INTECH
open science | open minds

OPEN ACCESS
ARTICLE

International Journal of Advanced Robotic Systems

Designing High Performance Factory Automation Applications on Top of DDS

Regular Paper

Isidro Calvo[1,*], Federico Pérez[1], Ismael Etxeberria-Agiriano[2] and Oier García de Albéniz[1]

1 Department of Systems Engineering and Automatic Control, University of the Basque Country (UPV/EHU), Spain
2 Department of Computer Languages and Systems, University of the Basque Country (UPV/EHU), Spain
* Corresponding author E-mail: isidro.calvo@ehu.es

Received 24 Jul 2012; Accepted 7 Mar 2013

DOI: 10.5772/56341

Abstract DDS is a recent specification aimed at providing high-performance publisher/subscriber middleware solutions. Despite being a very powerful flexible technology, it may prove complex to use, especially for the inexperienced. This work provides some guidelines for connecting software components that represent a new generation of automation devices (such as PLCs, IPCs and robots) using Data Distribution Service (DDS) as a virtual software bus. More specifically, it presents the design of a DDS-based component, the so-called *Automation Component*, and discusses how to map different traffic patterns using DDS entities exploiting the wealth of QoS management mechanisms provided by the DDS specification. A case study demonstrates the creation of factory automation applications out of software components that encapsulate independent stations.

Keywords Factory Automation, Robot Programming, Industrial Communications, Middleware, DDS

1. Introduction

Today's factory automation applications are growing in size and complexity. In modern factory automation applications, industrial devices such as robots, Programmable Logic Controllers (PLCs) and Industrial PCs (IPCs) are changing from preprogrammed, stationary systems into machines capable of modifying their behaviour, based on interactions with the environment and other devices. Changes have also come about from users demanding higher efficiency, functionality and performance in the control of industrial plants [1]. Moreover, modern applications involve a large amount of information and an increasing number of industrial devices [2]. In this scenario, the integration of both horizontal and vertical communication is becoming a critical and complex issue [3]. Traditionally, factory automation communications have been organized in layers according to the so-called automation pyramid (Figure 1). This approach allows different types of technologies to be used in different layers of the automation pyramid. Historically, fieldbuses like Profibus or CAN were used at device and control levels, whereas typical office networks like Ethernet or Wi-Fi were preferred at plant level. Nowadays, these latter technologies, namely Switched Ethernet or Ethernet-modified architectures, are being increasingly used even at the bottom layers of the automation pyramid. This is due to several reasons [4]: (1) decentralization of applications is shifting intelligence to the distributed components; (2) vertical integration is gaining importance and (3) the adoption of IT standards is bringing about important cost reductions. Another driving force in factory automation results from the adoption of more powerful computing platforms by most industrial device vendors. These new platforms allow the introduction of modern software engineering techniques that improve performance and flexibility while, at the same time, reduce costs and time-to-market. However, modern factory automation applications require the integration of heterogeneous devices that execute different hardware platforms and Operating Systems (OS) [2].

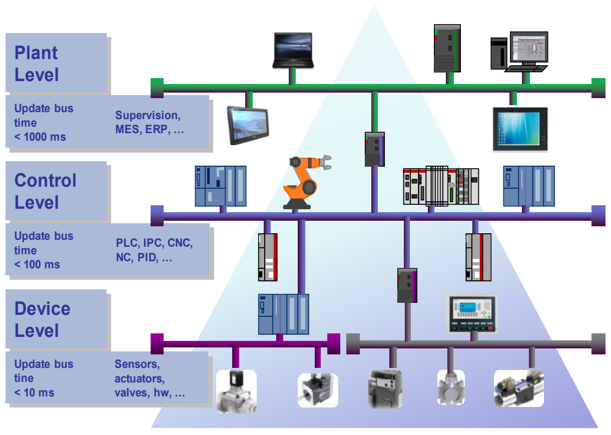
Plant Level
Update bus time < 1000 ms
Supervision, MES, ERP, ...
Control Level
Update bus time < 100 ms
PLC, IPC, CNC, NC, PID, ...
Device Level
Update bus tine < 10 ms
Sensors, actuators, valves, hw, ...

Figure 1. Automation pyramid

In this context, middleware technologies provide high-level abstractions and interfaces to facilitate integration, reuse and development [5]. Some middleware specifications, such as CORBA [6, 7], ICE [8], OPC [9, 10], Web Services [11, 12] and DDS [13, 14] simplify the development of distributed factory automation applications. Even though middleware is typically organized in a hierarchy of several layers [15], most of the above cited specifications only address communication issues. Some authors have defined higher-level middleware architectures that provide new abstractions and services [16, 17] addressed to specific domains and applications.

In particular, the OMG Data Distribution Service (DDS) is an emerging middleware specification that follows the publisher / subscriber paradigm and implements a broad set of mechanisms to specify and manage Quality of Service (QoS) requirements in real-time applications. It has been used in several application domains such as avionics or defence. Although a few industrial automation systems have also been implemented with DDS, this standard has not yet gained much significance in factory automation applications. There are two main reasons for this fact: (1) It is a relatively new standard and (2) there is still a lack of development of environment support to facilitate enhancing de facto standard control systems like PLCs with DDS-based communication without deep programming knowledge [14]. In addition, although DDS is a very powerful technology it may prove complex to use for non-experienced users.

This paper fills the gap by providing some guidelines and abstractions to use DDS as a virtual software bus in the control layer for a new generation of factory automation devices (PLCs, IPCs and robots). More specifically, this work presents the generic architecture of a DDS-based component capable of providing QoS communication requirements in factory automation applications. Also, it identifies different types of traffic patterns among the controllers and maps them over DDS entities exploiting its wealth of mechanisms for QoS management.

The design of applications that use automation components allows concepts such as maintainability, reusability, flexibility and reconfigurability to be consolidated. For some time attempts have been made to integrate component architectures in industrial automation systems. One alternative is to use the IEC61499 standard [18] with component-oriented technologies. This standard proposes an open architecture for designing distributed automation applications capable of introducing modularity, reconfigurability and flexibility into distributed control systems. Vyatkin [19] defines the term IMC (Intelligent Mechatronic Component) for automation software design. This concept is close to the "Technological Module" introduced by Profinet-CBA [20]. Cengic et al. [21] introduce the term "automation component" over IEC61499, inspired by VHDL components. Black et al. [22] implemented IMCs as IEC61499 FBs (Function Blocks) within an approach to create multiagent systems that introduce autonomous and collaborative behaviour according to holonic principles.

The layout of the paper is as follows. Section 2 discusses several related works in the field. In particular, it analyses different middleware architectures used in robotics, as well as the most relevant kinds of traffic found in factory automation applications. Section 3 is the heart of the paper. It presents the design of a generic DDS-based component, the so called *Automation Component* and provides certain guidelines to map the different traffic patterns found in factory automation into DDS entities exploiting the richness of QoS management mechanisms provided by the DDS specification. Section 4 illustrates the use of this generic component in factory automation applications by means of a simple case study. Finally, Section 5 draws some conclusions.

2. Related work

This section describes some middleware architectures used in automation and robotic domains and discusses the major communication needs of factory automation applications.

2.1 Middleware for robotics and factory automation applications

Different architectural paradigms have been proposed to support the development of distributed and concurrent systems. In particular, object-oriented, component-based and service-oriented approaches are among the most relevant paradigms for the integration of software artefacts that require interprocess communication and event synchronization. Ref. [23] discusses its application to robotics. There have been several initiatives based on these paradigms to create frameworks and libraries for industrial robotic applications. Most of them are aimed at solving the following challenges [24]:

- Simplify the development process
- Ease integration
- Support communications and interoperativity
- Provide abstractions that hide heterogeneity and use the available resources efficiently
- Offer frequently needed services

These frameworks have been analysed in [24, 25] and Orocos [26], Miro [27], Orca [28], Player/Stage [29], Aseba [30] and Robot Operating System (ROS) [31] are the most representative ones. Some of them are restricted to specific types of robotic applications such as mobile robots (e.g., Miro). One interesting initiative, Nerve [32], has developed a lightweight middleware aimed at networked social robotic applications. However, the robotics community still lacks well-accepted common software architectures, middleware and shared low-level functionality. In addition, since most of the previous initiatives have been developed within academic environments, the authors write software solutions that tend to be firmly based on their own architecture, middleware and robots [33]. Finally, these frameworks typically focus on solving the requirements of robotic applications only, without considering the integration of other industrial devices like PLCs or IPCs.

Previous initiatives have been built on top of different distribution middleware technologies. In particular, CORBA is used by Orocos and Miro, Orca is built on top of ICE, Nerve uses DDS, whereas the rest follow proprietary client/server communication approaches. Most of the previous frameworks barely implement the client/server paradigm. Only Orca, ROS and Nerve supply publish/subscribe one-to-many communications, which improve the overall performance of data exchanges and decouple discovery and communication tasks [34]. However, in most factory automation applications both client/server and publisher/subscriber paradigms should be combined.

Other authors propose alternative approaches for factory automation applications, such as using Service Oriented Architectures and Web Services [12, 35], but as shown in [36] these technologies may not provide an adequate performance in real-time applications like those usually found in factory automation.

It may be concluded from the previous discussion that simple approaches on top of more efficient technologies could be helpful in building factory automation applications with open industrial devices. When compared with previous approaches, the use of DDS as distribution middleware in the control layer of the automation pyramid provides several advantages. On the one hand it is a high performance middleware with low overhead, which is capable of supporting real-time communication on top of the TCP/IP stack. In addition, DDS is an open standard promoted by the OMG [13] that has been designed to work in different programming languages, on top of different computing platforms and operating systems providing a certain degree of independence. However, abstractions on top of DDS are needed in order to ease the creation of factory automation applications. This work, as well as [37], goes in this direction. More specifically, DDS is used directly as a communication backbone to connect software components based on concepts adopted from the IEC61499 standard [18].

2.2 Traffic types in factory automation applications

This subsection discusses the main communication needs in the control layer of the automation pyramid. This layer connects different types of process controllers (mainly IPCs, robots and PLCs). In factory automation applications, it is necessary to provide at least three different communication types, associated with different operations [4].

1. **Periodic data communication**. This covers the needs of the distribution of process data, typically with soft real-time constrains. It is the most relevant communication type used to execute distributed control algorithms among several devices within a system. Periodic data are used to deliver the transfer relationship of the automation signals. Process variable update times may reach up to several tens of milliseconds (10, 50, 100ms). Usually, a certain jitter in the update times of this type of data may be acceptable if kept below a certain threshold (typical admissible values are around ten percent of the period). The maximum accepted latency depends on the kind of application and is typically linked to the value of the distributed task periods.
2. **Aperiodic data communication**. Typically used to distribute sporadic data, alarms and events. This information must be delivered in a way that satisfies

certain real-time constraints such as a maximum latency, which varies quite a lot since it is closely related to the nature of the applications. The maximum acceptable jitter for transmissions of this kind of data must also be below a specific threshold.

3. **Non-time-critical communication**. This is required by operations involving configuration, parameterization and diagnostics of distributed devices; sending reference signals for process control and providing eventual access to process data. These operations support configuration and monitoring aspects of industrial control applications. Even though this kind of communication is not time-critical, it may require sending large amounts of data such as configuration, report or log files.

Each of the previous traffic types adapts best to a particular communication paradigm. Periodic and aperiodic data communications may be distributed more efficiently by using the Publisher/Subscriber paradigm, ensuring that certain QoS constraints are met. In such cases the same information is frequently sent to a set of recipients. However, the Client/Server paradigm is better suited for most non-time-critical operations, which may be synchronous (a Client makes a request to a Server, which provides a response) or asynchronous (a Client makes a request to a Server, which is processed without response).

In this paper DDS has been chosen as the communication backbone since it is a high-level, high-performance standardized middleware that follows the Publisher/Subscriber paradigm. In addition, it provides platform/programming language independence. As shown in [34, 38] the use of the Publisher/Subscriber paradigm is well suited for distributed real-time systems like those found in factory automation applications. Moreover, DDS is a unique middleware that provides a rich set of parameters to control QoS constraints, such as transport priorities, deadlines, reliability or liveliness, as described in next section. However, since some typical operations found in factory automation (both request/response and request/non-response) adapt best to the Client/Server paradigm they should be adequately mapped in terms of DDS with no interference from other more stringent operations.

3. Building factory automation applications with DDS

This section presents the main contribution of the paper, consisting of the internal design of a DDS-based Automation Component for building factory automation applications. It also describes the mapping of the different types of communication needs in this domain in terms of DDS entities and QoS parameters. The section starts with a short overview of the DDS fundamentals.

3.1 DDS Fundamentals

The Data Distribution Service for real-time Systems (DDS) [13] is an emerging specification released by the Object Management Group (OMG). It provides a high performance platform-independent middleware for Data Centric Publish/Subscribe many-to-many communications. There exist several implementations of DDS that allow different programming languages (mainly C, C++ and Java) and general purpose OSs, like Windows, Linux and some RTOS to be combined. Open source and community distributions of DDS products, such as OpenDDS [39] or OpenSplice [40] are available.

DDS defines a virtual Global Data Space (see Figure 2), which is accessible to all applications: Publishers produce/write data, which are consumed/read by subscribers. Publishers and subscribers are decoupled in time, space and synchronization. For example, late joiners may obtain data issued before their activation even if the application that produced it has already finished or crashed. This behaviour must be properly configured by tuning the QoS properties. Data are described in a platform independent way, inherited from the CORBA specification, the so-called IDL (Interface Definition Language).

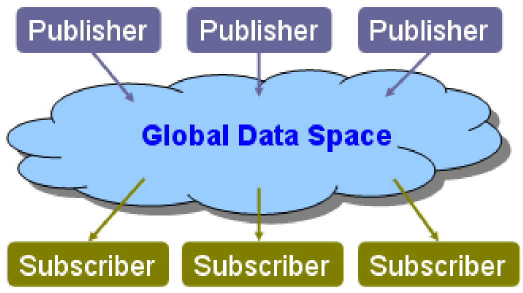

Figure 2. DDS Global Data Space

DDS also defines the Data-Centric Publish-Subscribe (DCPS) model, a topic-based API involving several entities [13] as represented in Figure 3. These entities are described below.

- **Topic** – An information unit that can be produced or consumed, identified by a name. It allows anonymous and transparent communications. Topic instances are associated with a key, defined by IDL and a set of QoS parameters.
- **Domain** – A communication context, which provides a virtual environment, encapsulating different concerns and thereby optimizing communications.
- **Domain Participant** – An entity taking part in an application within a domain.
- **Data Writer** – An entity intending to publish a Topic, providing type-safe operations to write/send data.
- **Data Reader** – An entity used to subscribe to a Topic, providing type-safe operations to read/receive data.

- **Publisher** – An entity created by a Domain Participant to manage a group of Data Writers.
- **Subscriber** – An entity created by a Domain Participant to manage a group of Data Readers.

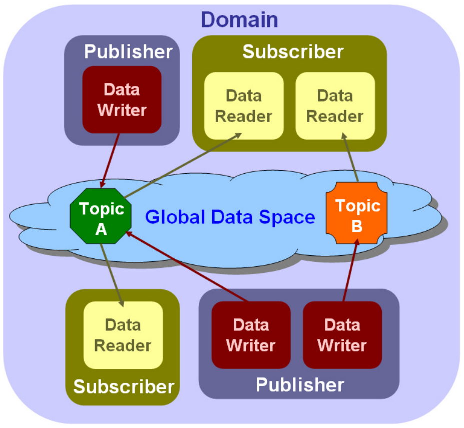

Figure 3. Major DDS entities and their interactions

One of the key features of DDS is the availability of a broad number of mechanisms that allow the management of QoS policies. These policies are configured at the different DDS entity levels (topics, data writers, data readers, publishers and subscribers). Table 1 shows the available parameters for specifying and managing the QoS policies in the current DDS specification. These parameters must be publisher/subscriber compatible under an RxO (Request vs. Offered) contract.

DDS has been adopted in many application domains. In particular, a few works deal with the use of DDS in factory automation applications. For example, in [14] a modular active fixturing system built with DDS has been presented. Also, DDS is analysed in [37] as a communication backbone for IEC61499 communication SIFBs [18]. Other works analyse the use of DDS in Electrical Substation Automation Systems [38].

DDS QoS policies	
Deadline	Ownership Strength
Destination Order	Partition
Durability	Presentation
Durability Service	Reader Data Lifecycle
Entity Factory	Reliability
Group Data	Resource Limits
History	Time-Based Filter
Latency Budget	Topic Data
Lifespan	Transport Priority
Liveliness	User Data
Ownership	Writer Data Lifecycle

Table 1. List of most relevant DDS QoS parameters

3.2 Internal design of the DDS-based Automation Component

One of the major benefits of using DDS as a communication backbone is that it provides a hierarchical set of entities with a rich set of QoS parameters, which, when configured appropriately, may be used for fine tuning the communication needs of the applications. Even though this characteristic makes DDS a very powerful and high performance technology, it may prove complex to use by non-experienced users, especially due to the variety of combinations of QoS parameters. This subsection presents the internal design of a DDS-based *Automation Component* architecture that includes all the functional operations provided by every component, as well as the necessary entities to use DDS as a communication backbone for factory automation applications. Thus, programmers will be able to create factory applications by connecting *Automation components* that encapsulate the functionality of factory automation controllers, such as PLCs, robots or IPCs, to the DDS software bus, as shown in Figure 4.

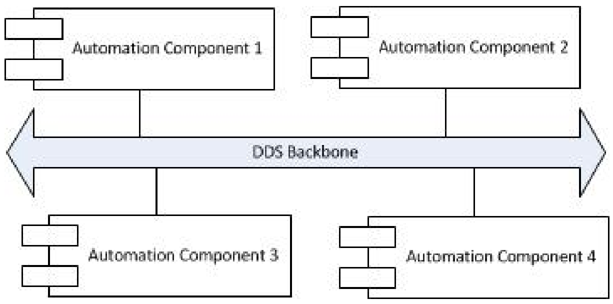

Figure 4. Building factory applications with DDS-based Automation Components

The DDS-based *Automation Component* is based on a hierarchical skeleton that could be semi-automatically generated with model driven engineering (MDE) techniques. Figure 5 presents the UML class diagram for this component. Application programmers must insert the functionality of the component by instantiating several objects of the *FunctionalWrapper* class. These objects implement the services that require access to the DDS backbone as described above through objects of the *DDS:Topic* class. The Figure also shows the hierarchy of DDS classes needed to access the *DDS:Topic* objects in every component. Finally, according to the type of service, each DDS entity is configured with different DDS QoS policies, as discussed in the following subsection.

For example, since DDS is capable of managing the distribution priorities of the data inside the Topics: (1) aperiodic communications for delivering sporadic data, alarms and events may be assigned the highest priority levels; (2) periodic data will be distributed by using medium to high priority values, thereby guaranteeing basic update times and (3) non-time-critical communications can be mapped into topics with lower priority values, improving the traditional best-effort paradigm. However, due to the nature of TCP/IP protocols, some QoS aspects will not be enforced but just a hint given to the underlying infrastructure.

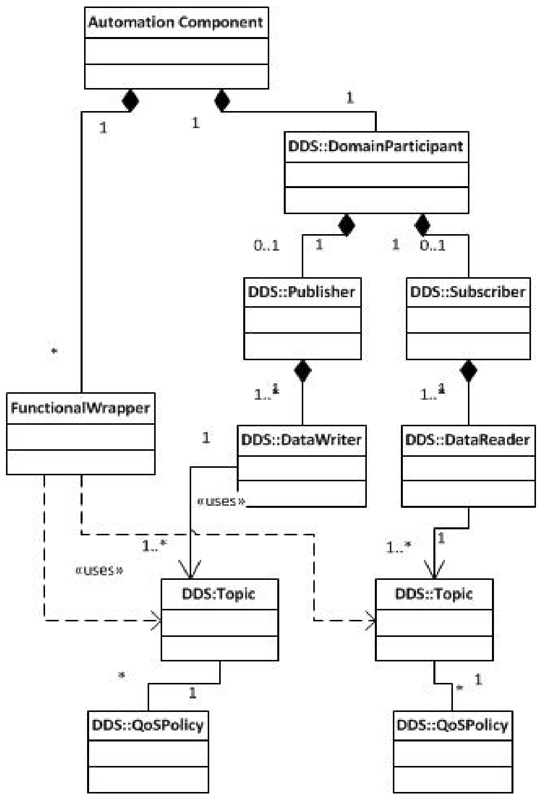

Figure 5. UML class diagram for the DDS-based Automation Component

3.3 Mapping factory automation communications over DDS

Even though DDS allows a large number of QoS policies to be configured, the authors have tried to simplify the configuration of the middleware by focusing only on those considered most relevant in factory automation applications. A short description of these parameters follows:

- **Deadline**: significant for periodic and aperiodic real-time critical communications. This parameter represents the maximum separation between two topic updates.
- **Latency budget**: significant for real-time communications. It specifies the maximum acceptable delivery delay from the Data Writer to the Data Reader.
- **Reliability**: specifies whether to attempt to deliver all samples or if it is acceptable not to retry any failed data propagation.
- **History**: specifies the number of samples kept by the middleware engine.
- **Transport priority**: integer value used to specify the priority of the underlying transport.
- **Durability**: specifies if data should outlive their writing for late joiners. Provided variants include *volatile*, *transient* and *persistent*.

This subsection also presents guidelines for specifying the DDS QoS policies in order to map the different types of traffic found in factory automation applications, as identified in Section 2. This kind of application involves different operations, each of them requiring particular traffic flows. Thus, since DDS is a Publisher/Subscriber based technology, all Client/Server operations require mapping into the Publisher/Subscriber paradigm. Efficiency will be lost, but only on the request/response operations, which frequently require lower priorities. This approach allows middleware to be left to deal with the communication requirements. A discussion about how to map factory automation traffic in terms of DDS follows. Table 2 summarizes this mapping.

- **Distribution of Aperiodic Station Events**: these adapt best to the Publisher/Subscriber many-to-many paradigm, i.e., there may be several *Automation components* that produce/receive data. Each alarm or event (or a group of them, depending on the desired granularity of the system) should be mapped into a DDS Topic. This kind of traffic must be delivered at the highest transport priority (*DDS:QoSPriority*) and in a reliable manner (*DDS:QoSReliability*).
- **Distribution of Periodic Variables:** these also adapt best to the Publisher/Subscriber many-to-many paradigm. Each periodic variable (or a group of them) is mapped into a DDS topic. Typically, in most automation systems only the last sample is relevant. Thus, the history window (*DDS:QoSHistory*) should be assigned the *keep last* value and the reliability parameter (*DDS:QoSReliability*) should be set to *best effort*. Priorities for every topic (*DDS:QoSPriority*) should be set in the range of medium to high according to the urgency in the distributed application. Due to the cyclic nature of this kind of traffic, the Deadline (*DDS:QoSDeadline*) should be enforced with a value marginally higher than the sampling period or maximum process time.
- **Invocation of Request/No response Services**: this kind of traffic adapts best to the Client/Server paradigm, since there is only one producer and one receiver of information. A DDS topic is used per service and the destination Device Component must be identified in the body of the Topic. Since no response is required, reliability (*DDS:QoSReliability*) must be enforced. In this case, the priority (*DDS:QoSPriority*) is set to low, since in most cases they are non-time-critical operations.
- **Invocation of Request/Response Services**: unlike the previous communication type, this is a synchronous Client/Server case, where the client remains blocked until a response from the server is received. This behaviour can be achieved over a Publisher/Subscriber framework by using a pair of topics: one for the request and another one for the response. The client writes the request into the request-topic and performs a blocking read upon the response-topic. The priority of this traffic

(*DDS:QoSPriority*) is set to the lowest, since these services are typically used for non-time-critical operations.

	Services			
	Aperiodic Station Event	**Periodic Variable**	**Request/ No Resp.**	**Request / Resp.**
Deadline	-	Maximum process time	-	-
Latency Budget	Low	Medium	-	-
Reliability	Reliable	Best effort	Reliable	Reliable
History	Infinite	Keep last	Keep last	Keep last
Transport Priority	Highest	High	Low	Lowest
Durability	Persistent	Volatile	Volatile	Volatile

Table 2. Recommended QoS parameter values for every kind of factory automation traffic

4. Case study

This section presents a case study that illustrates the application of the above-proposed DDS mapping. It is based on a factory automation cell (see Figure 6) in which four stations assemble different parts over a pallet. More specifically, the assembled final piece is composed of a base, a bearing, a shaft and a lid.

4.1 Description of the case study application

The cell is capable of assembling different types of pieces. Individual stations are capable of distinguishing the type of piece by means of a piece code located at the bottom of the pallet and execute different operations accordingly. Each station has a conveyor belt to move the pallet across the station. After correctly processing one piece, the conveyor moves the pallet with the piece to the next station, in order to continue with the assembly process. The four stations involved in the case study are the following:

1. **Loader.** This station is responsible for correctly locating the base over the pallet placed on the conveyor belt.
2. **Robot.** This station obtains the bearing and the shaft from the feeding points of the station and fits them to the base placed on the pallet.
3. **Sealer.** This station puts the lid over the assembled part, which was previously located by the robot on the pallet.
4. **Storage Cell.** This station picks up the assembled piece from the pallet and places it in the storage area.

There is also a supervisory control application that initializes the controllers of the stations at the start-time and monitors the process status at the run-time.

The control application for the factory automation cell has been created with five automation components that encapsulate the functionality of each of the four stations described above, as well as the supervisor.

Figure 6. Factory automation cell

These automation components have been distributed over three physical devices connected by an Industrial Ethernet, as shown in Figure 7. One device holds the supervisory automation component whereas the other two devices are IPCs that play the role of station controllers. Each IPC contains the software of two automation components that encapsulate one physical station. In the proposed case study Controller 1 holds the Loader and Robot automation components whereas Controller 2 holds the Sealer and Storage automation components.

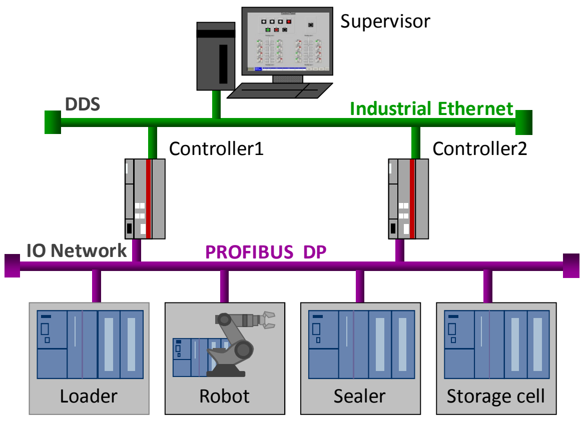

Figure 7. Topology of the case study application

As shown in Figure 7, there are two industrial communication networks involved in the system: (1) Profibus-DP fieldbus, which is used by the controllers to access the I/O signals in each station in a multimaster configuration and (2) Industrial Ethernet network, which implements the control communication network and links the controllers and the supervisory computer. This is the network in which the DDS virtual bus is implemented.

4.2 Information exchange and mapping into DDS topics

The case study presented above requires both synchronization of the stations in order to follow the processing sequence and delivery of the messages simultaneously to several devices (e.g., the following station and the supervisor). Consequently, an efficient publisher/subscriber technology like DDS seems an adequate choice.

Figure 8 represents the messages exchanged among the automation components that encapsulate the station functionalities of the factory automation cell. These messages are described below. Different colours have been used to represent each type of service.

Once the piece identification code is read from the pallet at each station, its code and error/status information are published in DDS as one topic (*Set*) that is used by the next station (*Ready*) for two purposes: (1) to anticipate the parts required for a specific type of piece and (2) validation. This information is also delivered to be used by the supervisor.

When any station finishes processing a piece, it publishes its code and error/status information by means of a DDS topic (*End*). This topic informs the next station (*Begin*) to start its procedure and it also contains information used by the supervisor.

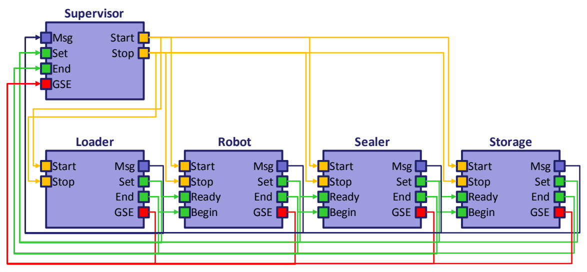

Figure 8. Information exchanged among the Automation Components via the DDS backbone

The supervisor may start and stop the stations using two DDS topics with configuration data (*Start*/*Stop*). Both of them allow configuration data in the message body to be sent. Internal faults at the stations are communicated immediately to the Supervisor by means of another DDS topic, the so-called General Station Event (GSE), which includes error and status information of the fault.

Table 3 provides a map of the identified services in the described industrial automation environment into DDS. Since GSE represent alarms and events, they are mapped into Aperiodic Station Event (ASE) services whereas *Set* and *End* are mapped into Periodic Variables (PV). Both types of services adapt best to the Publisher/Subscriber paradigm as described above. As a consequence, they require a single DDS topic for each variable with the same names (GSE, *Set* and *End*). In the case of GSE the supervisor subscribes to a topic shared by several station components (Loader, Robot, Sealer or Storage), which may publish the events. In the case of the *Set* and *End* topics, any station component may act as a Publisher, whereas the next physical station as well as the Supervisor may act as Subscribers.

	Services			
	Aperiodic Station Event	**Periodic Variable**	**Request / No Response**	**Request / Response**
Paradigm	Publish / Subscribe		Client / Server	
Topics (per variable)	1	1	1	2
Distribution	Many to one	Many to many	One to one	One to one
Content Filtered	No	Yes	Yes	Yes
Topic names	*GSE*	*Set, End*	*Start, Stop*	*Msg_Req, Msg_Res*
Variables	*GSE*	*Set, Ready, End, Begin*	*Start, Stop*	*Msg*

Table 3. Summary of the mapping into DDS topics and services

Start and *Stop* operations involve sending configuration information from the Supervisor to the stations in the body of the message. Since these operations do not need a response from the station component to the supervisor, they should be mapped as Request/Non Response (RNR) services. The supervisor component publishes these topics and station components (i.e., Loader, Robot, Sealer and Storage) subscribe to them.

The stations may also require additional information from the Supervisor, such as a code description or the status of any station. These operations are achieved by using Request/Response (RR) services. In this case, two topics are needed: *Msg_Req* and *Msg_Res*.

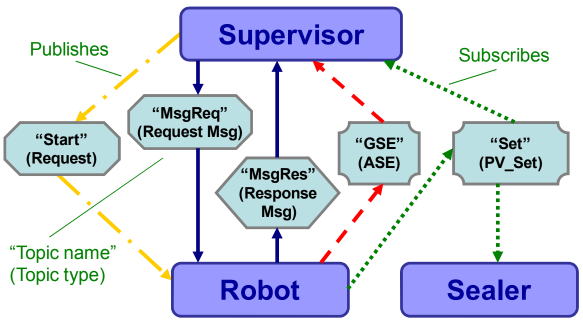

Figure 9. Topics exchanged among the Supervisor, Robot and Sealer automation components

Figure 9 shows some of the topics exchanged among the Supervisor, Robot and Sealer components. This figure is focused on the Robot component and, for the sake of clarity, the figure is not exhaustive: neither are all the topics and nodes depicted nor all connections of the Storage station shown. The boxes represent the automation components that wrap the stations. The direction of the arrows represents whether the application publishes one topic (one outgoing arrow) or is subscribed to a topic (one incoming arrow). Arrows have different colours and styles depending on the type of communication: Red is for ASE, green is for PV, yellow for RNR and blue is for RR. Finally, topics are located in the middle of the figure and are managed by the DDS middleware according to the QoS parameters as described below.

DDS uses a subset of IDL to describe the topics of the applications. Table 4 provides sample IDL interfaces that illustrate how to define the data of the case study in order to distribute them among the nodes of the distributed application for the above-described services. It should be noted that RR and RNR services share the same type for their requests, namely *Msg_Req*.

4.3 Building the automation components

This section describes the construction of the automation components according to the internal design proposed in Section 3.2. Even though this section is focused on the Robot automation component, it could be easily extended for the other components. Two major points are discussed hereafter: the object hierarchy of the robot component, as well as the values for the most relevant QoS parameters for the DDS entities.

Figure 10 represents an UML object diagram that implements the UML class diagram proposed in Figure 5 for the robot component. The *RobotComponent* contains two major objects, the *RobotWrapper* which holds the robot functionality by wrapping its programs and the *CellParticipant*, which implements an object of a *DDS:DomainParticipant* class for connecting the *RobotComponent* to the DDS middleware. All *DDS:Topics* described above (*Set, End, GSE, Start, Stop, MsgReq* and *MsgRes*) are connected to the DDS middleware by means of the *RobotPublisher* and *RobotSubscriber*. There are also several implementations of the *DDS:DataWriter* and *DDS:DataReader* classes, one for each kind of service defined above (i.e., Aperiodic Station Events - red, Periodic Variables - green, Request Non Response Services - yellow and Request/Response Services - blue).

	IDL

Aperiodic Station Events	`struct ASE_GSE {`
	`boolean ERROR;`
	`unsigned long STATUS;`
	`};`
	`#pragma keylist ASE_GSE // no key`

Periodic Variables	`struct PV_Set {`
	`string STATION;`
	`unsigned long CODE;`
	`};`
	`#pragma keylist PV_Set STATION`
	
	`struct PV_End {`
	`string STATION;`
	`unsigned long CODE;`
	`boolean ERROR;`
	`unsigned long STATUS;`
	`};`
	`#pragma keylist PV_End STATION`

Request Services	`enum Order_t {START, STOP, MSG};`
	`struct Msg_Req { // Request`
	`string DEST_STATION;`
	`string BODY;`
	`Order_t ORDER;`
	`};`
	`#pragma keylist Request_Msg`
	`DEST_STATION ORDER // two keys`

Response Services	`struct Msg_Res { // Response`
	`string DEST_STATION;`
	`unsigned long CODE;`
	`boolean ERROR;`
	`unsigned long STATUS;`
	`string BODY;`
	`};`
	`#pragma keylist Response_Msg // no key`

Table 4. DDS topic sample definitions grouped by type of service

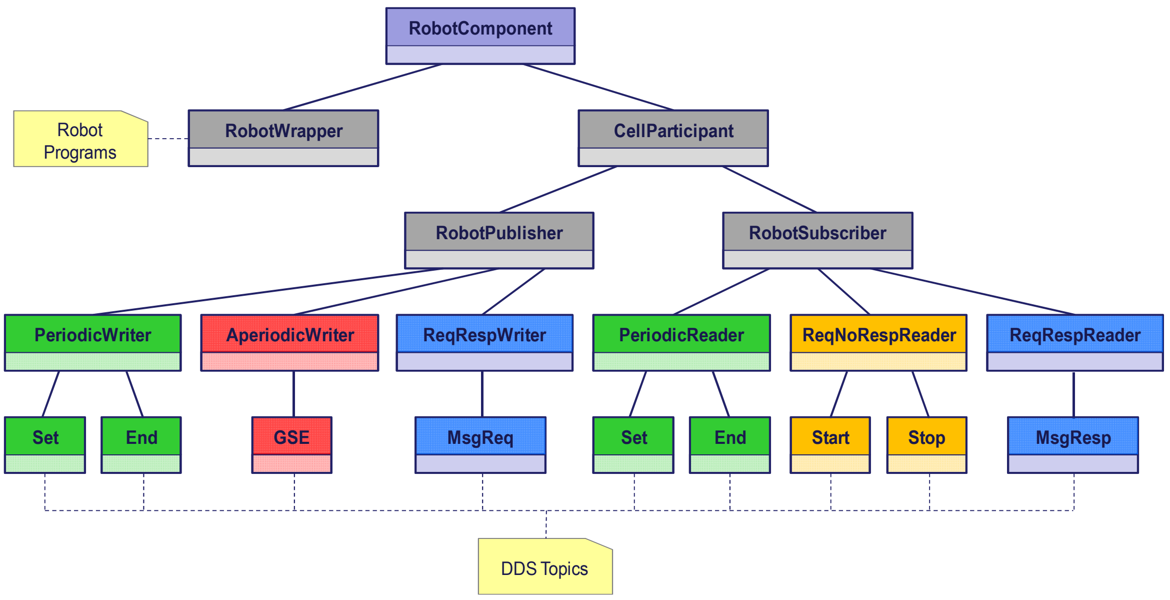

Figure 10. UML object diagram for the Robot automation component

With regard to the QoS parameter settings, Table 2 summarizes the values assigned to the different DDS QoS parameters. Below is a brief justification of the chosen values:

The **Deadline** parameter is only used for the Periodic Variables (*Set* and *End*) with a value related to the maximum process time for all individual stations since it represents the maximum separation between two topic updates.

Latency budget, which specifies the maximum acceptable delivery delay in the distribution of a DDS topic, is set to a relatively low value for the Aperiodic Station Events (*GSE*), meaning that they should be delivered very rapidly and at a relatively medium value for the Periodic Variables (*Set, End*).

The **Reliability** parameter is set to *best effort* for the Periodic Variables (*Set*, *End*), meaning that an efficient delivery is more important than resending a particular message. For this kind of data the current value has no meaning after a specific interval of time has elapsed and it is more important to distribute a newer value. For the remaining variables (*GSE, Start, Stop, MsgReq* and *MsgResp*) this parameter is set to *reliable*, meaning that the message must be resent if necessary.

History is configured so that the middleware keeps all samples of the Aperiodic Station Events (*GSE*) and only one for the remaining types of topics (*Set, End, Start, Stop, MsgReq, MsgResp*).

The **Transport priority** is set to the highest value for the Aperiodic Station Events (*GSE*) since the distribution of this information is considered the most relevant. Periodic Values (*Set*, *End*) are assigned a medium-high level of priority, whereas Request No Response messages (*Start, Stop*) are given a medium-low level of priority. Finally, Request Response messages (*MsgReq*, *MsgRes*) are given the lowest priority.

Durability is set to *Persistent* for Aperiodic Station Events (*GSE*) since it represents alarms. These alarms must be distributed to the nodes (e.g., the supervisor) even if they are disconnected from the application for a while. For the remaining topics (*Set, End, Start, Stop, MsgReq, MsgResp*) this parameter is not considered relevant and is set to *Volatile*.

5. Conclusions and future work

This work illustrates how to use DDS as a middleware virtual bus for connecting industrial controllers at the control layer of the automation pyramid. Every physical device is encapsulated by means of a generic software component, the so-called *Automation Component*, capable of accessing (reading and writing) information (Topics in DDS terminology) from the virtual bus. Assuming that the underlying network infrastructure is capable of enforcing the selected QoS policies, the use of a unique middleware specification if properly configured, may solve all communication needs for factory automation applications becoming a real communication backbone.

This paper also proposes an internal design for the *Automation Component* as well as a mapping for the most frequent operations in factory automation applications. This mapping distinguishes among different types of services and provides guidelines to configure the values of the DDS QoS parameters. The use of the proposed mapping is illustrated by means of a case study consisting of a factory automation application.

The authors are currently working on the creation of a basic IEC61499 function block set [37] so that they may constitute structural components from which distributed factory automation applications may be created on top of DDS.

6. Acknowledgments

This work has been financed in part by the Spanish Ministry of Economy and Competitiveness (MINECO) under project DPI2012-37806-C02-01; by the Basque Government through the S-PE11UN061 SAIOTEK project and by the University of the Basque Country (UPV/EHU) through grant EHU11/35.

7. References

[1] Amoretti, M., Caselli, S., Reggiani, M. (2006) Designing Distributed, Component-Based Systems for Industrial Robotic Applications, In: Industrial Robotics: Programming, Simulation and Application, Low Kin Huat (Ed.), ISBN: 3-86611-286-6, InTech.

[2] Pereira, C.E., Carro, L. (2007) Distributed real-time embedded systems: Recent advances, future trends and their impact on manufacturing plant control, Annual Reviews in Control, Vol. 31, pp. 81-92.

[3] Sauter, T. (2007) The continuing evolution of integration in manufacturing automation, IEEE Industrial Electronics Magazine, Vol. 1, No. 1, pp. 10-19.

[4] Skeie, T., Johannessen, S. and Holmeide, O. (2006) Timeliness of real-time IP communication in switched industrial Ethernet networks, IEEE Transactions on Industrial Informatics, Vol. 2, No. 1, pp. 25-39.

[5] Al-Jaroodi, J., Mohamed, N. (2012) Middleware is STILL everywhere!!!, Concurrency & Computation: Practice and Experience, DOI:10.1002/cpe.2817.

[6] OMG, Object Management Group (2004) Common Object Request Broker Architecture: Core Specification, Version 3.0.3.

[7] Calvo, I., Cabanes, I., Etxeberria-Agiriano, I., Sanchez, G., Zulueta, E. (2011) A CORBA wrapper for applications with multiple robots, International Journal of Online Engineering, Vol. 7, No. 4, pp. 4-9. DOI: 10.3991/ijoe.v7i4.1724.

[8] Henning, M. (2004) A new approach to object-oriented middleware, IEEE Internet Computing Vol. 8, No. 1, pp. 66-75.

[9] OPC foundation, http://www.opcfoundation.org/.

[10] Perez, F., Orive, D., Marcos, M., Estévez, E., Morán, G. and Calvo, I. (2009) Access to Process Data with OPC-DA using IEC61499 Service Interface Function Blocks, In Proc. of the 14th Conf. on Emerging Technologies and Factory Automation (ETFA2009) DOI: 10.1109/ETFA.2009.5347024.

[11] W3C (2004), Web Services Architecture, Available at: http://www.w3.org/TR/ws-arch/.

[12] Jammes, F., Smith, H. (2005) Service-oriented paradigms in industrial automation, IEEE Trans. Industrial Informatics, Vol. 1, No. 1, pp. 62-70, DOI: 10.1109/TII.2005.844419.

[13] OMG, Object Management Group, (2007) Data Distribution Service for Real-time Systems, v1.2.

[14] Ryll, M., Ratchev, S. (2008) Application of the Data Distribution Service for Flexible Manufacturing Automation, Proc. of World Academy of Science, Engineering and Technology, vol. 31, pp. 178-185.

[15] Schmidt, D.C. (2002) Middleware for real-time and embedded systems, Communications of the ACM 45(6), pp. 43–48.

[16] Noguero, A., Calvo, I., Almeida, L. (2012) A Time-Triggered Middleware Architecture for Ubiquitous Cyber Physical System Applications, In Ubiquitous Computing and Ambient Intelligence, LNCS 7656, Springer Berlin Heidelberg, pp. 73-80, 2012

[17] Elmenreich, W., Haidinger, W., Kopetz, H., Losert, T., Obermaisser, R., Paulitsch, M., Peti, P. (2006) A standard for real-time smart transducer interface, Computer Standards & Interfaces, 28, pp. 613-624.

[18] IEC, International Electrotechnical Commission (2005) IEC 61499-1: Function Blocks – Part 1. Architecture International Standard: Technical Communication 65, Working group 6, Geneva IEC.

[19]Vyatkin, V. (2003) Intelligent mechatronic components: control system engineering using an open distributed architecture. In Proc. of the 8th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'03), Vol. 2, pp. 277-284.

[20] Profinet-CBA, Profibus-Profinet International, Available at: http://www.profibus.com/, Last access April 2012.

[21] Cengic, G., Ljungkrantz, O., Akesson, K. (2006) A Framework for Component Based Distributed Control Software Development Using IEC 61499. In Proc. of the 11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'06), pp. 782-789.

[22] Black, G., Vyatkin, V. (2010) Intelligent Component-Based Automation of Baggage Handling Systems With IEC 61499, IEEE Transactions on Automation Science and Engineering, Vol. 7, No. 2, pp. 337-351.

[23] Amoretti, M., Reggiani, M. (2010) Architectural paradigms for robotics applications, Advanced Engineering Informatics, Vol. 24, No. 1, pp. 4-13.

[24] Mohamed, N., Al-Jaroodi, J., Jawhar, I. (2008) Middleware for Robotics: A Survey. In Proc. of the 2008 IEEE Conf. on Robotics, Automation and Mechatronics, (RAMECH2008), DOI: 10.1109/RAMECH.2008.4681485.

[25] Elkady, A., Sobh, T. (2012) Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography, Journal of Robotics, vol. 2012, Article ID 959013, 2012. DOI: 10.1155/2012/959013.

[26] Bruyninckx, H., Soetens, P., Koninckx, B. (2003) The real-time motion control core of the Orocos project. In Proc. of the IEEE International Conference on Robotics and Automation, pp. 2766-2771.

[27] Utz, H., Sablatnog, S., Enderle, S., Kraetzschmar, G., (2002) Miro - middleware for mobile robot applications, Robotics and Automation, IEEE Transactions on , Vol.18, No.4, pp. 493- 497 DOI: 10.1109/TRA.2002.802930.

[28] Makarenko, A., Brooks, A., Kaupp, T. (2006) Orca: Components for Robotics, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2006). Workshop on Robotic Standardization.

[29] Kranz, M., Rusu, R. B., Maldonado, A., Beetz, M., Schmidt, A. (2006) A Player/Stage System for Context-Aware Intelligent Environments. In Proc. of UbiSys'06, System Support for Ubiquitous Computing Workshop, at the 8th Annual Conference on Ubiquitous Computing (Ubicomp).

[30] Magnenat, S., Rétornaz, P., Bonani, M., Longchamp, V., Mondada, F. (2011) ASEBA: A Modular Architecture for Event-Based Control of Complex Robots, Mechatronics, IEEE/ASME Transactions on, Vol.16, No.2, pp.321-329 DOI: 10.1109/TMECH.2010.2042722.

[31] Robot Operating System, ROS, (2011) Available at: http://www.ros.org.

[32] Cruz, J. M., Romero-Garcés, A., Rubio, J. P. B., Robles, R. M. and Rubio, A. B. (2012), A DDS-based middleware for quality-of-service and high-performance networked robotics. Concurrency Computat.: Pract. Exper. DOI: 10.1002/cpe.2816.

[33] Smart, D.C. (2007) Is a Common Middleware for Robotics Possible?, In Proc. of the IROS 2007 Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware.

[34] Oh, S., Kim, J.H., Fox, G. (2010) Real-time performance analysis for publish/subscribe systems, Future Generation Computer Systems, Vol. 26, No. 3, pp. 318-323, DOI: 10.1016/j.future.2009.09.001.

[35] Valera, A., Gomez-Molinero, J., Sánchez, A., Ricole-Viala, C., Zotovic, R., Vallés, M. (2012) Industrial Robot Programming and UPnP Service Orchestration for the Automation of Factories, International Journal of Advanced Robotic Systems, Vol. 9, 123: 2012

[36] Saiedian, H., Mulkey, S. (2008) Performance Evaluation of Eventing Web Services in Real-Time Applications, IEEE Communications Magazine, Vol. 46-3, pp. 106-111. DOI: 10.1109/MCOM.2008.4463780.

[37] Calvo, I., Pérez, F., Etxeberria, I., Morán, G., (2010) Control Communications with DDS using IEC61499 Service Interface Function Blocks, In Proc. of the 15th Conf. on Emerging Technologies and Factory Automation (ETFA2010) DOI: 10.1109/ETFA.2010.5641207.

[38] Calvo, I., García de Albéniz, O., Noguero, A., Pérez, F. (2009) Towards a Modular and Scalable Design for the Communications of Electrical Protection Relays, In Proc. of the IEEE 35th Annual Industrial Electronics Conf. (IECON 2009), DOI: 10.1109/IECON.2009.5415221.

[39] Busch., D. (2011) Introduction to OpenDDS, Available: http://www.opendds.org/Article-Intro.html.

[40] OpenSplice. http://www.opensplice.com/.