A Robust and Scalable Architecture for Airborne Radar Simulation

K N Rajanikanth
Aeronautical Development
Agency
PB 1718, Vimanapura Post
Bangalore-560017, India
91-080-5087328

knrajanikanth@rediffmail.com

Y Narahari
Indian Institute of Science
Computer Science and
Automation
Bangalore-560012, India
91-080-3942773

hari@csa.iisc.ernet.in

N N S S R K Prasad
Aeronautical Development
Agency
PB 1718, Vimanapura Post
Bangalore-560017, India
91-080-5087224

nnssrkprasad2001@yahoo.com

R S Rao
Aeronautical Development
Agency
PB 1718, Vimanapura Post
Bangalore-560017, India
91-080-5232873

r_srao@lycos.com

Abstract - This paper presents the evolution of a robust and scalable software architecture for airborne radar simulation. We make a comprehensive analysis of the problem domain for building the architecture. Both structural and run-time views are provided. Proposition of generic interfaces for inter-operation with external systems, environment, and internal radar blocks is a novelty that the system offers, and makes the architecture extensible. Robustness is built into the architecture by considering real-time issues by investigating aspects like critical threads of execution analysis and schedulability analysis. Architectural scalability and resilience are achieved by proper definition of subsystems. We have used UML (Unified Modeling Language) to capture the important artifacts.

1. INTRODUCTION

In the future, complex and expensive systems like radar [4] would be designed, built, and tested with computer modeling and simulation techniques. Comprehensive simulation must take into account the diverse needs of different stakeholders like analysts, designers, field technicians, and operators. In this paper, we describe the analysis and architecture construction process for developing a comprehensive tool, which addresses various needs that arise during radar simulation. Current practices in radar simulation do not place emphasis on building a software architecture taking into account various issues pertinent during the construction of such a system. Architectural design brings out the top-level design decisions and permits the designer to reason about satisfaction of system requirements in the design elements.

We have used UML 1.3 to capture the software artifacts.

1.1 Motivation

The avionics system of a modern combat aircraft features an array of electronic equipment, each of which performs a specific function, and hence ensures success of a mission. The Radar onboard such an aircraft is the primary sensor for fire control and avionics system. The radar features a fully coherent pulse doppler system with excellent detection and ranging in both look-up and look-down modes. A simulation system for the airborne radar caters to the following objectives:

- Test the radar component module of an interacting system like the Central Computer

- Provide development life cycle support for the airborne radar (for example, validation of software enhancements)

- Investigate functionalities pertaining to the radar system vis-à-vis its environment

1.2 Contributions and Outline

This paper evolves a robust and scalable architecture for airborne radar simulation by a compelling use of the following:

- Determination and effective use of architectural drivers (Modifiability, Balanced Specificity, Reusability, Integrability, Performance, Functionality, and Usability) in evolving the software architecture

- Object-driven subsystem design

- Innovative discovery of IDLs (Interface Definition Language) for the problem domain

- Treatment of time as a first class entity

- Building real-time features like concurrency and schedulability

Section 2 deals with capturing requirements of the system through use cases. In Section 3, we present considerations that provide the architectural drivers for the

system. In Section 4, we use OO-Analysis to evolve a preliminary analysis model. This model is transformed into a system of subsystems. We provide a method of discovering the IDLs, the cornerstone for a mature architecture. Section 5 looks at ways to guarantee response times. Section 6 evaluates the architecture using a scenario-based method, demonstrating the superiority of the architecture. Section 7 concludes the discussion on architectural issues. In passing, we make a reference to the design patterns that have been used in building the research prototype 'ARSENAL' (Airborne Radar Simulation aNd EvALuation) [7].

2. CRYSTALLIZING REQUIREMENTS

In collecting the *Requirements Specification* we have resorted to capturing a detailed description of the proposed uses of the system using the viewpoint oriented requirements gathering technique. We have identified the stakeholders of the simulation system to be the following: Pilot, Radar System Designer, Radar Component Designer, System Tester, Avionics System Integrator, Simulation User Roles Played: Target Configuration Manager, Flight Path Manager, Simulation Designer, Radar Environment Designer, Simulation Manager, Radar Operator. With each stakeholder in focus, we have captured the use cases of the system. We used preliminary user interface screens for helping the process of detailing the use cases. The GUIs were designed with four points in mind: (1) activation of entities in the simulation (2) control of the simulation process (3) monitoring the simulation activity (4) provision for a point of entry to enable radar system component configuration and interacting systems configuration.

3. ARCHITECTURAL DRIVERS

Architectural drivers are primarily quality attributes, which must be accommodated in a system's architecture for technical and operational considerations. These qualities are over and above that of functionality.

Modifiability: usage of the simulation system for diverse purposes like evaluation of radar subsystems, training, evaluation of interfaces.

Balanced specificity: architecture should provide a meaningful base of interfaces sufficiently detailed enough for subsystem developers and component builders to build the individual elements and general enough to allow situation specific features and optimizations to be added.

Reusability: portions of the subsystems or the subsystems can be reused separately.

Integrability: is needed since it is also required that the simulator be used for testing radar system interfaces with external interacting systems. .

Performance. The radar simulator must respond to control inputs as quickly as the real radar system would. Fixed frame rates at high frequencies are required to achieve fidelity.

Functionality: the simulator is envisaged to perform a wide variety of functionalities (ranging from radar mode simulations to system configurability).

Usability: the simulator's user interface portion is separated from the actual application and the right information needs to be exchanged between the user and the system for successful operation of the system.

Modifiability, balanced specificity, reusability, and integrability are system quality attributes not discernible at run-time, while performance, functionality and usability are those discernible at run-time.

4. STATIC ARCHITECTURE

The radar simulation environment models the radar system and its environment. The modeling approach incorporates a realistic treatment of the interaction of the simulated radar model with terrain, electronic counter measures, and targets. Interaction with water, weather can also be modeled. The environment model provides the basic inputs that are applied to the radar model. Terrain modeling (clutter activity) involves clutter modeling described by topography, radar reflectivity and surface attributes. Electronic counter measures modeling (jamming activity) involve generating false targets and smart noise. Target modeling (target generation activity) involves generation of flight profiles of the targets by defining segments of straight lines or curves, and specifying initial and terminal speeds. These profiles may incorporate a fluctuation model such as swerling1. To provide increased target modeling fidelity, efficient multiple-scatterer models can be used. The rest of the model can be partitioned in much the same way as the actual radar. It consists of an antenna, transmitter, receiver, a signal processor, data processor and interfaces to other systems such as central computer, inertial navigation system, and display system.

The inertial navigation system and central computer are elements that support the radar system for its proper functioning, and we need not choose complete and detailed models of these systems. Simple input/output models are sufficient.

Robustness analysis using sequence diagrams, leads to a first-cut set of rich conceptual classes. The objects found are classified into stereotypes: Boundary objects, entity objects, and control objects. Examples for boundary objects are:

Input Keypad, Multifunction Input/Output Device, Radar Controls Panel. Examples for entity objects are: Central Computer, Inertial Navigation System, Radar System, Space, Targets, Background, Clutter, and ECM (Electronic Counter Measures). One should note that not all use cases require control objects. If we choose the decentralized control scheme, examples of control objects would be: Radar Controls, CC (Central Computer) Controls and 'INS Controls'. It should be noted that "time" is an entity which sort of controls the classes, since with every clock tick, the control objects are driven first, which in turn drive the main objects in the system like radar system, central computer and inertial navigation system to perform their actions. 'Configuration Manager' for configuring the system and a 'Target Control' class that handles all target related changes are candidates for control classes. This leads to definition of the analysis model through a class diagram.

4.1 Defining the Subsystems

The process of architectural construction involves taking the analysis model we built in the previous section and performing a coarse decomposition of the system into groups. Groups decompose into systems, which, in turn, decompose into subsystems. Analysis classes provide a way of identifying large-scale organizational entities. Classification into groups is done based on physical partitions within the domain. The large-scale groups are shown diagrammatically in Figure 1, which represents the reference model.

To arrive at the subsystems we apply atomic operations of architecture, called unit operations to the groups of subsystems. These operations are compression, abstraction, resource sharing, uniform decomposition and replication. The relationships between unit operations and qualities are given in [5]. In the first step, we prioritize the quality requirements, since the order of composition of the unit operations has to be determined and these software transforming operations are not, in general commutative:

- P1 Sequential Performance
- P2 Concurrent Performance
- P3 Modifiability/Extensibility
- P4 Integrability
- P5 Balanced Specificity
- P6 Reusability

To come up with a reference architecture (a reference model mapped onto software components) for the application, we strive to achieve the quality attributes from P1 through P6, taking the application portion of the reference model provided in Figure 1.

Applying the unit operations associated with each of the quality requirements results in arriving at the architecture shown in Figure 2, which provides the reference architecture of ARSENAL application.
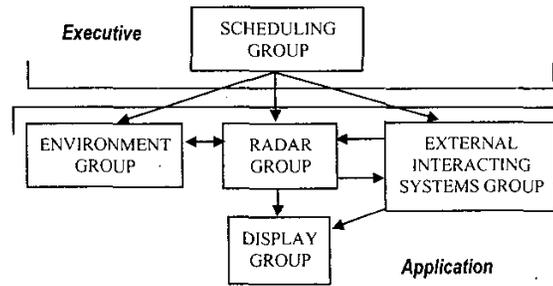

Figure 1: The Large Scale Groups

4.2 Subsystem Integrity

We have chosen the *Horizontally integrated subsystems* approach since we foresee that the user interface changes can occur independently from rest of the model. Changes to system services are neatly handled in the respective subsystems. We have also looked at aspects like cohesion and coupling between the classes to validate our decisions about the subsystems within the major systems.

4.3 Defining the Interfaces

An important contribution is the definition of generic buses for the application. Our method of eliciting the ARSENAL interfaces is through the use of *n*-square charts [5]. A close observation of the main *n*-square chart (for the entire radar simulation environment) reveals presence of two kinds of "generic buses". The Beam bus centered on the Beam subsystem and the Radar System Interaction Bus centered on the communication subsystem. Table 1 is an example of a sub table obtained from the main n-square chart, when the cells surrounding the generic bus are extracted.

Table 1: Discovering the Beam Bus – An Example

	Echo Signal	
Input Signal	Beam Subsystem	Rx Beam
	Tx Beam	

5. REAL-TIME ISSUES

A real-time architecture must not preclude meeting response time constraints. The runtime architecture models concurrency, by examining two aspects of the problem: (1) Task definition and communication, and (2) Timeliness and schedulability issues in terms of timing details associated with object communication. For examining the first aspect, we use the UML component view (showing tasks and component links) and for the second, we use UML sequence diagrams to convey the timing details.
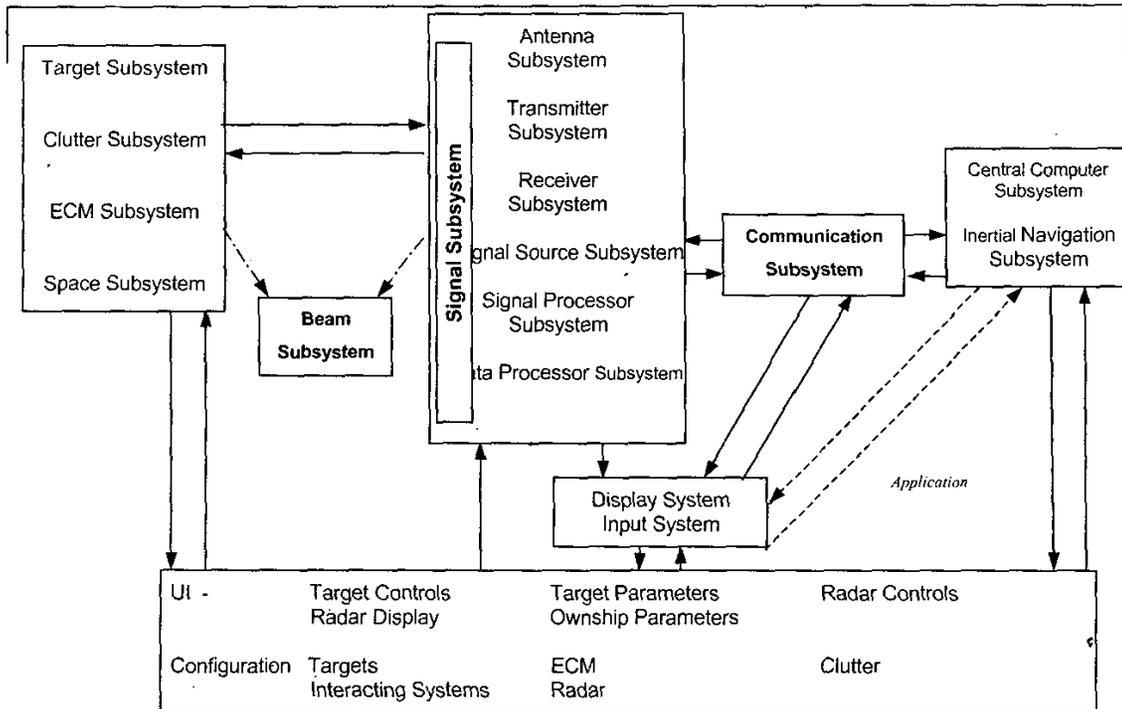
Figure 2: The Reference Architecture

5.1 Defining the Threads/Tasks

We have used the approach [1] of grouping events in the system so that a thread handles one or more events, and each event is handled by a single thread. This leads us to the definition of threads and processes in the system. Table 2 provides a listing of the final threads.

Next, we fix the timing constraints for each (critical) service along a critical thread of execution. This phase involves identifying events that have a corresponding hard, real-time deadline [3]. We look to the analysis sequence diagrams to help us in this activity. In each of these sequence diagrams we place timing marks to make estimates of the task execution times and their deadlines. These sequence diagrams helps us make an estimate of the worst-case execution time of the critical threads. While estimating the time on the sequence diagrams, we take into account issues like hardware requirements (sensor computations), algorithmic requirements (numerical stability), and human factor (response times) requirements.

5.2 Schedulability Analysis

Scheduling tasks may be done either by using a run to completion policy or a preemptive policy. The most common scheduling policy for real-time systems is the preemptive scheduling policy. Preemptive scheduling policy schedulers assign each task a priority, which establishes the precedence of the task when multiple tasks are ready to run. Two keys to good real-time design are repeatability and predictability. The key to determining predictability is to define the timing characteristics of tasks and to properly schedule them using a predictable scheduling algorithm. We employ one such algorithm called Rate Monotonic Analysis (RMA) for predictable real time task scheduling [6]. In the basic RMA, tasks are assigned priorities as a monotonic function of the rate of a periodic process, given by the simple inequality

$$\sum_{i=1}^{n} [C_i / T_i] <= n (2^{1/n} - 1) \qquad (1)$$

C_i and T_i represent the execution time and period respectively associated with periodic task τ_i, and n is the number of tasks. Applying the RMA to the set of ARSENAL tasks we have the intermediate results shown in Table 3. The utilization bound (term on the RHS of inequality (1)) for the five tasks is 0.7435. The computed utilization of 0.7125 is less than the utilization bound for the five tasks and hence the system is guaranteed to always meet its deadlines. The schedulability analysis fixes the upper bounds on each task. We use an iterative process to

arrive at the performance/timing constraints for the subsystems.

Table 2: The Main Processes

Event	Event Type	Event Grouping Strategy	Process
User presses a key on the UI (MFD/RCP...)	External	Event Source	UI
Central Computer sends data to Radar {Timing Tick}	Internal	Related Information	Central Computer
Central Computer sends data to Display {Timing Tick}	Internal		
INS sends Data to Radar {Timing Tick}	Internal	Event Characteristics	INS
INS computes Data (present position) for Radar {Timing Tick}	Internal		
Radar Sends Tx signal	Internal	Sequential Processing	Radar
Radar captures Rx Signal	Internal		
Radar Processes Rx Signal	Internal		
Target enters Search space	Internal/ External	Sequential Processing	Radar Environment
Target changes position (moves)	Internal		
Clutter interacts with beam	Internal		

6. ARCHITECTURAL QUALITY

Table 4 presents a method to evaluate the architectural quality of the application ARSENAL, wherein we have used Software Analysis Architecture Method, a scenario-based method for analyzing architecture [5]. An examination of the Table indicates low coupling and high cohesion characteristics of ARSENAL. Extremely low scenario interactions reveal good separation of concerns.

7. CONCLUSIONS

A well-defined macro-architecture alone does not ensure an efficient software solution. Design of a system is equally important. Design of the top-level foundational subsystems is accomplished using design patterns [2, 7, 8]. ARSENAL is a discrete state/continuous time, non-distributed, well-architected air-borne radar simulation tool, which functions as both a system design tool as well as an engineering research simulator, with huge potential for extension, easy modification, and diverse usage scenarios. The front end has been implemented in VC++, and the design classes using C++. All radar signal generation, transformation, and processing is done using a MATLAB computational engine running in the background.

The architecture was conceived to answer the question "How does one simulate various aspects of a radar system and its environment using a single tool?".

Table 3: RMA values with 5 Tasks

Task	Exec Time (Ci)	Period (Ti)	Ci/Ti
UI	5	80	0.0625
Central Computer	10	80	0.125
INS	2	40	0.05
Radar	8	20	0.4
Radar Environment	6	80	0.075
Computed Utilization			0.7125

Table 4: Scenario Evaluation for Architectural Quality

Scenario	Direct/ Indirect	Changes Required
Conf Radar Comp. Impl	Direct	None
Change Radar Comp. Implementation	Indirect	Modification to Radar System SS, UI & Config.Sys SS.
Change data exchanged b/wn Rdr & Ext. Int. Sys	Indirect	Modifications to Communication SS.
Conf. Radar Env. Chars	Direct	None
Change Clutter/ECM/Tar models	Indirect	Modif req. to Radar Env. Sys SS, UI, & Config. Sys SS
Change Component Scheduling strategy	Indirect	Modifications required to Scheduling System SS
Integrate new External Systems	Indirect	Modifications. req. to Ext Int. Sys . UI, and Config. Sys SS.

8. REFERENCES

[1] Bruce Powel Douglass. Doing Hard Time – Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns. Addison-Wesley Longman Inc, 1999

[2] Erich Gamma, et al. Design Patterns: Elements of reusable O O software, Addison-Wesley, 1995

[3] Hasan Gomma, Designing Real-Time Applications with the COMET/UML Method. www.realtime-info.be/magazine/01q1/2001q1_p044.pdf, 2002.

[4] Hovanessian S.A. Radar Detection and Tracking Systems. Artech House Inc., 1973.

[5] Len Bass, Paul Clements, Rick Kazman. Software Architecture in Practice. Pearson Education Inc, 1998.

[6] Lui sha, Mark H. Klein, John B. Goodenough. Rate Monotonic Analysis for Real-Time systems. CMU/SEI-91-TR6 ESD-91-TR6, SEI, CMU, March 1991.

[7] Rajanikanth K N. ARSENAL: Airborne Radar Simulation and Evaluation, Object-Oriented Analysis, Design and Implementation of Radar Simulation Software, MSc (Engg) Thesis, Dept of Computer Science and Automation, Indian Institute of Science, Bangalore, India, 2002.

[8] Rajanikanth K N, Y Narahari, N N S S R K Prasad, R S Rao. A Robust Design of Airborne Radar Simulation Software using Design Patterns, NCOOT 2003, India.