REPORT DOCUMENTATION P

AD-A277 503

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	
	March 1994	Professional Paper

4. TITLE AND SUBTITLE

A PROPOSED SHIPBOARD READINESS REPORTING SYSTEM

5. FUNDING NUMBERS

PR: EE03
PE: 0602233N
WU: DN301014

6. AUTHOR(S)

M. Vineberg and T. Sterrett

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Command, Control and Ocean Surveillance Center (NCCOSC)
RDT&E Division
San Diego, CA 92152-5001

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

David Taylor Research Center
Bethesda, MD 20084

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

94-09663

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. ABSTRACT (Maximum 200 words)

This paper describes a Shipboard Readiness Reporting System (SRRS) proposed to automate readiness reporting aboard ship. The SRRS will report operational capabilities to tactical users, repair needs to technical users, and ship status to higher echelons. It will support maintenance scheduling based on preserving operational capability.

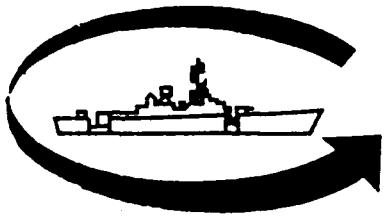
The SRRS is designed in the context of a layered open architecture in anticipation of future shipboard systems. The highest layer of the architecture includes both operational functions (relating directly to the ship's mission) and readiness reporting functions. The SRRS will receive frequent status reports from the resources necessary to perform functions and synthesize resource status reports to determine operational capability (in terms of ability to perform operational functions) and repair requirements. The SRRS is designed to operate in spite of system damage.

Published in *Tenth Ship Control Systems Symposium*, vol. 3, October 1993, pp. 3•281 to 3•298.

14. SUBJECT TERMS	15. NUMBER OF PAGES
local area network (LAN) readiness reporting	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

21a. NAME OF RESPONSIBLE INDIVIDUAL	21b. TELEPHONE *(Include Area Code)*	21c. OFFICE SYMBOL
M. Vineberg	(619) 553-6557	Code 016



PROCEEDINGS

ACTES

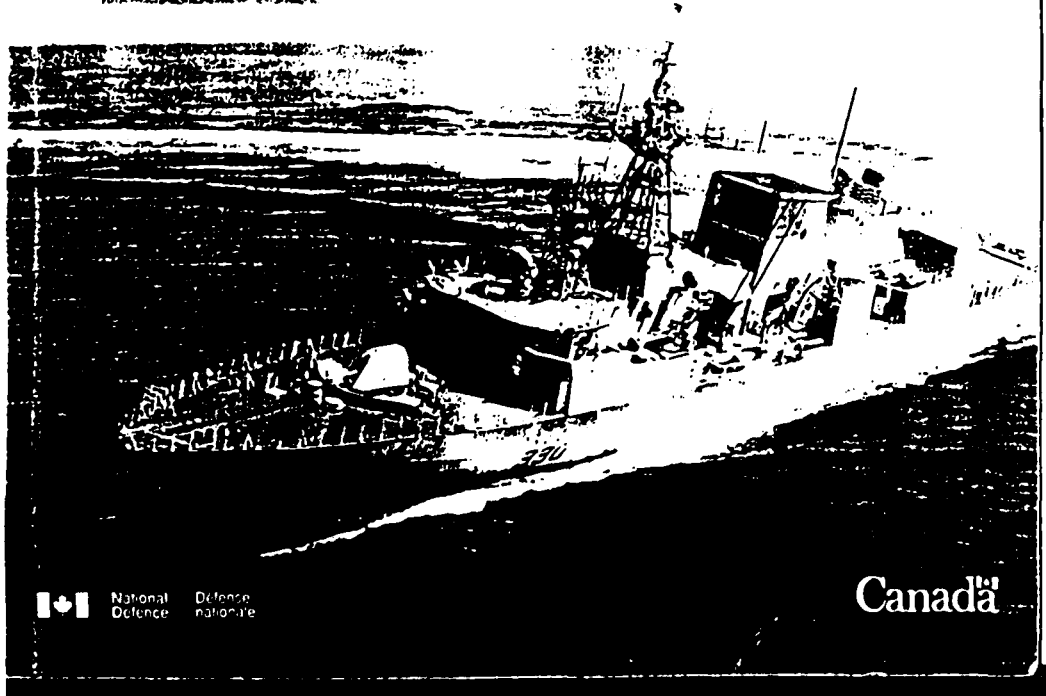
VOLUME 3

TENTH

SHIP CONTROL SYSTEMS SYMPOSIUM

DIXIÈME

COLLOQUE SUR LES SYSTÈMES DE COMMANDES DES NAVIRES



A PROPOSED SHIPBOARD READINESS REPORTING SYSTEM

Maniel Vineberg
Tony Sterrett
Naval Command, Control and Ocean Surveillance Center
Research, Development and Evaluation Division
San Diego, California

ABSTRACT

A Shipboard Readiness Reporting System (SRRS) is proposed to automate readiness reporting aboard ship. The SRRS will report operational capabilities to tactical users, repair needs to technical users, and ship status to higher echelons. It will support maintenance scheduling based on preserving operational capability.

The SRRS is designed in the context of a layered open architecture in anticipation of future shipboard systems. The highest layer of the architecture includes both operational functions (relating directly to the ship's mission) and readiness reporting functions. The SRRS will receive frequent status reports from the resources necessary to perform functions and synthesize resource status reports to determine operational capability (in terms of ability to perform operational functions) and repair requirements. The SRRS is designed to operate in spite of system damage.

1. INTRODUCTION

The US Navy mission and structure continue to undergo rapid change: "...From the Sea," a recent Navy and Marine Corps White Paper, defines a combined vision for the Navy and Marine Corps which includes "support for long-term operations, flexible and sustainable sea-based forces, and Naval Forces [that] can be continuously tailored to developing events;" units and platforms in five media (space, air, land, sea-surface, and underwater) are now integrated into single warfighting entities; multiple command levels must be appropriately supported for operating and maintaining all resources under their control in separate and overlapping warfare areas.

The new surface combatant is more complex than ever: (a) it must cope with threats of increased capability and sophistication; (b) it must combine with other ships, submarines, aircraft, and land and space assets to form combined task forces; (c) shipboard systems must support independent, coordinated, and cooperative operations; (d) ship systems and personnel are widely distributed for survivability; and (e), time to assess readiness status and make "fight or flight" decisions continues to diminish.

A US Navy surface combatant must be integrated with respect to readiness assessment and reporting in order to operate with other surface ships, submarines, and aircraft within a joint task force. Force commanders need readiness information to accomplish required functions as well as to report status to higher echelons. To determine the operational and maintenance status of a force, the status of each ship must be known. A ship comprises systems, where a system requires combinations of resources to perform user functions. The SRRS will address systems on a single ship. This paper focuses on the combat system; hull, mechanical, and electrical (HM&E) system issues have been reported separately [Nickerson 1990].

The SRRS addresses these technical problems: (a) accurate assessment of platform capability and repair needs is difficult and time-consuming; (b) there are no design standards for reporting status; (c) there is no consistent methodology for distributing, synthesizing, and displaying status reports; (d) no distinction is made among an equipment that has failed, an equipment that is in other than the normal operational mode or state, and an equipment that is not fully initialized; (e) testing periodicity and fault detection and isolation coverage are substantially different from equipment to equipment and this is frequently not accounted for in current readiness reporting and assessment approaches; (f) operational and maintenance data and information are mixed and indiscriminately provided to both tactical and technical users; (g) data need to be filtered such that they are more timely, concise, and meaningful to the user; (h) readiness terminology varies widely from equipment to equipment - different terms are used to mean precisely the same thing while one term frequently has multiple meanings; (i) maintenance scheduling aboard ship is not supported by automated tools linked to operational planning; and (j) readiness reporting to higher echelons is labor intensive.

1.1 Scope and Objective

The objective of this work, illustrated in Figure 1, is to develop the SRRS by writing new software to benefit from the advent of shipboard data networks (such as the Interior Integrated Command and Communications (IC)2, currently under development), and from equipment built-in-test and calibration hardware and software. The SRRS will synthesize status reports, collected from widely distributed shipboard resources automatically via the networks, to determine operational capability and repair needs and to distribute relevant reports to tactical and technical users. The synthesized reports will include the readiness data and information needed to make correct and timely operational and repair decisions. The SRRS will support maintenance scheduling by showing the impact of equipment down time on operational capabilities. The SRRS will support the reporting of shipboard readiness status to higher echelons.

1.2 Approach

The SRRS approach features a context based on a layered architecture and the use of software and hardware prototyping. The architecture delineates the partitioning of the SRRS and also holds the key to the synthesis process. Software prototyping, that is the building of a model of the SRRS in software, has been used for concept development. Hardware prototyping, the development of a model of the SRRS on a network testbed, is fundamental to concept validation.

The SRRS is least likely to be available when most needed, that is when the ship has just been damaged. To successfully report and assess damage, the SRRS must be able to survive damage to the resources it uses. Therefore, SRRS survivability has received considerable attention.

The SRRS must be designed into the ship architecture. The SRRS must present the "right amount" and "right kind" of information to tactical and technical users and must provide timely access to extensive detail on request.

1.3 Organization of the Paper

Results of the SRRS development are reported here under four headings: design; software prototype; hardware prototype; and, summary and plans.

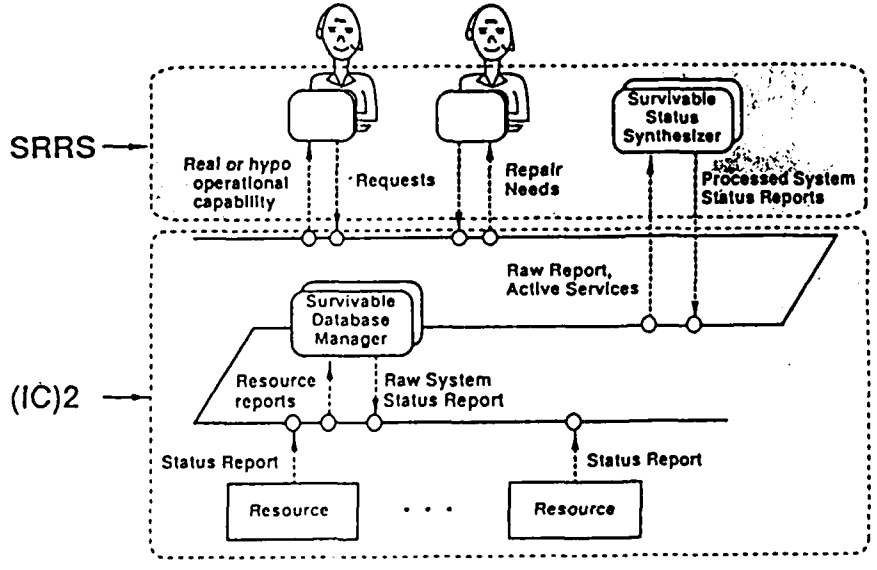


Figure 1: Objectives: SRRS Block Diagram

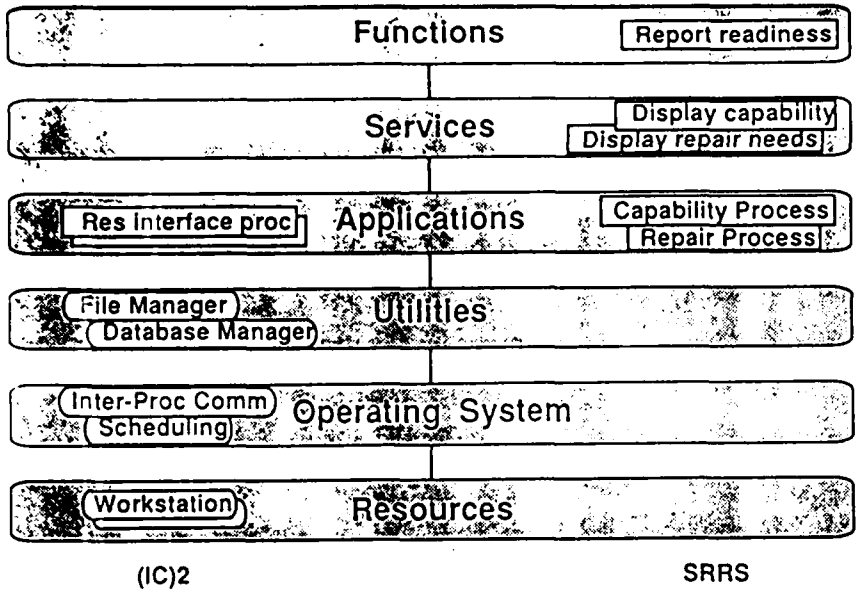


Figure 2: Layered Architecture

2. DESIGN

The architecture is described first. Then, features of the SRRS are described within the context of the architecture.

2.1 Architectural Context

The SRRS Layered Architecture, depicted in Figure 2, reflects directions in US Navy tactical system design [Vineberg 1991, Copernicus 1991, DISA 1992]. It also represents a perspective on the shipboard combat system. The closer this view corresponds to the actual shipboard combat system design, the more effective the SRRS will be in reporting operational capability.

Each layer of the architecture contains hardware resources or software processes which support processes in the next higher layer. Holding interfaces between layers stable, over time, minimizes the negative impact that enhancements at one layer have on other layers.

The various layers carry the following meanings: *functions* are system operations which may be requested by a user; *services* are primitive system operations used to implement functions; *applications* and *utilities* are special-purpose (mission-oriented) and general-purpose (generic) software processes respectively which support the services; the *operating system* is software that makes the resources usable; *resources*, including both special-purpose (weapons, sensors, etc) and general-purpose (workstations, networks, etc) hardware, are the devices which host the applications and utilities.

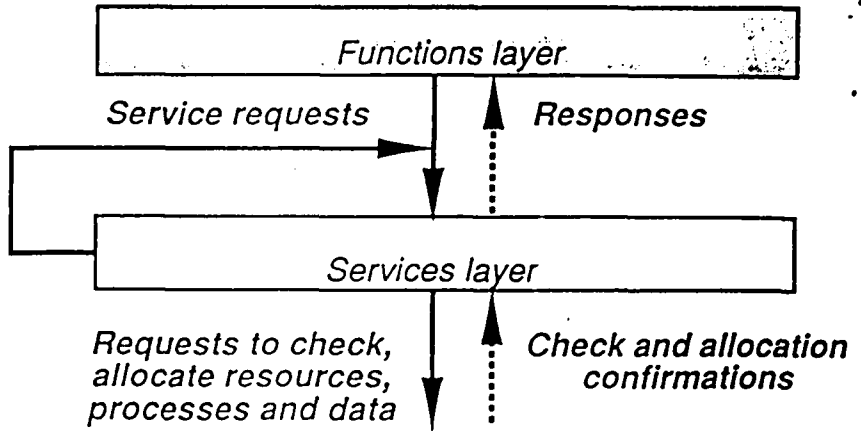


Figure 3: Functions Layer

The architecture makes it possible to define *readiness functions* which portray operational capability with respect to the sets of mission-related *operational functions* and services which the system can (or cannot) perform with the resources available.

2.2 Functions

A function is an operation that a user may request of the system, as shown in Figure 3. To each request, he assigns a unique priority which is carried by the system throughout all operations related to that function and is used (along with scheduling priorities) in the resource-contention-resolution process.

From the SRRS viewpoint, there are three classes of functions: *administrative functions* allow users to configure the system; *operational functions* directly support the mission of the ship; *readiness functions* support the assessment of operational capability and of maintenance requirements. At present, one readiness function, *report readiness*, is defined.

Function requests are translated into one or more service requests and passed to the services layer. Unless all requested services can be performed (e.g., for lack of available resources), the requested function will not be performed. This satisfies an underlying design rule that no single function request be capable of causing system deadlock. This rule, together with a mechanism to ensure that every active function carries a unique priority, will prevent system deadlock.

Three example functions are currently defined: (1) *find submarines*; (2) *report current platform position and velocity*; (3) *report readiness*. The first two are operational functions; the third is a readiness reporting function.

2.3 Services

Services are system primitives which implement functions, as shown in Figure 4. One service may implement more than one function. Two services defined to support the function report readiness are *report system status* and *recommend resource change*. Only the first service is necessary to accomplish the function.

Several operational services have been defined to support the concept development process for the SRRS. In support of the function find submarines, the services *deploy sonabuoys* and *process sonabuoy data* have been defined. Both services are necessary to accomplish the function; logical

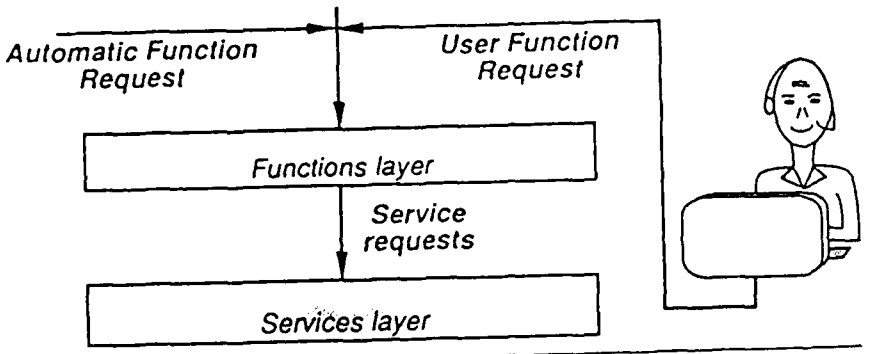


Figure 4: Services Layer

dependency trees which illustrate combinations of resources adequate to perform these services are shown in the Appendix. In support of the function report current platform position and velocity, the services *determine platform course and position* and *determine platform speed* have been defined.

Once it has been determined that all services required to implement a function can be performed, a resource manager will schedule or dedicate the resources necessary to perform the supporting services.

2.4 Applications

Applications, portrayed in Figure 5, are software processes which support services. Several applications may be required to support a single service. Readiness applications, discussed below, must be active at all times; an approach to increasing process survivability is discussed under utilities below.

Resource Readiness Interfaces comprise a class of readiness applications, one for each resource. A Resource Readiness Interface accepts the results of tests within the resource and reports them in a standard way. Ideally, those

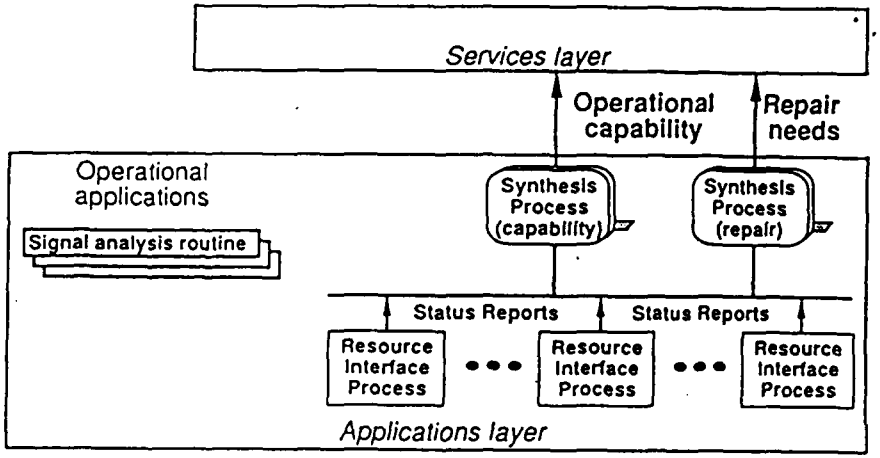


Figure 5: Applications Layer

reports will be in SRRS formats and will be immediately available to SRRS synthesis processes; practically, the evolutionary process toward Resource Readiness Interfaces and standard reports could be a long one.

There will be two SRRS synthesis processes, a *Capability Process* and a *Repair Process*. The Capability Process is a readiness application which synthesizes reports from Resource Readiness Interfaces to determine operational status. Operational status is characterized by which services and functions the system is capable of performing, separately or in combination, as determined from the relevant logical dependency trees. This information is directly available to the tactical user who may use it to establish priorities for repair or changes to the system configuration. In addition to resource status reports, inputs to the Capability Process generally include the following: (1) a correlation between services and required resources; (2) proposed changes to resources or services; and (3) the present configuration of active services.

The Capability Process supports these services: (1) "report system status;" (2) "assess impact of resource change;" and (3) "recommend resource change to restore service." For the three services, the Capability Process produces the following: for 1 and 2, reports of services which cannot be performed at all, which cannot be performed if other services are ongoing, and which cannot be initiated under the present configuration of services; and for 3, alternative modifications to resources to restore requested service. Planned, but not yet designed, are features to synthesis total ship status for reporting to higher echelons and to support maintenance scheduling based on assuring continuous availability of key operational capabilities.

Figure 6 illustrates how maintenance can be scheduled to assure that an operational capability, in this case piloting, can be kept available through scheduled maintenance when there are redundant resources.

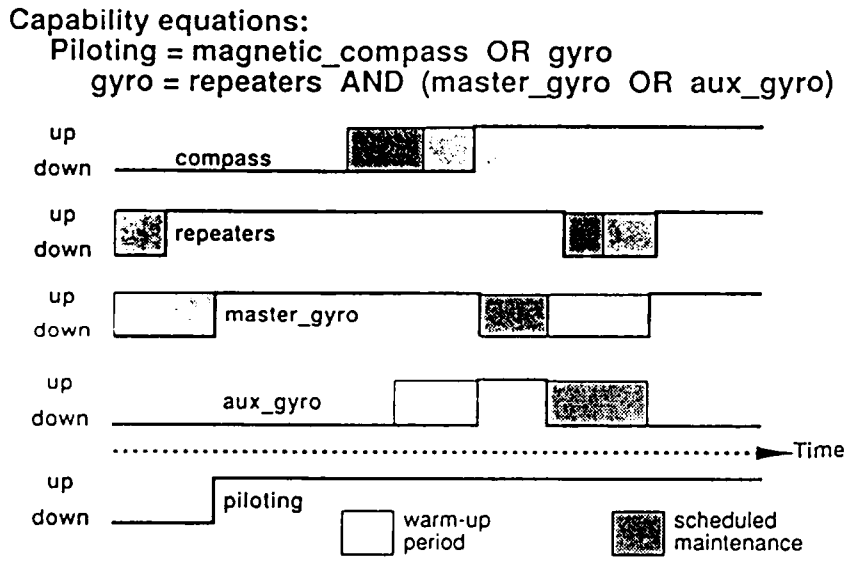


Figure 6: Maintenance Scheduling

The Repair Process is a readiness application which uses reports from the Resource Readiness Interfaces and priorities from the Capability Process to provide reports to technical users to support scheduling of repair actions. The technical user may initiate repair actions in response to faults and with respect to priority (as determined by tactical users) and expected time to repair. Some repairs may be in cabinets which are collocated and may therefore be scheduled accordingly; others may be less urgent because they are in cabinets protected by fault tolerance. The outcome of all repair actions - successful, incomplete (and expected time required to complete) or unsuccessful (and reason, e.g., requiring external support) are reported to the Repair Process. The Repair Process must accept "special conditions," such as sea state, which may influence tolerances or use.

2.5 Utilities

Utilities are general-purpose processes which support applications, as shown in Figure 7. A *Database Manager* maintains all resource status

reports. These reports are dispatched to the Capability and Resource Processes at regular frequent intervals.

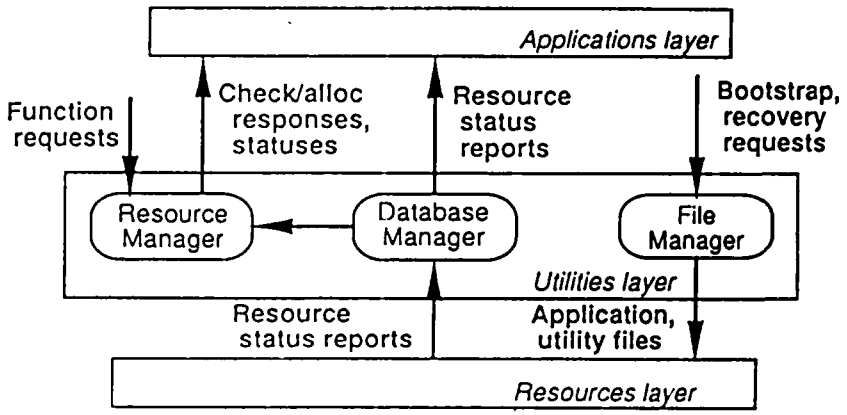


Figure 7: Utilities Layer

The *Resource Manager* is a central resource which monitors function and service execution and allocates resources. In response to a service request, the resource manager checks and reports resource availability. In response to an allocate request, the resource manager allocates resources to a service at the given (function) priority and marks any dedicated resources unavailable.

A *File Manager* distributes all applications and utilities to where they are needed. The File Manager supports process survivability, a feature required by the SRRS. The file manager load twin processes, a *primary* and a *backup*, of each utility and certain applications to assure continual operation in the event of damage.

The approach to process survivability is illustrated in Figure 8, which shows a file manager and a protected (twinned) utility. During normal operations the twins operate identically except with respect to operating messages. Normally, the backup generates an operating message each time the primary does. These messages are handled as follows: the primary sends its message, m1, to its intended destination and a duplicate message, m2, to the backup; the backup stores its message, m3, in a recovery buffer; when it receives m2, the backup deletes m3 and discards m2; if the backup receives m2 before generating m3, it saves m2 until it generates m3 (failure to generate m3 indicates an anomaly). Both twins periodically send "I am OK" reports to the File Manager. If the File Manager fails to hear from one twin in an expected time, it assumes a failure and designates the other twin as the

primary. If the new primary was originally the backup, it sends all the messages in its recovery file. Then, the File Manager installs a new backup and initializes it to the state of the new primary.

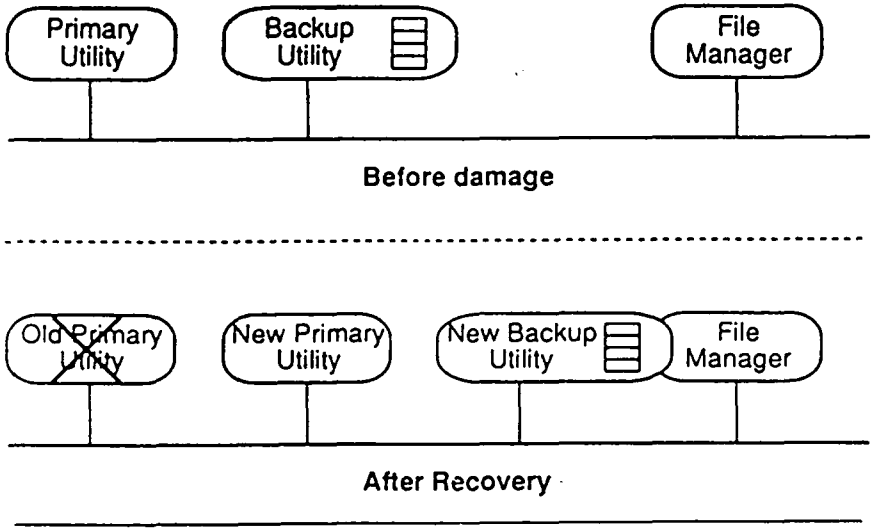


Figure 8: Survivable Processes

2.6 Operating System

Operating systems are the software which make resources usable by supporting use of the native machine language and of the network for interprocess access and communication. Several standards programs are active in operating system definition. No SRRS development is anticipated in this area.

2.7 Resources

Many devices now common in the commercial sector will be used in the construction of Navy systems. Use of these general-purpose resources will facilitate the building of systems, will simplify special-purpose resources, and will support evolution (as the commercial sector continues to provide more capable versions of these devices). Operational and readiness functions are related to general- and special purpose-resources in Figure 9.

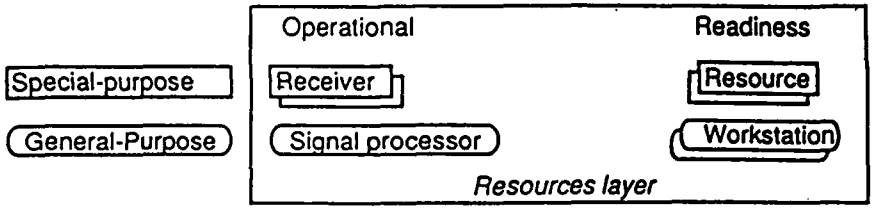


Figure 9: Resources Layer

To realize the full potential of the SRRS, resources will need to report status in standard ways in order for system capability and repair needs to be accurately reported. Software prototyping has shown that the SRRS will be most effective when resource status reports include the following:

1) Resource name;
2) *Capability - up, substantial, marginal* or *down* - including expected times to recover to higher levels (if not already *up*);
3) *Failures - none, redundant, minor, major* or *total* - including expected times to repair (if not *none*);
4) *Initialization - on, standby, energized, support* or *off* - including expected time to *on* (if not already *on*);
5) *Configuration -normal, alternate, casualty* or *unavailable* - including expected time to *normal* (if not already *normal*);
6) *Remarks*. These conclusions confirm earlier results [NAVSEA 1986].

3. SOFTWARE PROTOTYPE

The SRRS software prototype is being developed to expose and confront issues inherent in designing a distributed, survivable, hierarchical, service-based, shared-resource, multi-user, priority-driven SRRS. The software prototype is being written in MODSIM II (a product of CACI Products Company). MODSIM II is an object-oriented, discrete-event simulation language based on MODULA II, linked to a color-graphics package. The software prototype is hosted on a SPARCstation 2 (a product of Sun Microsystems).

The word *system* is used below to refer to the set of hardware resources and software processes resident on the platform. The purpose of the system is to perform functions for the user on command.

3.1 Two Stage Approach

An interactive software prototype is being built in two stages: in the first stage, the 1992 prototype [Vineberg 1992] accepted only scripted input. User interaction with the operating prototype was limited to obtaining graphical information concerning the simulation state. All events - user logon, requests for service, and damage - were specified prior to the start of a prototype session and could not be altered during that session.

In the second stage, now in progress, the user will interact with the prototype. He will be able to log on, request functions, cause damage, and read the resulting SRRS reports during the simulation.

3.2 Organization

As illustrated in Figure 10, the interactive software prototype is organized into the following parts: a *user-graphics module* is the interface between user and system; *configuration files* are user-created specifications of system parameters such as available applications, numbers of various types of resources on board, and res e characteristics; and, various *program modules* include components of the SRRS simulation.

Each program module contains two parts: a *definition part* specifies variables, procedures, and objects contained in that module and the calling conventions for the procedures and object methods; an *implementation part* contains the procedure and object method code which is executed when the respective procedures and methods are called.

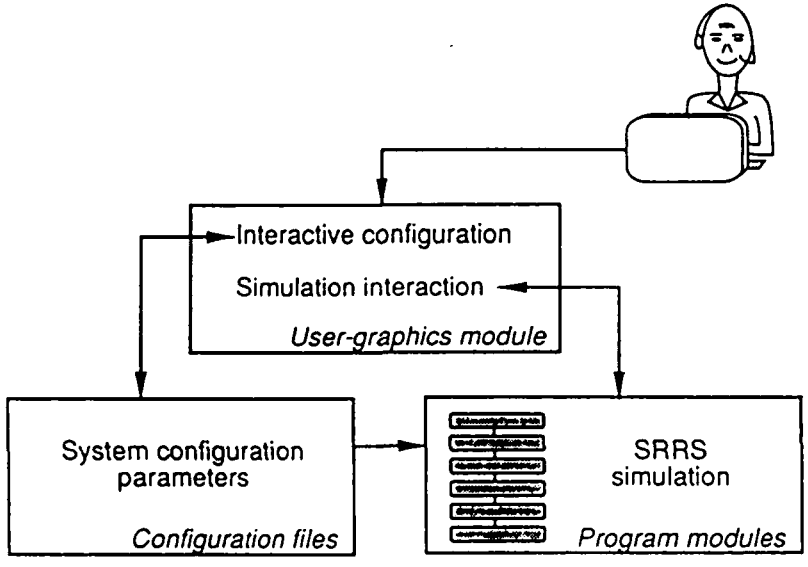


Figure 10: Software Prototype Block Diagram

3.3 Configuration Files

Users build files specifying (1) service names and the resources, applications, and utilities required to support each service, (2) function names and the services required to support each function, and (3) available resources and such resource characteristics as startup times, maintenance intervals, and fault characteristics. These files are reusable from one experiment to the next.

3.4 Operation

To operate the prototype, the user first enters a command from the host SPARCstation keyboard. The prototype is initialized by the main module from the configuration files and then started. The user is then presented with a graphical interface window showing various user operations such as

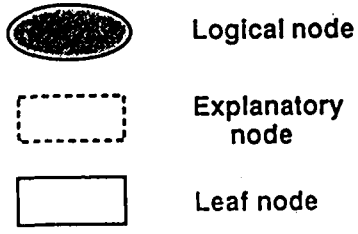


Figure A1: Symbology

bootstrap system, start system, log onto system, initiate operational functions, cause damage, etc. These options are selected, in a logical order (e.g., the user must start the system before logging on, and he must be logged on before he can initiate an operational function) from pull-down menus using a mouse. Displays of the results of operations depend on user selection.

3.5 Modules

The object-oriented program which comprises the SRRS software prototype is organized into several modules, described briefly below.

3.5.1 Common Module

Certain objects are specified in a *common module* for use throughout the program. A *message object* has a type, a priority, an origin and a destination. It also has a content queue which can contain zero or more content objects (data words). A *content object* can contain a string, a Boolean value, an integer, an object, or a combination of these. A *mailbox object* can contain a set of message objects, ordered by priority. A *subscriber object* has a name and a mailbox and can send and receive mail. A variety of types, variables, constants, and procedures are defined for common use.

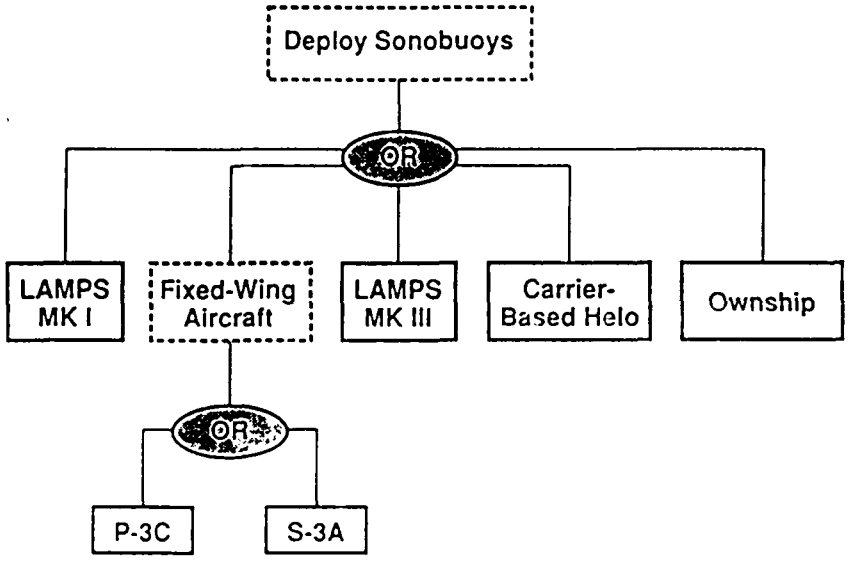


Figure A2: Deploy Sonobuoys Logical Tree

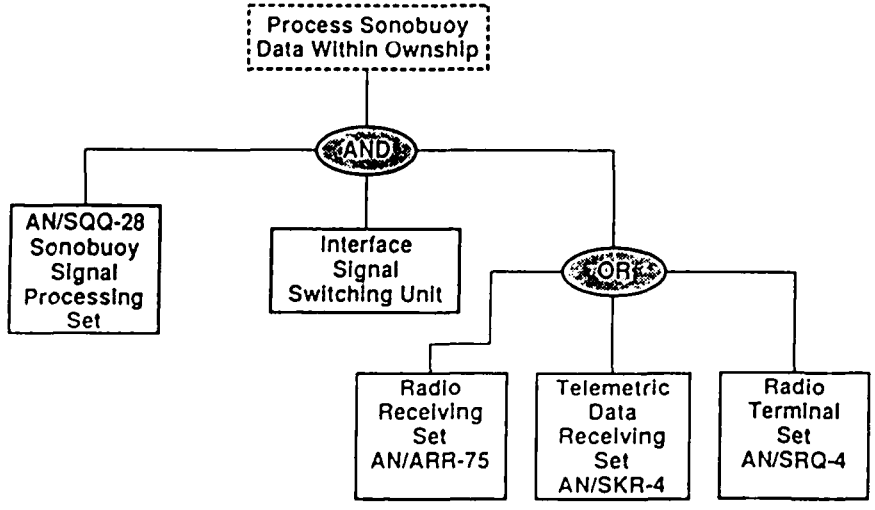


Figure A3: Process Sonobuoy Data Logical Tree

3.5.2 *User Module*

A user is a person authorized to issue commands to the system. Within the *user module*, a *user object* is defined as a subscriber object (see common module) and inherits all subscriber object attributes and methods. User attributes include a *host* (workstation), indicating if and where a user is logged on.

3.5.3 *Function Module*

Within the *function module*, an array of function records, initialized from the configuration file, shows the services that support each function. A run-time queue shows active functions ordered according to priority. A *function object*, needed for function instantiation, has attributes indicating the function type, priority, creator (a user), and a list of required services.

3.5.4 *Service Module*

Within the *service module*, a *service object* is defined. Attributes of a service object include name, parent function, parent service, child services, and element. A check field is used to indicate whether or not service can be performed. An array of service records, initialized from the configuration file, shows the applications and resources that support each function.

3.5.5 *Application Module*

Within the *application module*, a basic *application object* is defined as a subscriber object (see common module) and inherits all subscriber object attributes and methods. Key additional application object attributes include host (a resource), and a message log to be used by a backup (some applications may run as twins).

A *resource interface object* is defined as an application object. Every resource which is not a workstation has a manager (workstations have their own, more capable managers) whose primary duties are to receive mail for that resource and to report the resource status.

A combined *capability/repair* object is defined as an application object. The key attributes are status reports on all ship's resources. Key methods allow functions to be checked against available resources and both operational capabilities and resource status information to be displayed.

3.5.6 *Utility Module*

Within the *utility module*, a basic *utility object* is defined as a subscriber object (see common module) and inherits all subscriber object attributes and methods. Key additional utility object attributes include twin (all utilities run with a primary and a backup), station (a host workstation), and a message log to be used by a backup twin to recover from the loss of the primary twin.

A *file manager object* is defined as a utility object. The file manager has attributes which describe active workstations and triggers which are used during system bootstrap. The file manager controls the system bootstrap, allocating primary and backup twins of utilities and certain applications to operational workstations. It also receives "I am OK" messages from twins, and in the event of a lost twin, installs a new backup twin if a host is available.

A *dbms object* is defined as a utility object. The dbms forwards and retains the most current status report information.

A *resource manager object* is defined as a utility object. A function execution method is supported by methods to request service checks, resource allocation, service performance, and service preemption. Other methods allow individual services to be checked, resources to be allocated, and services to be preempted.

A *workstation manager object* is defined as a utility object. Every workstation has a manager whose primary duties are to receive mail for that workstation, report workstation status, log users on and off, and load software processes (applications and utilities) onto the workstation on request.

3.5.7 Resource Module

A *network interface unit (NIU) object* is defined. Attributes include name, port (an object which in turn identifies a network and a tap), a send (outgoing) message queue, and a resource.

A *resource object* is defined. Attributes include name, interface, NIU, class, fault state, initialization state, configuration, and a list of mailboxes (belonging to subscribers on the resource). The resource receives mail from the network for those subscribers.

A *workstation object* is defined as a resource object. An additional attribute is user (to show who is logged on). An additional method allows a workstation to load a file manager (in preparation for a system boot).

A *network object* is defined. Attributes include name, the first and last tap, and a list of connected NIUs. The key network method supports the transmission and reception of messages by NIUs.

3.5.8 Graphical Interface Module

A *graphic manager object* is defined as a window object (supplied by MODSIM II to manage a graphic window). Attributes include various icons, arcs, text, and menu bars. Several methods allow items to be retrieved, displayed in required configurations, and deleted.

In addition, various objects are defined to support user operations and the relevant menus and checking features associated with those operations.

4. HARDWARE PROTOTYPE

The reasons for developing a hardware prototype are to validate the concept developed in software and to expose issues related to distributed system implementation. The Survivable Adaptable Fiber Optic Embedded Network (SAFENET) is a leading candidate for a standard shipboard local area network (LAN). SAFENET uses the Fibre Distributed Data Interface (FDDI) protocol, a commercial standard. SAFENET development has attracted industry and tri-service participation, which bodes well for joint and commercial dual use.

A SAFENET testbed, currently under development at the Naval Command Control and Ocean Surveillance Center (NCCOSC), has been chosen as the environment in which to implement SRRS concepts shown to have merit in the software prototype. The focus in this stage of development, which has just begun, is to make the SRRS an integral tool in the use and trouble-shooting of the testbed.

5. SUMMARY AND PLANS

An SRRS concept and an architectural framework, consisting of an open, service-oriented architecture in support of a hierarchical organization, have been described. Work remains to be done in the definition of functions and in the decomposition of functions into services. Much work also remains in the development of standards for resource interfaces and for readiness reports to users. Once prototyping is complete, efforts are planned to transition the work to a shipboard networking program such as the (IC)2 Program.

6. ACKNOWLEDGMENTS

This work was sponsored by the Office of Naval Research under the direction of Commander David Bennett (USN) and the Logistics Block managed by Ray Brengs (NSWC). The authors take pleasure in acknowledging the extensive contributions of Stan Connors (NCCOSC, retired), Bill Nickerson (NSWC), and Don Kover (NSWC).

7. REFERENCES

[1] Copernicus Project Office, "The Copernicus Architecture, Phase I: Requirements Definition," August 1991.

[2] Defense Information Systems Agency, Center for Information Management, "Technical Reference Model for Information Management," Version 1.3, 30 September 1992.

[3] Naval Sea Systems Command, PMS-400, "Readiness Standards: Combat System Equipment Readiness Assessment and Reporting," S9410-AN-STD-010/AEGIS.

[4] Nickerson, W., "System Study of Condition-Based Maintenance

System for Shipboard High Pressure Air Compressors," DTRC-PASD.A-91, December 1990.

[5] Vineberg, M., "A Proposed Navy Battle Management Architecture," BRG C2 Research Symposium, National Defense University, Washington DC, June 1991.

[6] Vineberg, M. et al., "Shipboard Readiness Reporting System (SRRS) Software Prototype," NCCOSC Technical Document 2313, July 1992.

8. APPENDIX: SERVICE PROFILES

Prototyping has uncovered the need to represent services using logical dependency trees. In the example below, symbology is shown in Figure A1, and the two services which support the "find submarines" function are shown in A2 and A3.