Real-time modelling of DDS for event-driven applications

Héctor Pérez and J. Javier Gutiérrez
Computers and Real-Time Group
Universidad de Cantabria
Santander, SPAIN
{perezh, gutierjj}@unican.es

Abstract—**The Data Distribution Service (DDS) standard defines a data-centric distribution middleware that supports the development of distributed real-time systems. To this end, the standard includes a wide set of configurable parameters to provide different degrees of Quality of Service (QoS). This paper presents an analysis of these QoS parameters when DDS is used to build reactive applications normally designed under an event-driven paradigm, and shows how to configure DDS to obtain predictable applications suitable to apply traditional schedulability analysis techniques.**

I. Introduction[1]

The OMG DDS specification [1] is based on the decoupled interaction paradigm provided by the publisher-subscriber communication model. DDS also supports a wide range of QoS parameters that enable communication control, which makes this standard suitable for developing distributed real-time and embedded systems (e.g. cyber-physical systems [2]). Furthermore, the OMG is addressing the use of distribution middleware in future high-integrity systems by extending DDS for this kind of critical systems, in which real-time issues should be also considered. Although DDS was initially designed to develop data-centric systems, it is also possible to apply this standard to event-driven systems with precedence constraints in which the real-time behaviour should be ensured by addressing the schedulability analysis of the system as a whole.

The event-driven end-to-end flow model defined by the MARTE standard [3] enables the modelling and analysis of distributed real-time systems via the traditional scheduling theory for both linear [4][5] and nonlinear systems [6][7]. This model plays a central role in the development of real-time distributed systems, as it is part of a relevant modelling standard and also includes Computer-Aided Software Engineering (CASE) tools such as MAST (Modelling and Analysis Suite for Real-Time Applications) [8] to facilitate the development process for real-time engineers.

MAST is a software toolsuite that offers an open set of tools for modelling, analysis and design of real-time systems. It proposes a system model also based on the aforementioned real-time end-to-end flow model, and it is currently being aligned with MARTE's standardized terminology [9].

1. This work has been funded in part by the Spanish Government under grant number TIN2011-28567-C03-02 (HI-PARTES).

The complexity associated to the use and management of the QoS parameters defined by DDS has motivated the application of design patterns to prevent inconsistent combinations of QoS policies that may also lead to unpredictability [10]. Furthermore, and although the distribution model proposed by DDS primarily aims to develop data-centric applications where the source of the data samples may not be known, it is possible to apply this model to other kinds of real-time applications in which the processing of each data sample may have a specific deadline associated. This scenario was addressed in a previous study [11] which concluded that an application using the DDS distribution model can be represented as a set of end-to-end flows. This paper provides a further step in this study and adds an analysis of how the set of QoS parameters defined by the DDS standard can be modelled using the real-time end-to-end flow model in order to make the distributed application analyzable via the traditional scheduling theory.

The document is organized as follows. Section 2 reviews the main concepts of the real-time end-to-end flow model used by MAST. In Section 3, we introduce the distribution model proposed by DDS. In Section 4, we describe the QoS parameters supported by DDS-based systems. The modelling of these parameters using the end-to-end flow model is discussed in Section 5. Finally, Section 6 draws the conclusions.

II. The Real-Time model

We will use the general model of event-driven distributed systems defined in [3] and [8], in which there is a set of external events that cause the execution of operations, such as the execution of tasks on the processors or the transmission of messages through the networks. The operations activate each other through internal events, which can be generated both by tasks or messages. As an example, the end-to-end flow model shown in Figure 1 consists mainly of the following entities: (1) *step,* which is a kind of *Event_Handler* that includes the operation to be executed and the schedulable entity to execute the operation; (2) *merge*, which is a king of Event_Handler that represents a precedence relationship that generates its output event every time one of its input events arrives; (3) *workload* and *internal events*, which are the elements responsible for triggering the execution of a step or are managed by other Event_Handlers; and (4) *observers*, which are responsible

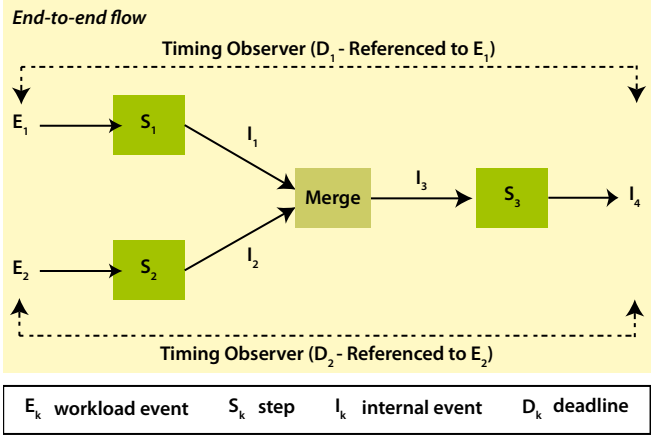

Figure 1. End-to-end flow model

for monitoring a set of requirements associated with events (e.g. deadline or queue size).

The end-to-end flow model enables the representation of complex interactions among the responses to different event sequences such as these proposed by the DDS specification, and the application of schedulability analysis techniques to determine whether timing requirements can be met or not in a distributed real-time system.

III. Distribution Model for DDS

The DDS conceptual model is based on the abstraction of a strongly typed *Global Data Space*, where publisher and subscriber respectively write (produce) and read (consume) data, leading to a middleware focused on obtaining data independently from its origin. To better handle the exchange of data, the standard defines a set of entities involved in the communication process. Applications that wish to share information with others can use this Global Data Space to declare their intent to publish data through the *Data Writer* (DW) entity. Similarly, applications that need to receive information can use the *Data Reader* (DR) entity to request particular data. *Publisher* and *Subscriber* entities are containers for several DWs and DRs respectively, which share compatible QoS settings. Likewise, these entities are grouped in *Participants* of a *Domain*. Only entities belonging to the same Domain can communicate. At a higher level of abstraction, the Participant entity contains all DWs, DRs, Publishers and Subscribers that share compatible QoS parameters in the corresponding Domain.

To exchange information among entities, Publishers only need to know about the specific *Topic* (i.e. the data type to share) and Subscribers require registration of their interest in receiving particular Topics, while middleware will establish and manage the communication almost transparently.

Finally, the DDS standard was explicitly designed to build distributed real-time systems. To this end, this specification adds a set of QoS parameters to configure non-functional properties. In this case, DDS provides high flexibility in the configuration of the system by associating a set of QoS parameters to each individual entity. Furthermore, DDS enables the modification of some of these parameters at runtime while performing a dynamic reconfiguration of the system. This set of QoS parameters is briefly reviewed in the next section.

IV. Supported QoS Parameters

The DDS standard defines several compliance profiles which provide different kinds of services, but only the *Minimum profile* is mandatory for any DDS implementation. This profile defines most of the QoS parameters that can be used in a DDS-based system, while the others usually add optional settings to them. This set of QoS parameters enables several aspects of data, networks and computing resources to be configured and may be classified in the following categories.

A Data Availability

It comprises the parameters used for controlling queuing policies and data storage. The parameters that fall into this category are the following:

- *History* specifies how many data samples will be stored before their delivery. It is strongly linked to the optional *Ownership profile* [1], which allows the configuration of how many data samples can be stored. Under the Minimum profile, there are only two settings available: (1) to keep the most recent data sample or (2) to keep all the received data samples.
- *Durability* determines whether previous data samples should be delivered to late-joiners and where they should be stored (e.g. memory or hard disk). Under the Minimum profile, no samples must be delivered to late-joiners.
- *Lifespan* defines a time period in which data samples will be available to the application.
- *Lifecycle* controls the automatic removal of data samples when their associated entities are no longer available.

B Maximum Resources

It limits the amount of resources that may be used in the system through the following parameters:

- *Resource_Limits* specifies how much memory can be used.
- *Time_Based_Filter* defines a minimum separation period between received samples of each instance.

C Data Delivery

It specifies how data must be transmitted and presented to the application. The parameters that fall into this category are the following:

- *Reliability* configures whether or not middleware will automatically manage the communication to make it reliable (i.e. no missed samples).
- *Ownership* specifies whether multiple DWs can update the same data instance. Under the Minimum profile, DRs will receive all the data samples generated from any matching DW.
- *Presentation* controls how received samples are presented to the application when the read operation is called (i.e. ordered or coherent policies).
- *Destination_Order* determines the order in which multiple samples of the same instance should be read (i.e. by source or destination timestamp).
- *Partition* defines additional rules to match DRs and DWs (besides matching the Topic and having compatible QoS settings).

D User Configuration

User, *Topic* and *Group Data* represent extra information that is attached to an entity when it is discovered or instantiated. This information is used in an application-defined way.

E Data Timeliness

It controls the latency in the distribution of data. The parameters that fall into this category are the following:

- *Deadline*: This parameter indicates the maximum amount of time available to send/receive data samples belonging to a particular Topic.
- *Latency_Budget*: This parameter is defined as the maximum acceptable delay in message delivery.
- *Transport_Priority*: This parameter allows the specification of the priority of each network message for a particular Topic. It is only associated to DW entities.

F System runtime configuration

These parameters control different configuration mechanisms applied at runtime. The parameters that fall into this category are the following:

- *Entity_Factory* indicates whether the DDS entities should be enabled automatically or by the application.
- *Liveliness* determines whether or not a DDS entity is still alive. It can take three different approaches depending on how the liveliness is asserted:
  - *Automatic_Liveliness*. It is managed automatically by middleware which defines a built-in Topic (i.e. DCPSParticipantMessage [12]) to send an ”alive” signal.
  - *Manual_By_Participant*. Liveliness is maintained by middleware which must check periodically whether the application has called any data or liveliness related operations.
  - *Manual_By_Topic*. Liveliness is managed by the application by sending data or explicit *Heartbeat* messages [12].

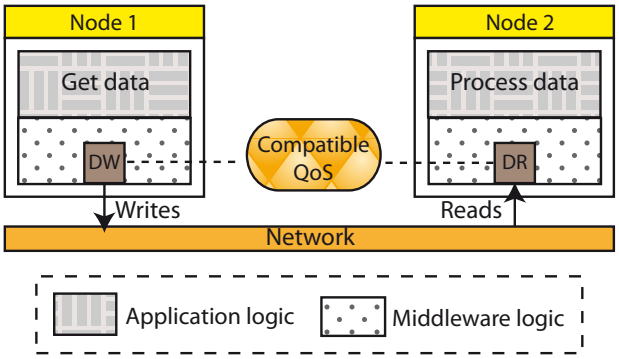

Figure 2. Real-time application using DDS

V. Discussion

Once the set of QoS parameters has been introduced, this section aims to discuss how to model them using the real-time end-to-end flow model. To better focus on the modelling issues, the following analysis assumes that the discovery process is finished and there are no late-joiners within the distributed system. Moreover, we will only consider those QoS parameters and settings included in the Minimum profile, and leave the optional profiles for future analysis.

The effect of the QoS parameters defined by DDS over the distributed system leads to different kinds of real-time applications. To illustrate this behaviour, we will use the example shown in Figure 2 which represents a distributed system where Node 1 gets data from some peripheral and send them to Node 2 to be processed. The first kind of real-time application is called synchronized (Figure 3-A), where each specific data sample has associated timing requirements, and represents a simplified version of the end-to-end flow model that includes the write and read middleware operations within the *Get data* and *Process data* steps, respectively. The second kind of real-time application is called non-synchronized (Figure 3-B and C), in which the operations for getting and processing data are decoupled. Thus, this kind of system is modelled by two independent end-to-end flows, which may be activated by different workload events with different periods.

Once the basis for the analysis is established, it will be performed following the same categories specified in the previous Section.

A Data Availability

The *Data Availability* parameters defined by DDS allow users to control how many data samples can be stored. In

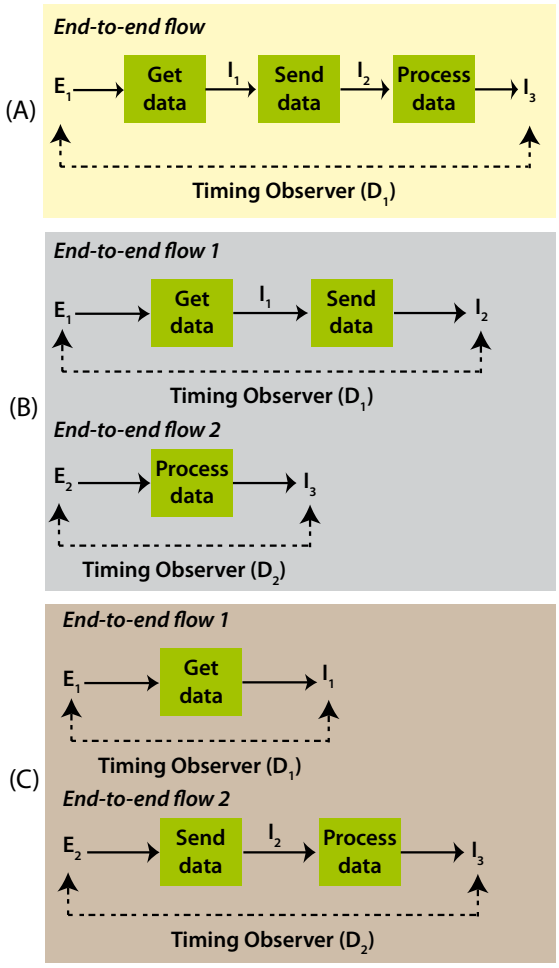

Figure 3. Synchronized (A) and non-synchronized (B,C) models for the real-time application

relation to synchronized applications, the schedulability analysis does not consider discarded samples and therefore this kind of systems should be configured to store each data sample (e.g. by setting the *KEEP_ALL* value [1] for *History* QoS parameter). In this case, the *Lifespan* and *Lifecycle* parameters should also be configured to avoid the removal of data samples.

For non-synchronized applications, the schedulability analysis can manage discarded data samples by defining different end-to-end flows (1) to publish data samples and (2) to process them (e.g., as shown in Figure 3-B). In this case, *History* should be set to *KEEP_LAST* [1].

Under the Minimum profile, the *Durability* parameter can only be set to *VOLATILE* [1] which has no effect on the schedulability analysis.

B Maximum Resources

The *Maximum Resources* parameters enable the use of DDS over resource-constrained systems (e.g. embedded systems). To deal with memory constraints, *Resource_Limits* controls the amount of memory that can be allocated to store data samples. In the case of synchronized applications, this value should be specified through the *Queue_Size_Req Observer* entity defined by the MAST end-to-end flow model. This entity represents the maximum number of pending activations that are admitted in the system in order to avoid the loss of any sample. Furthermore, the current schedulability analysis techniques included in MAST are capable of estimating this value through the *Num_Of_Queued_Activations* parameter.

The *Time_Based_Filter* reduces the number of data samples received by a DR and therefore it decouples a DR from its matched DW. Consequently, the use of this QoS parameter leads to a non-synchronized real-time application. According to the DDS-related standards [1] [12], time filtering can be applied on the reader or writer side. If on the former, data samples are transmitted through the network but are filtered before being processed. Therefore, this behaviour can be modelled as a stream of data samples that have a minimum interarrival time (i.e. sporadic workload events [8]) equal to the *minimum_separation* period [1]. If we again consider the example illustrated in Figure 2 but adding the time filtering, the resulting model is the one shown in Figure 3-B. When filtering is applied on the writer side, data samples are filtered before transmitting them through the network to avoid overusing network resources. This behaviour can also be modelled as a stream of data samples with a minimum interarrival time, which is shown in Figure 3-C. Furthermore, the standard specifies that additional network messages should be sent to disseminate the fact that those samples have been filtered and not simply lost, which should also be modelled and included in the schedulability analysis.

C Data Delivery

The *Data Delivery* parameters provide different mechanisms to manage data samples. From the schedulability analysis perspective, reliable communication increases the network traffic and this overhead must be taken into account for the analysis. Moreover, the standard does not determine the middleware behaviour for reliable communications, and leaves multiple aspects open to implementations (i.e. the number and timing characteristics of messages used for acknowledgments, or whether multiple RTPS submessages can be coalesced into the same message).

Under the Minimum profile, the *Ownership* parameter can only be set to *shared* and therefore DRs can receive data from any matching DW. This behaviour can be modelled as a *Merge* entity from the MAST model (see Figure 1).

The *Presentation* parameter would add an extra step to execute the desired policy when all data samples are available. However, the standard does not specify whether this parameter requires the use of reliable connections, in

which case the predictability of this policy would depend on each implementation.

The schedulability analysis based on the end-to-end flow model usually takes the delivery of data samples according to the order in which they are received, as network delays are already considered in the response time analysis. If *Destination_Order* is configured to deliver data samples to DRs in the order in which they were sent, some of these samples may be dropped as they can arrive out of order. When the real-time application can tolerate the loss of data samples, the schedulability analysis will assume that all the samples are delivered to the DR and processed by the application. However, the issue of how to set the source timestamp is not defined in the standard and remains open to implementations.

In relation to the *Partition* parameter, it does not add any extra modelling entity to the real-time end-to-end model.

D User Configuration

The parameters related to the *User Configuration* allow extra information to be added to each entity at application level. These parameters are transparent to the schedulability analysis.

E Data Timeliness

The three parameters of *Data Timeliness* are particularly important in the management of resources for real-time systems. The *Transport_Priority* parameter represents the scheduling parameters for the communication networks. However, as shown in [11], the DDS specification does not explicitly address the scheduling of tasks in the processors, as this is an implementation-defined aspect. Thus, for example, the *Deadline* parameter does not define any associated mechanism to enforce this timing requirement and therefore this QoS parameter only represents a notification service in which middleware informs the application that the deadline has been missed. Therefore, this notification service should be modelled as a new linear end-to-end flow.

In the case of the *Latency_Budget*, the standard emphasizes that this parameter must not be enforced or controlled by middleware and, consequently, indicates the urgency in the processing of data samples. Therefore, it can be considered as a *best-effort* parameter to configure the internal behaviour of middleware, and its modelling would depend on the selected implementation.

F System runtime configuration

Regarding the *Entity_Factory* parameter, any setting is suitable for timing analysis.

Finally, the *Liveliness* parameter allows DRs to detect when matching DWs have been disconnected. When the liveliness kind *Automatic_Liveliness* is used, the schedulability analysis must take into account the overhead introduced by the new Topic and its corresponding entities in order to manage the publication and subscription operations. Furthermore, this parameter would add a new end-to-end flow per Participant for the assertion of liveliness. However, the standard specifies that the kind of reliability of these built-in entities must be set to *RELIABLE*, and therefore this mechanism would be suitable for schedulability analysis depending on each specific implementation. Similarly, middleware will be in charge of refreshing the liveliness through the same built-in and reliable entities when the setting *Manual_By_Participant* is used. Lastly, the liveliness setting *Manual_By_Topic* can be managed by the application in an explicit way, which would add a new end-to-end flow per DW to the real-time model, or in an implicit way by sending data.

Summary

A comprehensive summary of the results obtained is shown in Table 1. This Table shows the different QoS parameters classified according to the established categories, their available values, whether these values are suitable for the schedulability analysis and their corresponding end-to-end flow modelling entities.

VI. Conclusions and Future Work

The paper presents a study of the real-time modelling of advanced configurations for DDS-based applications. In particular, it discusses how to represent the QoS mechanisms defined by the DDS standard using the real-time end-to-end flow model.

The modelling and timing analysis of DDS-based applications strongly depends on the nature of the real-time application. Thus, real-time synchronized applications should configure their QoS parameters to avoid the dropping of data samples. Furthermore, the DDS standard leaves multiple aspects dependent on middleware, such as the implementation of the *Latency_Budget* and *Reliability* QoS parameters. The latter also influences the modelling of other DDS features, such as those related to the *Automatic* or the *Manual_By_Participant* liveliness. Therefore, each DDS implementation should provide information on the modelling and usability of these QoS parameters for real-time systems.

Future work will include the completion of this study by integrating the optional profiles defined by the DDS specification, such as the Persistence or the Ownership profiles. Furthermore, there are still several aspects to be analysed that may affect the determinism of the distributed system, for instance, the influence of the built-in entities, dynamic systems or the discovery process.

Table 1: Comprehensive summary of the modelling of DDS QoS parameters

QoS parameters		Value	Analyzable	End-to-end flow modelling
Data Availability	History	Any	Conditioned	Synchronized / Non-synchronized app.
	Durability	Volatile[a]	Yes	-
	Lifespan	Any	Conditioned	Synchronized / Non-synchronized app.
	Lifecycle	Any	Conditioned	Synchronized / Non-synchronized app.
Maximum resources	Resource_Limits	Any	Yes	Queue_Size_Req Observer
	Time_Based_Filter	Any	Conditioned	Sporadic workload events
Data Delivery	Reliability	Best effort	Yes	-
		Reliable	Conditioned	Implementation-specific
	Ownership	Shared[a]	Yes	Merge
	Presentation	Any	Yes	Step
	Destination_Order	By reception timestamp	Yes	-
		By source timestamp	Conditioned	Implementation-specific
	Partition	Any	Yes	-
User Configuration	User, Topic and Group Data	Any	Yes	-
Data Timeliness	Deadline	Any	Yes	New end-to-end flow
	Latency_Budget	Any	Conditioned	Implementation-specific
	Transport_Priority	Any	Yes	Scheduling parameter
System Runtime Configuration	Entity Factory	Any	Yes	-
	Liveliness	Automatic	Conditioned	Implementation-specific
		Manual_By_Participant	Conditioned	Implementation-specific
		Manual_By_Topic	Yes	New end-to-end flow per DW for explicit assertion of liveliness

a. Available values defined by the Minimum profile

REFERENCES

[1] "Data Distribution Service for Real-time Systems" (v1.2), Object Management Group, Doc. formal/07-01-01, 2007.

[2] Kang, W., Kapitanova, K. and Sang Hyuk Son: "RDDS: A Real-Time Data Distribution Service for Cyber-Physical Systems", IEEE Transactions on Industrial Informatics, vol.8, no.2, 2012, pp.393-405.

[3] "A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems", Object Management Group, OMG Document ptc/2009-11-02, 2009, pp. --.

[4] Tindell, K. and Clark, J. "Holistic schedulability analysis for distributed hard real-time systems", Microprocessors and Microprogramming Journal (40), 1994, pp. 117-134.

[5] Palencia, J. C., Gutiérrez, J. J. and González, M. "On the schedulability analysis for distributed hard real-time systems", in Proceedings of the Ninth Euromicro Workshop on Real-Time Systems, 1997, pp. 136 -143.

[6] Fohler, G. "Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems", in Proceedings of the 16th IEEE Real-Time Systems Symposium, 1995, pp.152-161.

[7] Gutiérrez, J. J., Palencia, J. C. and González, M. "Schedulability analysis of distributed hard real-time systems with multiple-event synchronization", in Proceedings of the 12th Euromicro conference on Real-time systems' IEEE Computer Society, Washington, DC, USA, 2000, pp. 15-24.

[8] González, M., Gutiérrez J. J., Palencia J. C. and Drake J. M. "MAST: Modeling and Analysis Suite for Real Time Applications", in Proceedings of the 13th Euromicro Conference on Real-Time Systems, IEEE Computer Society, Washington, DC, USA, 2001, pp. 125-134.

[9] González, M. Gutiérrez, J. J., Drake, J. M., López, P. and Palencia, J. C. "Modeling distributed real-time systems with MAST 2", In Press, Journal of Systems Architecture, http://dx.doi.org/10.1016/j.sysarc.2012.02.001, 2012.

[10] Joe Hoffert, Douglas Schmidt, and Aniruddha Gokhale. "A QoS policy configuration modeling language for publish/subscribe middleware platforms". In Proceedings of the International Conference on Distributed event-based systems (DEBS), 2007. ACM, New York, NY, USA, 140-145.

[11] Pérez, H. and Gutiérrez, J. J. "On the schedulability of a data-centric real-time distribution middleware", Computer Standards & Interfaces (34:0), 2012, pp. 203-211.

[12] The Real-time Publish-Subscribe Wire Protocol. DDS Interoperability Wire Protocol Specification", Object Management Group, 2009.