Chapter 1

This chapter will cover the fundamentals in data distribution service (DDS), explaining the concept of middleware and DDS for real-time systems.

1.1 Middleware

In distributed systems, multiple independent computers are connected on a network to cooperate in achieving the same goal. These computers can be placed with different geographical locations and have different operating systems (OS) [**?**, p. 2].
To make the development of a distributed system easier developers can use middleware, which is software between the application and the physical layers on the computer. The middleware contributes to the development as it hides the problems that can occur due to different software, hardware and OS's from each application [**?**, p. 3].

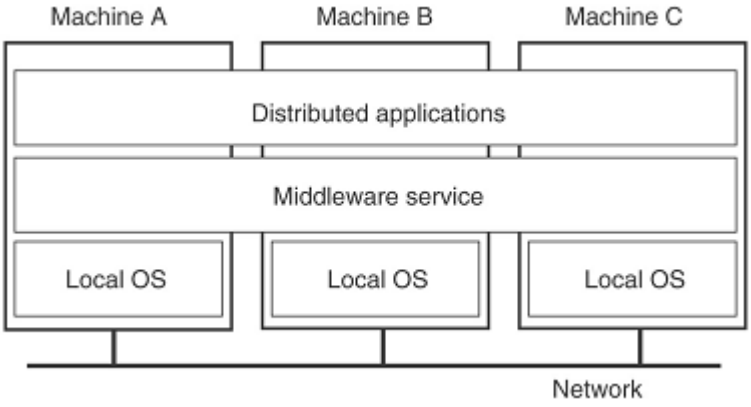

Figure 1.1: A distributed system with middleware [**?**, p. 3]

Each application is offered the same interface through the middleware layer which extends over multiple machines [**?**, p. 3]. The middleware is a principle that makes it easier for developers to scale a distributed system, as the developer can focus on the interface to the middleware and not the layers beneath. This makes the environment homogeneous as the differences in network technology, hardware architecture, OS, programming languages and geographical locations etc. is not to be taken care of by developers [**?**] [**?**, p. 68].
To support the different programming languages used in the applications, the middleware supports the Interface Definition Language (IDL). It is a standard language for defining the interfaces which enables communication between applications that do not share programming language. This means that each application transform the programming langauge into IDL and transfer it into the middleware and the middleware then transform it to the given target's applications programmering langauge. [**?**] [**?**] [**?**].
Middleware is useful when developing large and complex distributed systems as it connects the different parts with a "pipe" that makes data-sharing and communication efficient [**?**] [**?**, p. 68]. It should be considered that the middleware-software should be installed on all the involved computers, which can raise the CPU load. Depending on the type of hardware that is

used the developers should choose a sufficient middleware [**?**].

When choosing between middlewares it is important to know which standard the different middlewares are using. There is a lot of standards that the middleware can support, e.g. which network the middleware supports or if the middleware supports a medical standard for transferring images and data between devices in the medical industry [**?**].
There are various implementations of the standards e.g. OpenDDS by Object Computing Inc. which is an open source implementation. Among the commercial implementations Real-Time Innovations (RTI) has developed a middleware called Connext.

Mia - Skal der skrives mere her? Synes Christian sagde noget men har pt glemt det - ved ikke om du går lidt i dybden med det i dit afsnit? Det var noget de forskelilge standarder, mens synes vi går i dybden med det når vi forklare DDS. Lasse - nej, jeg synes det er fint som det står :) Men ja, vi mangler at skrive noget om de forskellige standarder og OMG. jeg kommer ikke rigtig ind på standarderne i mit afsnit, så vi skal have flettet det ind et sted...

A concrete implementation of this middleware in a distributed system is seen below.

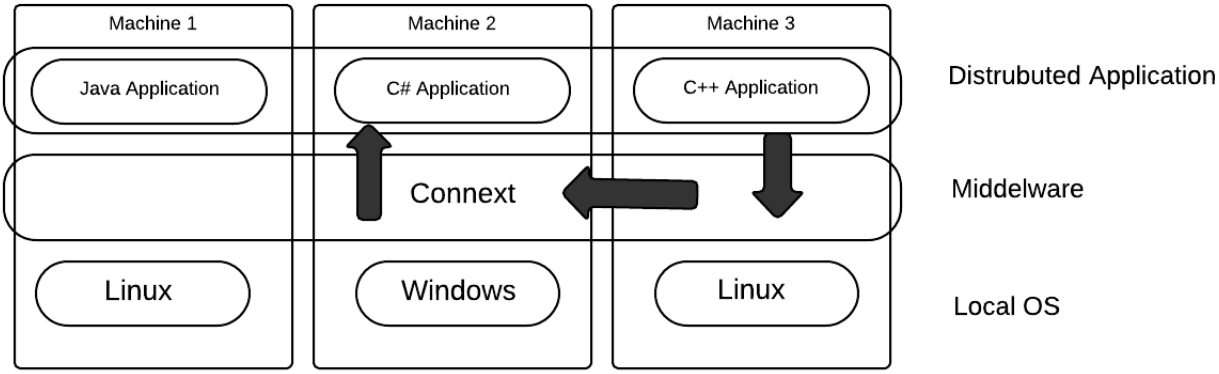

Figure 1.2: Connext in a distributed system

The figure shows, that a C++ application on Machine 3 is transferring data to the middleware. The middleware then handle and transform it, so it can be used on Machine 2, even if the programming languages and the machines are different from each other.
The communication in the middleware is performed by a publish-subscriber paradigm, called Data-Centric publish-subscribe, which is explained in details in the next section. Other forms of middelware communication paradigms is e.g. Message Passing, Message queuing, Remote procedure call and derivatives and etc. [**?**].
These communication paradigms in middleware, decouples data producers and consumers in different ways and can be categorized into three different decouplings; space, time and flow decoupling.
Space decoupling is if the producers and the consumers do not know each other, which means that they do not have a reference to each other and do not know the location of each other.
Time decoupling is if the interaction is asynchronous which means that the data being passed can be used later and not only when it is received. An example on an asynchronous communication could be a mail client which receives mails, but the mails can be read at a later time than when it arrived. A synchronous communication could be a phone call where it is only possible to talk if the recipient is answering the call.
Flow decoupling is if the data production and consumption are not blocking the flow for the producer or the consumer. This means that the producer and the consumer do not block each other when they use the data being passed between them.

1.2 Data distribution service for real-time systems

The DDS is a specification standard, approved by the Object Management Group (OMG), that facilitates data-communication through the data-centric publish/subscribe paradigm [**?**]. The

DDS standard supports deterministic data-delivery (if the underlying data-link and physical layer supports it), a large number of configuration parameters and a low overhead on memory and CPU [**?**].
The specification for DDS consists of two distinct sections; Data Logical Reconstruction Layer (DLRL) and Data-Centric Publish-Subscribe (DCPS) [**?**].

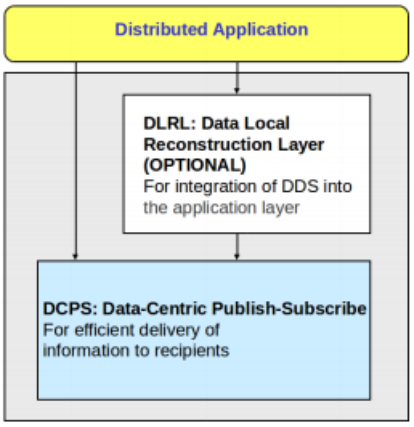

Figure 1.3: The two sections in the DDS specification [**?**]

The DLRL is an optional upper layer that allows a simpler integration of the DDS into the application layer. It provide a more natural access to data as it updates a local copy of the data, which allows the application to access the data as if local [**?**]. It outlines how an application can interface with DCPS data fields. According to Gerardo Pardo from RTI no DDS vendors supports the DLRL any more [**?**].
The DCPS is the lower layer which the application uses to communicate with other DDS-enabled applications; it provides efficient delivery of the proper information to the proper recipients [**?**]. The DCPS comprises the following entities:

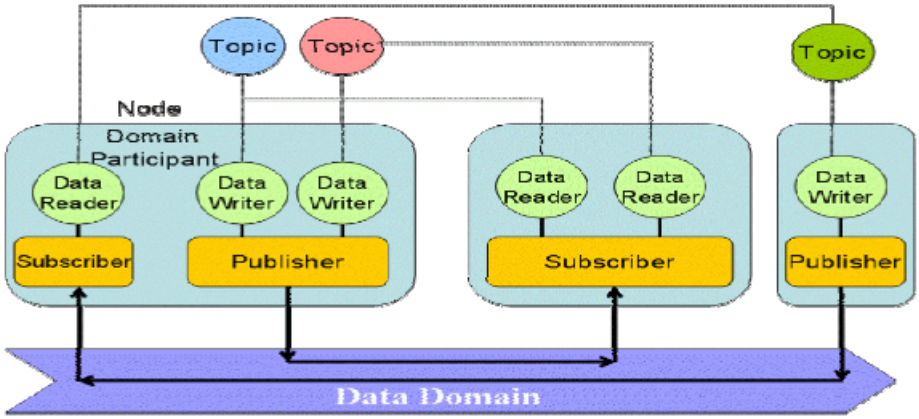

Figure 1.4: The DDS/DCPS entities [**?**]

- **Domain**
  The basic construct that binds the individual applications together and where data is send and received from. The domain is the defined communication-area in which the data is being shared.
- **Domain participant**
  This object enables the developer to specify default Quality of Service (QoS) parameters for all entities in the corresponding domain.
- **Data Writer**
  The primary access point for an application to publish data into the data domain. The Data Writer is configured with the wanted QoS-settings which enables it to write data of a particular type. All data that is going to be transferred is written by the Data Writer

with a simple write-method. When the write is executed the data will be passed to the publisher.

- **Publisher**
  The publisher is a container to group individual Data Writers; it can have many Data Writers. The publisher can also be configured with Quality of Service (QoS) parameters and can apply these parameters to all the Data Writers it contains. Furthermore the publisher is responsible for defining the topics to which subscribers can subscribe.
- **Data Reader**
  The Data Reader is the primary point to access data that has been received by a Subscriber. The Data Reader can read data from Data Writers if the data has the same type; the reader has a single Topic.
- **Subscriber**
  As for the Publisher the Subscriber is a container that groups the Data Readers; it can have many Data Reader. When data is available the Subscriber can notify the application in one of three ways:
  - Listener Callback Routine which makes DDS run our own specific software to access data as soon as it arrives.
  - Polling the Data Reader to check if data is available.
  - Conditions and WaitSets where the application access the data from the Data Reader when specified conditions are met.

  When the data has been received it can be accessed by "take()" which removes data or by "read()" which allows data to be used multiple times.
- **Topic**
  Topics are information about a single data type. The Topic of a given publisher on one computer must match the Topic of an associated subscriber - otherwise communication will not take place. A topic consists of a Topic Name (a string that uniquely identifies the Topic within the domain) and a Topic Type (the definition of the data contained within the Topic). Within the definition of the Topic Type one or more data can be chosen to be a Key. They are used by the DDS middleware to sort incoming data and provide scalability as the Key can be used to identify the data-source instead of using the Topic Name as an identifier. When Keys are used only one Topic is needed. As an example of the use of Keys refers to the Prototype-section. KEYS ARE USED IN OUR PROTOTYPE??
  [**?**] [**?**] [**?**]

The figure below shows another conceptual view on publication, subscription and topic:

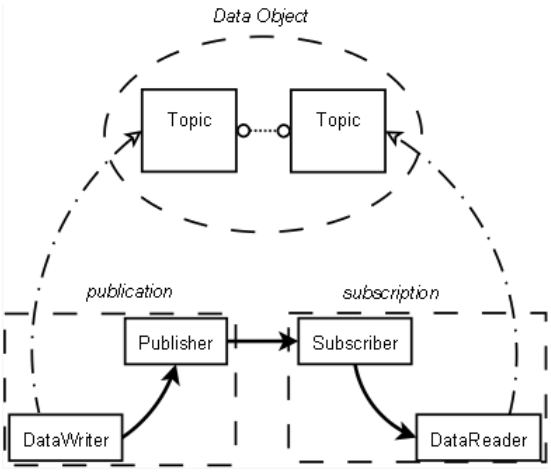

Figure 1.5: A simple conceptual view of publication and subscription [**?**]

Sending and receiving data
The application on the publishing side initiates the flow of data by writing a data value to the Data Writer. The Data Writer's publisher publishes the data. The publisher sends the data

to the associated subscription(s). Each associated Subscriber gives the received data to the Data Reader(s) that are associated with the sending Data Writer; they have the same Topic. Afterwards the application on the subscribing side retrieves the data from the Data Reader.

Quality of Service
The QoS parameters specify the data flow; the rate of publications and subscriptions, for how long data is valid ect. DDS provides QoS parameters on each of the above-mentioned entities which enables custom tailored communication as the developer can construct complex data communication patterns. This is the essence of data centricity within DDS [**?**] [**?**].

The Shapes Demo
Various DDS vendors participate in an interoperability demonstration each spring at the Object Management Group (OMG) Spring Technical Meeting. During the demo, each vendor publishes and subscribes to each others topics using a test suite called the Shapes Demo [**?**]. This leads to a quality check of the DDS standard as all the different implementations should be able to communicate if the standard has been followed.
The screen-shots below shows the Shapes Demo. Notice that the publisher (left) and the subscriber (right) need to be in the same Domain on order to communicate.

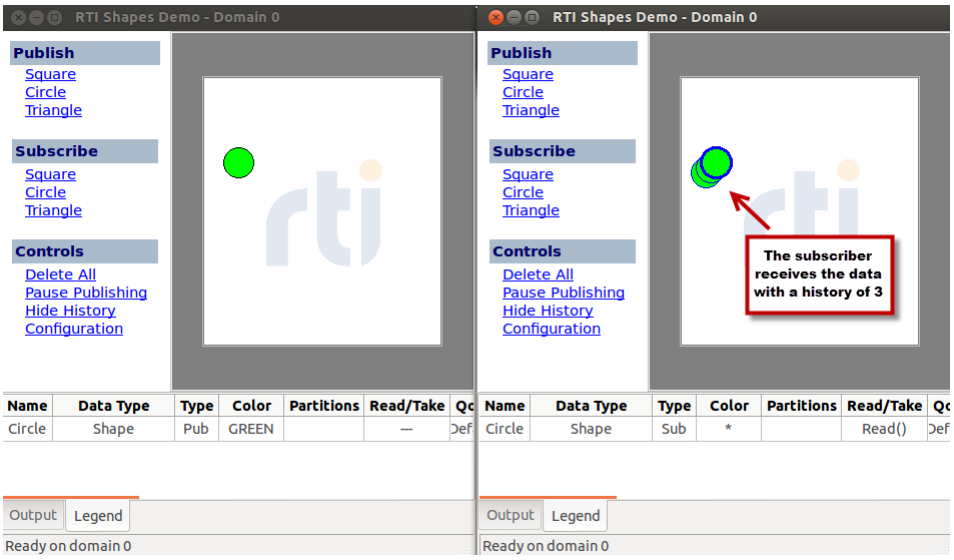

Figure 1.6: The publisher on the left is publishing data with the Topic Name "Circle" and the Topic Type "Green". The subscriber on the right is receiving the data as it subscribes to the same Topic.

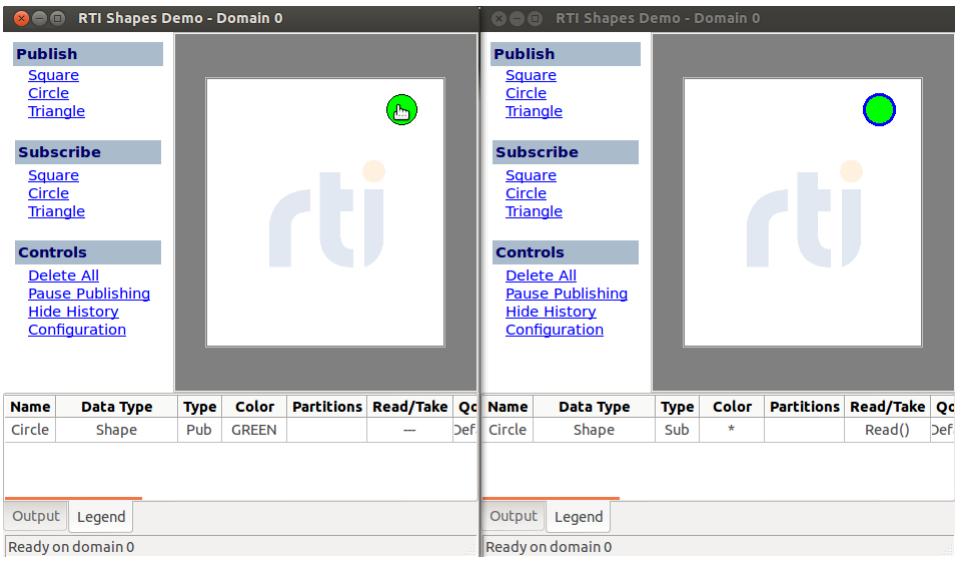

Figure 1.7: With the mouse it is possible to change the speed and the orientation of the circle and this behavior is mirrored in the subscriber on the right

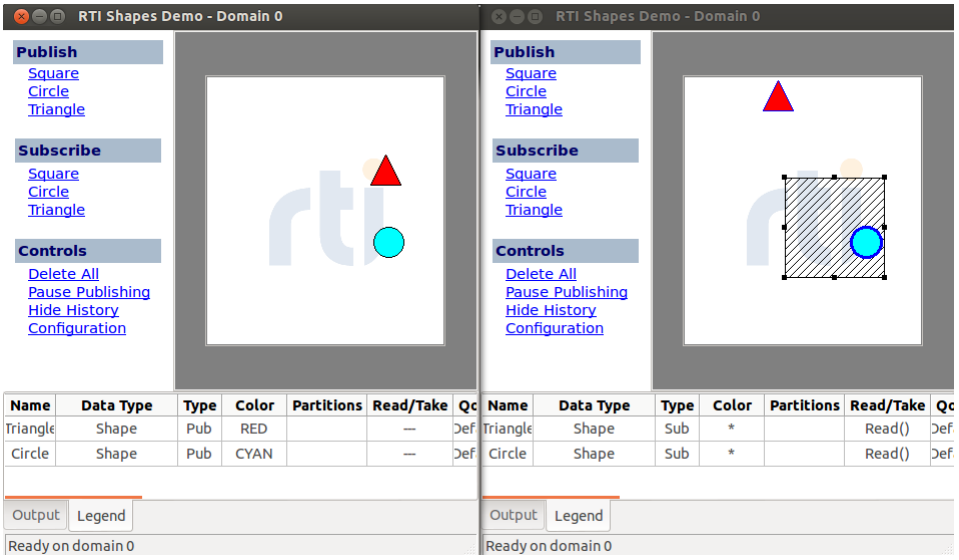

Figure 1.8: The subscriber on the right is using two QoS parameters; it only wants data every 1000 milliseconds; the red triangle is only mirrored once every second, and content-filtering; the subscriber only wants data that matches the shaded area. If the cyan-coloured data is outside the content-filter then the Data Reader will not read the data, and nothing is being provided to the application