Using DDS middleware in distributed partitioned systems

Marisol García-Valls[†], Jorge Domínguez-Poblete[†], Imad Eddine Touahria[†*]
[†]Universidad Carlos III de Madrid, Spain
[*]University Ferhat Abbas Sétif1, Algeria
{mvalls,jdominguez}@it.uc3m.es , imad.touahria@univ-setif.dz

ABSTRACT

Communication middleware technologies are slowly being integrated also into critical domains that are also progresively transitioning to partitioned systems. Especially, avionics systems have transitioned from federated architectures to IMA (Integrated Modular Avionics) standard that targets partitioned systems to comply with the requirements of cost, safety, and weight. In the future developments, it is fully considered the integration of middleware to support data communication and application interoperability. As specified in FACE (Future Airborne Capability Environment), middleware will be integrated into mixed criticality systems to ease the development of portable components that can interoperate effectively. Still nowadays, in real-time environments, communication middleware is perceived as a source of unpredictability; and still there are very few contributions that present real applications of the integration of communication middleware into partitioned systems to support distribution.

This paper describes the usage of a publish-subscribe middleware (precisely, DDS –Data Distribution Service for real-time systems–) into a fully distributed partitioned system. We explain the design of a reliable communication setting enabled by the middleware, and we exemplify it using a distributed monitoring application for an emulated partitioned system with the goal of obtaining the middleware communication overhead. Implementation results show stable communication times that can be integrated in the resource assignment to partitions.

1. INTRODUCTION

Communication *middleware* and *virtualization* technologies are two main contributions to the development and maintainability of software systems as well as to machine consolidation [6]. These were initially used in mainstream applications, but are progressively entering into the critical environments and complex systems, where their role is increasingly important. In fact, in the avionics domain, the combination of IMA [24] and FACE [21] require the usage of both *virtualization technologies* to develop partitioned systems and *middleware* to ease interoperability and portability of components. This satisfies key requirement regarding cost, space, weight, power consumption, and temperature.

On the one hand, *middleware* brings in the capacity to abstract the low-level details of the networking protocols and the associated specifics of the physical platforms (e.g. endianness, frame structure, and packaging, among others). Consequently, the productivity of systems development is augmented by easing the programmability, maintanability, and debugging.

On the other hand, the penetration of *virtualization* technology has opened the door to the integration of heterogeneous functions over the same physical platform. This effect of virtualization technology has also arrived to the *real-time systems* area. The design of *mixed criticality systems* (MCS) [3] is an important trend that supports the execution of various applications and functions of different *criticality levels* [28] in the same physical machine. The term *criticality* refers to the levels of assurance over the system execution in what concerns failures. For example, in avionics systems, software design follows DO-178B that is a de facto standard for software safety; software is guided by DAL (Design Assurance Levels), and failure conditions are categorized against their consequences: from catastrophic (DAL A) to no effect (DAL E). Then, an MCS is one that has, at least, two functions of different criticalities on the same physical machines.

Over the past 30 years, middleware technology has been applied in critical domains but in those subsystems of lower criticality levels. This is the case of, e.g., CORBA applied to combat systems [26] or, recently, DDS [18] applied to control of interoperability of unmanned aircraft and air traffic management[1], mainly for ground segment control. Still middleware is mostly used directly on bare machine deployments; yet it is not used in partitioned software systems.

This paper provides an initial design of a fully distributed partitioned deployment that integrates DDS middleware. We exemplify this concept on a data monitoring application that has been developed to provide hands on the actual technology, to analyze the temporal behavior of the overall distributed partitioned setting. The system is fully distributed across different physical machines to perform data

[1]http://www.atlantida-cenit.org

sampling and transmission that is later received, processed and displayed at a remote node. The nodes of the monitoring system emulate a mixed criticality system. The setting replicates that of a partitioned system in a FACE compliant architecture. We concentrate on the design of the software stack and the analysis of the middleware performance over an Ethernet network that emulates an AFDX (Avionics Full DupleX) [1] compliant communication.

The paper is structured as follows. Section 2 presents the work background and a selected related contributions, including concepts and technologies relative to partitioned systems and distribution middleware technologies for critical domains. Section 3 analyzes the most important characteristics and properties of DDS for partitioned systems within FACE. Section 4 presents the design of the distributed partitioned system, illustrated for a remote monitoring application. Section 5 provides the implementation of the system and presents the results obtained for the communication. Section 6 draws some conclusions and describes future work.

2. BACKGROUND

IMA has been a very successful approach to transition from the former federated architectures to a more efficient design and final deployment into avionics systems. In this context, different standards are proposed to facilitate componentization, portability and interoperability at different levels of a system. In this way, ARINC 653 standard decouples the real-time operating system platform from the application software. For this purpose, it defines an APEX (APplication EXecutive) where each application software is called a *partition* having its own memory space. Each partition has dedicated time slots allocated through the APEX API, so that each partition can have multi-tasking and its own scheduling policy. Overall, the execution is embodied in a hierarchical scheduling policy where the top level is a cyclic schedule. Current work is to enhance ARINC 653 for multi-core processor architectures. The underlying network is AFDX that uses commercial technology with redundancy to support safe transmission. The integration of networked and distributed systems follows ARINC 429 that is the data bus defining the characteristics of the data exchange across connected subsystems. It defines the physical and electrical interfaces and a data protocol for a local avionics network.

In other real-time systems, network scheduling typically relies on either: network scheduling off-line network transmission plan given the a-priori knowledge of the generated traffic (e.g. [4]); architectures such as TTA (Time Triggered Architecture) [16]; or distributed component models highly related to the hardware on-chip design such as Genesis [17].

Nevertheless, in the last decade the trend in the development of complex systems is to move to more productive ways of designing the communication and interaction. The avionics industry has developed FACE (Future Airborne Capability Environment) standard to facilitate the development and easy integration of portable components. The communication middleware is given a key role as an interoperability enabler. Different technological choices can be used in FACE such as CORBA, Web services, or DDS, among others. The most popular technology at the moment is (probably) OMG's DDS standard [18] that has been applied in a number of domains such as remote systems control [9]. It provides an asynchronous interoperability via a publish-subscribe (P/S) paradigm that is data-centric, and the communication can be fine tuned through quality of service (QoS) policies.

Most recent works on the literature provide improvements to different aspects of the middleware such as service times making it aware of the underlying execution hardware [11] or improvement of the number of serviced clients [12]. On the performance side, there are some related works that contribute a thorough performance study of DDS for desktop virtualization technologies [7] but was not dealing with partioned systems; or the execution of DDS over a real-time hypervisor [23] although the actual network stack processing was not measured; or [5,15] for network level P/S evaluation, and [25] for bare machine deployments. Overall, there is not sufficient analysis on the actual execution characteristics of specific middleware technologies in general partitioned environments. Moreover, there are no practical design models of partitioned systems that can comprehensively put forward the required software levels integration and there actual performance results. This paper contributes in this direction with a practical design of a distributed partitioned enviroment based on DDS, providing a study of the behavior of a partitioned system that communicates using this technology.

3. MIDDLEWARE IN PARTITIONED SYSTEMS

3.1 Middleware in FACE Standard

FACE standard defines the software computing environment and interfaces designed to support the development of portable components across the general-purpose, safety, and security profiles required by the avionics domain. Its goal is to define the interaction points between the different technologies that ease systems integration. Actually, FACE uses industry standards for distributed communications, programming languages, graphics, operating sytsems, among other areas. Version 2.0 further promoted application interoperability and portability, with enhanced requirements for exchanging data among FACE components; also, v2.0 emphasizes on defining common language requirements for the standard. Precisely, the operating system segment of FACE defines different levels: the portable components segment (the applications), the transport services segment (where middleware technologies are employed for interoperability), the platform specific services segment (for the common services of a given domain), and the I/O services segment (related to the low-level adapters and drivers to access peripherals including the actual network).

At the transport services segment, many technological choices can be employed (e.g. CORBA, Web services, DDS, etc.). At the moment, the most widely accepted is DDS.

3.2 Data Distribution Service for real-time systems

The Data Distribution Service (DDS) is an OMG standard that provides a publish-subscribe (P/S), i.e., a decou-

pled interaction model among remote components. DDS relies on the concept of a *global data space* where entities exchange messages based on their type and content. Such entities are *remote nodes* or *remote processes*, although it is also possible to communicate from within the same local machine.

QoS policy	Entity	Description
Deadline (t)	DR	Max expected elapsed time between arriving data samples (unkeyed data) or instances (keyed data)
	DW	Max committed time to pulish samples or instances
Resource Limits (o)	DP	Limit to the allocated memory (message queues for history, etc.). It limits the queue size for History when the Reliability protocol is used.
History (o)	DR, DW	Stores sent or received data in cache. It affects Reliability and Durability (receive samples sent prior to joining) QoS policies
Latency Budget (t)	T, DR, DR	Indication on how to handle data that requires low latency. Provides a maximum acceptable delay from the time the data is written to the time it is received by the subscribing applications.
Timed based Filter (t)	DR	Limits the number of data samples sent for each instance per a give a time period
Transport priority (t)	DW	Establishes a given priority for the data sent by a writer. Underspecified: It is dependent on the actual transport characteristics and also supported by only some OSs.
Reliability (o)	DR, DW	Global policy that specifies whether or not data will be delivered reliably. It can be configured on a per DataWriter/DataReader connection.

Table 1: Subset of QoS policies affecting overhead

Entities can take two roles; they can be *publishers* or *subscribers* of a given data type. Types are based on the concept of *topics* that are constructions supporting the actual data exchange. Topics are identified by a unique name, a data type and a set of QoS policies; also, they can use *keys* that enable the existance of different instances of a topic so that the receiving entities can differentiate the data source.

Applications organize the communicating entities into *domains*. Essentially, a domain defines an application range where communication among related entities (an application) can be established. A domain becomes alive when a *participant* is created. A participant is an entity that owns a set of resources such as memory and transport. If an application has different transport needs, then two participants can be created. A participant may contain the following child entities: *publishers*, *subscribers*, *data writers*, *data readers*, and *topics*. Publishers and subscribers are an abstraction that manages the communication part, whereas the data writers and data readers are the abstractions that actually inject the data.

One of the most successful elements in DDS is the set of quality of service parameters that it defines, namely *QoS policies*. In fact, not all of them are related to the temporal behavior of the communication. Most QoS policies provide other guarantees over the data transmission. A short summary of policies that influence the communication time (marked as t) and others affecting the actual overhead of the system (marked as o) are provided in table 1. The entities[2] to which they apply are also indicated.

3.3 Communication in partitioned systems

The mainstream design of a critical partitioned system consists of isolating the communications of a system into one single partition that is selected as the communications partition (see figure 1). Transmissions take place purely during the time of the cycle assigned to the partition. When any partition needs to transmit or receive data, it sends it to the communication partition via a shared memory space. When the kernel schedules the partition at its assigned time slot, the transmission takes place.

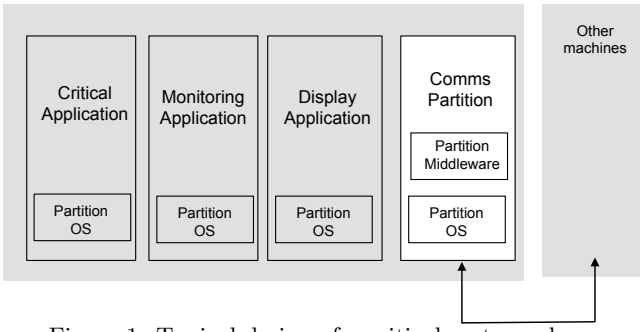

Figure 1: Typical design of a critical system where communication is isolated in one partition

This design approach has not included middleware, but has used direct implementation of network protocols for ARINC 429. So, standards such as DDS are progressively considered into the actual implementations of critical partitioned environments in the context of FACE.

4. SYSTEM DESIGN

4.1 Software design and middleware communication

Figure 2 shows the major software blocks for the distributed monitoring application. The system has two differentiated subsystems: *System A* that is a *meteo proximity server* and *System B* that is the *Global Monitor*.

Two sensors (humidity and temperature) gather data samples that are passed to System A that is able to perform an initial basic processing of the data to take basic

[2]PU: Publisher, SU: Subscriber; DR: Data Reader, DW: Data Writer; T: Topic; DP: Domain Participant

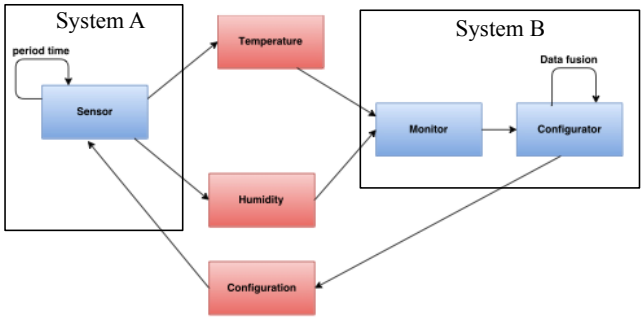

Figure 2: Application software design

decisions on the sensors operation and detect faults. System A passes these data to System B that is able to perform more complex analysis over the data, and also it is able to make decisions on the configuration of the sensors and the overall system operation. The proximity sensor can do an initial processing of the samples and later, it sends the data to the global monitor.

Systems A and B are two different physical machines that are partitioned emulating an ARINC 653 deployment. They are connected via an Ethernet link as compliant with AFDX. All partitions can integrate the communication middleware, so that any of them can send or receive data to and from other either local or remote partitions as shown in figure 3. In this way, communications may take place during the time slots that the kernel assigns to each of the partitions.

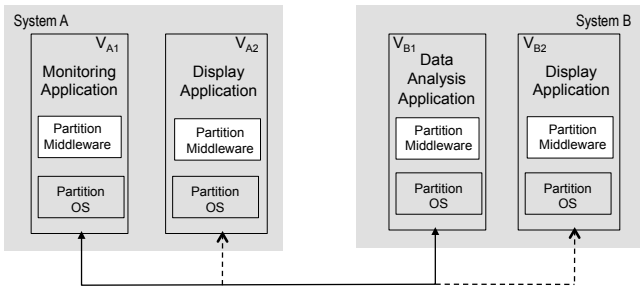

Figure 3: Proposal architecture- Distributed monitoring application design in a partitioned system

System A has two partitions: V_{A1} that monitors the sensor reads and performs basic analysis, and V_{A2} that performs data displays to the operators. System B has two partitions: V_{B1} that receives the sensor reads and performs complex analysis to guide the system operation, and V_{B2} that performs data displays to the remote operators.

Partition V_{A1} receives the sampled data, and it sends them to V_{B1} through the appropriate topics: *temperature* and *humidity*, to send temperature and humidity values, respectively. Also, a *configuration* topic is used by system B to change the operation parameters of A such as the sampling rate.

The topic based communication between System A and B is shown in figure 2. Topics are associated to a data types declared in a given programming language. The system defines a *dynamic topic* so that it is possible to update the its parameters (e.g. name or payload in this case). This favors flexibility and portability. This class contains a method for the creation and deletion of the topic. Code 1 shows the topic template.

Code 1: Topic structure template

```
struct DataSampleType{
 string<TOPIC_NAME> prefix;
 long sampleId;
 sequence<octet, TOPIC_MAX_PAYLOAD_SIZE>
     payload;
};
```

As the system handles sensor data samples, the needed topic parameters are *sequence number* and *payload*.

The communication settings for this application must be reliable. The subset of the QoS policies are shown:

- *RELIABILITY* is set to RELIABLE for guaranteed message delivery, and it is set both for data reader and data writer.
- *HISTORY* is set to KEEP_ALL that guarantees that samples will be retained until the subscriber retrives them. It is set for data reader.

Other properties as TRANSIENT need not be set as no late joiners are allowed. All connections are configured at the system start up time. Other parameters such as *Availability*, *Durability, Durability Service, DataWriterProtocol, DataReaderProtocol*, etc., ensure aspects as the ordered delivery of items, cache storage, usage of ACK/NACK and sending frequency. As our system is mapped to a UDP transport protocol over Fast Ethernet, it is sufficient to use the maximum allowed frame size as our application data messages payload is of 1024KB. Also, IP routing is used as compliant with AFDX in a switched Ethernet setting; this is enabled by the wire protocol of DDS that is RTPS.

The Real-Time Publish-Subscribe [19] is the protocol that enables interoperability across different implementations (vendors) of DDS. Its design requires, by default, the usage of the Internet Protocol (IP) multicast (one-to-many communication) and an unreliable, connection less transport such as UDP (User Datagram Protocol).

Apart from the default best-effort, RTPS also provides reliable communication for IP networks. RTPS offers distributed knowledge, preventing the centralised management of the system; this is an inherent fault tolerance mechanism which prevents the network from having a single point of failure.

The general implementation of the reliability mechanism for RTPS is the usage of acknowledgment messages (ACK). They are similar to a heartbeat mechanism and ACKNACK. Using a sequence number and a storage space (cache), it is posible to know which messages have arrived to the subscriber and which have not. Data writers have access to this cache that per message stores the assigned sequence number, the history of the sent data values, and the history of whether the sample has been delivered to the reader.

4.2 Temporal bounds of the communication within the partitions'schedule

The hierarchical scheduling of partitions is related to the temporal bounds of the middleware communication across the partitions. In order to incorporate the transmission times to the schedule, the temporal bounds on the communication among the partitioned nodes are analyzed.

In IMA systems, temporal partitioning is guaranteed though partition windows. The operating system kernel applies a deterministic scheduling algorithm based on a static configuration file that indicates the time windows that are assigned to each partition. This is defined in different environments such as VxWorks [22] like other ARINC 653 compliant operating systems or the MultiPartes approach [27].

We undertake a local schedulability analysis for each system and consider the middleware cost within the partitions as follows. Let $\mathcal{V}$ be the set of n partitions of a system such that $\mathcal{V} = \{V_k\}$, $\forall k = 1..n$. The execution requirements of each partition are expressed as follows. A partition V_k requires to use the processor for C time units every period of T time units: $V_k = (C_k, T_k)$.

The execution life of a partitioned system follows a hierarchical approach of two main levels. At the top level, there is a sequence of equal duration time slots namely *major frames*. Each major frame F^M is divided into a number (q) of time slots of equal duration called *minor frames*, F^m: $F^M = \sum_{j=1}^{q} F^m$. Each minor frame is divided into a number r of time slots of different durations (C_k). Each of these time slots is assigned to a specific partition V_k: $F^m = \sum_{k=1}^{r} C_k$. During its assigned time slot, a partition has uninterrupted access to common resources.

A part from the numbers provided by middleware vendors, a middleware stability analysis is performed off-line. The analysis considers the message size required by the specific application (1024KB in our case), the required load conditions, and the specific hardware. In the case of the used implementation, DDS RTI Connext, this yields efficient values and a stable behaviour in different situations. The maximum value of the middleware communication cost (c_{mw}) is associated to very exceptional situations (< 0.1), although possible. Consequently, the maximum value c_{mw} is considered in the schedule as part of the corresponding partition(s) C_k that make use of it. For a partition, $c_{mw} = \frac{1}{\delta} C_k$ where δ is the ratio between the overall partition time and the middleware cost.

5. RESULTS

Experimental results are presented with the goal of assessing the cost of using DDS in a partitioned context for supporting remote communication across local and remote partitions. We intend to validate the suitability of the proposed system design; we analyze the system in terms of the communication time from the side of the invoking node. Even for an unreliable setting, we have measured a partition-level reliable communication implementation, i.e., including the client response reception. The goal is to show that the chosen middleware provides communication bounds for this distributed partitioned system. Also, results show the stability of the middleware in the partitioned system, that is compared against the control group (a bare machine deployment in an unrealiable DDS configuration). Results of the execution of the implemented system show its feasibility, given the performance of the middleware in different scenarios over a sufficiently large number of trials to obtain the maximum values of the communication enabled by the middleware. The presented data is compiled from 1000 iterations to provide meaningful information. The communication results show the overhead of the whole processing stack of a partition [7].

The monitoring system is implemented with DDS RTI Connext 5.2.0 over two networked machines connected by a 100Mbps Ethernet link. The hardware of the physical nodes is a double core Intel E3400 at 2.6Ghz and with 2GB RAM.

Initially, a control group experiment was performed to analyze the performance of the middleware on a favorable situation, i.e., a distributed setting with a best effort configuration set at the data writer and data reader entities of the distributed partitions. This scenario was analysed on,

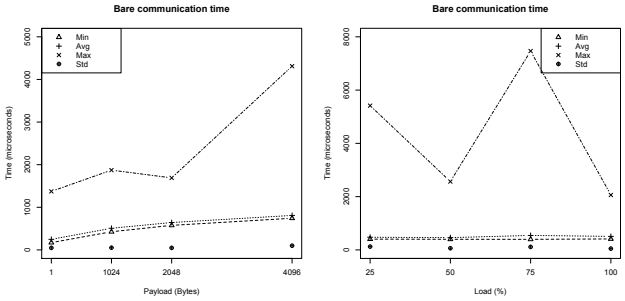

Figure 4: Bare machine; reliable communication; varing load.

both, empty load conditions and with progressive load up to 100%. The communication cost was increased by 7x in the partitioned scenario. Nevertheless, it was shown that the average case times ranged from 3x to 7x lower than the worst case.

Figure 4 shows the results of a bare machine deployment of the emulated scenario for the distributed monitoring application. This scenario fully replicates the system and it also provides a DDS reliable communication configuration. The scenario of figure 4(a) (left side graph) is executed with no additional load, whereas 4(b) (at the right side) presents a progressive loaded system. Resulting times follow the expected pattern, and a similar phenomena as in the previous scenario is observed. Average behavior for (a) are between 3.7x and 5.33x smaller than the maximum times. Again, the communication proves to be very stable; dispersion is around $50\mu s$ for all the cases except for the largest message size that is $98.2\mu s$. For the scenario (b) that shows progressive load, the average times are between 11x and 4x smaller that the worst case. Nevertheless, the system shows to be stable, and the dispersion is between $47.2\mu s$ and $123.20\mu s$.

In the last scenario, the full fledged deployment of the system is shown. Figure 5 shows the reliable communication

setting on a distributed partitioned system deployment with no additional load. Figure 6 shows the same scenario with progressive load.

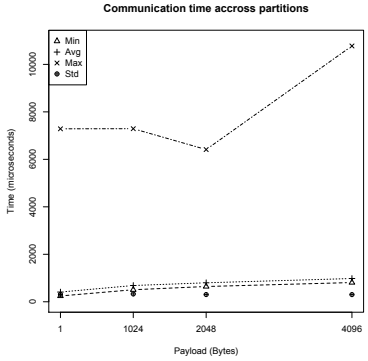

Figure 5: Partitioned system; reliable communication; no load.

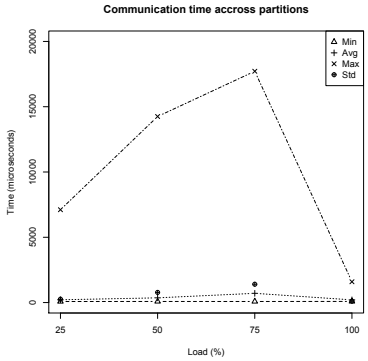

Figure 6: Partitioned system; reliable communication; varing load.

Results show that the system is very stable as standard deviation is consistently around the same magnitude and in the order of $300\mu s$ for the empty scenario and between 212 and $1400\mu s$ for the progressive load. Moreover, the worst case is, for the empty scenario, between 8x and 18x larger than the average case. For the progressive load tests, the average case is between 8x and 39x smaller than the worst case. Also, it can be seen that the minimum and average cases are very close. Results show that the fully distributed partitioned envirornment yields a stable execution and the communication overhead of the middleware follows the expected pattern.

6. CONCLUSION

The paper describes the design of a distributed partitioned system that supports communication of remote partitions through DDS middleware. Topics are defined to support the data centric model and it is exemplified for a distributed monitoring application. The communication overhead caused by the middleware and the partitioned setting is analyzed for a sufficient number of trials. The novelty of the paper is the exhaustive trials on the specific DDS technology and the measurements that provide the overhead of the whole middleware processing stack. Results show that the communication is stable even in presense of very high loads. The average case times are significanly smaller that the worst case (from 8x to 39x) and dispersion is 1.4ms for the worst possible scenario. We show that the overhead can be obtained for its integration in the partition resource assignment.

Acknowledgment

This work has been partly supported by the Spanish Ministry of Economy and Competitiveness through projects REM4VSS (TIN 2011-28339) and M2C2 (TIN2014-56158-C4-3-P).

7. REFERENCES

[1] Airbus Deutschland GmbH AFDX. *Avionics Full Duplex Switched Ethernet.* `http://www.afdx.com` (Accessed 2016)

[2] – *ARINC Specification 653: Part 1, Avionics Application Software Standard Interface, Required Services.* `https://www.arinc.com/` (Access 2016)

[3] A. Burns, R. Davis. *Mixed criticality systems – A review.* 8^{th} edition. Report. Univesity of York. July 2016.

[4] R. Davis, A. Burns, R. J. Bril, J. J. Lukkien. *Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised.* Real-Time Systems, vol.35(3), pp. 239–272. April 2007.

[5] C. Esposito, D. Cotroneo, S. Russo. *On reliability in publish/subscribe services.* Computer Networks, vol. 57(5), pp. 1318–1343. April 2013.

[6] M. García Valls, T. Cucinotta, C. Lu. *Challenges in real-time virtualization and predictable cloud computing.* Journal of Systems Architecture, vol.60(9), pp736–740. October 2014.

[7] M. García-Valls, P. Basanta-Val. *Analyzing point-to-point DDS communication over desktop virtualization software.* Computer Standards & Interfaces, vol. 49, pp. 11–21. January 2017.

[8] M. García-Valls. *A Proposal for Cost-Effective Server Usage in CPS in the Presence of Dynamic Client Requests.* 19^{th} IEEE International Symposium on Real-Time Distributed Computing (ISORC), pp. 19–26. York , UK. May 2016.

[9] M. García-Valls, P. Basanta-Val. *Usage of DDS Data-Centric Middleware for Remote Monitoring and Control Laboratories.* IEEE Transactions on Industrial Informatics vol. 9(1), pp. 567–574. 2013.

[10] M. García-Valls, L. Fernández Villar, I. Rodríguez López. *iLAND: An enhanced middleware for real-time reconfiguration of service oriented distributed real-time Systems.* IEEE Transactions on Industrial Informatics, vol. 9(1), pp. 228–236. February 2013.

[11] M. García-Valls, C. Calva-Urrego. *Improving service time with a multicore aware middleware.* 32^{nd} ACM/SIGAPP Symposium on Applied Computing (SAC). Marrakech, Morocco. April 2017.

[12] M. García-Valls, C. Calva-Urrego, J. A. de la Puente, A. Alonso, *Adjusting middleware knobs to assess*

scalability limits of distributed cyber- physical systems. Computer Standards & Interfaces, January 2017. DOI: 10.1016/j.csi.2016.11.003

[13] S. Groesbrink, S. Oberthr, D. Baldin. *Architecture for adaptive resource assignment to virtualized mixed-criticality real-time systems.* 4th Workshop on Adaptive and Reconfigurable Embedded Systems (APRES12), vol.10(1). ACM SIGBED Review, 2013.

[14] C. Gu, et al. *Partitioned mixed-criticality scheduling on multiprocessor platforms.* In *Proc.* of IEEE Design, Automation and Test in Europe Conference and Exhibition (DATE), pp.1–6. 2014.

[15] A. Hakiri, P. Berthou, A. Gokhale, D. C. Schmidt, T. Gayraud. *Supporting end-to-end quality of service properties in OMG data distribution service publish/subscribe middleware over wide area networks.* Journal of Systems and Software, vol. 86(10), pp. 2574–2593. October 2013.

[16] H. Kopetz, G. Bauer. *The time-triggered architecture.* Proceedings of the IEEE, vol 91(1), pp. 112–126. 2003.

[17] R. Obermaisser et al. *Fundamental Design Principles for Embedded Systems: The Architectural Style of the Cross-Domain Architecture GENESYS.* IEEE ISORC 2009, pp. 3-11. 2009.

[18] Object Management Group – OMG. *A Data Distribution Service for Real-time Systems Version 1.4.* `http://www.omg.org/spec/DDS/1.4` 2015.

[19] Object Management Groug (OMG). *The Real-time Publish-Subscribe Wire Protocol DDS Interoperability Wire Protocol Specification, v2.2.* September 2014.

[20] C. M. Otero Pérez, et al. *QoS-Based Resource Management for Ambient Intelligence.* Chapter on *Ambient Intelligence: Impact on Embedded System Design*, pp. 159–182. Kluwer Academic Publishers. 2003.

[21] The Open Group. *Future Airborne Capability Environment – FACE.* `http://www.opengroup.org/face` (Accessed 2016)

[22] P. Parkinson, L. Kinnan. *Safety-Critical Software Development for Integrated Modular Avionics.* VxWorks. White Paper.`www.windriver.com` (Accessed 2016)

[23] H. Pérez, J. J. Gutiérrez, S. Peiró, A. Crespo. *Distributed architecture for developing mixed-criticality systems in multi-core platforms.* The Journal of Systems and Software, vol.123, pp. 145–159. 2017.

[24] RTCA DO-297/ED-124. *Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations.* `www.rtca.org` and `www.eurocae.org` (Accessed July 2016)

[25] T. Rizano, L. Abeni, L. Palopoli. *Experimental evaluation of the real-time performance of publish-subscribe middleware.* In Proc. of 2_{nd} International Workshop on Real-Time and Distributed Computing in Emerging Applications (REACTION 2013). Vancouver, Canada. December 2014.

[26] R. Schantz, et al. *Towards adaptive and reflective middleware for network-centric combat systems.* Encyc. of Software Engineering. Wiley& Sons. 2002.

[27] S. Trujillo, A. Crespo, A. Alonso, J. Perez. *MultiPARTES: Multi-core partitioning and virtualization for easing the certification of mixed-criticality systems.* Microprocessors and Microsystems, vol.38, pp.921–932. November 2014.

[28] S. Vestal. *Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance.* In *Proc.* of the IEEE Real-Time Systems Symposium (RTSS), pp. 239–243. December 2007.