8.4 # SOFTWARE ARCHITECTURE OF THE NEXRAD OPEN SYSTEMS RADAR PRODUCT GENERATOR (ORPG)

Michael Jain[1], Zhongqi Jing[1,2], Allen Zahrai[1],
Arlis Dodson[1,2], Hoyt Burcham[1,2], Dave Priegnitz[1,2], and Steve Smith[3]

[1]NOAA/ERL/National Severe Storms Laboratory, Norman, OK
[2]Cooperative Institute for Mesoscale Meteorological Studies,
University of Oklahoma, Norman, OK
[3]NOAA/NWS/WSR-88D Operational Support Facility, Norman, OK

1.0 INTRODUCTION

The National Weather Service (NWS) is currently involved in an effort to evolve the existing NEXRAD WSR-88D system so that changing Tri-Agency operational requirements can continue to be met. This activity falls under the NWS project of NEXRAD Product Improvement (NPI) (Saffle, et al., 1997). The initial objective of the NPI project is to rehost the existing Radar Product Generator (RPG) functionality from the current, proprietary platform to one that is open systems compliant. The National Severe Storms Laboratory (NSSL), working with the WSR-88D Operational Support Facility (OSF), has been tasked with the responsibility of establishing the software architecture and performing the software development for this project.

A common tendency in the design of software systems of the past has been an approach that treats the entire system as a "tightly coupled", monolithic entity where functions are highly interrelated and dependent on one another. This design approach results in a system that is generally difficult to extend and maintain. Software systems of the past have also been typically developed on a single proprietary hardware/operating system (OS) platform making the system highly dependent upon the life cycle and expandability limitations of the vendor specific platform.

Current software architectures are addressing issues such as scalability and extensibility of systems. Functionality demands on software systems are rarely static as new requirements and operational needs are often identified after a system is fielded. The ability to easily integrate new functionality in a well engineered way is crucial in today's evolving systems.

Software portability is also seen as an increasingly important characteristic in today's software systems. The capability of the software system to execute on a wide variety of hardware/OS platforms protects the system from technological obsolescence and dependencies upon vendor specific hardware/OS platforms. It also provides the economic benefit of being able to select from a broad base of competitively priced, commercial-off-the-shelf (COTS) platforms.

The ORPG is being designed as a "loosely coupled", distributed system that is composed of a collection of independent modules functioning in an interdependent fashion. Further, a layered software architectural model is being employed for the ORPG that is characterized by the encapsulation of services that will enhance the portability, extensibility, and maintainability of the new ORPG.

Corresponding author address: Michael Jain, National Severe Storms Laboratory, 1313 Halley Circle, Norman, OK 73069-8480; e-mail <mjain@nsslgate.nssl.uoknor.edu>

2.0 ARCHITECTURAL OBJECTIVES

Saffle, et al. (1997) and Zahrai, et al. (1997), discuss the primary system and software architectural requirements of this NPI project. Further, Zahrai, et al. (1997) provides a systems level discussion of the software model and the use of open system standards. The ORPG software architecture is being designed such that the resultant software is portable, expandable, scalable, maintainable, and contributes to high system availability.

2.1 Portability

The ORPG software is being developed so that it will be portable at the source code level to a variety of open system standards compliant hardware/OS platforms. A layered software architecture model, as depicted in Figure 1 and discussed in Zahrai, et al. (1997), and an adherence to open systems standards will contribute to minimizing the impact of moving ORPG software to different host platforms.

The key to portability of the layered software model is in isolating the application level programs from the underlying system infrastructure and OS dependent layers. If the need arises to change to a new OS, or more likely, a revision upgrade to an existing OS occurs, only those modules at the lower software layers will require modification. This portability attribute protects the ORPG system from becoming locked into a proprietary hardware/software system, avoids technological obsolescence, and provides the capability to keep the ORPG on the leading edge of cost effective technology.

2.2 Expandability

Over the life cycle of the ORPG, computing resource demands are expected to increase due to research advances and the development of new meteorological algorithms and products. To meet these demands the ORPG must be capable of accommodating the addition of new processors and peripherals. The ORPG software is being designed to be distributable across multiple computing platforms. This ability is supported by the Distributed Application Services layer depicted in Figure 1. Additional hardware can be incorporated into the system as required by simply modifying ORPG system configuration data.

2.3 Scalability

As mentioned in 2.2, new functionality is expected to be identified and introduced to the ORPG system over the course of its life cycle. As requirements change, the ability to easily incorporate new functionality is imperative. The ORPG software is being designed to be a collection of highly modular and independent processes. This functional encapsulation of ORPG applications and services, along with distributed processing capabilities, and the establishment and effective use of an ORPG Application Programming Interface (API) will contribute to easily incorporating new applications into the ORPG.

2.4 Maintainability

The ORPG software architecture will provide for effective software maintenance. The layered software model contributes to easing the maintenance of the ORPG software. By encapsulating and layering the system services and OS interface, changes in these areas will restrict code modifications to these layers and will not propagate changes to the application level. The use of a consistent, well engineered ORPG API in new application development will result in easing the incorporation and maintenance of these new applications in the ORPG. In the future, if an improved system service is identified, a complete replacement of this service layer module is possible without any effect on the application layer of software.

Further, in the open systems environment, there are many COTS products available that aid in software development and maintenance. Several of these Computer Aided Software Engineering (CASE) tools are being used in support of the development process and are expected to be transferred to the OSF when they assume responsibility for the software.

2.5 Availability

The ORPG software is being developed with a number of fault tolerant features so as to maintain a high level of system availability. Encapsulating applications will aid in localizing the

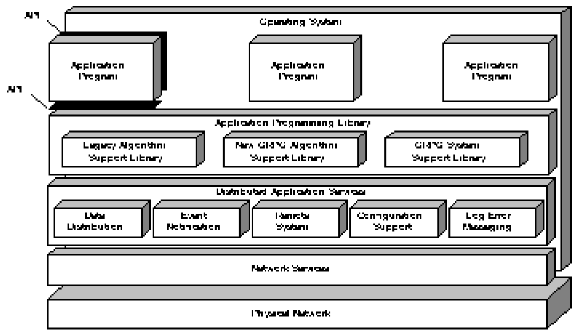

Figure 1. ORPG Layered Software Model

impact of application failures by preventing the error or failure from propagating throughout the system. The distributed processing attributes of the system allows for the use of multiple processing nodes. This provides protection in the event of a catastrophic failure of one of the ORPG nodes. Should a node fail, applications that were running on the failed system would be automatically redistributed to the surviving nodes. Similarly, in the event that a particular node becomes compute bound, the system will have the ability to dynamically redistribute tasks across all participating nodes to achieve a leveling of processing load, thus minimizing potential load shedding. Properly sized data buffering space, together with replication of critical data across ORPG storage locations will enhance failure recovery.

3.0 INFRASTRUCTURE SERVICES

The following is a brief description of the ORPG infrastructure services. These services are depicted in Figure 1 by the Distributed Application Services Layer. One should note that these services are not specific to RPG functionality. Instead they represent services required of any distributed processing system.

3.1 Data Distribution Services

ORPG applications are being designed and implemented to be encapsulated processes, not concerned with knowing what application(s) produces the input data required to perform its function, nor what application(s) might consume its products. ORPG applications are designed to be responsible for retrieving input data and writing output data to well-known system locations. This is in contrast to a message passing model where the source and destination of each piece of information is predefined and managed by a control module. This ORPG model allows for the building of more flexible and expandable distributed applications. It will also support building fault tolerant applications with relative ease.

3.1.1 Linear Buffer (LB)

The ORPG uses LBs for distributing radar data, products, and other system information. The content of an LB will be hereafter referred to as messages. The LB is a tool designed for building distributed applications that apply an INTERNET/WEB-like model. In this model, messages are stored in an open and distributed way for convenient system-wide access.

An LB is an allocated piece of storage space, a file or a segment of shared memory, with a well-known name. Messages are stored in and retrieved from an LB via LB function calls. These functions, in addition to reading/writing messages, maintain control information in the LB for message management and provide access control and data integrity protection, which are critical in supporting reliable applications in an unreliable networked workstation environment. The LB provides the following support:

- Multiple readers and multiple writers can effectively access an LB simultaneously. Multiple writers are automatically prevented from actually writing to a single LB simultaneously. No locks are needed among multiple readers and between a reader and a writer. This can substantially improve the access performance when there are a number of readers. The LB also supports random access of its contents through the use of internal identification numbers assigned to each message.
- The LB is a reliable message passing channel. No messages written to an LB are ever lost before their expiration and no corrupted messages are ever read as long as the OS is running. A reader or a writer can terminate or crash at any time without destroying the LB or leaving corrupted messages.
- The LB automatically manages the allocated storage space. When there is no space available for a new message, the oldest messages are expired by default to make room for the new message. While new messages keep arriving, the older messages keep expiring. Only a certain number of the most recent messages are available in the LB at any given time. The LB tracks the entire message expiration history. The set of available messages is considered as a line segment moving forward on an infinitely long time axis. Every reader has its own read pointer pointing to a particular message. When the reader returns to read that message later and the message is

expired, the reader will be notified that the message is gone and can move on to the next message. Thus, the reader can traverse the entire message history to point to any message that is either expired, available or yet-to-come. The expired messages are lost but their identities in the message sequence are still traceable.

- The LB supports transparent accesses across hosts in a networked environment, which means by using the same LB functions, one can read messages from or write to an LB on any host on the network.
- The LB is an efficient message storage and retrieval tool designed for distributed applications requiring extensive data exchange. It supports both shared memory and file implementations with an identical API. When a file is used, the file is mapped into the workstation memory for maximum performance. The storage type can be selected at run time. When the shared memory type is chosen, the reader can directly access the buffer in the LB, which eliminates a message copy.

3.1.2 Data Distribution Services: Reliable Broadcast and Receive

The ORPG is expected to be composed of multiple processing nodes on a Local Area Network (LAN). Therefore, a service to assure the reliable distribution of the basedata to multiple nodes is provided through the implementation of two tools: bcast, a broadcast tool and brecv, a receiving tool. Reliable distribution means that any message sent by the broadcast tool is guaranteed to be received, in the correct sequence, by each receiver running on various remote nodes. These routines (bcast/brecv) provide the following support:

- There is no limit to the number of nodes that can receive the messages.
- bcast, running in the background, reads incoming user messages from a Linear Buffer on the local node. It then divides the messages into segments and broadcasts the message segments via UDP packets using a socket port. The receiving task (brecv) runs on each host node receiving the UDP packets. The received packets are re-assembled according to the header information to regenerate the original user messages. The regenerated messages are then written to an LB for local tasks to retrieve.
- A TCP connection (socket) is set up between the broadcaster and each receiver for passing acknowledgments, retransmission requests, and retransmitted packets. When a certain number of message packets are received by a receiver, it sends an acknowledgment to the broadcaster. If a message segment is detected as being lost by a receiver, the receiver sends a retransmission request to the broadcaster. Upon receiving a retransmission request, bcast resends the lost message segment. Since TCP is a reliable protocol, no further error control or retransmission is needed. Message segments received by all receivers are discarded from the broadcaster's buffer and new message segments are then sent. With buffers set up on both broadcaster and receiver sides, message reception acknowledgment does not need to be sent for each packet reception.

3.2 Event Notification Services

The Event Notification (EN) Service allows an application to notify other applications running either locally or on another ORPG node that a particular event has occurred. It also allows an application to be notified by one or more applications running either locally or on another ORPG node that particular events have occurred. The EN service provides a degree of responsiveness that may be required to supplement an otherwise data-driven system. The EN Service provides the following:

- The EN Service consists of a daemon process that runs on every node in the ORPG and a supporting library of functions that allow applications to post and receive events. The EN node daemon handles event notification, registration, deregistration, and posting for a given node on the system. The EN daemon disseminates locally-posted events throughout the system.
- When a receiving process requires notification of events, it registers with the daemon process. When a process posts an event, notification of that event posting is disseminated throughout the system. This dissemination is accomplished both locally and remotely. Locally, the node EN daemon notifies its registered node processes of the posting of the event. Remotely, the node EN daemon broadcasts notification of the posting to every other node EN daemon within the system. Note that this broadcast will be reliable inasmuch as remote node EN daemons may request retransmission of event notifications.

3.3 Distributed Computing Services

The Distributed Computing Services represents a family of services that provide the ORPG the ability to perform in a distributed computing environment.

3.3.1 Remote System Services

The Remote System Services (RSS) module implements remotely callable common system support functions that are useful for developing distributed applications. These functions include a set of selected UNIX system calls and standard C library functions as well as LB functions. These allow an application to make function calls on a remote node as if it were running on that node. The RSS provides the following support:

- A library is provided that allows the user to create the server as well as client applications. The server, containing the remote functions, runs on the remote node. An application that needs to perform a remote function call links the library into its executable. When an application makes a remote function call, the routine first looks for an existing connection to the server on that node, and, if the connection does not exist, it makes a connection to the server through a socket. Then it sends a request to the server and waits for a return message. The server then responds to the request by performing the required procedure call and sending the results to the client.
- The remote function call facility is built on top of UNIX TCP/IP sockets. Thus the communication is reliable unless there is a network or host failure.

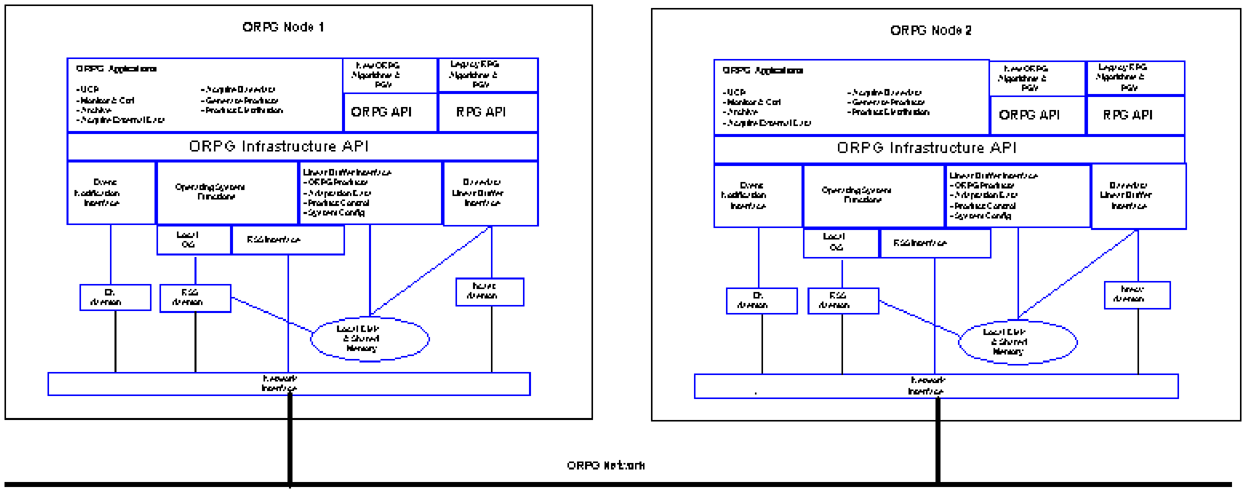

Figure 2. ORPG Software Architecture in a Multiprocessor Environment

3.3.2 Configuration Support Services

The Configuration Support (CS) module is designed for supporting dynamically reconfigurable, distributed applications whose processes, files, and other resources can be moved from one node to another without shutting down and restarting the entire application or system. The CS Services provides the following:

- An application can find dynamically, changing configuration items, such as resource locations.
- The CS module reads in configuration data from a file or an LB and stores it in memory for later look-up. The stored configuration data can be updated through a CS function call or from a specified event received from the Event Notification (EN) service.
- Allows the user to register a configuration status variable in the CS module. This mechanism provides an efficient way for calling routines to determine whether or not a configuration item has changed.
- It is designed for both full-feature and light-weight uses. In the first case, an LB tool and event notification services may be used for allowing reliable, timely, and efficient dynamic modification of the configuration data. In the latter case, a simple static configuration file can be used.

3.3.3 Log-Error Messaging Services

The log-error messaging (LE) module is a formatting and distribution tool for log and error messages. An LE message contains an error code and a text message. The error code may be used to indicate the message characteristics such as category, group, type, and severity of the message. The message code is application specific and interpreted typically by application processes (e.g. by the ORPG system monitor/manger).

4.0 DISCUSSION

Figure 2 illustrates a more detailed representation of the ORPG software architecture in a multiprocessor environment. This figure represents two nodes on the ORPG network. Both nodes are shown as being functionally equivalent with the exception that the ORPG Node 1 is running a bcast daemon, reading basedata from a local LB and broadcasting the basedata to the LAN whereas ORPG Node 2 is running a brecv daemon receiving the basedata and writing the data to a local LB. All application software access the basedata linear buffers the same on all nodes. Any number of such nodes could be configured to run on the ORPG LAN.

The top levels of Figure 2 represents the application level of the ORPG. Here, the ORPG applications, legacy and new generation algorithms and product generators (PG) are depicted. Note that a separate RPG API and support library (Fig.1) is provided for the legacy algorithms/PG's (Jing & Smith, 1997). This API preserves the existing algorithm API and enables these FORTRAN algorithms to be ported to the ORPG environment with little if any modification. A separate API and support library is being developed for future algorithm/PG development and implementation. This will aid in providing efficient and reliable routines that have been designed for the open systems environment. The ORPG specific applications are shielded from the infrastructure of the ORPG via an API.

5.0 ACKNOWLEDGMENTS

The authors would like to acknowledge the Office of Systems Development of the National Weather Service for funding this project. We would also like to thank members of the SEDT and other OSF personnel that have participated in this project.

6.0 REFERENCES

All references are to papers included in this preprint volume. Preprints 13th International Conference on Interactive Information and Processing Systems (IIPS) for Meteor., Ocean., and Hydro., Long Beach, CA, Amer. Meteor. Soc.