Radar Open System Architecture Provides Net Centricity

John A. Nelson
MIT Lincoln Laboratory

ABSTRACT

The second generation of the Radar Open Systems Architecture has been developed and put into practice. This approach consists of a layered architecture that isolates applications from underlying hardware and software elements such as operating systems, middlewares, communication fabrics, and computer platforms. The framework also consists of a set of component libraries that are being populated as the framework is applied in an expanding series of application domains. Easy swap ability for the library components or newly-developed components along with a high degree of hardware independence allows systems built using this infrastructure to be easily maintained and upgraded.

INTRODUCTION: OPEN SYSTEM CHARACTERISTICS AND QUALITIES

Radar sensor and similar device control systems have commonly been developed from very basic building blocks, using proprietary hardware and software architectures. This development model is usually expensive and requires long design and development lead times. Because each resulting device employs a unique architecture and supporting technology, it is difficult and expensive to upgrade and maintain the considerable assortment of radar systems [1].

Acquisition reform thrusts and the proliferation of open systems (OS) and commercial off-the-shelf (COTS) technologies have prepared the way for major changes and cost reductions in the development process of defense-acquisition programs. However, OS are about more than limiting development costs:

Author's Current Address:
J.A. Nelson, MIT Lincoln Laboratory, Lexington, Massachusetts, USA.

Based on a presentation at Radar 2010; review handled by J. Entzminger.

0885/8985/10/ $26.00 © 2010 IEEE

- *they speed up the development process,* and
- *provide easy access to the latest technological advances.*

Further, OS facilitate the use of common architectures, alternate vendors, and a more competitive procurement model. A standard open architecture applied to radar systems has been shown to streamline the development process for these systems and greatly improve future technology insertion opportunities [1].

An OS has several salient characteristics:

- *It is generally a complex system that is made more manageable by breaking it down into subsystems,* and
- *further into components.*

The smaller parts of the OS interact with each other in a predictable fashion that involves inter-component interfaces that are well-defined and published without reservation. This approach allows individual parts, i.e., subsystems or components, to be replaced without affecting the remainder of the system as long as replacement pieces conform to the published interoperability behavior and interfaces. The decomposition of the problem described herein has major benefits:

- *The sub-problems associated with the development of the parts become more manageable* as fewer engineers and developers need to work on any given part.
- *The parts are more easily tested, and multi-level testing* (unit, component, integration, validation) are easily carried out.

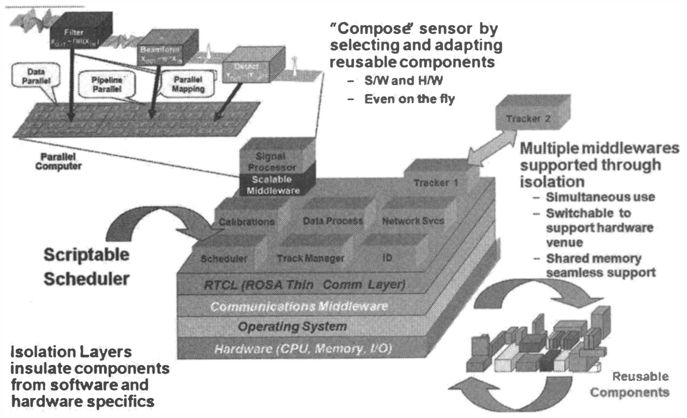
Individual parts may be replaced by other like-function parts that share the proper behavior and interfaces. This factor allows the integrating entity of an OS to be a different entity than those that may have developed the individual sub-systems or components, thus breaking down barriers to competition within a system development environment.

The openness of a system is determined largely by the level to which parts are described with respect to their interfaces. It is quite possible that an open system may contain some closed or proprietary parts as long as their function is well-known and understood and they obey the common interface definitions. An obvious example of this circumstance is the highly integrated electronic circuit – for a complex example; the CPU chip. The behavior of these chips is well understood publicly because the interface rules as well as the programming model are generally made public by the vendor. However, the details of the chip design under the hood are often held as a trade secret. This does not inhibit the use of the chip in an open system context in any way, and allows for commercial entities to function in a particular and common OS business model.

When developing an OS, it is desirable to ensure that the architecture can support the following important aspects. The architecture should be applicable to a wide variety of different device instantiations. These include, for example, ground-based radars and optical sensors, dish and phased array radars, airborne sensors including Synthetic Aperture Radar (SAR) used for ground surveillance, and other devices that use the notion of processing chains and open loop data collection or closed loop control approaches. Another important dimension for an open architecture to address is common support for diverse computing frameworks. There are several types of computing frameworks that are of interest in sensor and device development, including symmetric multi-processor computers, cluster computation, embedded device computation, and other specialized high performance computation including graphics processing units, etc. An open architecture should support these different computing venues with minimal (if any) changes to software components as they are shifted from one compute platform to another.

Net-centricity is another equally important aspect of the OS. Generally the system must be able to accept commands, requests for behavior modification, as well as provide data and results to the outside world. Not only must the system be connected to networks; in many cases, it should use the common technology being developed for the Web centric world to enhance its usability and configurability. The Net Centric Enterprise Services (NCES) [2, 3], being in part developed and provided by the Defense Information Systems Agency (DISA), can serve as a common basis for the net centric aspects of an open system.

The other aspect of the OS that is very important is the availability of a library of component functionality that can be reused across projects and programs. This library availability is a key aspect of the OS because it will promote the reuse of the components that have generic functionality. For example, an integration module that conducts pulse-by-pulse integration of raw radar data is a component with wide applicability.



Filter
Beamform
Detect
Data Parallel
Pipeline Parallel
Parallel Mapping
Parallel Computer
"Compose" sensor by selecting and adapting reusable components
– S/W and H/W
– Even on the fly
Tracker 2
Multiple middlewares supported through isolation
– Simultaneous use
– Switchable to support hardware venue
– Shared memory seamless support
Signal Processor
Scalable Middleware
Tracker 1
Calibrations
Data Process
Network Svcs
Scheduler
Track Manager
ID
Scriptable Scheduler
RTCL (ROSA Thin Comm Layer)
Communications Middleware
Operating System
Hardware (CPU, Memory, I/O)
Isolation Layers insulate components from software and hardware specifics
Reusable Components

Fig. 1. The ROSA II infrastructure consists of a layered construct that ioslates application modules and their code from the specifics of middlewares or other inter-process communication mechanisms, such as shared memory. Application components may be swapped in and out of the system easily, and libraries of domain-related components may be used to help construct systems

With a suitable library of components, system integration and testing is simplified as long as the components behave in a fashion consistent with the system engineering design. To allow this, it is important that the standards to which the components have been designed be robust. The component library can also serve as a baseline from which components can be taken and modified. This capability is enhanced if the components are based upon an object-oriented component model.

THE MIT LINCOLN LABORATORY RADAR OPEN SYSTEMS ARCHITECTURE

ROSA II is the second generation of the Radar Open Systems Architecture (ROSA) originally developed at MIT Lincoln Laboratory for modernization of test range radars [1]. This second generation architecture adds more flexibility, scalability, modularity, portability, and maintainability than was offered by the original version. ROSA II focuses on enhancing these features, as well as providing a robust infrastructure for instrumentation and test bed development, as well as that for tactical turnkey radar system development. A key enabler for this is abstraction, where interfaces among components are separated and defined.

The ROSA II system also adds the capability to easily and directly support phased array radars. The phased array is very important for future radar system development. For example, closed loop tracking and pulse scheduling components need to be more robust for use in a phased array system. Furthermore the signal processing requirements of a phased array system can be very substantial and may require dedicated signal processing hardware sub-systems. These requirements may be accommodated using specialized parallel computers without affecting the general purpose computers used to service other aspects of the system. An abstracted communications layer in the system called the ROSA Thin Client Layer (RTCL) allows connecting disparate sub-systems without the need for each sub-system to have knowledge of the internal workings of the others. It also allows the software components of the system to be location-transparent with respect to computing platform and network infrastructure. For example, a system may be based upon a symmetric multiprocessing platform, or a networked cluster of computers, or both. The ROSA II software components may be placed within these platform architectures without making changes to the components themselves.

Abstraction of specific details of hardware and operating systems allows ROSA to easily use different types of machines for control and signal processing. System designers are able to use inexpensive commodity computers, rugged space- and power-efficient computers, or the traditional high performance symmetric multi-processing computers as appropriate for their application.

Figure 1 depicts ROSA for componentized software and mixed software and hardware systems. The important aspects of this architecture that provide openness are described below.

The architecture consists of a layered structure that isolates (in the case of software) the application modules, or components, shown in the boxes in the figure, from the lower level details of the structure. These common components are written to a specific API that includes the common interface to the RTCL as well as common POSIX-compliant interface to the operating system. The RTCL isolates the components from the specific aspects of the middleware or middlewares in use. This approach allows the components to be used with any middleware supported by the RTCL, and indeed to be used with multiple middlewares at the same time, if required by the system engineering. This is done without any changes necessary to the application code of the components, just configuration changes. This feature is a boon that allows location transparency for the components.

The components are contained in a component library. The system is built using an appropriate collection of the components or new components that the system developer generates. Any system developer can maintain a selection of components, as necessary and appropriate. Well-engineered components are loosely-coupled, and rely upon well-defined inputs and outputs defined by Interface Control Documents (ICDs). The ROSA component approach provides support for a common component object model (provided in the form of an object-oriented base class) that supports common code for data input and output as well as component control and status logging functionalities. The common component model also supports a component state machine, and timing and time control functionality. These functionalities are provided in a component base class that all ROSA components inherit from and therefore obtain by default. Application code needing to use these functionalities can inherit and populate the relevant parts of the base class in the finished component code.

The time control functionality is used to support the system from the view of time budgeting. The system engineer may determine that a particular component has to respond with a result or action within a certain period of time to be viable. This type of strict timing control can be avoided in most components; however there typically are some time critical components in a system that have to respond reliably with respect to a time budget. This need is supported in the component base class and can be used in components with a time-criticality requirement. Structure is provided in the form of timing routines, including call-backs that can be executed as a pre-set time budget limitation is approached. The application programmer can provide the code to handle time-out exceptions as necessary. An example would be a component that refines estimates of target position based upon available time and information. When its time budget for a new target report runs out it will be able to provide the best estimate that is available.

The RTCL allows simultaneous or individual support of any number of middlewares. The requisite middlewares are supported in the RTCL in one place, not in any application code. With this architecture there is no reason to settle on a particular middleware or support more than one middleware at the application level. The application components simply are isolated from the details of the individual middlewares, and therefore, of any details of inter-component communication below the RTCL API layer.

An example of the above is the support within a system of two middlewares: for example, one can be optimized for transport between components that exist on a single symmetric multiprocessor (SMP) platform; and the other can be optimized for support across a network. In the reference model for ROSA, the former is a shared memory transport that allows transfer of data between components without data copying, which is important for ultimate throughput and low latency. The latter transport is a Data Distribution Service (DDS) publish-subscribe middleware. DDS is a standard of the Object Management Group (OMG) [4].

NET-CENTRIC FEATURES

ROSA directly supports net-centricity by including web servers within the base class of the standard component model. This support, along with orchestration components and data streaming components, allow system designers to easily build fully net-centric systems that support multi-mission requirements as well as interfaces necessary for sensor and data collection brokering, streaming data production, and traditional request response data sharing paradigms.

EXAMPLE USE CASES

ROSA has been used successfully in building a prototype mobile instrumentation radar [5, 6], modernizing unique signature radars at a test range [1], and unique radars at a space surveillance facility, among many others. A surveillance radar system has been developed with a wide latitude of configurable features. An optical system back-end has been developed as well. This shares some components with radar systems. ROSA has been used to develop a benchtop demonstration phased array radar system complete with multifunction scheduler capability and phased array pulse level simulator.

ROSA supports the open system model by decomposing a radar processing and control system into functional building blocks constructed using COTS hardware and modular software. This decomposition provides loosely-coupled operational sub-system components that, when tied together using well-defined interfaces, form a complete radar-processing and radar-control system. Building blocks can be easily added or modified to allow new technology insertion, with minimal impact on the other elements of the radar system.

More importantly, existing radar building blocks can be shared and used to create new radars or to modernize existing systems. This modular OS architecture has led to improvements in time-to-market, reduced cost, and increased commonality.

SUMMARY

The ROSA II architecture is a publish-subscribe approach to data distribution. Components that need published data listen in on named data topics for the data that they need. The components subscribe to input data, and publish output data, as well as status information on a status topic, and receive control information on a control topic. Components linked in these publish-subscribe chains usually (but not always, and not of necessity) communicate in a one-to-many approach.

The components interact with the communications layers through an isolation layer. The isolation layer allows for a common component API that is not dependent upon the particular middleware in use. This prevents the need for *"mandating"* a particular middleware or set of middlewares.

The reason for avoiding this is that middleware technology is always being improved and refined, and it would be unwise to lock into a particular solution. Also middlewares are often specialized for particular purposes, so that the isolation layers allow more than one specialized middleware to be implemented in the system as appropriate, without need for any changes in the application components.

With ROSA II, Lincoln Laboratory has the architecture, design, and reference implementations for the system, the middleware, and components suitable for use in radar systems, optical control systems, and other device control domains. A number of projects are making use of ROSA II going forward.

ACKNOWLEDGMENT

This author acknowledges the support of the MIT Lincoln Laboratory Technology Office in the conduct of this work.

This work was sponsored by the Department of the Air Force under Contract No. FA8721-05-C-0002. Opinions, interpretations, recommendations, and conclusions are that of the author and are not necessarily endorsed by the United States Government.

REFERENCES

[1] S. Rejto, (2000),
Radar Open System Architecture and Applications,
Proceedings of IEEE International Radar Conference, pp. 654-659.

[2] Net Centric Enterprise Services (NCES) Overview,
Defense Information Systems Agency,
<www.disa.mil/nces/about.html>
and references contained therein.

[3] Net Centric Enterprise Services (NCES) User Guide, March 2008,
Defense Information Systems Agency,
<www.disa.mil/mces/users_guide_03122008.pdf>.

[4] Object Management Group,
Data Distribution Service (DDS) Specifications,
<http://www.omg.org/technology/documents/dds_spec_catalog.htm>
and references contained therein.

[5] J.T. Mayhan, R.M. O'Donnell and D. Wilner,
COBRA Gemini Radar,
Proceedings of IEEE National Radar Conference,
pp. 380-385, (1996).

[6] W.W. Camp, J.T. Mayhan and R.M. O'Donnell,
Wideband Radar for Ballistic Missile Defense and Range Doppler Imaging of Satellites,
Lincoln Laboratory Journal, 12(2), pp. 267-280, (2000).