A Study on the Distributed Real-time System Middleware Based on the DDS

Ren Haoli and Gao Yongming

Department of Information Equipment, Academy of Equipment, Beijing 101416, China

Abstract. This paper study on the distributed real-time system middleware. Presents a comprehensive overview of the Data Distribution Service standard (DDS) and describes its benefits for developing Distributed System applications. The standard is particularly designed for real-time systems that need to control timing and memory resources, have low latency and high robustness requirements. As such, DDS has the potential to provide the communication infrastructure for next generation precision assembly systems where a large number of independently controlled components need to communicate. To illustrate the benefits of DDS for precision assembly an example application was presented.

Keywords: Real-time System, DDS, middleware, Distributed system.

1 Introduction

In many distributed embedded real-time (DRE) applications have stringent deadlines by which the data must be delivered in order to process it on time to make critical decisions. Further, the data that is distributed must be valid when it arrives at its target. That is, if the data is too old when it is delivered, it could produce invalid results when used in computations [1]. This paper Presents a comprehensive overview of the Data Distribution Service standard (DDS) and describes its benefits for developing Distributed System applications. DDS is a platform-independent standard released by the Object Management Group (OMG) for data-centric publish-subscribe systems [2].

1.1 Distributed Applications

There are many distributed applications exist today, one requirement common to all distributed applications is the need to pass data between different threads of execution. These threads may be on the same processor, or spread across different nodes. You may also have a combination: multiple nodes, with multiple threads or processes on each one. Each of these nodes or processes is connected through a transport mechanism such as Ethernet, shared memory, VME bus backplane, or Infiniband. Basic protocols such as TCP/IP or higher level protocols such as HTTP can be used to provide standardized communication paths between each of the nodes. Shared memory (SM) access is typically used for processes running in the same node. It can also be used wherever common memory access is available. Figure 1 shows an example of a simple

Z. Du (Ed.): Proceedings of the 2012 International Conference of MCSA, AISC 191, pp. 1–6.
springerlink.com

distributed application. In this example, the embedded single board computer (SBC) is hardwired to a temperature sensor and connected to an Ethernet transport. It is responsible for gathering temperature sensor data at a specific rate. A workstation, also connected to the network, is responsible for displaying that data on a screen for an operator to view. One mechanism that can be used to facilitate this data communication path is the Data Distribution Service for Real Time Systems, known as DDS [3].

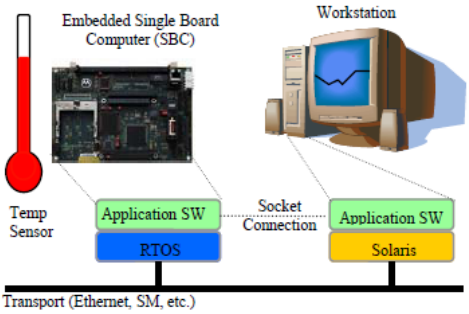

Fig. 1. Simple Distributed Application

1.2 Real-Time Publish-Subscribe Middleware DDS

Distributed systems connect devices, since devices are faster than people, these networks require performance well beyond the capabilities of traditional middleware. The Object Management Group's Data Distribution Service for Real Time Systems (DDS) Standard [4]. The OMG Data-Distribution Service (DDS) is a new specification for publish-subscribe data distribution systems. The purpose of the specification is to provide a common application-level interface that clearly defines the data-distribution service. DDS is sophisticated technology; It goes well beyond simple publishing and subscribing functions. It allows very high performance (tens of thousands of messages/sec) and fine control over many quality of service parameters so designers can carefully modulate information flow on the network [5].

DDS provides common application-level interfaces which allow processes to exchange information in form of topics. The latter are data flows which have an identifier and a data type. A typical architecture of a DDS application is illustrated in figure 2.

Applications that want to write data declare their intent to become "publishers" for a topic. Similarly, applications that want to read data from a topic declare their intent to become "subscribers". Underneath, the DDS middleware is responsible to distribute the information between a variable number of publishers and subscribers.

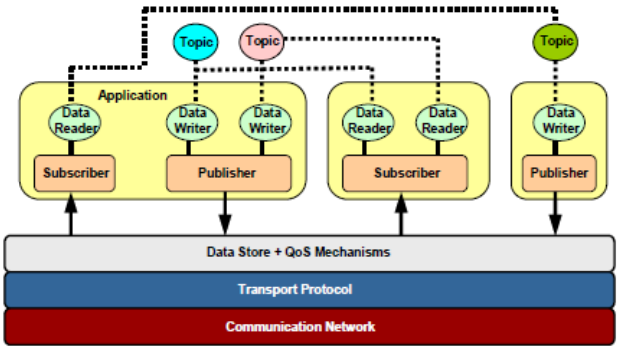

Fig. 2. Decoupling of publishers and subscribers with the DDS middleware

2 The Data Centric Publish/Subscribe Model

The DDS publish-subscribe model connects anonymous information producers (publishers) with information consumers (subscribers). The overall distributed application (the PS system) is composed of processes, each running in a separate address space possibly on different computers [6]. We will call each of these processes a "participant". A participant may simultaneously publish and subscribe to information.Figure3 illustrates the overall DCPS model, which consists of the following entities: DomainParticipant, DataWriter, DataReader, Publisher, Subscriber, and Topic. All these classes extend DCPSEntity, representing their ability to be configured through QoS policies, be notified of events via listener objects, and support conditions that can be waited upon by the application. Each specialization of the DCPSEntity base class has a corresponding specialized listener and a set of QoSPolicy values that are suitable to it. Publisher represents the objects responsible for data issuance. A Publisher may publish data of different data types. A DataWriter is a typed facade to a publisher; participants use DataWriter(s) to communicate the value of and changes to data of a given type. Once new data values have been communicated to the publisher, it is the Publisher's responsibility to determine when it is appropriate to issue the corresponding message and to actually perform the issuance.

A Subscriber receives published data and makes it available to the participant. A Subscriber may receive and dispatch data of different specified types. To access the received data, the participant must use a typed DataReader attached to the subscriber. The association of a DataWriter object with DataReader objects is done by means of the Topic. A Topic associates a name, a data type, and QoS related to the data itself. The type definition provides enough information for the service to manipulate the data. The definition can be done by means of a textual language or by means of an operational "plugin" that provides the necessary methods.

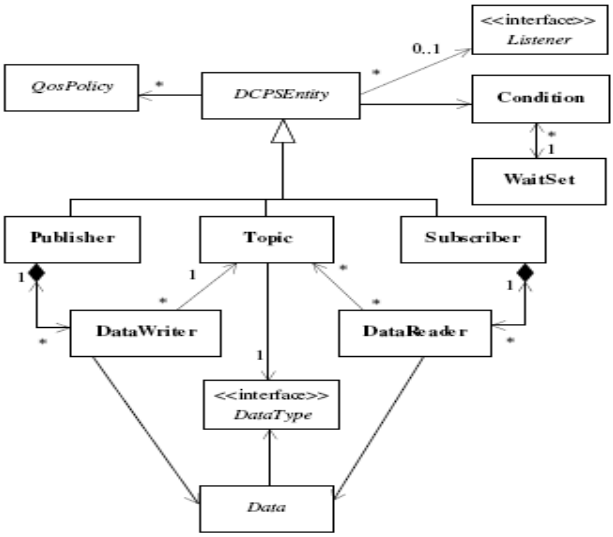

Fig. 3. UML diagram of the DCPS model

3 Example

This section presents the development of a software-based prototype for a modular active fixturing system. The prototype was developed using the commercially available DDS implementation OMG DDS 1.2 from Real-Time Innovations [7].

3.1 Design the System Framework

The system consists of a variable number of physical fixture modules, a fixture control software, a variable number of Human Machine Interfaces (HMI). For the sake of simplicity, each module consists of one linear actuator and three sensors. The former acts as the locating and clamping pin against the workpiece, while the sensors feedback reaction force, position and temperature of the contact point. The fixture modules are implemented as smart devices with local control routines for their embedded sensor/actuator devices. It is further assumed that each fixture module is configured with a unique numerical identifier and with meta information about its sensors and actuators. This way, the module is able to convert the signals coming from the sensors (e.g. a voltage) into meaningful information (e.g. reaction force in Newton) which is then published via DDS. The fixture control implements the global control routines of the fixture. It processes the data coming from the various modules and controls the movement of the actuators by publishing their desired status.

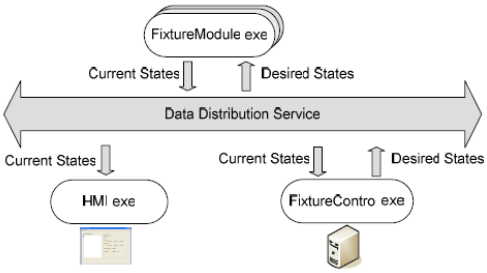

Fig. 4. Overview on the example application

3.2 Design the Data Structures

The first step of the data-centric application development is the definition of the data structures to be exchanged between the processes. In this context, there is a trade-off between efficient data transfer and flexible interpretation of data. On one hand, it shall be allowed to add fixture modules with varying capabilities, i.e. different hardware characteristics and representations of data. This makes it necessary that each module informs other systems about its capabilities which define how data has to be interpreted. On the other hand, it is not efficient to publish this meta-information with every sensor update. For this reason, it is proposed to separate actual state data from meta-information to interpret it. Therefore, two data structures are created for each capability of a fixture module. The first data structure is used for the transmission of the sensor readings or desired states for actuator devices during the manufacturing process. It is a very simple data structure that only consists of a field for the numeric module-ID and the data itself. Below an example is provided for the data structure for force sensor readings. Each attribute is defined with a data type, followed by a name.

```
struct Force {
    long module_id;
    double value;
};
```

Since this structure does not contain any information on how to use the data, an additional data structure is defined for each capability. It contains attributes describing the characteristics of the relevant capability like measuring range, resolution etc. In this prototype the meta-information only contains the measuring range for capabilities that result from the existence of sensor devices. This is further defined by attributes for the minimum and maximum measuring value, as well as the measuring unit. For the latter unique numerical constants have been defined. The following listing provides the data definitions for the capability that results from the existence of a force sensor. Similar structures have been defined for the other capabilities.

```
struct MeasuringRange {
  double min;
  double max;
  long unit;
};

struct SenseReactionForceCapability{
MeasuringRange measuringRange;
};
```

```
struct FixtureModuleCapabilityDef{
  long id;
  SenseTipPositionCapability
  senseTipPositionCapability;
  AdjustTipPositionCapability
  AdjustClampingForceCapability
  adjustClampingForceCapability;
  SenseTemperatureCapability
  senseTemperatureCapability;
};
```

Based on these data type definitions, the source code for all the subscribers, publishers, data readers and data writers are automatically generated in the specified programming language. These classes have to be used by the application programs that implement the fixture modules, the fixture control and any other participating system like monitoring applications.

4 Conclusions

In this paper, a new standard for data-centric publish-subscribe communication has been presented and put in context to the development of next-generation precision assembly platforms. The standard is called Data Distribution Service and is particularly targeting real-time applications which need to manage resource consumption and timeliness of the data transfer. DDS allows platform-independent many-to-many communication and alleviates a number of common problems which are of particular interest for the development of distributed assembly systems.

References

1. Cross, J.K., Schmidt, D.C.: Applying the Quality Connector Pattern to Optimize Distributed Real-time and Embedded Middleware. In: Rabhi, F., Gorlatch, S. (eds.) Patterns and Skeletons for Distributed and Parallel Computing. Springer Verlag (2002)
2. Object Management Group, Data Distribution Service for Real-Time Systems, Version 1.2 (June 2007), http://www.omg.org
3. Joshi, J.: A comparison and mapping of Data Distribution Service (DDS) and Java Messaging Service (JMS), Real-Time Innovations, Inc., Whitepaper (2006), http://www.rti.com
4. Real-Time CORBA Specification Version 1.2, Object Management Group (January 2005), http://www.omg.org/cgi-bin/apps/doc?formal/05-01-04.pdf
5. Joshi, J.: Data-Oriented Architecture, Real-Time Innovations, Inc., Whitepaper (2010), http://www.rti.com
6. Richards, M.: The Role of the Enterprise Service Bus (October 2010), http://www.infoq.com/presentations/Enterprise-Service-Bus
7. Data Distribution Service for Real-Time Systems Specification Version 1.1, Object Management Group (December 2005), http://www.omg.org/docs/formal/05-12-04.pdf