Event Management Proposal for Distribution Data Service Standard

José-Luis Poza-Luján, Juan-Luis Posadas-Yagüe and José-Enrique Simó-Ten

University Institute of Control Systems and Industrial Computing (ai2). Universitat Politècnica de València (UPV). Camino de vera, s/n. 46022 Valencia (Spain).
{jopolu, jposadas, jsimo}@ai2.upv.es

Abstract This paper presents a proposal to extend the event management subsystem of the Distribution Data Service standard (DDS). The proposal allows user to optimize the use of DDS in networked control systems (NCS). DDS offers a simple event management system based on message filtering. The aim of the proposal is to improve the event management with three main elements: Events, Conditions and Actions. Actions are the new element proposed. Actions perform basic operations in the middleware, discharging the process load of control elements. The proposal is fully compatible with the standard and can be easily added to an existing system. Proposal has been tested in a distributed mobile robot navigation system with interesting results.

1. Introduction

Currently, the Event-Based Control (EBC) paradigm technology (also called event-driven control) is adopted to implement systems where the periodic sampling is not possible (i.e. when no discrete-time model is available) or recommended (i.e.: in distributed control systems to improve communications performance decreasing the load of the network) [1].

In networked control systems (NCS), distributed control elements are connected by a network. A middleware enhances and offers to control elements a set of services in order to facilitate the access to the network. Therefore, if EBC is used in NCS, the middleware will be an essential component [2].

Middleware architecture is based on communications paradigms: message passing, client-server, Publish/Subscribe (P/S) or blackboard. The Data Distribution Service for Real-Time Systems (DDS) is an Object Management Group (OMG) standard based on the P/S paradigm. DDS offers time-controlled communications between components using Quality of Service (QoS) policies [3].

NCS needs certain event management, i.e.: to select only messages with a meaningful value to trigger the control action. DDS can be used to send any kind of Data, including event data. DDS allows the application (control component in NCS) to perform flexible filtering of events but DDS does not define a built-in

event type and advanced event management. To provide support to NCS using DDS we are developed an extension of the DDS event components. This article presents the specifications of this extension. The proposal is fully compatible with the DDS model and maintains the OMG philosophy of simplicity and ease of use.

The paper is organized as following. Next section outlines the DDS elements involved on event management. Section 3 presents the proposal elements to add to the current DDS model. Section 4 presents an example of the use of the proposed elements, Section 5 the results of a simple experiment. Finally, evaluation and future work are discussed in conclusions.

2. Data Distribution Service Event Management

Data Distribution Service (DDS) provides a platform independent model that is aimed to real-time distributed systems. DDS is based on publish-subscribe communications paradigm. Publish-subscribe components connect information producers (publishers) and consumers (subscribers) and isolate publishers and subscribers in time, space and message flow [4].

When an application (producer) wants to publish some information, it should write it in a "Topic" by means of a component called "Data Writer" which is managed by another component called "Publisher". Both components, Data Writer and Publisher, are included in another component called "Domain Participant". On the other side of the communication, a Topic can be received by two kinds of components: "Data Readers" and "Listeners" by means a "Subscriber". A Data Reader provides the messages to application when the application requests them, A "Listened" sends the messages without waiting for the application request (Figure 1).

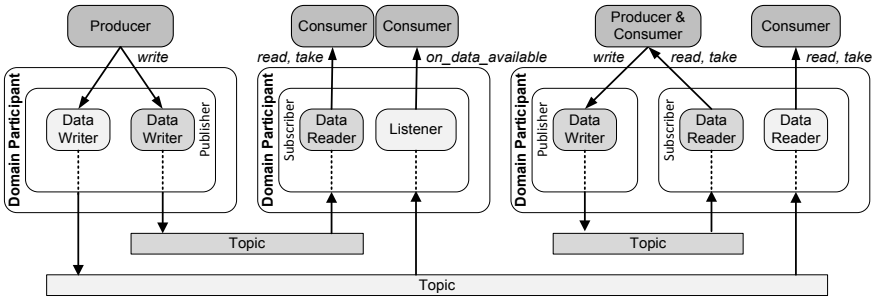

Fig 1. DDS elements and functions.

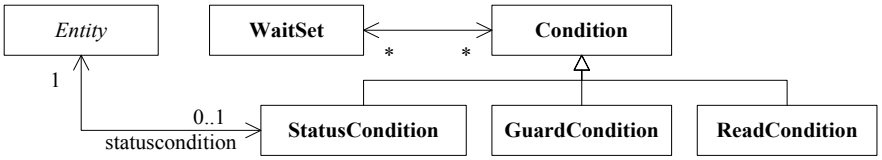

Fig 2. UML class diagram of the DDS elements involved in event management: conditions and WaitSet.

To configure communications, DDS uses QoS policies. A QoS policy describes the service behaviour according to a set of parameters defined by the system characteristics or by the administrator user. An Entity is the base class for communication components: Publisher, Subscriber, Data Writer, Data Reader and Listener. Details about the protocol can be obtained from [5]. Figure 2 shows the main elements of DDS involved on the event management.

Entities apply for relevant information by creating one of the types of Condition objects (StatusCondition, GuardCondition or ReadCondition) and attaching it to a WaitSet object. So that, a WaitSet object allows an Entity to wait until one of the attached Condition objects has a "triggervalue" of TRUE or else until the timeout expires.

3. Event Management Proposal

The event management used in DDS standard allows to perform a wide set of operations, such as filtering messages to be transmitted from the middleware to the application. The DDS QoS policies allow the application to change the characteristics of the communications between nodes, such as the message frequency or the deadline. When EBC is applied in a NCS, each control component sends messages based on the internal events (i.e.: when a new control action is calculated) but in distributed systems it is necessary that control nodes coordinates messages between them. In NCS, coordination requirements are based on the internal characteristics of control algorithms. DDS offers adequate support to coordinate communications between nodes, but it does not provide a mechanism to change the internal characteristics. To extend the power of DDS, a new component, called Action, is added.

Figure 3 shows the event management proposed. Main components are "Events", "Conditions" and "Actions". Events are situations that are necessary to be monitored. The Event component is similar to the Condition DDS element. Conditions are group of events using logical operations; the Condition component increases the WaitSet ability. Finally, Actions are the component that implements the effects on the system that are associated with Conditions. Events are categorized on three types: component operations, quality alarms and message filters.

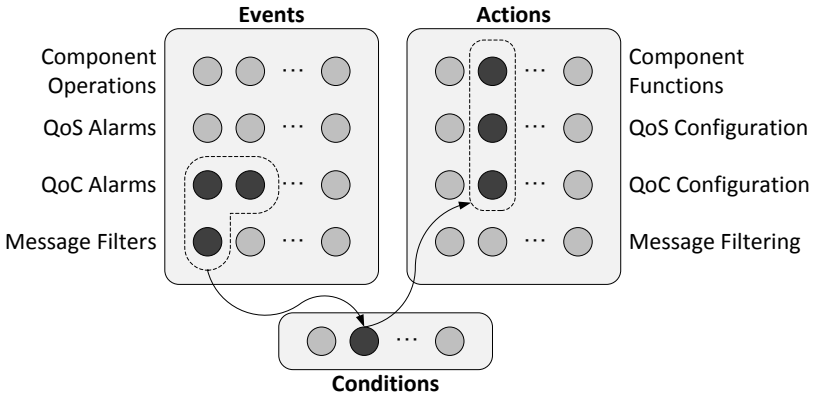

Fig. 3. Conceptual model of event management with the source of events, conditions and actions.

A component operation happens when a middleware or control component method is called; for example, when a control component starts the control action or when a middleware element is disconnected from a communications channel. Alarms are events associated with the compliance of the quality parameters. In our proposal we include the QoS and Quality of Control (QoC) parameters. QoC parameters [6] are associated to the efficiency of the control action and are directly related with the QoS parameters [7]. Thus, both parameters are considered. An example of QoS alarm is when a message arrives after the deadline. QoC alarm is related to the content of the message, for example when the reference value exceeds the reference value. Finally, message filters events are triggered when the content of the message is (or not) identical compared to a content pattern; for example, when the message field "source" (user defined) corresponds to a particular control node.

Conditions are associations of Events by using basic logic operations (AND, OR). For example, if a message from the node one (message filter) arrives in time (QoS alarm) and the internal value doesn't exceed the reference value (QoC alarm). The Condition is associated with a DDS element; so that, when the condition occurs, the element knows the existence. The method to trigger the alarm and to filter the message is the same used on the Remote Network Monitoring protocol (RMON) [8] due to its simplicity and efficiency.

When an element is triggered by a Condition, the element can run some function to process the trigger. In our proposal we add a new element called Action. Actions are associated to a Condition and are processed internally in the middleware. Initially four types of actions have been considered: component functions, QoS and QoC configuration and message queue actions. Component functions are the same functions that can produce events. For example, an action can be disconnecting a middleware element.

The QoS and QoC configurations are the variation of the parameters. For example, in order to increase the temporal limit of messages (QoS configuration) or the control error (QoC configuration).

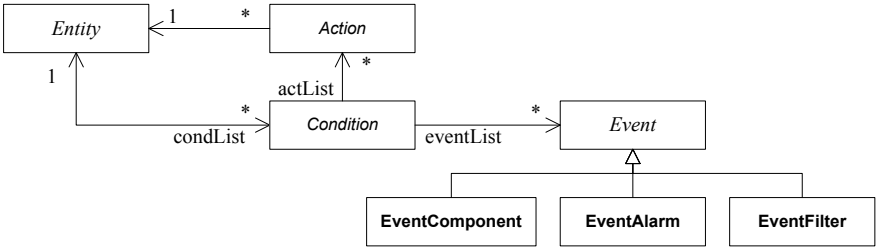

Fig. 4. UML class diagram of the proposal elements.

Finally, message queue actions are functions that change the behaviour of the queue, such as priority message, or message removal. So that, Actions discharge elements by processing of simple operations. Figure 4 shows the UML class diagram of the proposed elements. In our proposal Condition is similar to WaitSet DDS element, Event is similar to Condition and Action is the new element.

4. Implementation in a Distributed Control Architecture

A distributed mobile robot navigation environment has been used to test the validity and usefulness of the proposed model. The environment has been used in previous studies [9]. The mobile robot is controlled by a set of control algorithms. Some algorithms, generally algorithms that implement the reactive behaviours, are embedded within the robot and other algorithms, commonly the deliberative algorithms, are implemented in distributed nodes. However, in the case of study, all control algorithms are placed in distributed nodes in order to use a reduced dataset. The robot used in the study consisted of a simple two wheels and one distance eight sensors ring. The sensor distribution (figure 5) corresponds to a Khepera robot [10], thus developed algorithms can be compared with existing algorithms. The algorithm used is the "obstacle avoidance" based on Braitenberg vehicles behaviours [11].

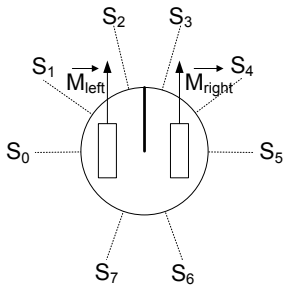

Fig. 5. The sensor distribution model used.

The obstacle avoidance algorithm is based on the speed variation of the motor depending on the distance detected by all sensors. For each motor, each sensor has a weighting value depending on its position on the robot. The motor speed is obtained combining the sensor weights by means the equation 1 where *m* is the concrete motor (left or right), *K* is the concrete weight factor applied to sensor *i* and motor *m*.

$$MotorSpeed_m = \sum_{i=0}^{7} K_i \cdot S_i \tag{1}$$

The maximum linear speed of the robot depends on the distance to the nearest obstacle and the sampling period to update motor velocities. The linear speed can be obtained from the equation 2, where *Sd* is the obstacle distance detected on the current robot path and T is the sampling period.

$$SpeedLimit = Sd/T \tag{2}$$

Figure 6, shows the results of the equation 2 to obtain the speed limit based on the sampling period and different distances. The communication channel defines the maximum sampling period, for example: a sampling period of 10 milliseconds needs a bandwidth to transfer at least 100 messages by second. The frequency of messages sent can be changed through the DDS QoS policies. Therefore Actions objects can increase or decrease this value automatically without the intervention of control components only with the distance value obtained from the sensors messages. The error in the control of obstacle avoidance behaviour is based on maximizing the distances to obstacles. Therefore, when the robot navigates far an obstacle, the robot does not need a high sampling rate. So, it can decrease the communications load without losing QoC.

This experiment is based on previous experiments performed in mobile robot navigation; these experiments are detailed in [12]. In previous works [9], QoC has been used to optimize the path regardless of the type of obstacle. The experiment presented below does have in mind the type of obstacle.

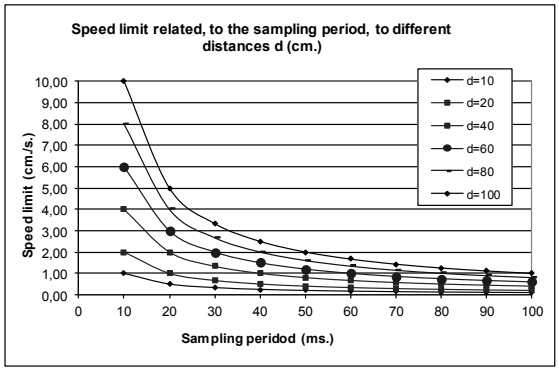

Fig. 6. Speed Limit obtained from the distance d and the sample period T.

5. Experiment and results

Two scenarios have been tested: robot navigation without event detection in the middleware and the same navigation with event detection and one Action linked by means a Condition. The Event detected in the middleware is the distance obtained from the robot by means a filter. The Condition set involves comparing the distance with the robot speed to obtain the optimal sampling period based on the equation 2. When middleware doesn't detect events, the sampling period is set to 10 milliseconds, and motor speed is obtained only with the equation 1. The experiment measures the sampling period and distance to an obstacle along time. In the case studied the sampling period changes internally in the middleware without the control component intervention. Figure 7 shows the graphs obtained for two obstacles and table 1 shows the summarised results.

Table 1. Experimental results for the two obstacles tested (average values).

Scenarios	Sampling period (1)	Distance average (1)	Sampling period (2)	Distance average (2)
a. Robot in corridor	10,0	2,1	62,6	2,1
b. Wall in front of the robot	10,0	1,1	9,4	0,6

When the robot navigates through a corridor (scenario a) the average distance is exactly the same for both middleware. This is because the corridor navigation doesn't involve avoiding an obstacle and the robot can run at maximum speed. However, the absence of obstacles allows navigating with the same efficiency but with fewer messages. Besides, robot reaches the target in less time. In this scenario the proposal does not improve the robot navigation, but reduces the network load. Scenario b is more complex, robot needs avoid an obstacle. As a result, the sampling period is similar. Nevertheless, the robot has more accuracy performing the manoeuvre.

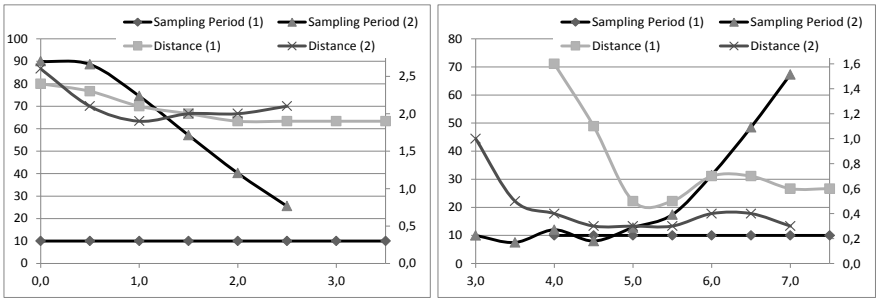

Fig. 7. Sampling period and distance variations along time robot navigation, without the proposal (1) and using the proposal (2) for two different obstacles.

6. Conclusions and Future Work

This article has presented a proposal to increase the event management system proposed in the DDS standard. The most significant contribution is the inclusion of a new object called Action. Actions automatically make changes on the middleware based on a combination of events.

The Action object has been tested with a simple mobile robot system. The test is based on the automatic variation of the QoS settings in function of the distance measured without control component intervention. The number of messages sent is reduced, and, as a result, the communications load is also reduced.

Future work is to test the middleware with complex combinations of events that generate different actions. The problem of generating different actions is the possibility to obtain contradictory actions, i.e.: increase and decrease the deadline QoS Policy. This problem can be solved by using priorities in Conditions or in Actions, but probably the inclusion of Actions in the middleware might be limited.

Acknowledgments The study described in this paper is a part of the coordinated project COBAMI: Mission-based Hierarchical Control. Education and Science Department, Spanish Government. CICYT: MICINN: DPI2011-28507-C02-01/02

References

1. Sánchez, J., Guarnes, M.Á., Dormido, S.: On the Application of Different Event-Based Sampling Strategies to the Control of a Simple Industrial Process. Sensors. 9, 6795-6818. (2009).
2. J. H. Sandee, W. P. M. H. Heemels, P. P. J. van den Bosch. Case Studies in Event-Driven Control. Lecture Notes in Computer Science, Vol. 4416, pages 762-765, Springer, 2007
3 Hadim S. and M. Nader, "Middleware Challenges and Approaches for Wireless Sensor Networks," IEEE Distributed Systems Online, vol. 7, no. 3, 2006.
4. Pardo-Castellote, G. OMG Data-Distribution Service: architectural overview. Proceedings of 23rd International Conference on Distributed Computing Systems Workshops. Providence, USA. Vol. 19-22, pp. 200-206. 2003.
5. Object Management Group. Data Distribution Service for Real-time Systems Version 1.2. 2007 (http://www.omg.org/)
6. R.C. Dorf and R.H. Bishop, Modern Control Systems, 11th Edition, Prentice Hall (2008)
7. Poza-Luján, J., Posadas-Yagüe, J., Simó-Ten, J.: Quality of Service and Quality of Control Based Protocol to Distribute Agents. ;In DCAI(2010)73-80.
8. Waldbusser, S. RFC 2819 - Remote Network Monitoring Management Information Base. Network Working Group. Lucent Technologies. 2000
9. Poza-Luján, J., Posadas-Yagüe, J., Simó-Ten, J.: Relationship between Quality of Control and Quality of Service in Mobile Robot Navigation. ;In DCAI(2012)557-564
10. K-Team Corporation. Khepera III robot. (http://www.k-team.com)
11. Braitenberg, V., 1984. Vehicles: Experiments on Synthetic Psychology. MIT Press, Cambridge, Massachusetts
12 Poza-Luján, J. (2012). Propuesta de arquitectura distribuida de control inteligente basada en políticas de calidad de servicio. Universitat Politècnica de València Press.