Data-centric distribution technology in ARINC-653 systems

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

Abstract—**Standard distribution middleware has recently emerged as a potential solution to interconnect distributed systems in the avionics domain, as it would bring important benefits throughout the software development process. A remaining challenge, however, is reducing the complexity associated with current distribution standards which leads to prohibitive certification costs. To overcome this complexity, this work explores the use of the DDS distribution standard on top of a software platform based on the ARINC-653 specification. Furthermore, it discusses how both technologies can be integrated in order to apply them in mission and safety-critical scenarios.**

Keywords—distributed systems; middleware; ARINC-653; hypervisor; DDS; real-time systems.

I. INTRODUCTION

Today's airborne systems typically rely on the integrated modular avionics (IMA) architecture to simplify the development of onboard software. One important aspect of this architecture is the ARINC-653 specification [1], where the partitioning concept can provide applications with strong temporal and space isolation, thus easing their verification, validation and certification [2].

In this kind of systems, partitions are interconnected using ARINC-653 communication services and through special purpose networks such as AFDX [3]. However, partitioned systems are shifting to rely on standard distribution middleware for communications, as it can bring important features to avionic systems (e.g., interoperability, location transparency or the abstraction of network services). One major effort in this direction is the Technical Standard for Future Airborne Capability Environment (FACE) [4], which aims to standardize approaches based on open standard solutions for airborne systems. Among others, FACE includes the Data Distribution Service for Real-Time Systems (DDS) [5] as a suitable candidate to provide distribution capabilities within avionic systems.

The DDS standard is starting to be applied to emerging real-time applications such as those related to cloud environments [6] or cyber-physical systems [7]. Nevertheless, the use of DDS in safety-critical systems is still an open challenge that is being addressed through the extension of DDS with a safety-critical profile [8][9] suitable for partitioned systems.

An early experience dealing with the integration of DDS into partitioned systems was introduced in [10], where the use of DDS in partitioned applications with low levels of criticality is discussed. However, safety-critical applications should rely on the ARINC-653 facilities for inter-partition communications. Therefore, this paper provides a step forward towards the integration of both technologies by exploring the use of DDS on top of the ARINC-653 communication services. To this end, it not only identifies a set of integration issues when using both standards, but it also proposes solutions for them. Furthermore, the work includes the development of a partitioned distributed real-time platform as a proof of concept, and a preliminary evaluation about the proposed integration.

To the best of our knowledge, few research papers have dealt with the use of DDS in safety-critical environments. The work in [11] proposes an architecture which relies on DDS for communications in the automotive domain. Furthermore, [12] presents an architecture for fractionated spacecraft in which a middleware layer provides high-level abstractions for client/server and publisher/subscriber communications based on CORBA [13] and DDS [5], respectively. The approach of this paper differs from the existing literature on DDS for safety-critical systems by integrating a new communication service into DDS for inter-partition communications instead of relying on the traditional UDP/IP transport. Similarly to our work, the use of other transports in DDS is addressed in [14] and [15] which explore the integration of the CAN bus and the FlexRay communications system, respectively.

The execution of DDS in a virtual environment was previously dealt in [16] and [17]. Unlike our approach, these works rely on general-purpose virtualization technology. More generally, comprehensive surveys on challenges for real-time virtualization can be found in [18] and [19]. While the former deals with real-time embedded systems, the latter

This work has been funded in part by the Spanish Government and FEDER funds under grant number TIN2011-28567-C03-02 (HIPARTES).

focuses on real-time virtualization for cloud computing systems.

This document is structured as follows. Section II introduces a brief review of the DDS and ARINC-653 specifications. Section III analyzes the differences between these standards from the communication perspective and proposes a system design to integrate both technologies. Section IV presents the development of the proposed partitioned distributed real-time platform and evaluates the feasibility of the approach. Finally, Section V summarizes the main contributions and the lines of future work.

II. Background

A Overview of DDS

The Data Distribution Service for Real-Time Systems [5] is a standard based on the publisher-subscriber paradigm which provides anonymous, decoupled and asynchronous communications. This standard also addresses the needs of real-time systems [20], as it provides applications with a wide set of configurable QoS parameters to control non-functional properties such as data timeliness [21].

The DDS distribution model is based on a fully distributed *Global Data Space* where data may flow from one or many publishers to one or many subscribers. The data to exchange are defined by means of *Topics,* which are written (produced) and read (consumed) by *DataWriters* (DW) and *DataReaders* (DR), respectively.

When a new application joins the distributed system, middleware will automatically detect its presence through a special service called *Discovery*. This service implements the process by which DDS entities can find out information (e.g., Topic name or QoS parameters) about the presence and characteristics of any other entity within the distributed system. Henceforth, this kind of information will be referred to as *discovery data*.

To guarantee the interoperability among different implementations, DDS relies on the DDS Interoperability Wire Protocol (DDSI) [22]. DDSI describes how data should be disseminated among nodes by defining a set of exchange information protocols and message formats. Although DDSI is particularly oriented to using the UDP/IP protocol, it does not preclude the use of other transport protocols.

B Overview of communications in ARINC-653

The ARINC-653 specification describes the baseline operating environment for avionics software used within large and complex IMA [1]. It consists of three main parts covering the mandatory and optional services, together with the conformity tests. Additionally, Part 4 [23] was introduced in 2012 and defines a strict subset of services for minimal onboard systems.

The key feature at the foundation of ARINC-653 is partitioning, which ensures time and space isolation between different partitions. As a result of this isolation, a set of applications with different levels of criticality can be executed in the same hardware platform, and they can also be certified together even if they have been developed by different companies.

The communication among partitions is performed by exchanging messages through the use of *channels*. A channel is composed of one single sending port and one or many receiving ports. A port can be configured to be either the source or the destination of messages, but not both. Furthermore, communication ports support two modes of transfer: (1) *sampling mode*, which supports unicast, multicast and broadcast messages; and (2) *queuing mode* in which only unicast messages are obligatory. The main difference is that queuing ports allow multiple messages to be buffered, whereas sampling ports do not (i.e. received messages always overwrite previous data).

As network links and workload are controlled in safety-critical scenarios, channels, ports and their attributes should be entirely defined at configuration time. In a system built around the space and time partitioning paradigm, the *application supplier* is only aware of the information which is sent or received within its own partition, while the *system integrator* has the overall responsibility of configuring the communication channels in order to ensure the correct routing of messages from source to destination/s. From the viewpoint of software development, this strict separation of roles provides important benefits [1][2] and so it should be addressed in the integration with distribution middleware.

III. System design

The DDS distribution model resembles the one proposed by the ARINC communication service, as it provides flexibility and decoupling features which are required for composability and reuse. For instance, ARINC proposes an asynchronous messaging service where the *application suppliers* only know about the specific data that must be produced or consumed. Similarly, the DDS conceptual model is based on publisher and subscriber entities that respectively write (produce) and read (consume) data independently of its location.

However, there is a set of differences among the family of DDS and ARINC-653 standards, which compromises the integration of the two technologies as described below in the list of issues and possible solutions.

A Communications

Issue #1. While DDS supports one-to-one, one-to-many and many-to-one communication models, the ARINC communication service supports the first two models

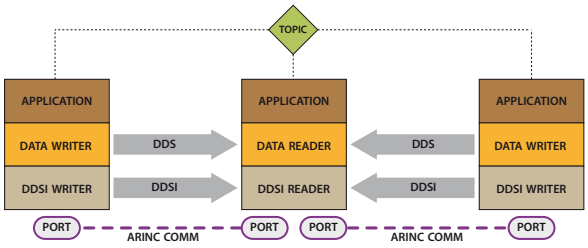

Fig. 1 Many-to-one communication model for DDS through ARINC-653 ports

through sampling ports, but only unicast communications are required for queuing ports. Although the many-to-one communication model is not explicitly prohibited in the standard, it is not allowed by some underlying networks (e.g., AFDX [3]) and implementations (e.g., XtratuM [24]).

Solution #1. The many-to-one communication model is one basic block for implementing redundancy at the DDS layer [25], as several DWs can be in charge of updating the same topic [5]. As shown in Figure 1, support for this communication model should be built at the DDSI level, as the underlying transport over ARINC-653 may only support one-to-one and one-to-many communications.

Issue #2. Although DDS communication mode is essentially asynchronous, it also supports mechanisms for synchronous access to data. However, the synchronous communication mode is not explicitly considered in the ARINC-653 Part 1 specification [1], which leaves the use of this feature as optional as long as it is guaranteed and provided by the underlying system when required. It is also worth noting that the ARINC-653 Part 4 specification [23] does not allow blocking calls to be used.

Solution #2. There is a set of different communication mechanisms supported by DDS. On the publisher side, the transmission of data is essentially asynchronous. Moreover, it also supports the synchronous behaviour by blocking the calling thread until the reception of acknowledgments from the matched DRs. On the subscriber side, data can be received through (1) polling, (2) listeners to asynchronously access the data, and (3) wait-sets for the synchronous access to data [5]. If the underlying system is not suited to support blocking calls or it must be ARINC-653 Part 4 compliant, middleware must rely on polling and listener mechanisms which should be implemented accordingly (i.e., without using blocking calls to listen for incoming messages).

B Transport protocols

Issue #3. One DDSI message is comprised by a fixed-size header and a variable number of sub-messages, as it may include not only user but also protocol data. It is particularly critical in ARINC-like systems where the maximum size of messages must be specified at configuration time. This point may also jeopardize the determinism of the distributed system [21].

Solution #3. Current DDSI protocol should be restricted in ARINC-like systems to preserve the boundary of messages from upper layers and therefore the maximum message size can be computed as the sum of the maximum size of the topic and the protocol headers. Otherwise, the maximum size of a DDSI message that can be sent or received by the ARINC-653 communication service must be restricted at configuration time.

Issue #4. For interoperability purposes, DDSI over UDP/IP relies on pre-defined communication ports to receive user and discovery data, and these ports can be shared among multiple sources. In ARINC-like systems, a receiving port should receive messages from a single sending port (see Issue #1 and Figure 1).

Solution #4. A new mechanism to associate the corresponding port name/s with each DR/DW entity should be implemented. For example, this kind of information can be automatically obtained from a static discovery service (see subsection D). As multiple DWs can be updating the same topic at the same time, it is important to remark that the use of the topic name to identify the corresponding port name is not suitable due to the single data source issue illustrated in Figure 1.

C Addressing scheme

Issue #5. Middleware requires a way to contact with remote nodes (i.e., the assignment of identifiers to entities in order to denote their location).

Solution #5. A data flow is uniquely identified within DDSI by the set {transport, address, port}. This addressing information or locator is used to send messages to matching readers through a specific transport protocol.

In the case of UDP/IP, the locator is composed of the UDP destination port and the IP destination address. For the ARINC-653 communication service, the locator shown in Table 1 is proposed. Unlike UDP/IP, the ARINC-653 communication service relies on the source port to send a message to the destination, as they are statically connected by means of a channel.

D Discovery

Issue #6. DDS requires information to be obtained about the presence of remote entities and their properties at runtime,

Table 1: DDS locator for ARINC-653 communications

Locator_t		
attribute	**type**	**value**
kind	long	LOCATOR_KIND_ARINC653
channel	unsigned long	Identifies the ARINC653 channel
port_name	string	Identifies the ARINC653 source port

but ARINC-653 channels are entirely defined at configuration time by the system integrator.

Solution #6. According to the standard, *Discovery* is a two-step process where new applications announce their presence within the distributed system. When two DDS applications have discovered one another, they exchange information about their DWs and DRs and then the possible matches are determined. As a result, discovery can be implemented following three different strategies:

- Dynamic, in which DDS entities are able to exchange information with any other entity on the network, and they are dynamically matched and tracked at runtime (i.e., plug-and-play system). The information is automatically exchanged in the context of discovery data by a pre-defined set of built-in entities. This represents the standard discovery mechanism defined by the specification [5].
- Static, in which all the information about DDS entities is statically configured before runtime (i.e., no data is sent through the network).
- Quasi-static, in which part of the information is sent through the network, while the rest is statically configured [12] [26].

ARINC-like systems are characterized by using pre-configured communication links with static workload. Under this scenario, it seems reasonable that each node does not have information about the whole distributed system (i.e., non-matching entities should not be discovered). Furthermore, since decoupled communications is a desired feature at the *application supplier* level, the location of DDS entities should be built upon the *system integrator* configuration. While the use of static discovery is the natural choice, the quasi-static approach can be considered with some restrictions. When the same ARINC port is used for the distribution of both application and discovery data, communication channels are simpler to configure at the cost of certain complexity at runtime (i.e., there is an initial discovery phase before transmitting any application data). Otherwise, the *system integrator* should be responsible for specifying the required configurations related to the DDS discovery process (e.g., channels for the built-in entities).

E Mapping of entities

Issue #7. DDS entities and their properties should contain enough information to comply with the attributes required by the ARINC-653 communication service.

Solution #7. An ARINC-653 communication port is basically defined by a set of 5 attributes:

- *Transfer direction.* DWs represent source ports, whereas DRs are identified as destination ports.
- *Port name.* New information to denote the corresponding port name/s should be attached to DWs and DRs, as proposed in Solution #4.
- *Message storage*. According to the DDS standard, the message storage requirements are specified by History QoS and its corresponding *depth* parameter [5]. Therefore, the maximum number of messages associated with a queuing port should be equal to the value of the *depth* parameter.
- *Mode of transfer.* This attribute is closely related to the message storage requirements, as multiple messages can only be buffered by queuing ports.
- *Message length.* DDS messages should be restricted in size, as proposed in Solution #3.

The basic mapping of DDS entities to the underlying ARINC-653 communication service is shown in Table 2.

Table 2: Mapping of DDS and ARINC-653 entities

ARINC-653	DDS
Port Name	New entity information
Message Storage	Depth parameter in History QoS
Message Length	Max IDL type + DDSI header
Transfer Direction (source/destination)	DataWriter/DataReader
Mode of Transfer (sampling/queuing)	Depth parameter in History QoS

IV. EVALUATION

This section aims to describe a partitioned platform to validate the proposed approach by obtaining a set of performance metrics. The hardware platform is composed of an Intel i5-4570 processor with a clock rate of 3.2 Ghz. The software platform is shown in Figure 2 and consists of a single C application distributed through RTI Connext Micro v2.2.3[1]. This application is running on top of a real-time operating system called MaRTE OS (v1.9) [27], which in turn is executed on top of an hypervisor called XtratuM

1. RTI Connext DDS Micro is available at http://www.rti.com

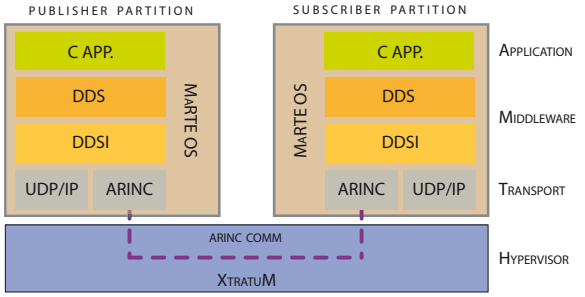

Fig. 2 Software architecture in the partitioned platform

(v3.7) [24]. This hypervisor is responsible for providing temporal and space isolation, and it allows a complete operating system to be executed in each partition.

A prototype implementation of the proposed approach has been developed as a proof of concept. The development of this prototype has focused on validating the proposed software integration, leaving the use of safety-related hardware and implementation efficiency for future work. To this end, the DDS implementation has been extended to provide support for the ARINC-653 communication service, as described below.

A Extensions to the DDS implementation

RTI Connext Micro is a minimal DDS middleware aimed at resource-constrained devices and which has been designed with certification requirements in mind. This middleware relies on a set of abstraction layers to support a variety of platforms and transport protocols. The abstract communication interface is used between different I/O layers in the core library and therefore the following extensions have been added:

- The ARINC-653 middleware interface to comply with the requirements imposed by the core library, such as operations to send or receive DDSI messages, create I/O threads, bind to destination addresses, etc.
- The ARINC-653 low-level interface to provide operations to transmit and receive data via the ARINC-653 communication service. This not only includes the functionality for sending and receiving data, but it also implements the operations to create the required communication ports. Furthermore, a blocking receive operation has been implemented to allow I/O threads to wait for incoming data instead of polling for it. This functionality relies on the extended interrupt mechanism provided by the hypervisor to notify partitions about incoming messages, and it can be of interest for a number of partitioned systems as specified in Issue #2.
- New addressing information for the ARINC-653 communication service, and the corresponding functionality to manage it within the core library.

B Extensions to the hypervisor

XtratuM is an ARINC-653-like hypervisor especially designed for real-time embedded systems developed at the Real-Time System Group of the Instituto de Automática e Informática Industrial of the Universitat Politècnica de València (Spain). This hypervisor features spatial and temporal isolation by (1) allocating partitions to a unique address space which is not accessible by other partitions and (2) through a fixed cyclic scheduler, respectively.

In the communication service implemented by XtratuM, channels, ports and their attributes are entirely defined through an XML configuration file. In the case of channels, their current configuration does not include any identifier that allows their information to be linked with the discovery service. Therefore, the discovery information is currently provided to the prototype by hand, although an automatic generation of this configuration from the hypervisor configuration file is planned.

C Results

For our purposes, this paper will only consider a steady and static partitioned system (i.e., initialization and discovery processes have been completed and late-joiners are strictly forbidden) in which applications are connected through the ARINC-653 communication service provided by XtratuM.

The test will measure the round-trip latency of a remote operation, that is, the time between the call to publish data and the return of the read operation. The data payload is bounded to 100 bytes, and the operation is executed 10,000 times to estimate the average, maximum, and minimum times, together with the standard deviation and the 99th percentile (i.e., the value below which 99 percent of the measurements are found). The results for the performance analysis are shown in Table 3, which includes three different evaluations that are detailed below.

The first evaluation aims to estimate the performance of the ARINC-653 transport for DDS. In this case, the application only consists of one partition and, therefore, this evaluation is not influenced by the overhead associated with the time partitioning. From the results obtained for the single partition scenario, it can be observed that the DDS implementation together with the proposed extensions is a

Table 3: Measurements of the round-trip latency (in µsecs)

	Min	Avg	Max	Std	Per99
SINGLE PARTITION	37	42	102	4	64
TWO PARTITIONS	333	384	741	16	414
TWO PARTITIONS WITH 4KB OF PAYLOAD	342	388	779	35	440

lightweight middleware which, as expected, presents a low standard deviation due to the low CPU utilization associated with the proposed test.

The second evaluation measures the performance of the partitioned application. In this case, the system holds two partitions: (1) the Publisher partition, and (2) the Subscriber partition which reads the data and sends the reply. The system has been configured to have a dedicated time window of 200μs for the Publisher, and 200μs for the Subscriber partition, resulting in a scheduling plan repeated every 400μs. As shown in Table 3, the operation for the single partition scenario takes a maximum of 102μs, while this value is 741μs for the partitioned system. Therefore, the variation in performance shown in Table 3 for the two partition scenario strongly depends on the nature of the partitioned system and its time window configuration.

An additional evaluation has been carried out to evaluate the impact of the proposed approach when the payload is increased from 100 bytes to 4 Kilobytes. In this case, the minimum, average and maximum times obtained are similar to the previous scenario, but the standard deviation has slightly increased. Again, an important part of the maximum latency is due to the time isolation property, but it is also due to the current prototype implementation. Further work on the implementation efficiency could minimize the extra-delay incurred when the processing is completed during the next time window.

Finally, it should be noted that most of the measurements are close to the average value, as can be deduced from the 99th percentile.

V. Conclusions and Future Work

This paper represents a first step towards the integration of the DDS and ARINC-653 standards. As has been highlighted through the set of integration issues listed in this paper, the current DDS specification requires applying some extensions to cope with the restrictions imposed by ARINC-like partitioned systems. Hence, the proposed extensions include:

- Bounded size of DDSI messages, as the current specification does not impose any limitations in the number of sub-messages.
- The use of static discovery, or quasi-static discovery under certain conditions.
- Conditioned support for synchronous communications, as its usage is strictly system-dependent.
- A new locator to uniquely identify a data flow using the ARINC-653 communication service.
- Support for the many-to-one communication model should be implemented at the DDSI level.
- New information attached to each DW/DR entity to denote the corresponding ARINC-653 communication port. This information also provides support for the many-to-one communication model.

Additionally, other integration issues have also been addressed related to the configuration and mapping between the DDS distribution entities and the ARINC-653 communication service entities, such as communication ports and channels.

As a result of the evaluation, we can conclude that the approach provides sufficiently low dispersion to build predictable applications. However, there is still room for improvement in terms of efficiency which deserves further research.

In the short term we plan to complete our implementation to evaluate its benefits and performance and to analyze its limits and constraints in complex scenarios. Finally, further investigation is also required to fully determine which features of the DDS specification can be applied in future airborne systems, such as the list of supported QoS configurations or the use of keys.

References

[1] Airlines Electronic Engineering Committee, Aeronautical Radio INC. “Avionics Application Software Interface, required Services”. ARINC Specification 653-1. November, 2010.

[2] P.J. Prisaznuk. “ARINC 653 role in Integrated Modular Avionics (IMA)”, Proc. of the 27th IEEE/AIAA Digital Avionics Systems Conference (DACS), pp. 1.E.5 1-10, 2008.

[3] Airlines Electronic Engineering Committee, Aeronautical Radio INC. “Aircraft Data Network, Part 7 - Avionics Full Duplex Switched Ethernet (AFDX) Network”. ARINC Specification 664-7. September, 2009.

[4] Technical Standard for Future Airborne Capability Environment, Edition 2.1. The Future Airborne Capability Environment (FACE) Consortium, The Open Group. 2014.

[5] Object Management Group. Data Distribution Service for Real-time Systems. OMG Document, v1.2, formal/07-01-01. 2007.

[6] A. Hakiri, P. Berthou, A. Gokhale, D. Schmidt and G. Thierry. “Supporting SIP-based end-to-end Data Distribution Service QoS in WANs”, Journal of Syst. Software, http://dx.doi.org/10.1016/j.jss.2014.03.078. 2014.

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

[8] R. Wahlin, and G. Hunt, “Towards a Safety Critical profile for DDS,” Real-time and Embedded Systems Workshop, Arlington, VA (USA), 2009.

[9] R. Karoui, and A. Corsaro. “Real time Data Distribution for Airborne Systems,” Workshop on Real-time, Embedded and Enterprise-Scale Time-Critical Systems, Washington DC, (USA), 2011.

[10] H. Pérez, and J. J. Gutiérrez, “Towards the integration of data-centric distribution technology into partitioned embedded systems”, Proc. of the 2nd International Workshop on Real-Time and Distributed Computing in Emerging Applications (REACTION), Vancouver (Canada), 2013.

[11] S. Sommer, A. Camek, K. Becker, C. Buckl, A. Zirkler, L. Fiege, M. Armbruster, G. Spiegelberg, A. Knoll. “RACE: A Centralized Platform Computer Based Architecture for Automotive Applications”, Proc. of the 2013 IEEE International Electric Vehicle Conference (IEVC), pp. 1-6, 2013.

[12] A. Dubey, W. Emfinger, A. Gokhale, G. Karsai, W.R. Otte, J. Parsons, C. Szabo, A. Coglio, E. Smith and P. Bose. “A software platform for fractionated spacecraft”, Proc. of the 2012 IEEE Aerospace Conference, pp. 1-20, 2012.

[13] Object Management Group. Realtime Corba Specification. OMG Document, v1.2. formal/2005-01-04. 2005.

[14] R. Ekik and S. Hasnaoui, “Application of a CAN bus transport for DDS middleware”, In Second International Conference on the Applications of Digital Information and Web Technologies (ICADIWT), pp. 766 –771. 2009.

[15] R. Bouhouch, H. Jaouani, A.B. Ncira and S. Hasnaoui. “DDS on Top of FlexRay Vehicle Networks: Scheduling Analysis”, International Journal of Computer Science and Artificial Intelligence, Vol. 3 Iss. 1, pp. 10-26. 2013.

[16] Y. Cho, J. Choi, and J. Choi, “An integrated management system of virtual resources based on virtualization API and data distribution service,” Proc. of the ACM Cloud and Autonomic Computing Conference, New York (USA), 2013.

[17] M. Garcia-Valls, P. Basanta-Val and R. Serrano-Torres. “Benchmarking communication middleware for cloud computing virtualizers”. Proc. of the 2nd International Workshop on Real-Time and Distributed Computing in Emerging Applications (REACTION), Vancouver (Canada), 2013.

[18] Z. Gu and Q. Zhao. “A State-of-the-Art Survey on Real-Time Issues in Embedded Systems Virtualization” Journal of Software Engineering and Applications, 5, 277-290, doi:10.4236/jsea.2012.54033. 2012.

[19] M. Garcia-Valls, T. Cucinotta and C. Lu. “Challenges in Real-Time Virtualization and Predictable Cloud Computing”, in Elsevier Journal of Systems Architecture, DOI: 10.1016/j.sysarc.2014.07.004. 2014.

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

[21] H. Pérez, and J. J. Gutiérrez, “A Survey on Standards for Real-Time Distribution Middleware”. ACM Computing Surveys, Vol. 46, No. 4, Article 49. 2014.

[22] Object Management Group. The Real-time Publish-Subscribe Wire Protocol. DDS Interoperability Wire Protocol Specification. OMG Document, v2.1, formal/2010-11-01, 2010.

[23] Airlines Electronic Engineering Committee, Aeronautical Radio INC. “Avionics Application Software Interface, Subset Services”. ARINC Specification 653-4. June, 2012.

[24] M. Masmano, I. Ripoll, A. Crespo, and J.J. Metge, “Xtratum a hypervisor for safety critical embedded systems,” Proc. of the 11th Real-Time Linux Workshop, Dresden (Germany), 2009.

[25] M. Ryll and S. Ratchev. “Application of the Data Distribution Service for Flexible Manufacturing Automation”, Proc. of World Academy of Science: Engineering & Technolog, Vol. 43, pp. 178, 2008.

[26] P. Pazandak. “Affordable Avionics through Commercial Certified Middleware”, Proc. of the Safe & Secure Systems & Software Symposium (S5), Dayton (USA), 2014.

[27] M. Aldea and M. González. “MaRTE OS: An Ada Kernel for Real-Time Embedded Applications”. Proc. of the International Conference on Reliable Software Technologies, Ada-Europe, Leuven, Belgium, LNCS 2043, 2001.