Towards Safety Critical Middleware for Avionics Applications

D.A. Haverkamp, R.J. Richards, Ph.D.,
Rockwell Collins Advanced Technology Center, Advanced Computing Systems Department,
Cedar Rapids, IA {dahaverk, rjricha1}@rockwellcollins.com

Abstract

Two factors influencing the design and development of avionics software are 1) the cost of verification, validation and certification 2) migration of avionics functionality from hardware to software, to decrease the weight and power consumption of the avionics. These two factors are inherently at odds. Lowering the development costs of engineering software for safety critical systems, while providing the abstractions necessary to build systems of ever increasing complexity, is key to achieving these two goals. Middleware seems to be the ideal vehicle to reach these goals.

Middleware is used to isolate the core application from the underlying distributed system and is constructed using object-oriented techniques. This has the benefit of increasing software reuse and minimizing the code that is verified to various safety criticality levels when the underlying system microprocessor and network are changed. The middleware that meets the criteria placed on safety critical software is faced with many challenges.

1. Introduction

The majority of system functionality in modern avionics systems is provided by software. This software functionality comes at the cost of increasing lines of code. Not only does the increase of software functionality place higher demands on the computing resources, but avionics software development is also painstaking and expensive. Traditionally, avionics system designers have made extensive use of proprietary hardware. The hardware used for avionics are not like a typical "off the shelf" personal computer. Some of the typical differences are, memory is error corrected (ECC) and microprocessor clock derated to increase reliability with built in self-test (BIST). This has hampered the ability to exploit commodity hardware, which has been improving at rates predicted by (and sometimes exceeding) Moore's Law. Avionics systems of the past have been very tightly coupled to the hardware capabilities, resulting in fragile software architectures that are sensitive to underlying hardware changes. Middleware allows us to focus more on the integration of system-level software components that are decoupled from the underlying RTOS and hardware.

A simple-minded use of middleware in a system might just create wrappers around legacy applications. This would not create the most effective usage of CORBA. A top down approach is necessary for efficiently creating an application component architecture (see Figure 1). This abstraction allows us to concentrate our efforts on specific component layers.

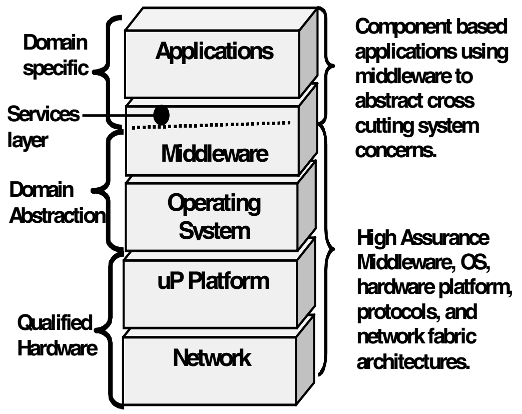
Domain specific
Services layer
Domain Abstraction
Qualified Hardware
Applications
Middleware
Operating System
uP Platform
Network
Component based applications using middleware to abstract cross cutting system concerns.
High Assurance Middleware, OS, hardware platform, protocols, and network fabric architectures.

Figure 1 High Assurance Middleware Systems

"Middleware" refers to a layer of interfaces and services that resides "between" the application and the operating system to facilitate the development, deployment and management of distributed computing systems. Middleware has emerged as an important architectural framework component in modern distributed real time systems because of its ability to decouple applications from many of the underlying hardware and real time operating system (RTOS) concerns. Some of this is being driven by the hardware changes and improvements predicted by Moore's Law. Use of middleware in a system allows a higher-level, platform-independent programming model (object-oriented or component-based) and hides some of the distributed programming complexities from the application (see Figure 1). Some examples of middleware platforms are CORBA, DCOM, .NET, and Java-based technologies (Enterprise JavaBeans, Jini, and RMI).

Businesses are deploying middleware into embedded and real-time domains to tackle heterogeneity problems and to decouple applications from underlying systems. Middleware technologies are being applied in a wider range of systems, including software radios, avionics mission computers, and many other types of embedded real-time systems [1].

2. CORBA

The Object Management Group (OMG) has been fostering CORBA (Common Object Request Broker Architecture) as a standard Middleware for common use, supported by the Unified Modeling Language (UML) as a modeling and design language (see Figure 2). CORBA and UML are open specifications developed by OMG members. These specifications provide support for many different programming languages. This allows CORBA-based software to execute on heterogeneous distributed systems.

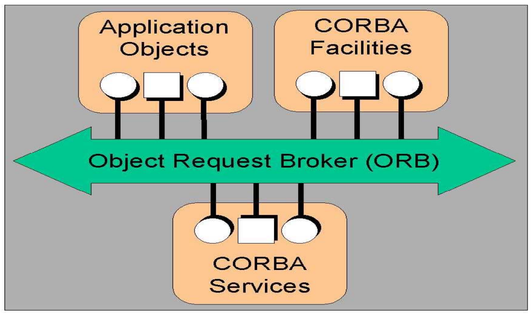

Figure 2 CORBA Middleware

Real-Time CORBA is a profile of CORBA tailored to allow Object Request Brokers (ORBs) to be components in Real-Time systems. An ORB implementation compliant with the Real-Time CORBA profile may offer a Real-Time Scheduling Service [21].

The CORBA standards are subject to interpretation and some, in an effort to adapt to an environment or application, implement a subset of the standard. This adaptation can have an impact on the overall performance of an ORB. In Figure 3 we see a full-featured general purpose ORB (ACE TAO implemented in C++) compared to a minimalist, specialized implementation, ORBit [15]. The ACE TAO is considered a real-time ORB implementation, however the full implementation has a performance cost. The smaller ORBit ORB has been developed largely as a special purpose ORB to support a windowing desktop GUI. ORBit performed better than the ACE TAO. ORBit does not support the real-time CORBA policies, priorities and the end-to-end predictability of activities in the distributed system with support for resource management. It should be noted that although ORBit performed better on this particular test, we had difficulty using it to send and receive periodic data, unless the data rates were slowed down considerably. This serves to illustrate that an ORBs performance and footprint characteristics can be improved greatly by adapting the ORB to its environment and removing extraneous functionality. However, the adaptation must be performed with care to insure that the resulting ORB runs well in all necessary situations.

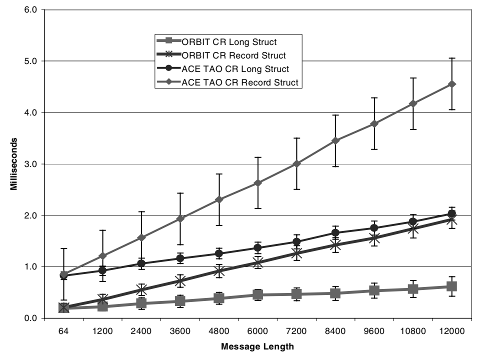

Figure 3 Real-Time vs. Specialized ORB (R-T DII COE Test)

3. Challenges for Avionics Middleware

The current OMG Real-Time CORBA specifications do not consider all of the forces driving application development for commercial avionics systems. The primary consideration that is not met is the development of software following RTCA DO-178B guidelines (e.g. safety criticality). Another application development constraint that is not met is the need for small size (memory footprint), due to hardware size, weight, and power requirements for avionics.

There is a need to research specific CORBA areas for use in safety critical avionics environments. The top issues needed to be researched and explored for using CORBA in avionics are (not in any order):

- System Partitioning
    - Time and Space Partitioning
    - Federated Architecture
- Fault Tolerance

        - Deterministic replica consistency
        - Fragmented Objects
        - Interoperability limitations
        - Analytic Redundancy
        - Leadership rules
        - Data integrity
- Quality of Service
- Reflective Middleware
  Aspects or Cross-Cutting concerns[3]
- Generative programming[5][10]
- Avionics Domain Services
- High Assurance

Improved software engineering tools are also necessary that support the research areas outlined.

3.1. System Partitioning

RTCA DO-255, an avionics industry standard for computing resources[20], dictates a time and space partitioned real-time operating system (see Figure 4). This requires the Avionics Computing Resource (ACR) to have the necessary hardware and software mechanisms to ensure that time, space, and I/O allocations are static. Static means that time, space, and I/O allocations are established at initialization time of a program or process, and are not altered during runtime. This ensures that safety critical applications are provided the computing resources necessary to fulfill their function. Chief among these resources are processor time and memory.

Conceptually, a real time operating system (RTOS) that supports multiple virtual machines capable of running a real time POSIX application or an operating system such as Linux is what RTCA DO-255 allows.

There are not currently any commercial ORBs running on an operating system supporting time and space partitioning. Running an ORB in this type of environment may have unforeseen consequences, mainly because legacy avionics applications have been tightly coupled to the hardware and system data flows. As discussed later, middleware offers new opportunities for developers to abstract the underlying system network and hardware.

The core ACR software performs access mediation between partitions, independent of other provided services. Importantly, access mediation between partitions must be complete, tamper-proof, and assured. Security minded people recognize this as being related to basic security requirements. An ORB could be the mechanism of cross partition communications, providing a consistent communication mechanism. This may mean ORB functionality should be provided by the core ACR system (i.e. OS kernel space).

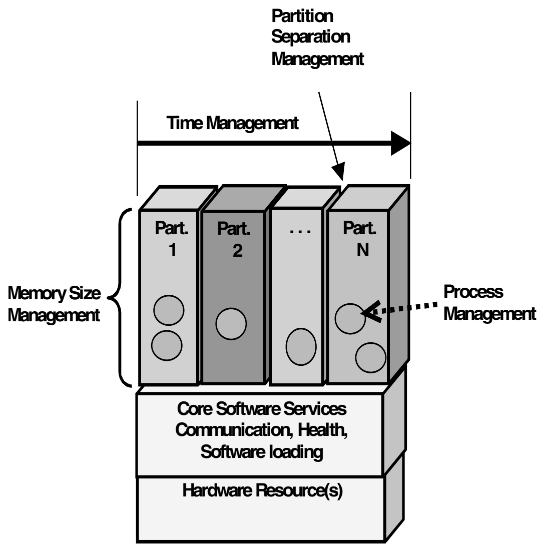

Figure 4 Partitioned Computing Platform

Avionics systems architectures are evolving from the federated architectures traditionally used. This type of system architecture and its impact on CORBA Fault Tolerant and Security Domains needs to be explored. Avionics are moving toward distributed architectures, providing a need for a middleware layer, such as CORBA, to provide the necessary abstractions to facilitate highly reliable distributed computing. How the system is partitioned has a flow down effect on Software Architecture, Fault Tolerance, Quality of Service, Security, etc.

3. 2. Fault Tolerance

Use of CORBA Fault Tolerance (F-T) [13] in avionics may require several different service levels depending on where and how an object is used in the system. For instance, the simplest type of Fault Tolerance is to have multiple instances of an object running in the system, with a fault detection mechanism that determines which instance is used. Another scenario is to have a Factory Object that replicates the object by creating instances on another computing resource in the system according to an *a priori* scheme. A third possible way to support F-T is to have the ORB support Object Fragmentation, in which parts of an object's functionality can be replicated in the system to support both F-T and performance concerns. This type of scheme is not in the current CORBA F-T specification, but is being researched.

Areas where the CORBA F-T specification can be improved are:

- Interoperability limitations dictate that F-T components be well specified for interoperability
- Determinism of varying degrees may need to be supported for replica consistency (Policy based Fault Tolerance.)
- Analytical Redundancy [4] to allow for temporal differences in system data. (i.e. Using knowledge of temporal data changes to predict correctness.)
- No Real Time (R-T) provisions in the current F-T spec (the OMG R-T SIG plans to work on this)
- Usage of known temporal data produced at expected intervals for Heartbeat/Liveness checks.
- Leadership rules are needed to support federated architectures.

3. 3. Quality of Service

Basic system Quality of Service (QoS) means providing consistent, predictable data delivery service. For avionics, this includes satisfying safety, availability, and integrity requirements. This provides network congestion control that is transparent to the endpoint applications. A useful (and relevant) analogy might be event delivery such that the endpoint applications can not differentiate whether a sensor is delivering the data directly or through an event service. The OMG is currently working on adoption of a specification that allows for a user framework permitting replacement of TCP/IP with other protocols as necessary for embedded R-T systems. We are evaluating embedding CORBA transport mechanisms into the network with the potential of improving QoS and overall performance.

The following characteristics are the most important system QoS considerations:

- minimizing end-to-end delivery delay (bounded latency)
- minimizing end-to-end delay variations (bounded jitter)
- consistent end-to-end data throughput

For safety critical systems, this means having a predictable Real-Time Operating System (RTOS), Communications Protocol Stack, and ORB data delivery service. Current R-T ORB implementations have some QoS considerations. However, the unpredictable "best effort" network protocol used in most operating systems does not ensure the predictable end-to-end requirements for safety critical systems.

Feature additions have a performance impact. Typical avionics applications only deliver functionality that is needed. Since not every application needs all of the real-time ORB functionality, a promising way to customize an ORB (and application) is through Aspect-Oriented Programming[3]. Through the development of feature sets the application developer can deliver a core set of features and the aspects that add features necessary to support the overall goals of the application.

Currently the major need for research in this area is on predictable end-to-end QoS policies, and consistent infrastructure (i.e. Predictable ORB, RTOS, and Predictable Network Protocols). Much of the current research has been on QoS applied at specific functional layers without consideration for meeting the stringent end-to-end predictability requirements of safety critical systems. Another issue for Safety Critical systems is how to continue applying the end-to-end QoS policies predictably when the system is reconfigured to compensate for component failure. There is a need to provide the application with a consistent set of end-to-end Real-Time adaptations to the services and the ORB functionality (i.e. naming, event/notification, etc.).

3. 4. Reflective Middleware

Reflective Middleware is a term that describes the capability of a system to reason about and act upon itself. For safety critical systems this can be used in conjunction with QoS, Integrity, and Fault-Tolerance policies. Current research has focused on meta-interfaces to manage internal operation and structure of the middleware for runtime changes.

A more generalized approach would be to have a rules approach allowing causal manipulation of the appropriate meta-interfaces. Safety critical systems could use a causal-net [14] applying pre-determined behaviors to direct adaptations in response to system changes. With causal adaptation of a system, changes are immediately reflected in state and behavior. This allows for safe systems that can react dynamically with predictable behavior. Some reflective properties may also be applied statically at compile or system generation time using generative programming.

3. 5. Generative Programming

Generative Programming [5][10] is the construction of a program by specification. This is done with a specification language that is processed by a "generator" to produce the implementation source code corresponding to the specification. This whole process should sound very familiar to those knowledgeable with the OMG Interface Definition Language (IDL). The IDL (an interface specification) is processed to produce the routines called by the client, known as "stubs" and the definitions (without implementation) of the server routines, called "skels".

The important aspect of this generation of interface code is the code necessary to handle the distributed nature of the program is generated from the IDL. In the future, it may be possible, through further enhancement of specification languages, to assist a developer in generating an increasing amount of the application by a generative process. The use of "Aspects" further customizes the generated code with only the application specific features (adaptations) used[7]. Moreover, IDL compilers should be enhanced to generate coverage tests which are a necessary part of the DO-178B software process.

3. 6. Avionics Domain Services

Definition of Standard Commercial Avionics domain services and their interfaces allows for a more open component oriented system. These standardized interfaces in turn lead to having a reusable system framework that allows for usage of well-defined architectural, component, communication, and synchronization patterns. This helps to increase the software reliability and decrease component integration problems. Developers in turn can concentrate more on the application requirements and ultimately improve software productivity.

Focusing on system services and standardized interfaces allows a System Engineer to have a more abstract view of the system. Rather than worrying about low-level data formats, the system engineer can design the system from a high level model based viewpoint. This in turn enables application providers to work with standard interfaces appropriate to an application. Either the platform or application provider can act as the system integrator as appropriate for the software or hardware being developed (see Figure 5).

We have developed a prototype data distribution service for avionics. The goal of the work was to move towards an 'Aircraft Abstraction Layer' (AAL). With such an abstraction an avionics application can request the requisite data and receive the data with high assurance. High assurance in this context can be taken to mean that the data's value correctly portrays the physical phenomenon that it represents and the data is not so old that it is no longer valid. The process of creating the high assurance for the data is moved from the application to the middleware and applied uniformly through out the system. Furthermore, the property of location transparency, provided by CORBA, removed any knowledge of the underlying transport mechanism from the application, rendering them more portable and reusable.

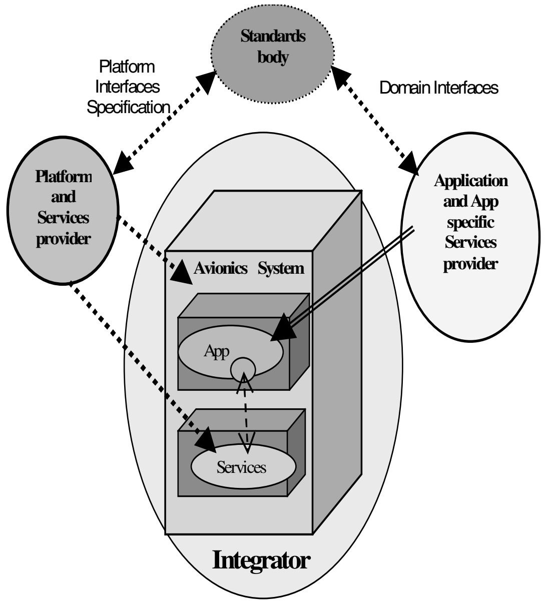
Standards body
Platform Interfaces Specification
Domain Interfaces
Platform and Services provider
Application and App specific Services provider
Avionics System
App
Services
Integrator

Figure 5 System Integration of Software and Hardware

Our prototype avionics data distribution service also assists the application with data interpretation. This includes the automatic data type and units conversions as well as automatically performing operations such as low-pass filtering on incoming data.

3. 7. High Assurance

The Boeing Company has paved the way by using CORBA in the mission computer of a military weapons platform[1]. CORBA is not currently used in any commercial safety critical avionics. However, CORBA is finding its way into less critical parts of commercial avionics systems.

For CORBA to move into Level-A critical systems, ORB developers must follow the RTCA DO-

178B Software Considerations in Airborne Systems and Equipment Certification guidelines. Most current Open Source ORBs have been developed in C++ without any consideration for the assurance processes necessary to certify the ORB. Recently, the Aerospace Vehicle System Institute (AVSI) guidelines [2] for certification of Object-Oriented software have become available. However, these guidelines have not yet been validated or applied to the construction of an ORB and associated Services.

CORBA security is essential for access mediation as part of the services provided by an ACR. Access mediation with respect to CORBA must be complete, tamper-proof, and assured as follows:

Complete. Software running in any partition must not be able to bypass the ORB to access system resources outside of its partition.

Tamper-proof. Applications running in any partition must not be able to tamper with the ORB or CORBA Services in any manner that subverts CORBA when accessing system resources.

Assured. CORBA Security routines must be evaluatable; that is, they must be amenable to investigation and analysis for correctness. Note that this means evaluatable with respect to both DO-178B [19] and the Common Criteria [6].

An important consideration in using CORBA security with a real-time system is how much of an impact Security will have on the systems laxity.

4. Conclusions

The Middleware technologies identified should enable a consistent layered development of large, complex avionics systems. Middleware abstracts the operating system and hardware platform providing widely used domain services. Avionics application developers can then use consistent software architectures. CORBA in avionics frees software developers to focus on application domain requirements. This allows developers to focus on better requirements and integration of the domain components rather than lower level details (see Figure 6).

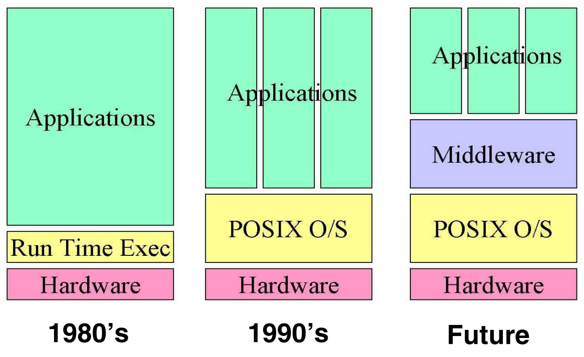

Figure 6 Avionics Application Environment

The recent Joint Strike Fighter award highlights the current emphasis on providing a modular component platform. CORBA is the leading technology for creating an open modular and reusable software base for complex software systems. The creation of "reusable software radio waveforms" for the Joint Tactical Radio System (JTRS) [11][17], is an example system with hard real time constraints using CORBA. The JTRS system providers are also leading the way toward creating lighter weight CORBA services[13]. Rockwell Collins is an active participant in the Software Radio and Real-Time working groups at the OMG.

It is apparent that CORBA technologies have matured considerably since the early 1990's. However, current middleware has some technical challenges to overcome before it will be acceptable for use in safety critical systems. Some of the current CORBA limitations are due to the black-box implementations of most current ORBs. Specifically, middleware must flexibly allow for small systems and scale to larger platforms offering various capabilities. A CORBA-based system must be flexible enough for developers to create high assurance systems with a consistent approach for cross cutting concerns.

5. Acknowledgements

Many thanks to the many reviewers of this paper and especially the considerable knowledge base in the research papers from the Distributed Object Group at Washington University St. Louis.

6. References

[1] ACE TAO research at **http://www.cs.wustl.edu/~schmidt/doc-group.html**

[2] Aerospace Vehicle System Institute "Guide to the Certification of Systems with Embedded Object-Oriented Software", AVSI, version 1.1, May 31, 2001

[3] Aspect Oriented Programming information at http://www.aspectj.org

[4] Carnegie Mellon Software Engineering Institute, Simplex Architecture. http://www.sei.cmu.edu/simplex

[5] Cleaveland, C. "Program Generators with XML and Java", Prentice Hall 2001. http://craigc.com/pg/index.html

[6] Common Criteria http://www.commoncriteria.org/

[7] FACET Framework for Aspect Composition for an EvenT channel http://deuce.doc.wustl.edu/doc/RandD/PCES/facet/

[8] Wolfe V, Dipippo L, Ginis R, et al "Expressing and Enforcing Timing Constraints in a Dynamic Real-Time CORBA System"

[9] Geihs K. "Middleware Challenges Ahead" IEEE Computer June 2001

[10] Generative programming information at http://www.generative-programming.org

[11] Joint Tactical Radio System. http://www.jtrs.saalt.army.mil/

[12] Leveson N. "Safeware: system safety and computers", ACM Press, New York, NY, 1995

[13] OMG Specifications information at http://www.omg.org

[14] Provan G., Chen Y. "Model-Based Fault-Tolerant Control Reconfiguration For General Network Topologies" IEEE MICRO Sept-Oct 2001.

[15] Robillard Lt. Col. Lucie M.J., Callison Dr. H. R., Maurer J. "Extending the DII COE for Real-Time" http://www.stsc.hill.af.mil/crosstalk/1999/sep/robillard.asp

[16] Rockwell Collins Advanced Technology Center response to the OMG Safety Critical RFI http://cgi.omg.org/docs/mars/02-03-01.pdf

[17] Software Defined Radio http://www.sdrforum.org/

[18] Quality of Service information at http://www.qosforum.org

[19] RTCA/Do-178B "Software Considerations in Airborne Systems and Equipment Certification"

[20] RTCA/Do-255 "Requirements Specification for Avionics Computer Resource"

[21] Schmidt D, Kuhns F. "An Overview of the Real-time CORBA Specification"

[22] Schantz R, Schmidt D. "Distributed Object Computing Middleware – Evolving the Common Structure for Network-centric and Distributed Applications