
Producer	Pentium IV, 3 GHz	Windows XP	JacORB 2.2.3
Consumer	Pentium IV, 3 GHz	Windows XP	JacORB 2.2.3

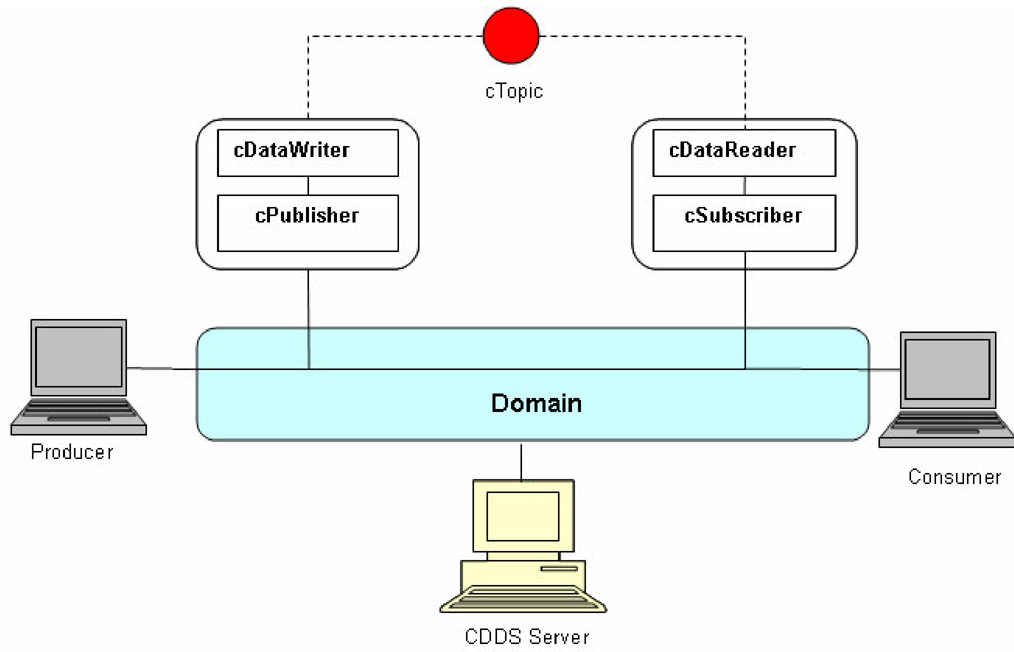

Figure 20 Experiment 1 Testbed Configuration

Table 3 shows average write operation latency for 1000 calls. Latency is measured as the time elapsed from invocation of write operation to return of write operation of DataWriter object. It is measured for JacORB DDS and CDDS for increasing data size.

Table 3 Roundtrip time of CDDS and JacORB DDS (μs)

	0,004 Kbytes	0.5 Kbytes	1 Kbytes	2 Kbytes	4 Kbytes	8 Kbytes	16 Kbytes
JacORB DDS	1753.191	2108.106	2494.723	2966.939	3721.404	4821.277	7241.000
CDDS	1728.478	2100.304	2444.065	2992.717	3742.826	4913.674	7109.674

As shown in Figure 21, JacORB and CDDS performances are similar, in spite of the fact that CDDS does not use JacORB Event Service. This means CORBA overhead in read and write operations has an important role in the performance of both applications.

It is possible to see the effect of CORBA overhead on throughput in Figure 22. Throughput of CDDS increases with increasing data payload.

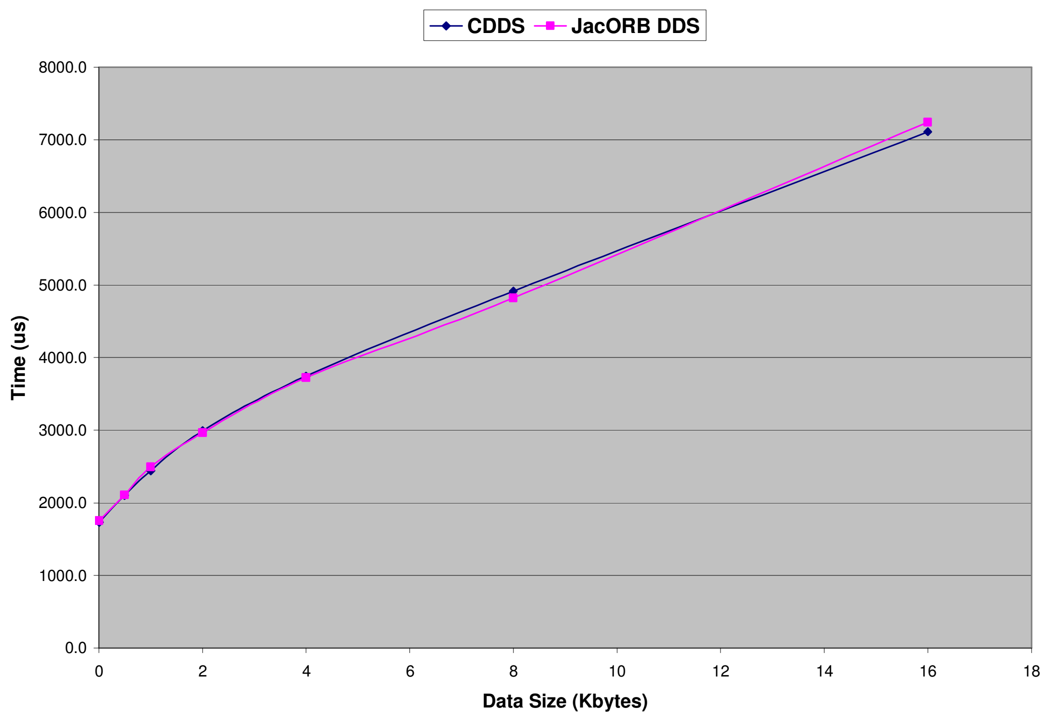

Figure 21 JacORB DDS versus CDDS performance

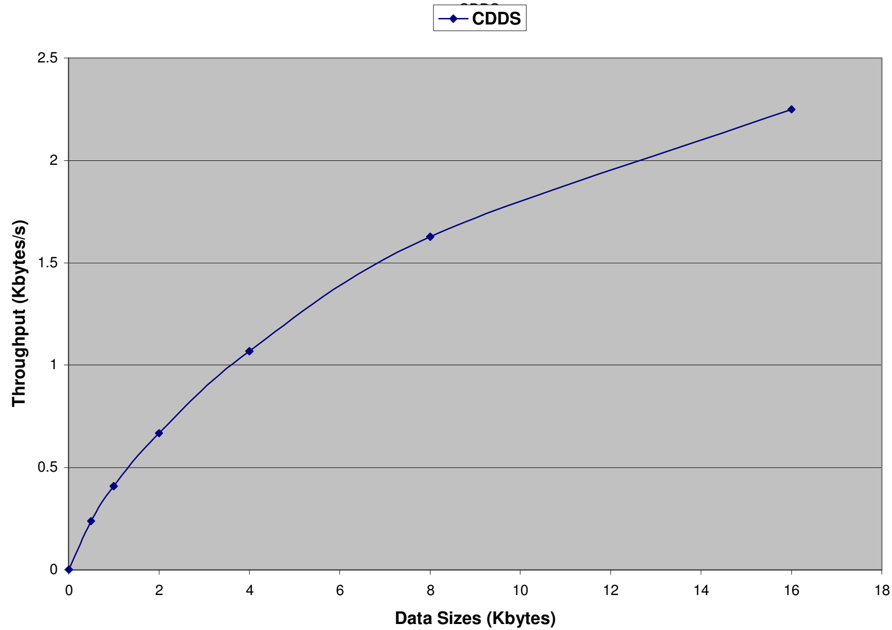

Figure 22 Throughput of CDDS

4.1.2 Experiment 2

Experiment 2 investigates interoperability property of CDDS implementation. The testbed is illustrated in Table 4 and Figure 23. In this context, interoperability is defined as the ability to distribute data from producer programs to consumer programs that may be coded in different programming languages and possibly run on different hardware and operating system platforms.

Table 4 Experiment 2 Testbed Properties

	CPU	OS	ORB
CDDS Server	Pentium IV,3 GHz	Windows XP	JacORB 2.2.3
Producer	Motorola, 400MHz	VxWorks 5.5	TAO 1.4
Consumer	Pentium IV, 3 GHz	Windows XP	JacORB 2.2.3

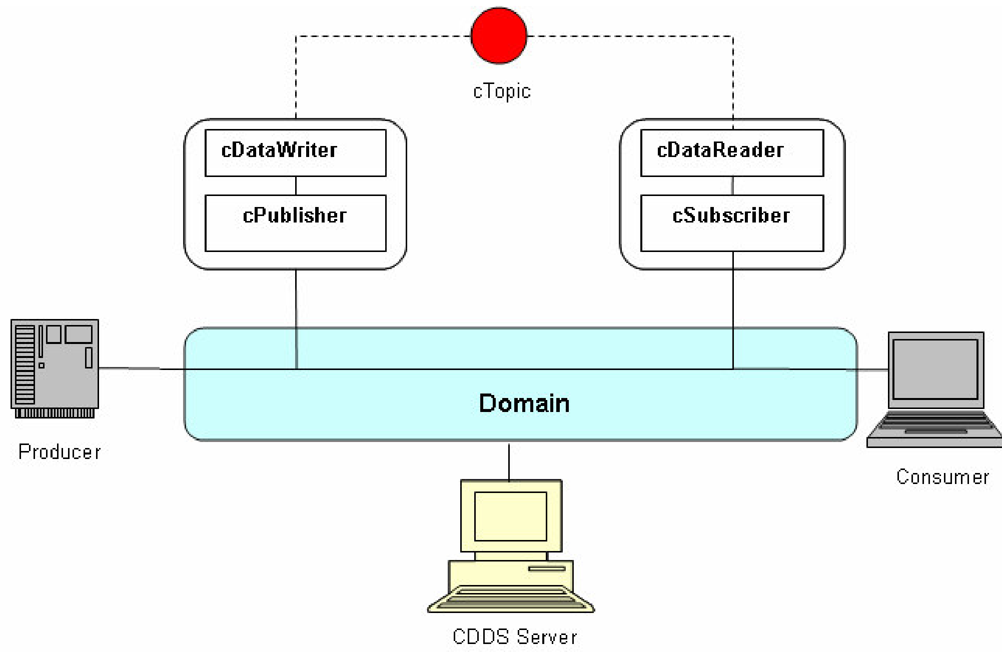

Figure 23 Experiment 2 Testbed Configuration

In this experiment producer coded in C++ publishes data to consumer coded in Java. Table 5 shows write operation latency for 1000 calls for JacORB producer and TAO producer for increasing data size.

Table 5 Roundtrip time of CDDS in heterogeneous distributed environment (µs)

	0,004 Kbytes	0.5 Kbytes	1 Kbytes	2 Kbytes	4 Kbytes	8 Kbytes
TAO Producer	2281.17	2878.91	3369.34	4047.24	4796.65	5929.99
JacORB Producer	1728.478	2100.304	2444.065	2992.717	3742.826	4913.674

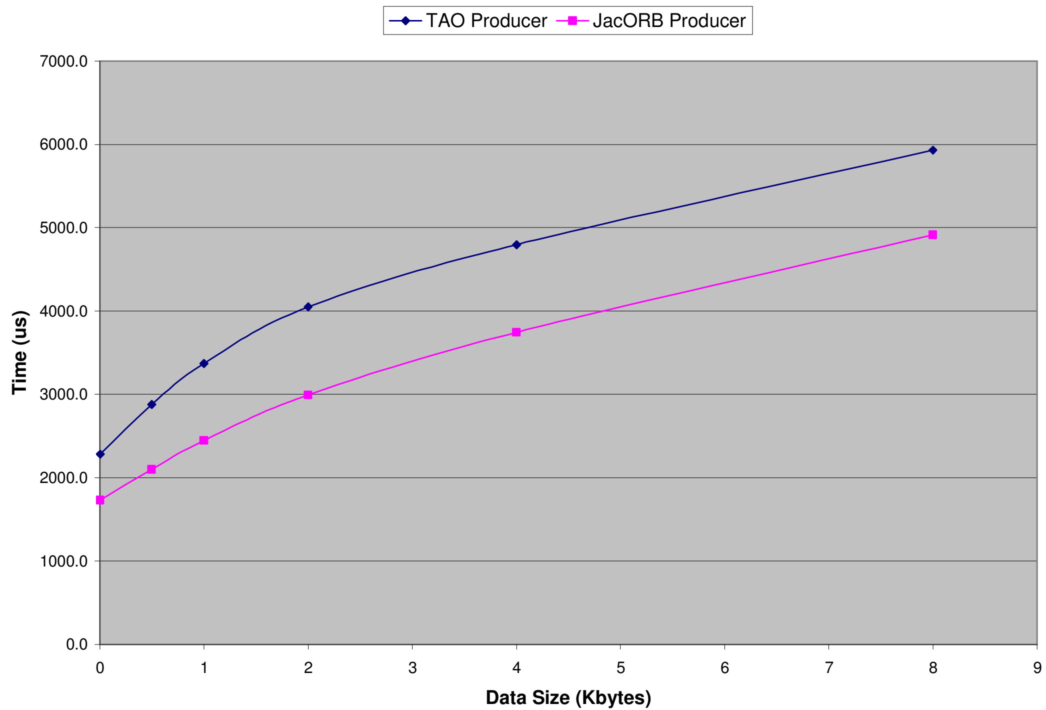

Figure 24 Performance of CDDS in heterogeneous distributed environment

As illustrated in Figure 24, TAO producer latency is higher than JacORB producer latency because of different CPU properties. JacORB producer run on a more powerful CPU than TAO producer.

Hence this experiment shows that CDDS has the same performance characteristic in heterogeneous distributed environment as in homogeneous distributed environment.

4.1.3 Experiment 3

Experiment 3 investigates the scalability of CDDS implementation. We examined the impact of increasing number of consumers and producers in Experiment 3.a and Experiment 3.b. In the context of this study, scalability is defined as the independence of communication latency from the number of producers and consumers.

4.1.3.1 Experiment 3.a

Experiment 3.a investigates scalability of CDDS by measuring variation of throughput and latency of write operation in CDDS with increasing data volume in data dissemination from one producer to three consumers. The testbed is illustrated in Table 6 and Figure 25.

Table 6 Experiment 3.a Testbed Properties

	CPU	OS	ORB
CDDS Server	Pentium IV, 3 GHz	Windows XP	JacORB 2.2.3
Producers	Pentium IV, 3 GHz	Windows XP	JacORB 2.2.3
Consumer	Pentium IV, 3 GHz	Windows XP	JacORB 2.2.3

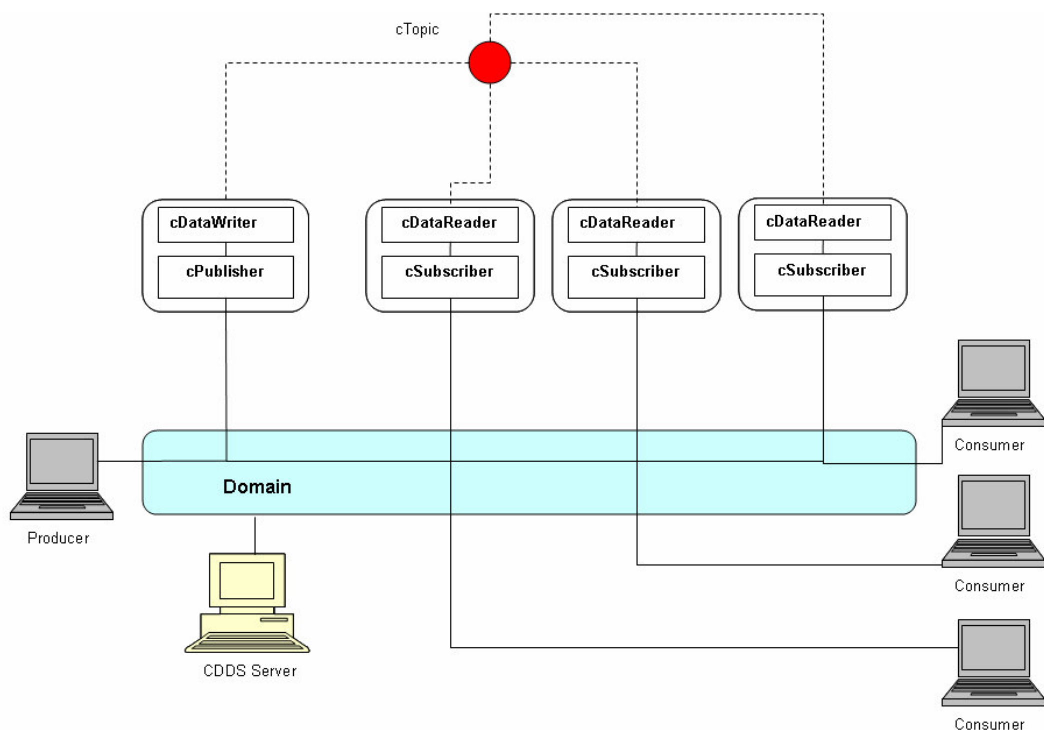

Figure 25 Experiment 3.a Testbed Configuration

Table 3 shows write operation latency for 1000 calls for 1, 2, and 3 subscribers. Latency is measured for CDDS for increasing data size.

Table 7 Roundtrip time of CDDS for different number of producers (μs)

	0,004 Kbytes	0.5 Kbytes	1 Kbytes	2 Kbytes	4 Kbytes	8 Kbytes	16 Kbytes
1 Consumer	1728.478	2100.304	2444.065	2992.717	3742.826	4913.674	7109.674
2 Consumers	3052.35	3693.47	4230.85	5083.57	6162.66	7811.19	12345.26
3 Consumers	4621.66	5474.74	6273.55	7520.98	8882.66	11370.32	17516.66

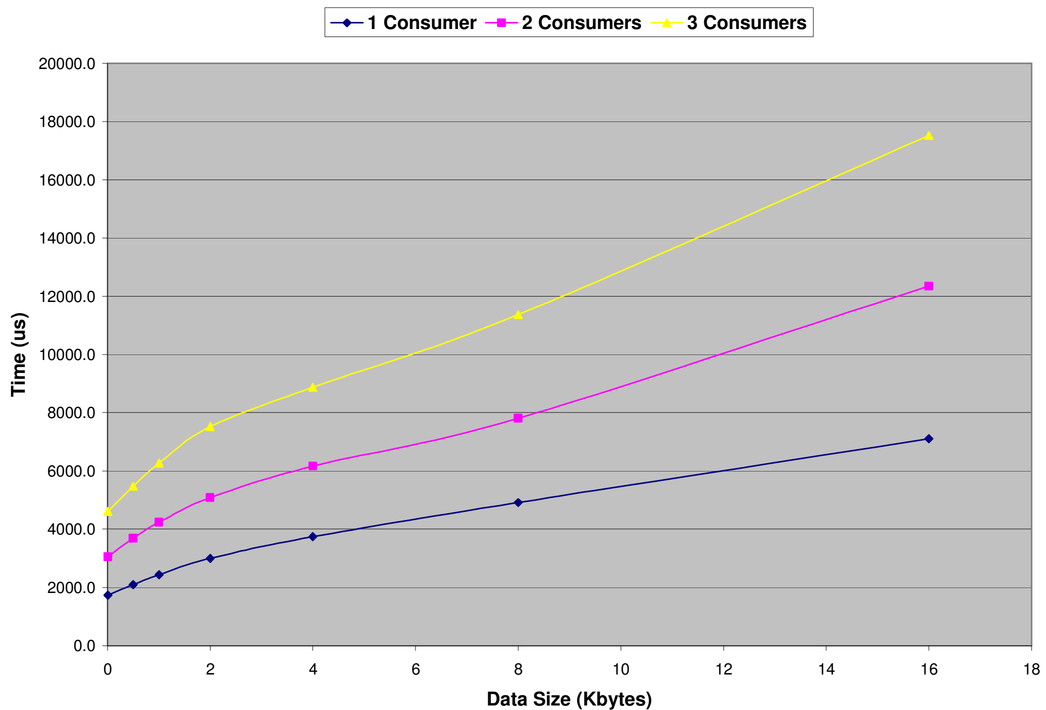

Figure 26 Performance of CDDS for different number of consumers

4.1.3.2 Experiment 3.b

Experiment 3.b investigates scalability of CDDS by measuring variation of throughput and latency of write operation in CDDS with increasing data volume in data dissemination from three producers to one consumer. Testbed is illustrated in Table 8 and Figure 27.

Table 8 Experiment 3.b Testbed Properties

	CPU	OS	ORB
CDDS Server	Pentium IV, 3 GHz	Windows XP	JacORB 2.2.3
Producers	Pentium IV, 3 GHz	Windows XP	JacORB 2.2.3
Consumer	Pentium IV, 3 GHz	Windows XP	JacORB 2.2.3

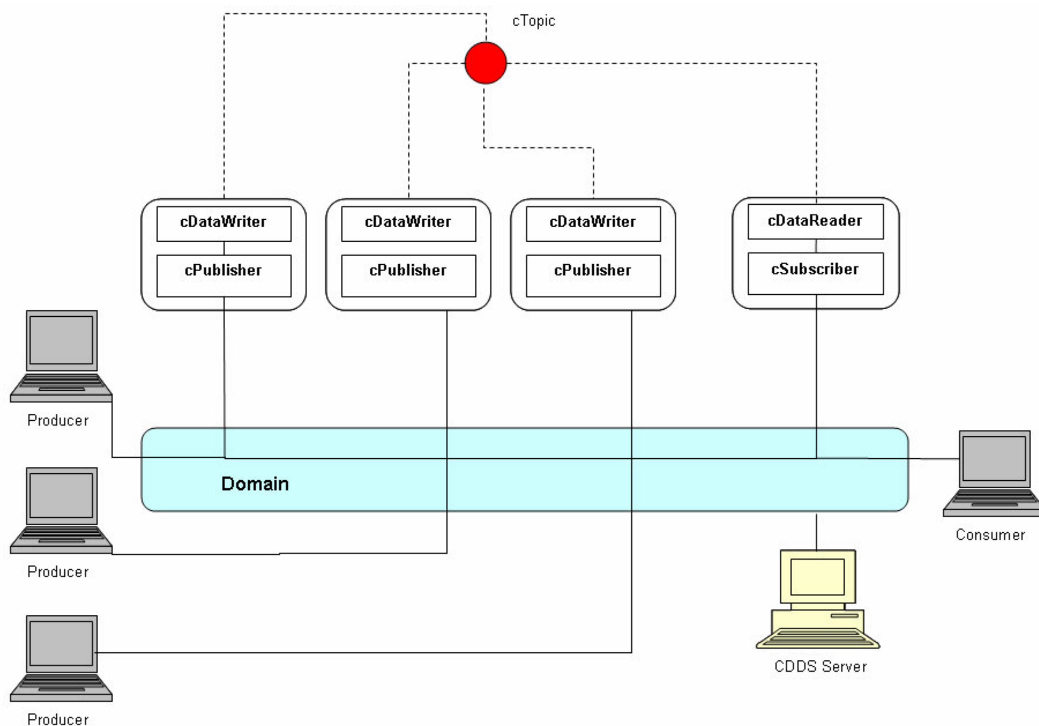

Figure 27 Experiment 3.b Testbed Configuration

Table 9 shows write operation latency for 1000 calls for 1, 2, and 3 producers. Latency is measured for CDDS for increasing data size.

Table 9 Roundtrip time of CDDS for different number of producers (μs)

	0,004 Kbytes	0.5 Kbytes	1 Kbytes	2 Kbytes	4 Kbytes	8 Kbytes	16 Kbytes
1 Producer	1728.478	2100.30	2444.06	2992.71	3742.82	4913.67	7109.67
2 Producers	2681.27	2738.81	3259.313	4678.75	4970.25	7127.33	11674.19
3 Producers	3391.97	4158.87	4815.469	5478.06	8719.04	9281.81	14196.84

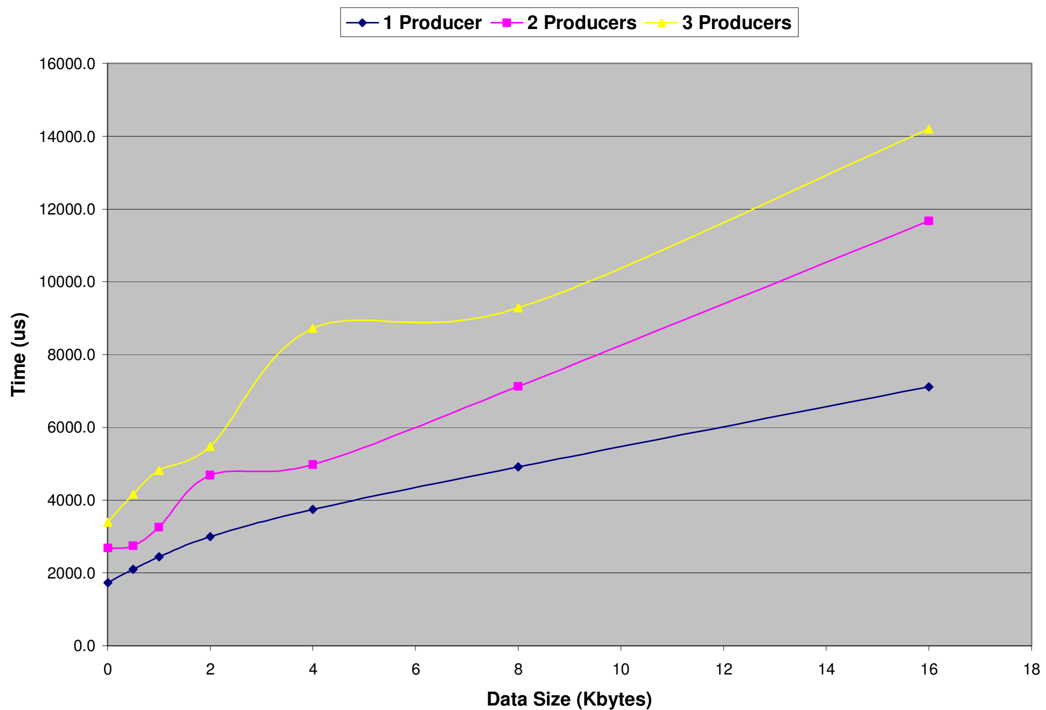

Figure 28 Performance of CDDS for different number of producers

As shown in Figures 26 and 28, latency of write operation increases linearly with increasing number of consumers or producers. This means that CDDS implementation is not scalable in the sense defined above in Section 4.1.3. The reasons are synchronous two-way function definition of DCPS entities in the DDS specification, and deployment of CDDS objects in a server.

4.1.4 Experiment 4

Experiment 4 investigates effect of BLOCK QoS policy by measuring the throughput and latency of data dissemination in CDDS with increasing data volume when blocksize value of BLOCK policy is 1,2,3,5,10,15,30. In this experiment, Experiment 1 testbed illustrated in Figure 20 and Table 2 is used.

Table 10 Roundtrip time of CDDS for different number of blocksize (μs)

	0,004 Kbytes	0.5 Kbytes	1 Kbytes	2 Kbytes	4 Kbytes	8 Kbytes	16 Kbytes
1 Block Size	1728.48	1050.78	833.87	634.57	515.89	471.39	421.17
2 Block Size	2099.77	1753.40	1362.94	937.13	778.89	736.68	644.26
3 Block Size	2446.19	1654.53	1384.94	1192.11	1037.53	884.60	826.96
5 Block Size	2992.72	2044.46	1796.46	1599.61	1374.11	1261.17	1212.61
10 Block Size	3744.85	2653.62	2370.74	2101.28	2022.19	1945.57	1877.13
15 Block Size	4914.74	3854.23	3601.74	3417.87	3280.60	3250.00	4835.51
30 Block Size	7144.70	6380.72	6072.00	5937.38	6375.41	7552.75	8855.68

Table 10 shows write operation latency for 1000 calls for increasing blocksize. Latency is measured for CDDS for increasing data size.

In Figure 29 and Figure 30, it is observed that increase in blocksize value causes a dramatic decrease in latency of write operation. The reason is reducing CORBA overhead with increasing data payload in read operation. With the blocksize value greater than 10 decreases in latency is minimal, that means data payload becomes dominant in operation calls.

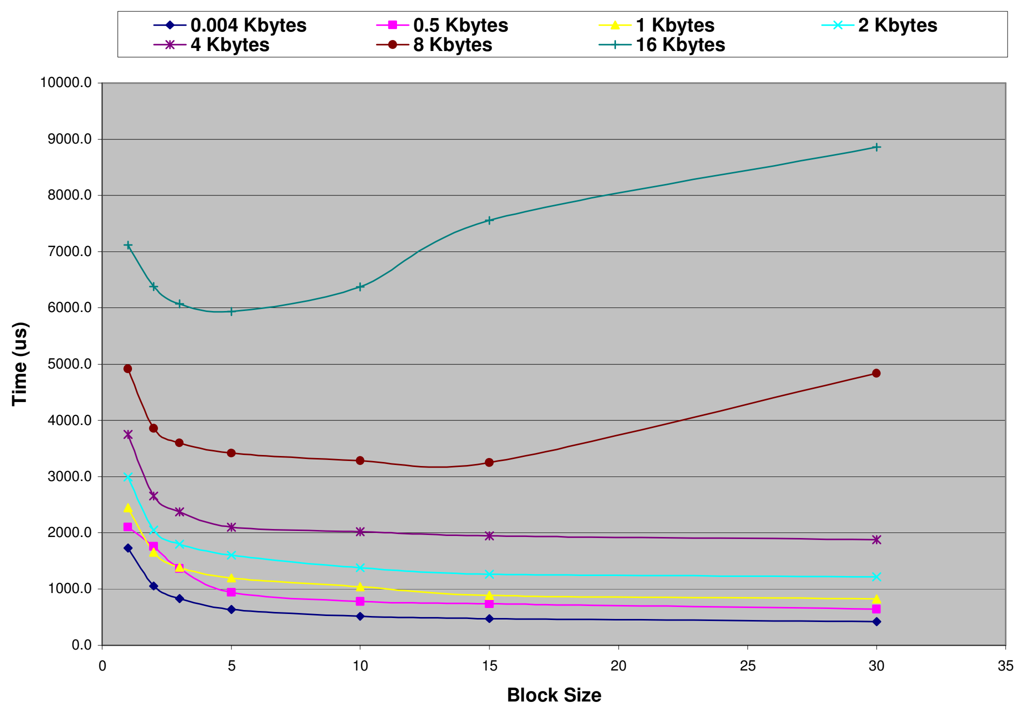

Figure 29 Performance of CDDS for different data volumes

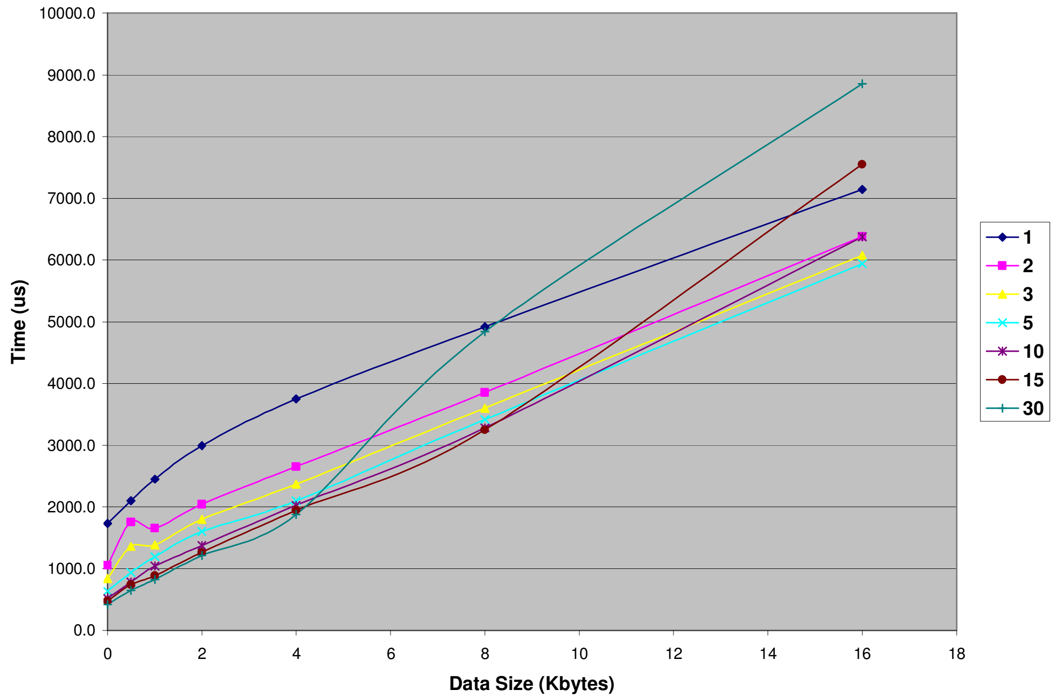

Figure 30 Performance of CDDS for different blocksize values

CHAPTER V

DISCUSSION AND CONCLUSION

A DDS implementation as a CORBA service and a QoS policy BLOCK QoS policy is proposed in this study.

CORBA Services, CORBA Event Service and CORBA Notification Service, which aims to transport publish-subscribe properties to CORBA are implemented by applying OMG specifications for CORBA Event Service and CORBA Notification Service. These specifications do not define a common API for publish-subscribe model. DDS specification defines API for publish-subscribe model while enhancing it with data-centricity property. DDS implementation transports data-centricity and decoupling properties of Data-Centric Publish-Subscribe model to CORBA by applying DDS specification and transports interoperability property of CORBA to DDS, which is not specified in specification. DDS implementation is compatible with DDS specification in API and responsibilities of DCPS entities. DDS implementation participants are deployed in a server machine as in CORBA Event Service and CORBA Notification Service.

DDS implementation uses CORBA features in data dissemination. This causes high CORBA overhead in data dissemination. Aim of BLOCK QoS policy is to minimize CORBA overhead in periodic data dissemination. In implementation of BLOCK QoS policy service blocks DataReaderListener to read data until the number of published data reaches the blocksize.

DDS implementation aims to bring a data dissemination mechanism, which is compatible with DDS specification and not cause an extra overhead. In JacORB

DDS implementation there exist incompatible points in data dissemination mechanism. As evaluated in Experiment 1, performance of DDS implementation is nearly the same with the performance of JacORB DDS implementation. Another aim of DDS implementation is to transport interoperability property of CORBA to DDS. Experiment 2 results show that interoperability between heterogeneous distributed system participants works in DDS implementation. Experiment 3 results show that DDS implementation violates scalability property of DDS specification. The reasons of violation are two-way invocation calls and deployment of DCPS participants into a server machine which is caused from the nature of CORBA Service deployment. Experiment 4 results show that implementation BLOCK QoS policy causes a dramatic decrease in latency time for periodic data distribution. Implementing BLOCK QoS policy achieves this by increasing data payload in CORBA invocation. BLOCK QoS policy differs from HISTORY QoS policy in notification of data. BLOCK QoS has blocksize value and concerns Topic and DataReader entities in the service.

In general, DDS implementation achieves data distribution as specified in DDS specification by using CORBA features and presents a QoS policy, BLOCK QoS policy, to reduce the CORBA overhead in distribution of periodically published data-objects. By using CORBA features DDS implementation transports interoperability and reliability from CORBA. On the other hand, DDS implementation violates scalability property of DDS specification.

As a future study, one-way write and read operation definitions can be added to interface definition of DataWriter and DataReader classes to increase the scalability of DDS implementation as a CORBA service.

Lack of DDS interoperability causes shortcoming in data distribution between different, heterogeneous DDS implementations. In the next versions of DDS specification, interoperability solutions should be defined.

REFERENCES

[1] H. Pinus, Middleware: Past and Present a Comparison, June 2004

[2] S. Güner, Architectural Approaches, Concepts and Methodologies of Service Oriented Architecture, August 2005

[3] Object Management Group, CORBA Specification Version 3.0.3, March 2004, http://www.omg.org/cgi-bin/doc?formal/04-03-01, last accessed August 2006

[4] Object Management Group, http://www.omg.org/, last accessed August 2006

[5] J. Zou, D.Levy, A. Liu, Evaluating Overhead and Predictability of a Real-time CORBA System, 2004 IEEE

[6] Object Management Group, Event Service Specification Version 1.2, October 2004, http://www.omg.org/cgi-bin/doc?formal/2004-10-02, last accessed August 2006

[7] Object Management Group, Notification Service Specification Version 1.1, October 2004, http://www.omg.org/cgi-bin/doc?formal/2004-10-11, , last accessed August 2006

[8] Object Management Group, Data Distribution Service for Real-Time Systems Specification, March 2005,

http://www.omg.org/technology/documents/formal/data_distribution.htm, last accessed August 2006

[9] S. Vinoski, New Features for CORBA 3.0, Communications of the ACM, October 1998

[10] JacORB, http://www.jacorb.org/, last accessed August 2006

[11] Object Computing Inc., http://www.theaceorb.com/, last accessed August 2006

[12] RTI, Build-Your-Own Middleware Analysis Guide, http://www.rti.com, last accessed August 2006

[13] D. Powell, Group Communications, Communications of the ACM, 39:4, pp. 50-97, April 1996.

[14] P. T. Eugster, R. Guerraoui, Distributed Programming with Typed Events, IEEE Software, March-April 2004

[15] H. K. Y. Leung, Subject Space: A State-Persistent Model for Publish/Subscribe Systems, September 2002

[16] P. Th. Eugster, P. A. Felber, R. Guerraoui, A. Kermarrec ,The Many Faces of Publish/Subscribe, ACM Computing Surveys, Vol. 35, No. 2, June 2003, pp. 114–131.

[17] Sun Microsystems, Java Message Service Version 1.1, April 2002, http://java.sun.com/products/jms/docs.html, last accessed August 2006

[18] Object Computing Inc., http://www.ociweb.com/products/dds, last accessed August 2006

[19] F. Allaoui, A. O. Yehdih, Implémentation de l'intergiciel DDS (Data Distribution Service), Juillet 2005

[20] E. Gamma, R. Helm, R. Johnson, J.Vlissides, Design Patterns, Addison-Weisley, 1995

[21] A. B. Arulanthu, C.O'Ryan, D. C. Schmidt, M. Kircher, J. Parsons, The Design and Performance of a Scalable ORB Architecture for CORBA Asynchronous Messaging, Middleware 2000, LNCS 1795, pp 208-230,2000

[22] R. Kindel, What Real-Time Data Distribution System Is Right for You?, AFRL Technology Horizons, August 2005

[23] M. Rogosin, Design Strategies for Real-Time Data in Distributed Systems, Data Management, June 2005