Ship Self-Defense System Architecture

Larry S. Norcutt

The Ship Self-Defense System (SSDS) was devised to provide self-protection and combat system capability to non-Aegis ships of the U.S. Navy. This automated combat direction system uses many commercial hardware and software elements to achieve the first Navy distributed processing combat system, integrating already developed weapon and sensor systems. The SSDS architecture was an innovation and somewhat of a risk, but well justified in light of its successful development and the continued benefits it has shown. The SSDS Mk 1 is currently operational on 11 Navy LSDs and its bigger variant, Mk 2, is well under development for the Navy's newest aircraft carrier and ship class. SSDS architecture concepts have succeeded in advancing both the state of the art and the tactical capabilities of the U.S. Fleet.

INTRODUCTION

The primary mission of the LSD (landing ship, dock) class of Navy ships is to support amphibious assault–conveying and –landing Marine troops onto potentially hostile shores. With the increasing capabilities of the anti-ship missile threat and the likelihood of Navy combat operations in the near-shore littoral regions came the requirement to significantly improve the self-defense capabilities of this ship class. In effect, the ships needed an automated Combat Direction System (CDS), smaller in scale than that of primary combatants such as destroyers, cruisers, and carriers, but highly capable of the detect, control, and engage functions necessary for self-defense.

Limited by budget and time constraints, Navy efforts focused on the automation, optimization, and integration of existing weapons and sensors to more effectively defend the ship. The resultant automation and integration, performed by a networked set of commercial computers and operator displays, was named the Ship Self-Defense System (SSDS) Mk 1.

The new SSDS design incorporated lessons learned from over 20 years of experience in Navy tactical software and sensor integration development, and applied those lessons to the particular characteristics of combat system data and processing needs in an open-architecture distributed-processing commercial-off-the-shelf (COTS) environment. This included interfaces, processors, data distribution, computer languages, operating systems, displays, software design concepts, and sensor integration concepts. The software architecture was based on many years of experience dealing with Navy combat systems, and on ship self-defense studies performed by APL in the 1980s as part of the NATO Anti-Air Warfare (AAW) Program.

SSDS Mk 1 formed the basis of the SSDS Mk 2 system currently in development. This larger variant adds

to the baseline self-defense capabilities to encompass more of the traditional shipboard combat system functions such as air control and tactical data links. The Mk 2 has a bigger role as the tactical combat system for the Navy's newest aircraft carrier USS *Ronald Reagan* (CVN 76) and the developing landing ship, platform–class USS *San Antonio* (LPD 17). The same basic architectural concepts apply to both SSDS variants.

EVOLUTION OF COMBAT DIRECTION SYSTEM SOFTWARE

The first Navy computer programs were developed in the early 1960s primarily to support manual operator functions (e.g., track plotting and track symbol display) and to send surveillance data to other ships (Fig. 1). Computer functions provided bookkeeping and numeric calculation to assist the operator. Because of the manual intervention needed for track maintenance, however, combat system computer loading in the early days was relatively low. Tracking accuracy was also highly dependent on the interest, dexterity, and energy level of the combat system operators. Additional operator support, such as the synthetic display of information, evolved into computer-based CDSs.

Early CDS computers allowed coordination of multiple ship operations for AAW by automating ship-to-ship data transfer via the radio transmitters and receivers of the tactical digital data link, Link 11. They also provided the control of real-time data communications and formatted digital information for exchange. Owing to the reliance on manual data input, initial digital link data rates were relatively low, and ship-to-ship data accuracy and consistency were poor. Correspondingly, demands for sophisticated CDS processing were relatively low.

CDS automation needs were greatly accelerated during the late 1960s owing to more stressing tactical environments and the introduction of sensor automation using digital computers. Improved sensor performance made it necessary to account for more tracks within the ship's surveillance region. These large numbers of tracks within the CDS area of interest accentuated the need to acquire and maintain timely and accurate information on each track. Naturally, more interfaces, functions, and operator displays and controls were added to the existing computers to automatically process the information.

As more sensors and weapons were automated, additional interfaces and processing software were added to the CDSs. Building on central computer concepts of the past, where simple functions were automatically supported and easily added to the software, the growth of CDS computer processing software continued by expanding the central computer program (Fig. 2). Additional memory was added when required, and speedier mainframe processors were phased in to handle processing loads. Further evolutions partitioned functionality and processing loads into two or three mainframe processors for improved performance and functional visibility. However, while these fixes relieved individual difficulties, they resulted in new and larger problems in software development and maintenance.

CDS functionality grew, but the basic software and computer architecture did not. After years of functional growth, the large, centrally oriented programs became very complex and functionally interconnected, as illustrated in Fig. 3. After years of maintenance, functional tweaking, and special fixes arranged among programmers in different areas, software became large and complicated. Software maintenance itself evolved into a specialized art. In such environments it is very difficult

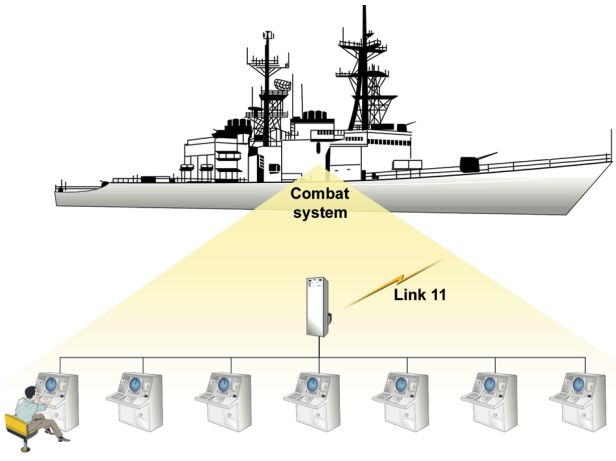

Figure 1. Early computer use in Combat Information Centers involved a central computer performing calculation functions involved in manual tracking and providing operator displays. The computer also formatted tracking data for communication to other ships.

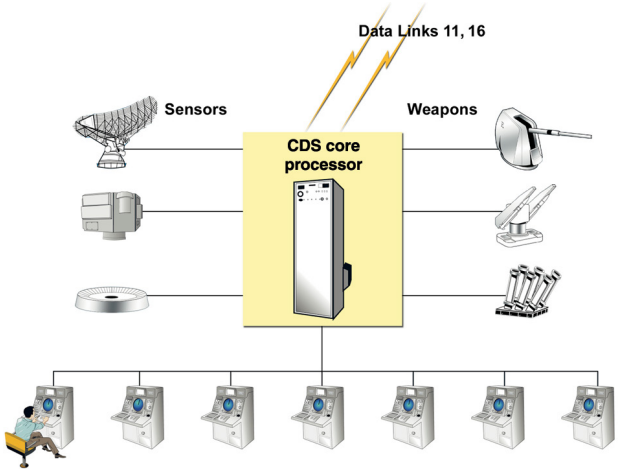

Figure 2. The CDS central computer architecture featured one or more computers that controlled data communications and began to provide automatic sensor processing and display. The hardware architecture remained simple; peripherals were attached to a core processor.

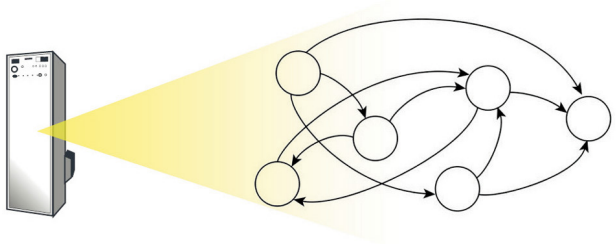

Figure 3. Many unique logical and data transfer "interfaces" among functions within the central processor lead to complex interactions that are difficult and expensive to expand, modify, or maintain. The central processor is easily CPU- and I/O-bound, particularly with the high-order language requirement and increasing numbers of sensors and weapons. This organization is not well matched to the characteristics of combat system data flow or processing.

to understand the whole program organization and provide changes or improvements in performance without incurring unpredicted degradations in other program functions. The large programs became subject to error due to numerous users of common databases and the difficulty of troubleshooting the complex functional interaction, data sharing, and program errors in unrelated functional code. Software maintenance and configuration management were further affected by the program size and logical complexities that exacerbated the training of new software support personnel and often produced unexpected consequences of code alteration.

CDS software development and maintenance also became severely hampered by the use of militarized, nonstandard equipment (computers, displays, I/O devices) and software language. Not only were the military computers relatively expensive, but they had limited availability and were relatively inflexible. This subsequently limited personnel productivity and increased both development and maintenance costs.

Large central processor programs became easily saturated in processing time demands. Logical processes had to be more sophisticated to form proper automatic evaluations and responses to a larger, more complicated set of information than ever before. The growing tactical environment encompassed larger ranges and required larger track capacities. Processing loads, from both logical complexity and data volume aspects, quickly exceeded the capabilities of individual processors. In short, the software/computer environment of shipboard CDSs (including sensors, weapons, command support, and communications) became a very complex problem.

Such was the typical CDS environment in 1991, the beginning of SSDS Mk 1.

SSDS EVOLUTION: THE SSDS OPPORTUNITY

From a computer system and software architecture perspective, the SSDS had the good fortune of an absent past. Since the ship for which it was developed had no prior computerized CDS and no major combatant air warfare mission, there was little justification for installing existing CDS components and then tailoring them to the LSD self-defense role. It was easier to provide new automation for this well-defined need. The SSDS computer, software, display, and data distribution architecture could use new concepts of software engineering and computer science that were not available in the early days of the CDS. Furthermore, the SSDS design could apply lessons learned from past CDS experiences.

The opportunity to develop a new combat system was offset somewhat by a lack of guiding specifications for system functions, computer architecture, and software architecture. As described in the next article by Thomas et al., the system's tactical functional requirements were defined by flowing down top-level operational requirements. This is a relatively straightforward process. The requirements for the supporting computer system and software architecture, however, were not so well defined. Typically, they had to be derived from the tactical functional requirements to extract data processing performance requirements implicit in the tactical needs and the nature of the data available to the combat system. For the SSDS, data processing performance requirements were derived from detect/control/engage reaction time requirements, from the volume of tracks expected in the ship's surveillance region, and from the observed and predicted data rates of the ship's sensors (the most demanding of the data providers).

Additional guidance came from higher-level requirements of a programmatic and historic nature. SSDS software and computer architecture designers recognized this ill-defined but critical feature of combat system development. The following general requirements contributed to the SSDS architecture. They are almost nonquantifiable and largely historic—reflecting 20 years' observance of Navy tactical software—but significant nonetheless.

One of the main contributors to the SSDS architecture was the desire to simplify the functional relationships among the software so as to logically and perhaps physically decouple the complex interactions seen in much of the historic tactical software. In addition, the Navy tactical processing architecture had to achieve cost and performance benefits derived from the quickly improving processing and data transfer performance offered by COTS products as well as improvements in computer languages; however, the architecture had to be easily adapted to the brief life of each new product.

Tactical software architectures must address the nature of computer processing and the system interfaces of the combat system. Tactical data processing is time-critical and must address large volumes of data from ownship sensors and offboard platforms. The features found in typical commercial operating systems such

as UNIX do not support the critical response times and predictable performance needs of tactical processing. Tactical processing must occur in real time at high volume and low latency. It must also address the "…ility" requirements that greatly affect system cost, development ease, and general long-term quality; e.g.,

Maintainability: Can software fixes and corrections be made easily?

Extensibility: Does the software architecture easily support growth in functionality, processors, interfaces, and languages?

Understandability/visibility/comprehensibility: Is the software system conceptually simple, or are its operations complex and obtuse?

Reliability: Can the software system give predictable performance?

Testability: Can the software be easily tested, and are there convenient measurement points?

The software must have a robust design to accommodate the unpredictable nature of tactical processing loads and potential equipment faults; function in a less-than-complete processing environment; not be susceptible to critical single points of failure; and be loosely coupled so that functions can be removed or added easily. Also, because of the frequent revision and improvement of COTS products, the software and processing elements must be easily updated, and a change to one must not generally affect the other.

These requirements, together with lessons learned from past CDS efforts and APL's sensor integration experience, resulted in an SSDS design that was new conceptually, physically, and functionally.

SSDS ARCHITECTURE DESCRIPTION

Architecture Concepts

Information-Oriented Design Concept

The SSDS software architecture incorporated the information-oriented design (IOD) concept that evolved from APL's NATO AAW studies. This software design concept was specifically applicable to a distributed combat system environment, satisfied all the "ility" requirements, and provided an "open," loosely coupled, logical processing environment. Rather than focus on point-to-point functional and physical interfaces, the IOD concept recognized the continuous and concurrent nature of combat system data processing, and concentrated on the information that was being produced and the responsibility for its creation. As illustrated in Fig. 4, IOD combat system functions do not interact with each other, but rather with the flow of combat system information.

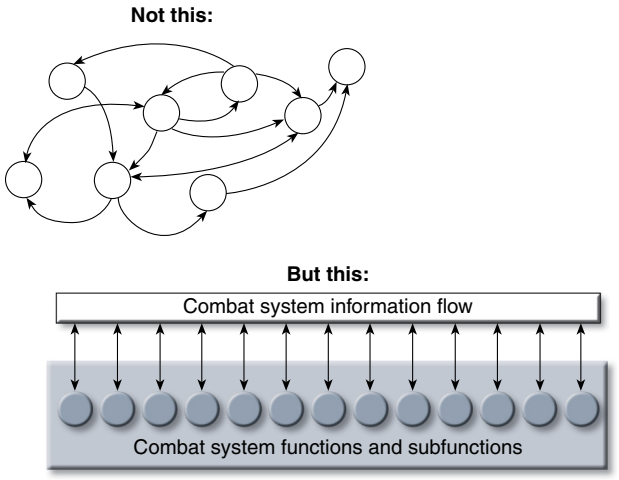

Figure 4. The information-oriented software design concept decouples complex, highly interrelated functions and instead features independent functions with access to a common information source.

Functions are simply assigned unique responsibilities to produce unique system information, which is broadcast as messages throughout the system. Each function is offered access to the system information flow from which it will perform its own duties. For robust design, each function must deal with potentially incomplete information and must recognize system information events that are particularly sensitive or pivotal to the function's purpose. Collectively, this robust information focus provides functional independence and a loose coupling of the software processes.

In general, information is only generated upon change. This serves two purposes: (1) the general revision and minimal data loading of information potentially used by other system functions (i.e., database update), and (2) the possible triggering of other function processes that are keyed to changes in particular information.

From a software system design/development point of view, the functional independence of the IOD removes the logical complications of sequential function coordination and communication. From an integration/testing point of view, if a function is not available, its only impact is the lack of its particular information in the system-wide database. Other functions must continue to operate to their best capacity on less information. Testing and development of individual functions may be performed in isolation, stimulated only by a controlled message flow and evaluated solely on their ability to generate their assigned information.

Given this form of functional independence, it is easy to develop physical independence in the form of distributed processor architectures. This allows massive computing power to be focused on particularly important functions, providing growth and change that are completely independent of other system processes.

The addition of a new system function in such an environment requires only the definition of the information (if any) it will add to the system information. Existing functions need to be changed only if the additional information is desired to improve the quality of their current output. By definition, the new function has access to all system information and will be informed upon update of any portion. In this environment there is total independence of functional definition and implementation. New processes may be added to the system to analyze or extract system information on a completely independent basis. Information display functions may similarly be added with total independence.

Information Attribute, Message, and Distribution Concepts

Following the IOD theme, SSDS concepts describe the system in terms of information "entities" having numerous "attributes" and relations among attributes. Furthermore, the data are broadcast upon change to all interested functions within the SSDS to use as each sees fit. In a sense, this is a form of object orientation where the objects are active rather than passive.

It should be noted that the combat system itself may be considered an entity described by status attributes. Also, surveillance tracks may be thought of as entities, having attributes of position, velocity, identification, engagement status, etc. Collectively, the attribute data define the state of the combat system. Its functions contribute to that state and react to changes in it, as illustrated in Fig. 5.

Additional features of these concepts are as follows:

- **The responsibility for data entities and attributes is assigned to particular and unique functions within the SSDS.** This method of segmentation allows a clear separation of the contributions from each SSDS element and function. System information has a distinct source, directly related to a specific portion of software and traceable to specific message output to the data distribution architecture.
- **System messages containing the entities and attributes are defined and broadcast.** Messages are more oriented to attribute assignment than to total entity description. The sum of system messages constitutes the total of the system information, which all the SSDS functions use as needed and contribute as assigned.
- **Messages are filtered upon receipt.** Upon initialization, each principal function within the SSDS registers its message needs with data distribution communications packages ("infrastructure" software). Using broadcast and multicast techniques, the communications packages transfer all function message outputs to all other required destinations and filter input messages to accept only those desired.
- **Each function uses system data and records the data as needed locally. There is no central data manager.** Each function outputs its contribution to the system in the form of (nominally) broadcast messages. Data are nominally sent only upon change. Aside from means to initialize functions that gain access to the distributed system after steady-state operations are achieved and for system reconfiguration after casualty, data transfer is to be minimized to operational need. Data transfer is used as much for change notification as for stimulation of other functions.
- **Functions do not use the system messages as a coordination device.** This incurs functional coupling, which is to be avoided in functional design.
- **Most data transfers are expected to be broadcast "send and forget."** This minimizes overhead in message acknowledgment and logical dependence. Special cases may require acknowledgment for safety or critical events, but these are expected to be relatively infrequent within the total data distribution environment.

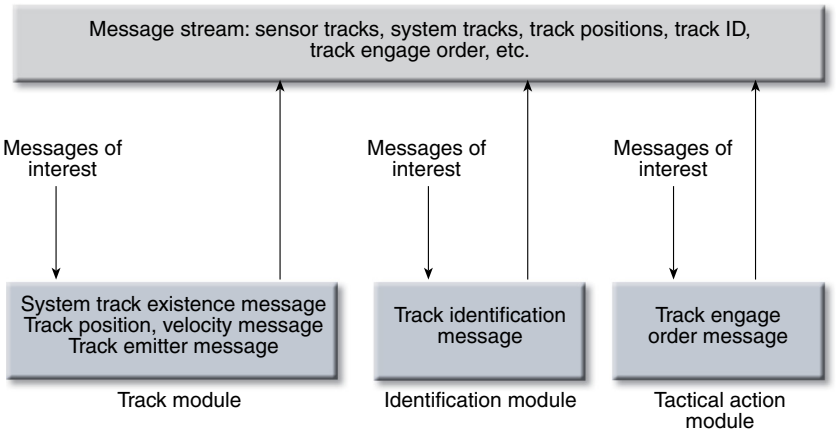

Figure 5. The SSDS functional examples shown are assigned entity and attribute responsibilities; their visible and physical mechanism of message generation and receipt allows simple, manageable function (process or processor) distribution. Processes may be as elaborate as necessary, but also may be totally independent of all other processes.

SSDS functions attached to the distributed architecture impact the system only through their message contribution to entity and attribute definition. Processing techniques, languages, and data structures within a function are independent of the remainder of the SSDS. Response to message receipt is totally the responsibility of the receiving function. Messages may be used to update a function's copy of track data

(organized optimally locally for the function's needs) or may be used as triggers for functional response.

The flexibility of the concept allows SSDS processes to be totally distributed within the system as long as an underlying communication system is available. To accommodate survivability requirements, duplicate processes are provided within separate enclosures to produce a passive monitor of data flow and to provide backup in the event of casualty to active processes.

Distributed Sensor Integration Concept

Another principal element of the SSDS conceptual design is the partitioning of tactical functions that collectively form and maintain the system's surveillance track information. The concept recognizes the continuity and accuracy benefit of frequent and complementary sensor detections; the ability of individual sensors to optimize their individual performance; the benefit of central track organization and flexible data access; the importance of *a priori* information; and the power of processing distribution. In the SSDS IOD concept, individual sensors take what information they need and do their best to contribute their piece (Fig. 6). The SSDS provides central management, feedback, and false system track control.

Robust Common Time and Time-Tagged Data Concepts

The use of time is critical to the architecture and technical processing performance of a distributed system. Although computers are very fast, the reliance on performing a process quickly so that the results have an epoch of "now" is very risky, subject to data inaccuracies, and intolerant of the randomness of processor loading and external events. To maintain accuracy in data calculations, the data must be time-tagged when it is measured or created. This "valid time" then maintains the epoch integrity of the measurement, which allows later processing with no loss in calculation accuracy.

Another element of time and data is latency, which may be thought of as the delay from data measurement to the subsequent processing of the data. In its most general terms, latency is the time between any two events of interest. It is critical in particular situations such as threat detection as well as in response and tracking loops. Latency within 10% of measurement periods is generally necessary for reasonable tracking maintenance.

A robust common time is a particular requirement of the SSDS. In the interest of system independence and the avoidance of single points of failure, the SSDS establishes a time base within its own collection of distributed processors, initialized and maintained over the network connections. In keeping with this system independence, the first processing node to be powered distributes its internal clock for all successors to receive and accept as the time base. (Each node listens briefly before it assumes it is the first.) After a node receives the initial clock value, subsequent tuning of the networked time is maintained through the functions of NTP (Network Time Protocol) implemented in the infrastructure software. In this manner, there is no requirement on the order of start-up of SSDS components.

Another important function and element of the SSDS distributed system concept is its feature of broadcasting system track information in a minimum period. Normally the frequency of sensor updates causes system track information to be broadcast within the minimum period (about 5 s), but a background process ensures the minimum for all tracks within the system. This feature allows each function within the SSDS to start up at any time and absorb the track picture within a few seconds.

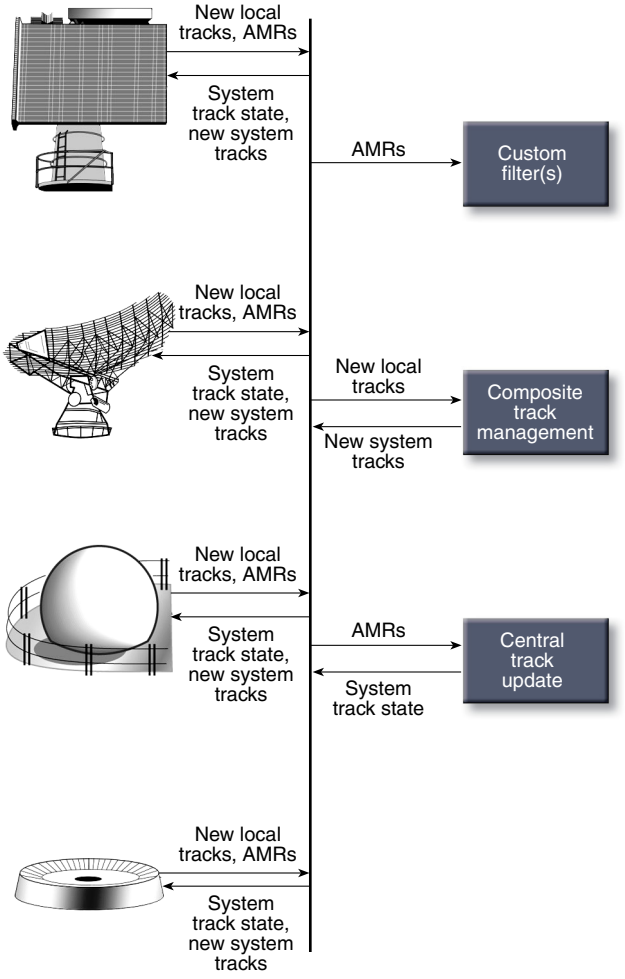

Figure 6. Conceptually, the SSDS provides each sensor with current system track information. If the sensor can associate a sensor observation with the track, it provides the system with the measurement data, forming an Associated Measurement Report (AMR). A central SSDS function uses this measurement to calculate an updated estimate of system track position and velocity and broadcasts this updated attribute information throughout the system. Special functions, such as custom filters, may selectively use measured data in the AMRs for particular needs.

To maintain the robust nature of the functional and physical architecture and to accommodate independent start-up of processors, each application processor is provided with common code to receive system track data messages over the network and populate its own local database. This code can be tailored to retain only the information of interest to the local user. An important feature of this common code is its logic to "age" each track and delete it from the user's local database if no updates have been received within a reasonable period of time (longer than the nominal system update period). This process, when coupled with the nominal system track broadcast feature, ensures that the user's local information is consistent with the remainder of the combat system.

The SSDS data broadcast feature also complements the SSDS concept of distributed, independent operator display support. This concept uses CDK (Common Display Kernel) software within each console to render the majority of operator displays. By operating totally from broadcast system messages, multiple displays can be added to the combat system with virtually no network load. Unique displays and controls are appended to the CDK software as specific application code to support particular user needs, with portions replicated in different consoles as equipment casualty backup. A successive variant of the concept, incorporated in SSDS Mk 2, retains a common display core in each display, efficiently supplied from the network broadcast. The low-demand, tailored operator controls and displays are implemented in CORBA and a server over the network. This hybrid display implementation provides console and operator mode independence while matching efficient processing and data loads with flexibility of software design and implementation.

Physical Architecture

CDS history, derived and implicit requirements, NATO AAW studies, the rapidly emerging COTS computer and networking environment, and IOD software architecture concepts all contributed to the design of the SSDS physical architecture. Being the ship's only combat system and having a critical self-defense role, the SSDS must be dependable, robust, and able to survive at least limited battle damage. Its processors have to handle full track and sensor data loading and must be able to adapt.

The SSDS physical architecture, shown in Fig. 7, consists of a local area network (LAN) connection of clustered VME-based single-board computers and interface cards that form LAN access units (LAUs) for the various functional elements of the SSDS.

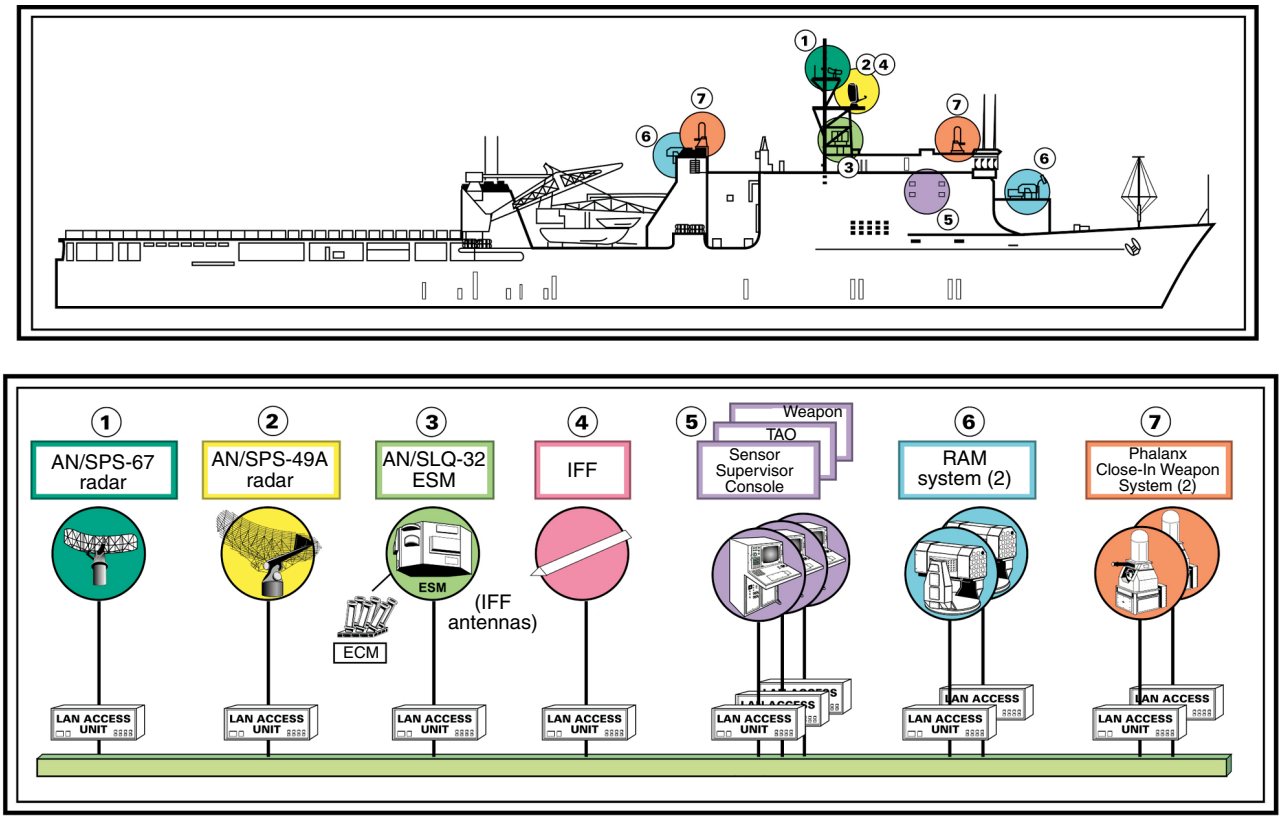

Figure 7. The LSD 41/49 combat system consists of sensor systems, SSDS components, and weapon systems, all connected to the fiber-optic LAN via similar LAUs.

LAUs typically contain single-board computers and interface cards to provide tactical processing and to communicate with specific shipboard systems such as sensors, weapons, data links, and operator displays. Because of the broadcast nature of SSDS system data, computer programs that do not service a particular physical external interface may execute in any processor, any LAU. Those functions that handle interfaces and integrate with particular external systems are placed in the same LAU and VME backplane as the corresponding physical interface boards. Figure 8 illustrates a generic LAU configuration, which contains a LAN interface, numerous general-purpose single-board application computers, and an external interface. LAUs, connected via the LAN, may be placed anywhere on the ship. SSDS message infrastructure software operates within the LAN interface and within each general-purpose application computer to provide message distribution, board start-up, and common time synchronization.

SSDS single-board computers are general-purpose, COTS, and non-proprietary to capitalize on cost, availability, language supportability, growth, and simplicity of software programming constructs. Multiple individual processors allow each principal tactical function to have its own computer, with the intent to operate at minimal CPU loading. This simplified distributed processing environment is a particularly effective means of avoiding resource contention problems among multiple programs sharing the same CPU, adding functionality, and allowing more predictable software processing performance. It also more easily accommodates the processing of data bursts that are typical of the tactical data flow environment.

Figure 8. The LAU concept provides both physical independence as well as functional and physical adaptation to particular interface and integration needs. While operating within the overall SSDS integration and broadcast information-oriented architecture, LAU components can support processors, languages, functions, and physical interfaces that are unique to a particular interface. This prototype LAU consists of a commercial VME card cage populated with LAN interfaces, device interfaces, application single-board computers, and tape drives.

Requirements for system survivability, ease of growth, and flexibility led to the networking of SSDS components. Network architectures, supported by data broadcast capabilities, also formed a physical complement to the IOD concepts of concurrent processing and universal access/contribution to system data. Further supported by general software infrastructure that facilitated message distribution, the SSDS architecture acquired both the physical and functional open-access nature intended for the IOD. The physical distribution of SSDS components, combined with redundancies of the chosen network, also provided a degree of system survivability in the event of battle damage.

Common network middle-layer protocols of the IP (Internet Protocol) family proved more than adequate for efficient data transfer and were universally available for software development. The ease, efficiency, and independence of the UDP (User Datagram Protocol) broadcast and multicast protocols were used to convey the bulk of SSDS network data, such as sensor track updates. In a few critical cases, data transfer acknowledgment was implemented to help ensure data receipt. In general, minimal network loading was desired to ensure the reliability and accessibility of network communications.

Despite the desire to use a minimal amount of the network data bandwidth, it is interesting to note that the Ethernet CSMA/CD (carrier-sense, multiple access/collision detect) physical layer protocol was not thought at the time to be predictable enough for use as the SSDS network backbone. The combination of numerous, frequent contributors of data (e.g., the numerous distributed processors and the characteristics of tactical data within the combat system) and the random, progressively longer back-off and retry characteristic of Ethernet could disastrously delay critical SSDS data transfers. For its automatic rerouting features, more predictable physical layer token "ring" protocol, 100-Mbit performance, and commercial software support, the FDDI (Fiber-Distributed Data Interface) was chosen as the physical network. The glass fiber for data transfer was selected for its performance, low weight (Fig. 9), and low electromagnetic susceptibility.

The SSDS network uses a dual home star topology incorporating network hubs that are positioned in different regions of the ship. Network star topology, illustrated in Fig. 10, uses a hub to connect to each network node. This provides ease of troubleshooting, simplified COTS growth, and ease of reconfiguration. In the case of FDDI, the hub avoided the problems of optical bypass relays when reconfiguring around breaks or inactive nodes in the FDDI token ring architecture. (Conversion of the network to another technology such as ATM or high-bandwidth Ethernet would use the same topology but different hub equipment and interface cards at the network nodes.) Dual hubs provide

Figure 9. The 144 copper cables for CDS parallel channels on the left are compared with a single fiber-optic cable containing 144 individual glass fibers. The change in physical media, resulting in a potential cable plant weight reduction from 35.3 to 0.05 lb/ft, was only part of the technology challenge. To exploit fiber optics, an evolution in equipment interfaces was also required.

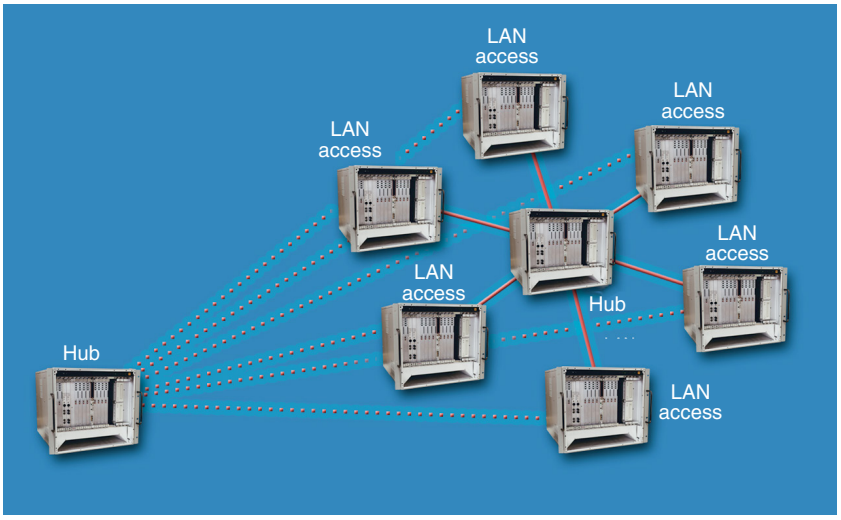

Figure 10. SSDS use of a double star topology provides significant network reliability and battle damage survivability by automatically reconfiguring data flow in the event of damage to any node interconnection or to the alternate hub.

battle damage resistance and automatic communications redundancy for system reliability.

Common Infrastructure Concept

Common infrastructure software (middleware) developed by APL provided message distribution, time synchronization, and processor start-up coordination services, in addition to offering common API (Applications Programmer Interface) functions that facilitated development of tactical code within the multiple processors. This software, like the concepts of sensor integration, evolved in performance and maturity over many years of use in the NATO AAW experiments, Cooperative Engagement Capability (CEC) development, and SSDS development. Synergism was achieved by combining the network concepts of the SSDS with the VME backplane message distribution features of the CEC (Fig. 11).

COTS Refresh Concept

Recognizing the rapidly changing (and the generally performance-improving) nature of commercial products, SSDS design and implementation focus on the use of the most common commercial hardware, software, and network standards; the avoidance of proprietary products; and the adaptability of its interfaces. Although early SSDS development followed the standard DoD language of Ada, the subsequent COTS commonality of the C and C++ languages and their familiarity to software developers have since led to their use in the SSDS. The CORBA language is also employed in lightly loaded display interface software owing to its flexibility and productivity.

The SSDS LAN design, through its fiber star topology and its reliance on the common IP middle-layer protocols, allows flexibility in the choice of the underlying physical implementation. Because SSDS tactical software uses the Transmission Control Protocol (TCP)/IP and UDP communication software commonly provided by network interface vendors, minimal changes are required to accommodate different physical network implementations.

Because the commercial market normally provides upgraded common language (such as C) compilers to complement upgraded processor boards, the impact of refresh on SSDS tactical code in such events is typically minimal. Since its inception, the SSDS has experienced COTS refresh twice, proceeding from Motorola M68020 processors to the M68040, and now the Power PC.

COTS refresh has minimal effect on SSDS software and no effect on the conceptual architecture. The flexibility of the network and LAU structure allows adaptation to handle exceptions to processors and languages in each node if required.

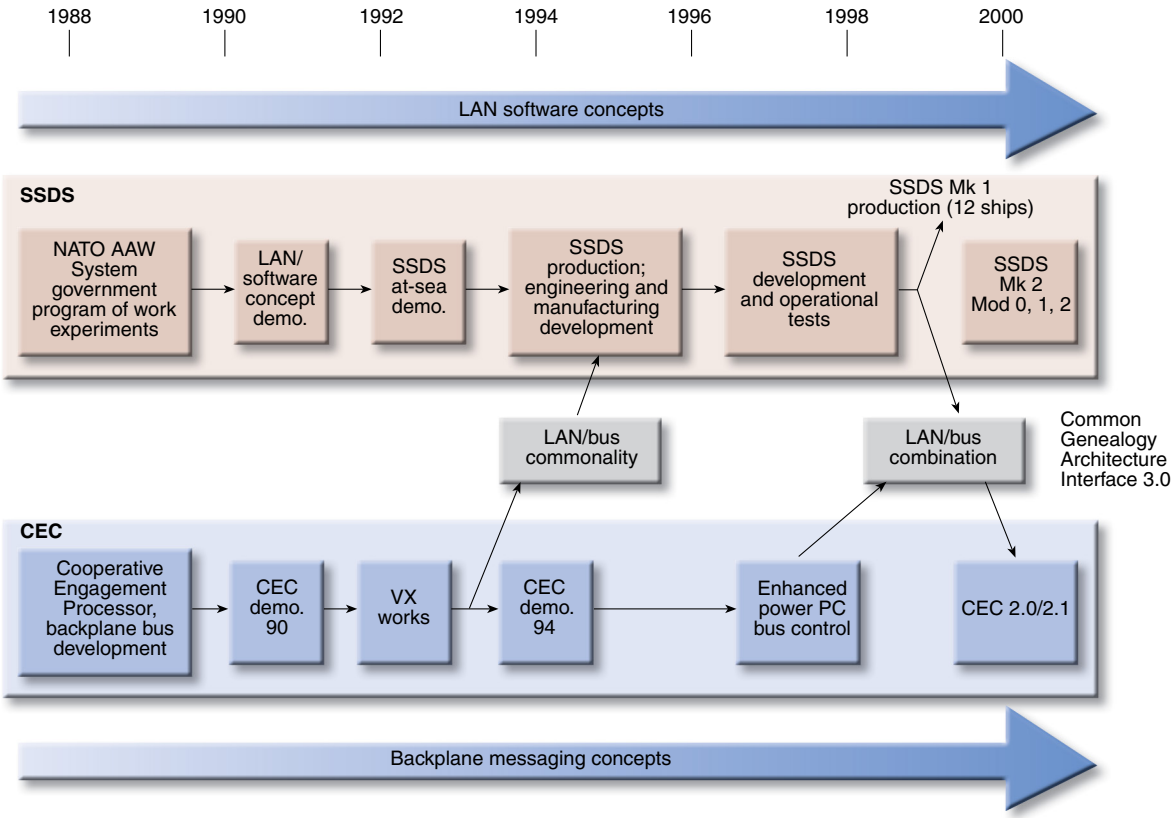

Figure 11. Common infrastructure software synergistically evolved over multiple programs, benefiting performance and development productivity.

Functional Architecture

In classic Navy combat system terms, the SSDS performs the functions of detect, control, and engage. In true IOD context, these may be shown as any combination of functional bubbles in a "flat" representation (Fig. 4) and could be implemented at various places throughout the network. A functional flow representation, shown in the IOD context, is illustrated in Fig. 12. This orients the functions in a left-to-right manner, illustrating the detect-to-engage sequence of operations that may occur. Tapping onto the stream of system information, and shown above the network flow, are the display functions of the SSDS Mk 1 Sensor Supervisor, Weapons Supervisor, and

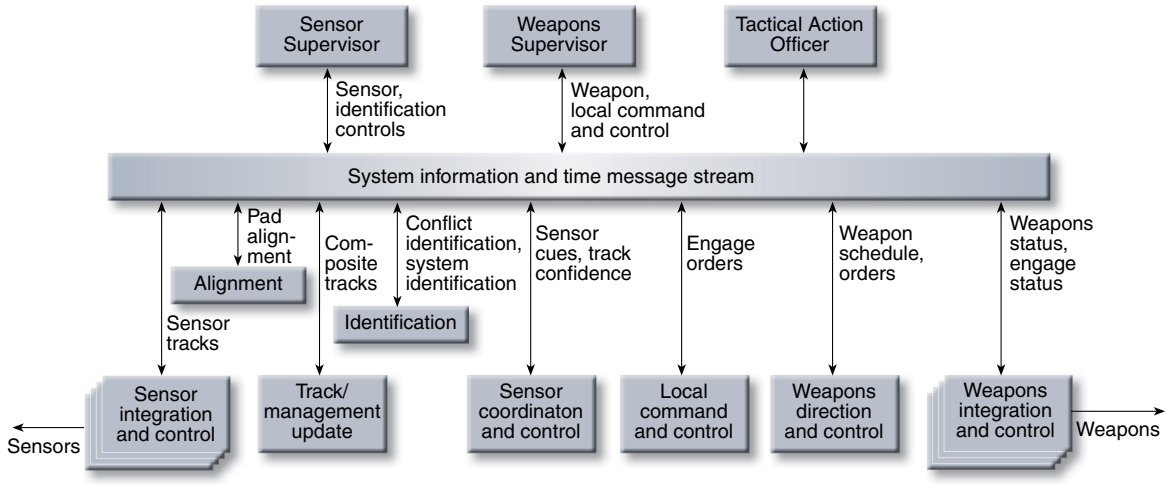

Figure 12. SSDS software functional architecture (sensor focus example). SSDS functions that collectively provide the detect, control, and engage combat system functionality are dispersed throughout the network, providing ease of extension, adaptation, and growth.

Tactical Action Officer. The display functions absorb whatever information is needed for display and provide operator actions back to the network to be interpreted by the interested functions.

SSDS DEVELOPMENT HISTORY

Prototype Demonstration Phase

The SSDS began as a proof-of-concept demonstration for the Quick Response Combat Capability Program in 1991. Incorporating some APL software reuse from the Navy Auto-ID and CEC programs and experimental network IOD demonstration software from the NATO AAW effort, the new SSDS distributed architecture was formed. Under Navy direction, APL provided lead system architecture design and developed the sensor integration, displays, and infrastructure software. Complementing the team were the Naval Surface Warfare Center and Hughes Aircraft (now Raytheon), who developed weapons scheduling and control and weapon interface software, respectively. This trio of software developers, building code at three independent sites, provided an early "ility" test of the SSDS architecture and integration concepts that proved quite successful through careful management of system message definition.

In June 1993, following land-based testing and shipboard installation and integration aboard USS *Whidbey Island* (LSD 41), SSDS—automatically integrating seven shipboard sensors and three weapon systems—performed a successful, near-simultaneous, fully automatic and coordinated detect-to-fire live engagement of two target "threats" (a towed decoy unit and a remotely piloted jet drone) using the Rolling Airframe Missile and Phalanx gun system.

Production Phase

After the successful demonstration efforts, the SSDS Mk 1 software and COTS components were ruggedized for shipboard operational use by Hughes Aircraft. Production-quality LAU equipment and multiple LAU enclosures were developed to house the COTS processors, and additional integration software was developed by the Laboratory to incorporate the AN/SPS-67 surface search radar and the AN/UPX-36 Identification, Friend or Foe (IFF) sensors. The first production SSDS Mk 1 was developed for USS *Ashland* (LSD 48), on which the system passed formal Navy operational testing and evaluation in the summer of 1997 in its first attempt.

The low cost and short schedule of SSDS Mk 1 development earned it the U.S. government's Hammer Award for efficiency of government procurement.

Current Status

The SSDS Mk 1 is now installed and operational on 11 of the 12 ships in the LSD 41/49 class, with the 12th nearing completion. The SSDS Mk 1 successor, SSDS Mk 2, is currently being developed for two more ships and classes, USS *Ronald Reagan* and USS *San Antonio*, where the system will provide full combat system functionality. In the Mk 2 configuration, much of the surveillance sensor integration is provided by the installed CEC, which incorporates the same shipboard sensor integration concept as the SSDS.

SUMMARY

The SSDS architecture was an innovation and somewhat of a risk. But the risk was well justified for the success of its development and the continued benefits the architecture has shown. SSDS architecture concepts have succeeded in advancing both the state of the art and the tactical capabilities of the U.S. Fleet.

SSDS Mk 1 development and capabilities are a success story, due largely to the contributions of many talented system and software engineers, the experience base of those people, new software design paradigms, and the effective use of COTS software and computer components.

THE AUTHOR

LARRY S. NORCUTT is a member of the APL Principal Professional Staff. He received a B.S.E.E. from Michigan State University in 1969 and an M.S.E.E. from The Johns Hopkins University in 1972. He joined APL in 1969 and has an extensive background in the integration and automation of Navy surveillance systems in the real-time combat system environment. He was a principal design and software engineer on the development of the AN/SYS-1 and AN/SYS-2 integrated automatic detection and tracking systems, led the development and integration of surveillance software for the Navy's ACDS Block 0, and was APL combat system architect lead for the NATO AAW Program. Recent efforts have included concepts for integration of passive sensors and improving Navy Surface Fleet interoperability and combat system training. Mr. Norcutt was the lead engineer for the SSDS Mk 1 system architecture design. His e-mail address is larry.norcutt@jhuapl.edu.