Adaptive Architectures for Future Highly Dependable, Real-Time Systems

Brian Ford[1], Peter Bull[2], Alan Grigg[3] , Lin Guan[4] and Iain Phillips[5]

[1] BAE Systems, Great Britain, b.ford@lboro.ac.uk
[2] Loughborough University, Great Britain, p.bull@lboro.ac.uk
[3] BAE Systems, Great Britain, a.grigg@lboro.ac.uk
[4] Loughborough University, Great Britain, l.guan@lboro.ac.uk
[5] Loughborough University, Great Britain, i.w.phillips@lboro.ac.uk

Abstract

Many present-day safety-critical or mission-critical military applications are deployed using intrinsically static architectures. Often these applications are real-time systems, where late responses may cause potentially catastrophic results. Static architectures allow system developers to certify with a high degree of confidence that their systems will provide correct functionality during operation, but a more adaptive approach could provide some clear benefits. In particular, the ability to dynamically reconfigure the system at run time would give increased flexibility and performance in response to unpredictable or unplanned operating scenarios. Many current dynamic architectural approaches provide little or no features to facilitate the highly dependable, real-time performance required by critical systems. The challenge is to provide the features and benefits of dynamic architectural approaches while still achieving the required level of performance and dependability.

This paper describes the early results of an ongoing research programme, part funded by the Software Systems Engineering Initiative (SSEI), aimed at developing a more adaptive software architecture for future military systems. A range of architectures with adaptive features (including object-based, agent based and publish/subscribe) are reviewed against the desirable characteristics of highly dependable systems. A publish/subscribe architecture is proposed as a potential way forward and a discussion of its advantages and disadvantages for highly dependable, real-time systems is given.

Keywords – Real-Time, Adaptive, IMS, SOA, DCPS

1 Introduction

Requirements for dependable systems are common within military applications and can often be categorised as either mission-critical, where system failure can lead to loss of mission effectiveness, or safety-critical where failure can lead to loss of human life. Generally to meet these dependability requirements the system must be certified against governmental or internationally recognised standards. These standards often require evidence of rigorous testing alongside formal analysis of the software system. To simplify this, software systems tend to have very static architectures, where no or limited changes are allowed to occur when the system is operational. This allows highly deterministic behaviour of the system to be shown.

This paper uses Integrated Modular Systems, an established approach for building distributed software and electronics architectures, as a case study to identify features of current dependable, real time, software systems. We then review a range of modern software architectural approaches to identify their adaptive features and their suitability for working within highly dependable environments. Finally, a publish/subscribe architecture is proposed as a potential way forward and a discussion of its advantages and disadvantages for highly dependable, real-time systems is given.

2 Background: IMS approach to developing highly dependable systems

Highly dependable software systems are designed to be extremely deterministic, where predictable and repeatable performance is a necessity. As mentioned earlier, traditional approaches to building highly dependable software systems are based upon using static architectures. Our research is aimed at extending one such approach called Integrated Modular Systems (IMS) to include more adaptive features. This section introduces IMS and discuses the static features within the IMS software architecture, with the intention of showing how conventional highly dependable software architectures are developed.

Integrated Modular Avionics (IMA) is an architectural approach for developing the electronics and software systems onboard aircraft. IMS extends the approach outside the established avionics domain. Key concepts in IMS include [1]:

- Modular and standardised hardware cards contained within integrated cabinets. Multiple cabinets may be distributed throughout the vehicle/platform.

- Distributed communication between modules through deterministic hardware buses or networks.
- The adoption of a multilayer software architecture, insulating application, operating system and hardware drivers from changes through common and open APIs.
- The ability to support mixed criticality levels within the application set.
- The use of open standards for both software and hardware.

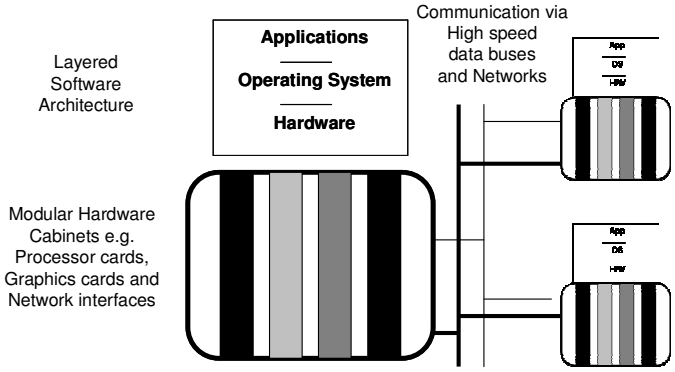

Figure 1: Key concepts in IMS

Standardisation attempts have been made for both commercial and military IMS systems. Our focus is on the UK MOD Defence Standard 00-74 [2]. The following is an overview of features that contribute to making the software and hardware architecture in IMS deterministic:

Runtime blueprints which contain static system configurations including resource allocation, real-time scheduling parameters and hardware support. Transitions between these configurations are also captured statically. This allows IMS systems to provide some reconfiguration capability, (for example adapting to faults), while maintaining high levels of integrity.

Inter-process communication channels which are statically defined within each configuration. These communication channels are unidirectional, connectionless links and can either be onboard individual hardware modules or offboard. The IMS API allows applications to write and read from these channels. Processes can not create or remove channels once a configuration has been loaded.

Executing processes controlled by a hard real time schedule. Each configuration in the blueprint defines a static set of processes. No processes can start dynamically. Process execution is controlled by a hard real-time scheduler which guarantees that predefined deadlines of each individual process are met.

Spatial partitioning of processes, meaning that each process is allocated its own memory space and no other process can access that space. This ensures no process can corrupt another processes memory. This is enforced by the operating system.

Static Device configurations are captured in the blueprint. At the software level this means that only device drivers specified in the blueprint configurations can be loaded.

These static features introduce limitations on how the system can adapt at runtime, however, they have been recommended within the standard to ensure that IMS systems are highly predictable. Although we discuss adaptive features in the next section, future research is required into the tradeoffs between predictability and adaptability before these features can be included within IMS systems.

3 Emerging Challenges for Dependable Architectures

Emerging challenges raised by modern and future military platforms are increasingly requiring support for more adaptive software architectures, whilst maintaining the same levels of dependability achieved by approaches like IMS. Some areas where these challenges arise include:

Autonomous systems which are gradually moving the responsibility of dealing with unpredictable environments from humans to software. These systems are likely to include software algorithms that are difficult or impossible to introduce static temporal and resource bounds to in advance. The ability to support increasingly non-deterministic algorithms with continually changing resource requirements is predicted.

Complex distributed systems which are becoming increasingly difficult to use traditional certification methods, due to the complexity of capturing and analysing all possible configuration and communication scenarios. Furthermore, when safety or mission critical components of these systems are being upgraded or changed, recertification of large parts of the system is often necessary. Adaptive architectures may be able to manage and enable these complex communication networks and allow for easier incremental technology insertion.

Embedded systems which often have significant size and weight restrictions, particularly in the field of military avionics. This means hardware resources like processing, memory and power may be limited. Static architectures can leave system resources underutilized as resources are often allocated based on worst case execution scenarios. Introducing adaptive techniques can help optimise resource usage.

Network Enabled Capability where communication networks are used to enable the armed forces to work more effectively together by increasing the sharing and exploitation of information between platforms and personnel. These networks are expected to be highly

dependable to support mission critical environments, but are also expected to be highly adaptable, for example allowing the ad-hoc creation of networks between platforms.

4 Adaptive Architectural Approaches

The following section gives a brief description of common mainstream software architectural approaches. Their adaptive features and suitability to highly dependable, real-time systems are discussed.

4.1 Object Oriented Approaches

Common Object Request Broker Architecture (CORBA) is an OMG standard for distributing functionality throughout a system [3]. CORBA facilitates interoperability at the object level, the main advantage of which is that most applications are currently based on object oriented design therefore little additional effort in design or redesign of applications is necessary. The use of objects does, however, tend to lead to tighter coupling of components, given the low level of granularity of object interfaces. Figure 2 shows a high level example of systems using ORB (Object Request Broker) to ORB based communication using the Internet Inter-Orb Protocol (IIOP) to communicate over TCP/IP. The ORB facilitates the communication allowing function calls between distributed objects and exposing higher level functionality such as discovery services.

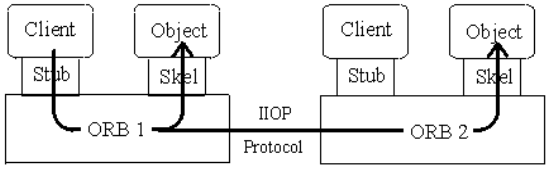

Figure 2 - ORB-to-ORB Communication (Object Management Group, Inc., 2007)

Henning [4] discusses the development of CORBA from a historical perspective noting some of the areas in which it fell short of expectations or failed to deliver on functionality. These include the high complexity of API's, a lack of support for key features such as security and fundamental design flaws in CORBA's interoperability protocol that made it unable to deliver high levels of performance.

Recent efforts have focused on the development of a real-time version of CORBA, which, as [5] details incorporates many features necessary for ensuring predictable performance including priority based scheduling and advanced resource management. As with other approaches this relies on the use of supporting real-time technologies such as predictable transport protocols and real-time operating systems.

4.2 Service Oriented Architectures

Service Oriented Architectures (SOA), are a model for distributing functionality amongst systems and components to facilitate loose coupling and late binding, therefore making a system with a greater potential for agility [6].

The basic model for SOA service fulfilment consists of three main components; the consumer, the service broker and the service provider, which work together in a publish/subscribe environment to fulfil a service requirement.

Services represent logical functional abstractions that promote reusability through a simple, well defined interface. For a service to be accessed in an ad-hoc manner the interface with which it communicates with external entities should be defined in a commonly accepted and widely known manner. To support this each service holds a service policy document that describes the functionality that it is capable of providing and the manner in which it may be accessed (for example the result of an operation could be given as an integer or a floating point number, etc.).

At a basic level a service broker can be described as a module capable of handling the necessary level of traffic for service announcements or requests. Additionally the capability is provided to store the service policies from announcing services in a service registry that can later be queried to find matches for requests (i.e. discovery of services)

Figure 3, as shown by [7], shows the basic SOA model, where the annotated numbers correspond to the following stages:

1. A service announces itself to the service broker, transferring a copy of its service policy document for storage in a service registry.
2. A consumer requests the fulfilment of a service from the service broker.
3. Wherever possible the service broker matches this request to the details of a service held within its service registry and replies with the location and interface details of this service.
4. The consumer contacts the service directly to negotiate service fulfilment.

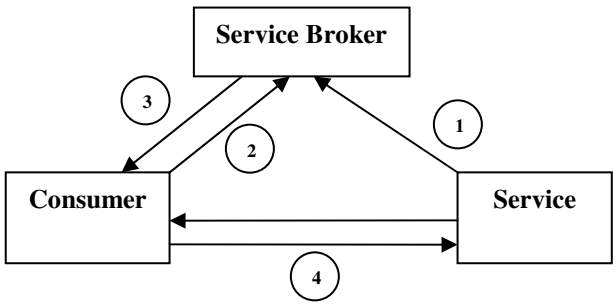

Figure 3 - SOA Model

Dependable, Real Time SOA: While O'Brien et al. [8] suggests that the loose coupling and unknown network structures inherent in SOA do not lend themselves well to dependable applications, there has been some work into adapting SOA for environments requiring real-time performance. RTSOA (Real-Time Service Oriented Architecture), as proposed by Tsai et al. [9] at Arizona State University, addresses the issues of real-time performance guarantees not only through the introduction of QoS constraints but from a wider perspective of the SOA environment. The main components identified by Tsai et al. as being key to the RTSOA framework are as follows:

- Real-time Communication
- Service Modelling for Real-Time Properties
- Repositories for Real-Time Composition
- Dynamic Service Composition
- Data collection & Policy Enforcement
- Real-time Service Execution Environment
- Mechanisms for Real-Time Guarantees.

Many of these key areas identified contain issues likely to have already been addressed in related research into real-time applications and communication.

4.3 Agent Based Architectures

Agent based architectures are a well established method of producing flexible, modular systems involving a degree of autonomy. An introduction to this field is given by Wooldridge [10], in which the basic premise of an agent based system is discussed. At a basic level an agent based architecture consists of a set of agents; components (either software objects or larger computer systems) with the ability to perform a unique function and the capability to manage their own actions through a small amount of Artificial Intelligence (AI). It is through the combined work effort of these agents that the systems goals are reached.

The supply of functionality by an agent is negotiated by the consumer and supplier to ensure that the final deliverable matches the consumer's requirements. This loose coupling and late binding provided by this negotiation step allows for easy upgrade or replacement of agents without creating disruption to the overall function of the system.

Agents are of their most use as an architectural choice when they are capable of interactions, sharing data or functionality. Allowing agents to communicate through broadcast messages may be the simplest solution, however, it is clearly not scalable and therefore an alternative approach must be employed. Multi agent systems, as discussed by van der Hoek & Wooldridge [11], typically make use of one of two strategies to solve this; using either an agent matchmaker or facilitator. An agent matchmaker identifies an agent capable of fulfilling the necessary functionality and passes details of this back to the consumer who then contacts the agent directly (in a similar manner to the SOA model). An agent facilitator matches a consumer to an appropriate agent and then acts as a router for the communication between the two parties.

These features can be seen to be very similar to those previously mentioned under the discussion of SOA, however, as Wooldridge [10] discusses, agents are unique to other modular architectures for several key reasons. They:

- Follow the Belief, Desire, Intention (BDI) model (as shown by Rao & Georgeff [12])
- Are aware of their environment.
- Are autonomous.
- Are goal directed.

Through the combination of these properties agents can be seen as a way in which to create a more autonomous and active distributed system in comparison to other architectures.

Dependable, Real-Time Agents: Many approaches to real-time agent based systems, such as Urbano [13] or DiPippo et al. [14] have focused on the use of agents themselves and how their properties can be exploited to meet deadlines. This can include for example, using faster executing but less accurate methods of determining a result with a lower accuracy or co-ordinating their behaviour in a manner that takes into account the higher priorities of certain tasks.

Urbano suggests that the AI methods employed by agents are well suited to adapting system characteristics to support real-time properties in dynamic environments. The example given is that of a network of cars with autonomous cruise control. When an emergency vehicle wishes to pass quickly through traffic (i.e. a high priority data packet) then the vehicles are capable of co-ordinating their movements in a manner that allows this.

While the use of agents in the previously described manners will certainly aid real-time systems the wider view of the system is perhaps of most importance. This is noted by DiPippo et al. [15] who highlight the importance of choosing an appropriate communication model and underlying framework.

4.4 Data Centric Publish Subscribe

The Data Distribution Service (DDS), as described by Pardo-Castellote [16], is an OMG standard for a real-time data-centric publish/subscribe system architecture. DDS shares certain properties with other publish/subscribe architectures (including SOA) such as the modularised design, loose coupling of participants and open interface, however, where DDS differs is that the focus is placed on the sharing of data between participants. (As opposed to invoking functionality)

DDS follows the publish/subscribe scenario closely. A client application places a subscription to a topic of

information (for example temperature readings or GPS coordinates), which is then matched to a publisher capable of dispersing data relevant to that topic. The overall DDS infrastructure is shown in Figure 4.

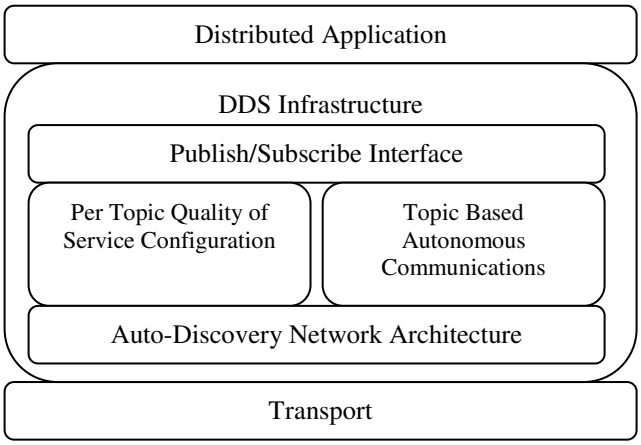

Figure 4 - DDS Infrastructure - Schlesselman et al. [17]

The DDS standard describes two levels of interfaces; DCPS (Data-Centric Publish-Subscribe) and DLRL (Data Local Reconstruction Layer). The DLRL is an optional higher level interface and allows for the integration of DDS into the application layer. DCPS (Data-Centric Publish-Subscribe) is a lower level interface and is typically composed of the elements found in Figure 5.

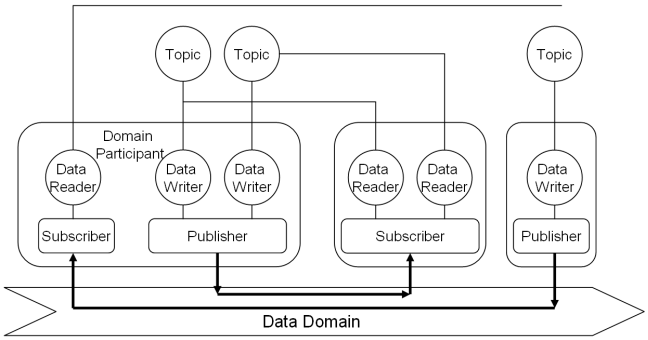

Figure 5 - DDS Entities - Schlesselman et al. [17]

Each node within the system maintains a record of the available publishers and the subscriber information relevant to them. Data is separated into domains in order to minimise the amount of data held by each node within the system and increase scalability. A domain participant is a physical (or logical) entry point to the network (or "data domain") and can contain both data readers and writers. A data writer is responsible for publishing instances of topic data. In order to distinguish between data originating from different publishers and to ensure that each value is treated separately to those previously received, each data entry is assigned a unique value or "key". Data readers declare their interest in a topic and the associated Quality of Service (QoS) properties that they require. The data writer then matches this request to the stored record of QoS characteristics available to offer.

The Real-Time Publish/Subscribe (RTPS) protocol is typically used in conjunction with DDS to provide a method of passing on the Quality of Service (QoS) requirements and ensuring that errors in transmission are detectable (given that transmission typically takes place over the unreliable UDP due to the importance of timeliness).

QoS

A key feature of DDS, as previously mentioned, is the support for QoS characteristics. Through the compliance with these QoS characteristics the necessary levels of performance are assured. The support for QoS characteristics greatly increases its suitability for those systems requiring performance guarantees. While this support allows for the specification and compliance with such performance requirements it does not strictly specify mechanisms for facilitating this and therefore these are dependent on the implementation.

4.5 Summary

A key requirement of the previously discussed approaches is real-time performance, which is vital where safety critical or mission critical systems are concerned. Table 1 shows a brief summary and comparison of the discussed architectural techniques with based on adaptive features and the maturity of support for dependable systems.

The four distributed architectures discussed here have for a large part show a lack of provision for dependable applications. With exception to DDS the architectures have placed little emphasis on the assurance of Quality of Service (QoS) characteristics (used to define an applications performance needs).

While the support for QoS parameters within DDS shows a progression towards dependable support, there is still a lack of focus for many key supporting technologies, including the role of the networks within such systems. It is assumed that these areas already contain the necessary means of assuring the required levels of service, a view that may be slightly short sighted

The maturity of the discussed architectural approaches with regards to their ability to facilitate highly dependable applications can be judged based on their actual use within industry today. In this respect only the data centric publish/subscribe approach DDS and real-time CORBA can be said to have reached such a level. DDS follows the current trend towards the use of a publish/subscribe environment to facilitate loose coupling and late binding within adaptive systems. DDS has also already proven itself to be capable of facilitating real-time communication between applications through its use by the US Department of Defense.

Table 1 – *Comparison of Architectural Approaches*

	Features of Architecture which contribute to adaptability	Maturity of Real Time Support
CORBA	- Runtime activation and deactivation of objects - Run time discovery of objects - Run time inter object network creation - Reflection of object interfaces	Real-time CORBA standard produced by OMG which includes predictable memory management and support for fixed priority scheduling. Implementations available.
SOA	- Connectionless communication model - Run time discovery of services. - Run time connection to services - Reflection of service interfaces	Some research conducted but no working real time standard or implementation produced
Agents	- AI methods employed to provide dynamic behaviour. - Agents are reactive to their environment	Some research into various approaches to optimise agent behaviour for real-time systems but no approach considering the wider architectural issues. Agent based implementations often rely on other software infrastructures for communication.
DDS	- Connectionless communication model - Run time discovery of publishers - Run time connection to publishers - Automated selection of 'best performing' publishers - Temporal decoupling between publishers and subscribers allowing matching of QoS deadlines - Runtime policing of QoS contracts	Real Time DDS standard produced by OMG. Includes aspects for QoS management related to real time performance. Implementations available. Is a mandated standard for publish-subscribe messaging by the U.S. Department of Defense (DoD) Information Technology Standards Registry (DISR).

5 Conclusion

This paper has presented IMS as a current architectural approach for building highly dependable systems. Features including inter-process communication networks, executing processes and real time schedules were identified as static. Then various challenges were discussed that highlighted the need for increasingly adaptive features within conventional approaches such as IMS. Next, a number of current open architectural approaches were discussed with regards to their adaptability features and suitability to highly dependable applications. Two approaches stood out as having the potential for use within highly dependable systems, Real Time CORBA and DDS.

Current work is assessing DDS for use within highly dependable systems. This involves understanding the tradeoffs between adaptability provided by the DDS standard and the predictability required by dependability requirements. Based on these understandings, recommendations will be made to include adaptive features in highly dependable architectures like IMS. Candidate adaptive features include

- Support of mixed hard and soft real time requirements within hybrid scheduling frameworks
- Integration of QoS parameters into IMS blueprints and QoS negotiation methods
- Runtime creation of inter-process networks
- Adaptive bandwidth management
- Runtime blueprint configuration generation

6 References

[1] Moir, I., Seabridge, A., & Jukes, M. *Military Avionics Systems*, ISBN: 978-0-470-01632-9

[2] Ministry of Defence Standard 00-74 Issue 2, *ASAAC Standards Part 1: Issues for Software*, 2008

[3] Object Management Group, Inc. (2007, September 11). *CORBA Basics*. Retrieved May 19, 2008, from Object Management Group: http://www.omg.org/gettingstarted/corbafaq.htm

[4] Henning, M. (2006, June). *The Rise and Fall of CORBA*. Retrieved August 2008, 7, from ACM Queue: http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=396

[5] Objective Interface Systems, Inc. (2008). *What is Real-time CORBA?* Retrieved August 8, 2008, from OIS: http://www.ois.com/Products/What-is-Real-time-CORBA.html

[6] Sim, Y. W., Wang, C., Gilbert, L., & Wills, B. (2005). An Overview of Service-Oriented Architecture. *University of Southampton* , 1-8.

[7] Gehlot, V., Way, T., Beck, R., & DePasquale, P. (2006). Model Driven Development of a Service Oriented Architecture (SOA) Using Colored Petri Nets. *First Workshop on Quality in Modeling, ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems*, (pp. 63 – 77).

[8] O'Brien, L., Bass, L., & Merson, P. (2005). Quality Attributes and Service-Oriented Architectures, Software Architecture Technology Initiative.

[9] Tsai, W. T., Lee, Y., Cao, Z., Chen, Y., & Xiao, B. (2006). RTSOA: Real-Time Service-Oriented Architecture. *Proceedings of the 2nd IEEE International Symposium on Service-Oriented System Engineering*.

[10] Wooldridge, M. (2002). *An Introduction to MultiAgent Systems.* Retrieved August 8, 2008, from http://www.csc.liv.ac.uk/~mjw/pubs/imas/distrib/powerpoint-slides/lecture10.ppt

[11] van der Hoek, W., & Wooldridge, M. (2007). Mulit-Agent Systems. In F. L. van Harmelen, *Handbook of knowledge Representation, 1*. Elsevier.

[12] Rao, A. S., & Georgeff, M. P. (1991). Modelling Rational Agents within a BDI-Architecture. *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning.*

[13] Urbano, P. (2002). Agent Based Approach to Distributed Real-Time Systems Development. *Proceedings of the 3rd International Symposium on Multi-Agent Systems, Large Complex Systems and E-Business*, (pp. 719 – 725).

[14] DiPippo, L. C., Fay-Wolfe, V., Nair, L., Hodys, E., & Uvarov, O. (2001). A Real-Time Multi-Agent System Architecture for E-Commerce Applications. *Proceedings of the 5th International Symposium on Autonomous Decentralized Systems*, (pp. 357 – 364).

[15] DiPippo, L. C., Hodys, E., & Thuraisingham, B. (1999). Towards a real-time agent architecture-a whitepaper. *Proceedings of the Fifth International Workshop on Object-Oriented Real-Time Dependable Systems*, (pp. 59-64).

[16] Pardo-Castellote, G. (2003). OMG Data Distribution Service: Architectural Overview. *23rd International Conference on Distributed Computing Systems Workshops Proceedings*, (pp. 200-206).

[17] Schlesselman, J. M., Pardo-Castellote, G., & Farabaugh, B. (2004). OMG Data-Distribution Service (DDS): Architectural Update. *IEEE Military Communications Conference*, (pp. 961-967).