

Challenges in Engineering Distributed Shipboard Control Systems

Lonnie R. Welch¹ and Binoy Ravindran

Computer Science and Engineering Dept.

The University of Texas at Arlington

Box 19015

Arlington, TX 76019-0015

welch@cse.uta.edu

Robert D. Harrison, Leslie Madden, Michael W. Masters and Wayne Mills

Code B35

Naval Surface Warfare Center

Dahlgren, VA 22448

1. Introduction

Although the end of the Cold War minimized the threat of massive open ocean naval combat, new littoral warfare missions and scenarios involving both combined arms and joint allied operations, have emerged. New threats, missions and operating environments present a greater challenge to target identification, reaction time, command and control, and tracking and weapons accuracy, requiring even more computer automation and data processing than in the past. As new systems designed to meet these challenges enter fleet service, it is increasingly difficult to engineer cost effective computing systems with traditional military standard computing resources. The expected growth in required computing capacity cannot be satisfied by scaling the current military standard computer system resources. The basic problem is that the current Aegis computer suite has run out of capacity, and the architecture, based on Navy standard computers and architecturally limiting point-to-point interfaces, lacks the scalability needed to add the required new capacity. A longer term, cost effective solution, based on full use of commercial computing resources and capable of being expanded over a distributed pool of resources as requirements grow, is needed.

In response to the need to develop high capacity, scalable computer systems for shipboard use, a program called the High Performance Distributed Computing Program (HiPer-D), was created. HiPer-D is intended to provide the technical design concepts and engineering data needed to enable the Navy to capitalize on commercial computing products. The program, conducted jointly by the Defense Advanced Research Projects Agency (DARPA) and the Aegis Shipbuilding Program, consists of simultaneous top down engineering studies and large-scale critical experiments using new computer technology.

¹ Supported in part by the Naval Surface Warfare Center.

HiPer-D is intended to validate distributed processing concepts not only for future Aegis ships but also for a new class of surface combatant ships, presently projected to begin construction in the year 2003. This ship, currently referred to as the Surface Combatant - 21st Century, or simply as SC-21, is intended to be a substantial departure from previous ships, and even future Aegis ships, with regard to computer technology. A key anticipated future direction is expansion of the role of computing to encompass not only combat system functionality but also ship systems and perhaps administrative tasks as well. Eventually, the ship's computing resources may be viewed in the same way that traditional hull, mechanical and electrical resources are viewed today, i.e., as part of the ship's infrastructure or "hotel services." Computing resources such as these would provide a genuinely new option for ship automation, bringing to reality the idea of "ubiquitous computing," that is, the ability to compute wherever and whenever needed to meet the threat.

The aim of the HiPer-D program, and the entire Aegis advanced baseline engineering process, in supporting the Aegis transition from military standard computing to COTS-based (COTS is an acronym meaning commercial off-the-shelf) computing has been to mitigate the risks associated with the major changes that will be required. To that end, a substantial portion of the HiPer-D investment focused on examining the available computing technologies and devising an evolutionary approach that would permit controlled and managed change to the combat system to occur without introducing undue cost and schedule risk into the baseline engineering process. The result of this study is to adopt a strategy of evolutionary transition toward a very advanced, fully distributed architecture that might supply the entire shipboard computing needs of a future surface combatant scheduled for construction beginning in fiscal year 2003.

The purpose of this paper is to present a representative Navy shipboard control system, to discuss its distributed implementation by the HiPer-D team, and to present infrastructure support for such a system. Section 2 provides an overview of the shipboard anti-air-warfare (AAW) control system which has been implemented within the HiPer-D Testbed. Section 3 describes a distributed implementation of the AAW system that was constructed by the HiPer-D team. The real-time, fault tolerant and scalable aspects achieved by HiPer-D are discussed in section 3.1. The system configuration is described in section 3.2, and the computing infrastructure is presented in section 3.3. Related work is discussed in section 4, and conclusions and future work are stated in section 5.

2. A Shipboard AAW Software System

This section provides an overview of the shipboard anti-air-warfare (AAW) control system which has been implemented within the HiPer-D Testbed. The Testbed implementation of the AAW system includes the following capabilities: a simulated track source; track correlation and filtering algorithms; track data distribution services; a doctrine server and three types of doctrine processing (semi-auto, auto-sm, and auto-special); an engagement server; a display subsystem including X-windows based tactical displays, submode mediation, and alert routing; surface operations (reengineered from CMS-2 to Ada), and simulated weapons control system and identification upgrade capabilities.

Figure 1 depicts most of the major software components of the AAW system, and categorizes them as *sensors*, *filter and sense* software elements, *evaluate and decide* software elements, *act* software elements, and *actuators*. The remainder of this section discusses the elements in each of these categories.

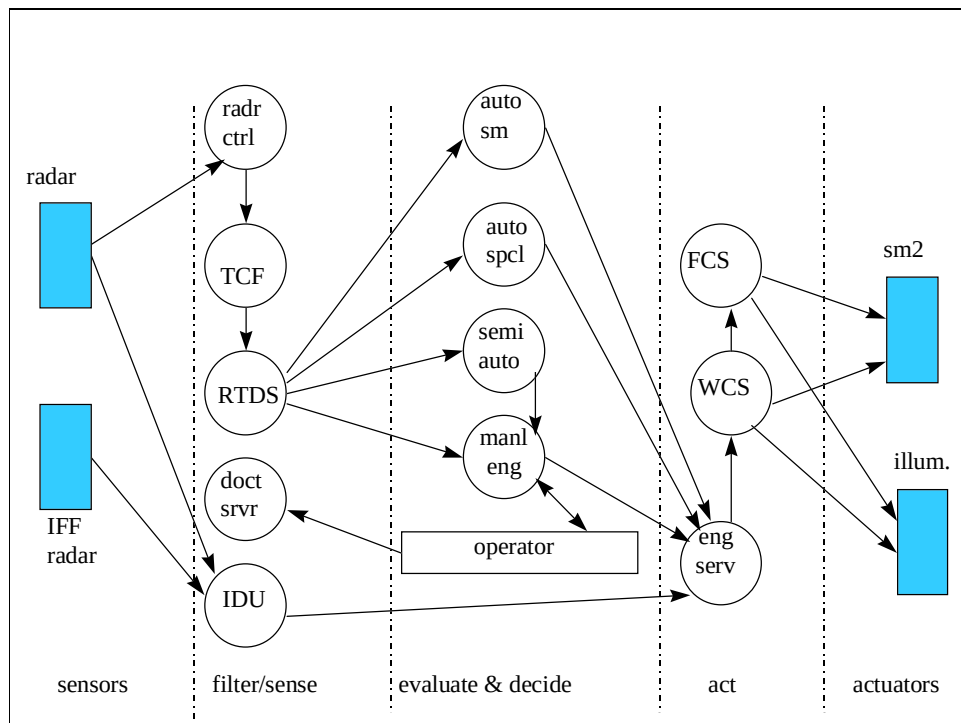


Figure 1. Elements of a shipboard AAW system.

2.1 Sensors and Filter and Sense Elements

The sensors are typically hardware devices (although occasionally sensors are software probes) that provide data regarding the status of various environmental entities. The “filter and sense” elements are typically software components that perform the tasks of collecting, filtering, correlating and classifying sensor data. Some of the sensors and filter and sense elements within the HiPer-D testbed are:

- **IDU Sim** - Identification Upgrade Simulator simulates the Aegis ID capability. It uses a simple random number generator to assign random ID's to new tracks in the system. It also uses an random number generator to perform a Quick ID on any newly engaged target in the system, when requested by the Engagement Server.
- **Track Correlation and Filter (TCF)** - TCF takes input from a sensor simulator and performs track initialization, radar track filtering, and related activities. The TCF component provides track data to the Radar Track Data Server (RTDS) for distribution to all other components of the system that require it.
- **Radar Track Data Server (RTDS)** - Clients of the RTDS register for distribution of track data, specifying a minimum periodicity and a maximum tolerable latency. The RTDS then distributes the data received from the TCF function to the various clients at their specified rates.

- **Doctrine Server** - The Doctrine Server maintains control of doctrine statements used by other components of the system. Doctrine statements are essentially rules that are applied to a set of data (generally track data). The doctrine server controls the activation and deactivation of doctrine statements, and provides doctrine statements to client programs.

2.2 Evaluate and Decide Elements

The “evaluate and decide” elements consist of software and human elements that are responsible for interpreting and assessing sensor data. If the data indicate that some action should be taken by the control system, then the evaluate and decide functions determine what actions should be carried out by the “act” elements. Some of the evaluate and decide elements within the HiPer-D testbed are:

- **AEC Display** - A tactical display capability is provided by the AEC Display program and the AEC State Data Server. The AEC display program provides an X-Windows based geographic display of the tactical situation. Operators may assume a set of responsibilities by signing on to any of the four currently implemented submodes, allowing them to interact with the AEC system. The currently implemented submode positions are Anti-Air Warfare Coordinator (AAWC), Tactical Actions Officer (TAO), Tactical Information Coordinator (TIC), and Computer Program Information Supervisor (CPIS). Once signed-on, an operator may perform any action allowed for that submode. These actions include, among others, listing, activating and displaying doctrine regions, initiating and monitoring engagements, reviewing operator alerts, and monitoring program status. The display window contains a map display, alert and tutorial areas, a VAB/FAB panel, a close control CRO, a tactical CRO area, and a menu for performing map actions such as recentering and rescaling.
- **Display State Data Server** - The Display State Data Server acts as a central point of control for submodes and operator alerts, as well as providing a repository of information about the status of all of the displays. It retains data on the location of each submode signed on; maintains a queue of unreviewed alerts associated with the various submodes; and maintains information on operator actions and display status. Services are provided to the Display State Server clients to retrieve this information, allowing the tactical and support programs to update CROs or act upon VAB actions as appropriate.
- **Manual Engage Control** - The Manual Engage Control program provides the operator interface to the tactical programs. It allows for the manual engagement of tracks and the subsequent sending of an Engagement Request to the Engagement Server as well as the processing of surfaced engagement alerts. Manual Engage Control is an Engagement client and receives updated target engagement status information from the Engagement Server.
- **Semi-Auto** - The Semi-Auto SM program receives tactical weapon doctrine from the Doctrine Server and then performs the processing to evaluate tracks against the currently active Semi-Auto doctrine regions. If a track's evaluated intercept point qualifies, a Recommend Engage alert is queued for the AAWC submode. Semi-Auto is an Engagement Server client and, as such, receives updated target engagement status information from that source.

- **Auto-SM** - The Auto SM program receives tactical weapon doctrine from the Doctrine Server and evaluates tracks against the currently active Auto SM doctrine statements. It sends an Engagement Request to the Engagement Server upon track qualification. Auto-SM is an Engagement client and receives updated target engagement status information from the Engagement Server.
- **Auto-Special** - Auto Special receives tactical weapon doctrine from the Doctrine Server and operates in two independent modes: SPY and AEC. The SPY mode evaluates tentative track and resolution messages reported by TCF in response to the Auto Special Parameters messages sent to Sensor Sim by the Doctrine Server. The Auto Special program attempts to qualify the tracks supplied by TCF against the active Auto Special statements. If a track qualifies, notification is forwarded to the Engagement Server. Auto Special also has an AEC mode which periodically evaluates all tracks against the active Auto Special doctrine statements and forwards an engagement request to the Engagement Server if a track qualifies. Auto-Special is an Engagement client and receives updated target engagement status information from the Engagement Server.

2.3 Act Elements and Actuators

Act elements respond to, and control, the environment by scheduling actuators to perform actions. Following the initiation of actuator responses, an act element monitors and controls the progress of the actuators, and assesses and reports the final outcome of an action. The actuators physically alter the environment to bring it to a desired state. Some of the act elements and actuators within the HiPer-D testbed are:

- **Engagement Server** - The Engagement Server maintains the engagement statuses of all current target tracks and arbitrates any race conditions that may occur due to multiple Engagement clients requesting the same target for engagement. The Engagement Server provides the tactical program interface with the WCS sim. It distributes updates of target engagement status to all Engagement clients and processes engagement requests from the Engagement clients: Auto-SM, Auto-Special, and Manual Engage Control and, if valid, requests the WCS sim to initiate an engagement.
- **WCS Sim** - The WCS Sim provides a simulation of the Weapons Control System. It simulates the actual engagement of a target by sending a missile flyout request to the Sensor Sim, which in turn causes a missile track to be introduced into the track distribution. The WCS Sim also simulates the stages of the engagement by a simple count down mechanism, sending status information back to the Engagement Server at appropriate intervals during the simulated engagement.

3. A Distributed Implementation of the AAW Software System

The current HiPer-D Testbed contains a distributed system providing functionality to support a standard missile path through the combat system. It supports up to 2000 tracks with up to 15 engagements in progress. It consists of more than 50 processes (not including COTS components). It has both fault-tolerant and scalable components. The Testbed currently runs over a heterogeneous network configuration that includes Myrinet, ATM, FDDI, and ethernet, on multiple

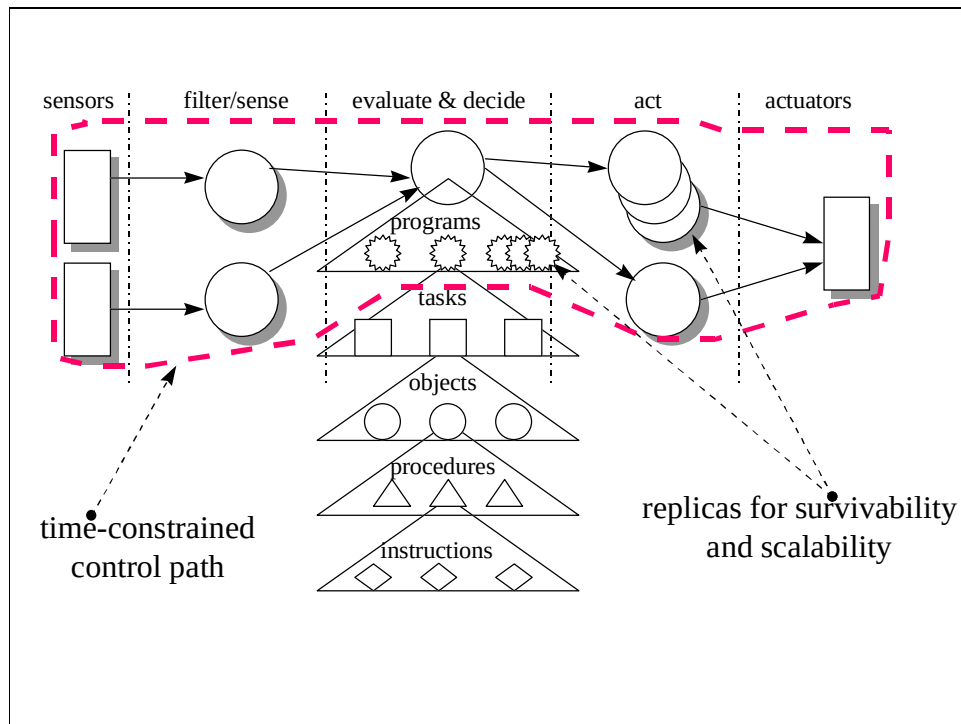


Figure 2. Large scale control system software architecture.

heterogeneous platforms, including Dec Alpha's with OSF-1, Dec Sable with OSF-1, TAC-4's with HP-UX, Sun Sparc 10's with Solaris, and Pentiums with OSF-1RT (real-time). In addition to supporting Aegis functionality, the HiPer-D Testbed also has facilities to support real-time

performance graphics display, and post-mortem performance analysis. HiPer-D has used these facilities to dynamically monitor system performance, to compare performance against the current Aegis Weapons System, and to support dynamic management of resources.

This section describes a distributed implementation of the AAW system, reflecting the level of sophistication achieved by the HiPer-D team. The real-time, fault tolerant and scalable aspects achieved by HiPer-D are discussed in section 3.1. The system configuration is described in section 3.2, and the computing infrastructure is presented in section 3.3.

3.1 Real-Time, Fault Tolerance and Scalability

The real-time, fault tolerance and scalability paradigms employed within HiPer-D's distributed implementation of the AAW system are illustrated in Figure 2. Real-time constraints are stated with respect to control paths which span one or more elements of the control system. Figure 2 depicts a time constrained path that begins with two sensors.; the sensor data feed two sense elements, which provide input to an evaluate and decide element; the evaluate and decide element triggers two act elements, which control a single actuator. Such an execution path is typically the unit for which timing requirements are specified and assessed in the HiPer-D project. Note the large granularity at which timing requirements are specified, which has ramifications on the granularity with which resource allocation and timing assessment are performed. In addition to referring to large grain entities, timing characteristics for the AAW system include: dynamically changing periods; sporadically arriving, transient periodic tasks with dynamically determined

PROCESSOR (QUANTITY)	OPERATING SYSTEM	NETWORKS
Sun UltraServer 4000, 6 processor SMP (1)	Solaris 2.5.1	Ethernet/FDDI/Myrinet
Sun Server 1000, 8 processor SMP (1)	Solaris 2.5.1	Ethernet/FDDI/ATM
Sun Ultra 1 (2)	Solaris 2.5.1	Ethernet/FDDI/Myrinet
Sun Ultra 2, 2 processor SMP (1)	Solaris 2.5.1	Ethernet/FDDI/Myrinet/ATM
Pentium 90Mhz (6)	OSF RT	Ethernet/FDDI/Myrinet
HP TAC-4 J210 (7)	OSF PA RT or HPUX 10.10	Ethernet/FDDI/Myrinet or Ethernet/FDDI/ATM
Alpha Server 2100 4 processor SMP (1)	DEC Unix 3.2	Ethernet/FDDI
Alpha 3000 (4)	DEC Unix 3.2	Ethernet/FDDI/ATM
Alpha 200 (5)	DEC Unix 3.2	Ethernet/FDDI

Table 1. The HiPer-D hardware platform.

periods; tactical loads which vary by several orders of magnitude; unknown upper bounds on required processing; adaptation to missed timing requirements; and high priority sporadics.

Application-level fault tolerance is achieved by replication of tactical programs, resulting in a primary (active) copy and secondary (passive) copies residing on different processors (see figure 2). Current state is maintained by all copies of a program. When a replica terminates abnormally, all replicas detect this automatically (through notification from middleware services). If the primary replica terminates, a new primary is selected automatically (by the middleware). This results in virtually no loss of functionality, as switchover to the new primary occurs very quickly. To maintain survivability, a new replica is automatically restarted by the resource manager. The determination of the location at which to restart the replica involves prediction of quality of service (for further discussion see section 3.3). The state is transferred to new replica automatically (by the middleware services).

Replication of tactical programs is also employed to achieve application-level scalability (see figure 2). Programs are made scalable by sharing tactical load among replicas. Individual replicas self-schedule their own portion of the tactical load, by considering the number of replicas currently in the system. Since replicas automatically detect when one replica dies, self-scheduling can adapt to varying numbers of replicas. When a load-sharing replica terminates abnormally, a new replica is automatically restarted by the resource manager.

3.2 System Configuration

The AAW system is implemented within the HiPer-D distributed platform, which contains a broad array of processor types, operating systems and interconnection networks. Specifically, the tactical programs are distributed over the architecture summarized in Table 1.

3.3 Computing Infrastructure

Figure 3 depicts a computational infrastructure that is appropriate for large grain, distributed control systems. A system engineer specifies the characteristics of the *paths* which constitute application software and also specifies the characteristics of the distributed hardware. Real-time

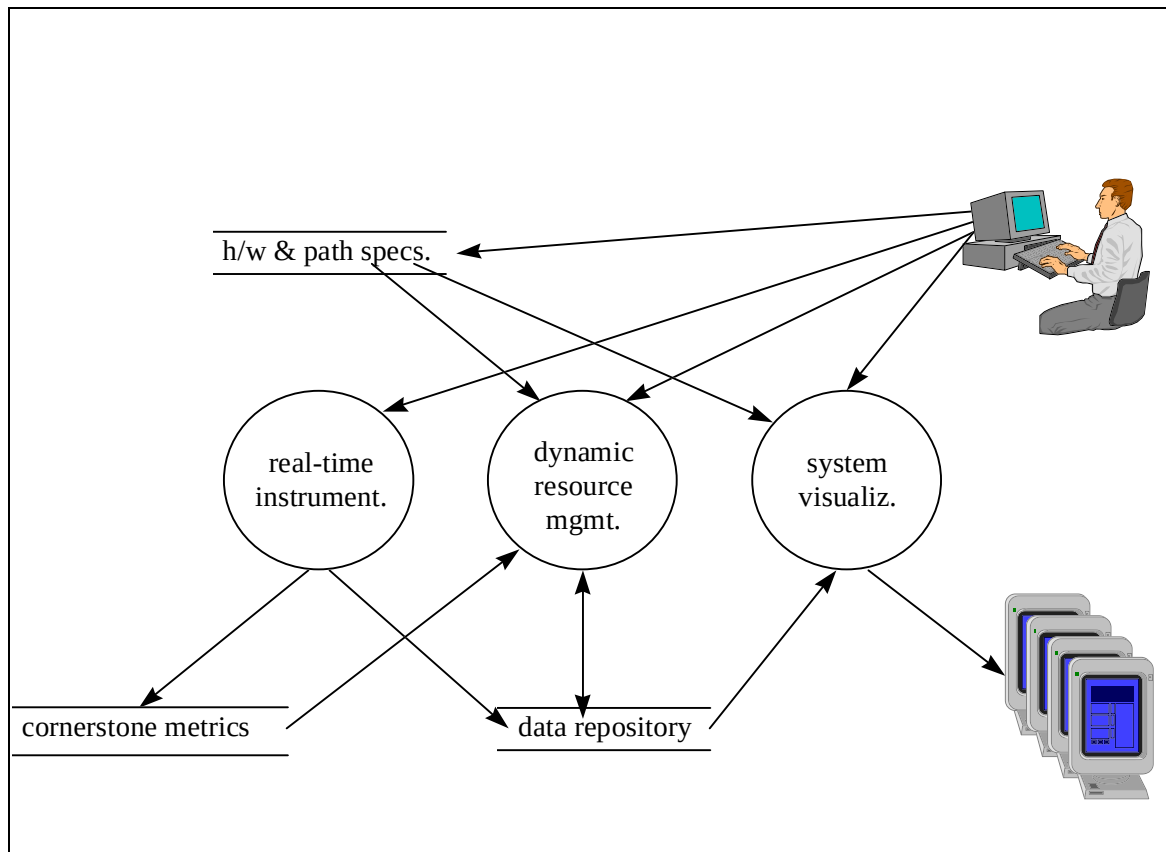


Figure 3. A computing infrastructure for distributed, real-time, fault tolerant control systems.

instrumentation provides cornerstone metrics which characterize such things as the performance and resource usage requirements of the paths. The information gathered during instrumentation is used by a dynamic resource manager to make allocation decisions, and is provided to system visualization components which provide various views of the quality of service being provided to the application system. HiPer-D has developed a computational infrastructure that contains prototypes of the major components depicted in Figure 3. An important goal is to evolve the infrastructure to address (1) quality of service assessment and (2) adaptive resource management for path-based systems.

4. Related Work

Resource management in real-time systems has been addressed in [1-5,7-33] (a non-exhaustive list). In much of this work, a task is defined as an independent program or a program component having a single thread of control, with timing constraints imposed on it. Schedule construction or schedulability analysis is performed statically using some heuristics - fixed priority algorithms as in [10],[12], and dynamic priority algorithms as in [2],[21] - with a task as the unit of scheduling, for all a-priori known tasks. For tasks arriving dynamically, the approach is to either statically pre-allocate computational resources (bandwidth preserving algorithms ---see [9],[24],[25]), or allocate unused computation time (slack stealing algorithms : [4],[8],[19],[28]) when the task arrives,

without effecting the successfully scheduled tasks. In distributed systems, the idea is to use a uni-processor scheme for local scheduling, and perform distributed scheduling [16] for tasks which are potentially subject to timing failures at run-time. Such tasks are typically those that arrive at run-time or those that were statically scheduled to realize time deadlines, but had to be rejected due to the arrival and scheduling of a dynamic task with high priority. To schedule a locally rejected task, the key idea is to search for a computational resource where it can be successfully allocated. Different schemes ([16],[18],[22],[23],[26]) have been presented in the literature for this purpose. Further, worst-case assumptions regarding execution times and communication delays have been made to provide hard guarantees.

Path-Based systems such as the AAW system (see sections 2 and 3.1) consist of multitudes of programs that communicate in an irregular, non-trivial manner, with each program having a hierarchical software structure. Timing constraints are specified for very large grain entities. A path may have large variability in its execution time (e.g., 3 orders of magnitude!). Also, many Path-Based systems have dynamically changing periods, high priority aperiodics, and scalability and dependability requirements. As stated by Koob in [6], the application of techniques which are designed to handle *anticipated* peak demands leads to (1) inefficient resource utilization and (2) inadequate crisis response. The engineering of systems having the dynamic, large grain characteristics of the AAW system present many challenges to technologists. Can the existing paradigms [1-5,7-33] for real-time computing be extended to meet the challenges? Is a new paradigm needed? The answers to these questions will become clear as systems such as the Navy's 21st Century Surface Combatant are engineered.

5. Conclusions

This paper describes a representative shipboard control system, and describes its distributed implementation. The distributed implementation has been successfully engineered by the HiPer-D team using primarily commercial technology. The HiPer-D prototype incorporates many critical components of the Aegis Weapon System, and thus provides a platform for experimental validation of concepts. An initial computational infrastructure has been engineered by the HiPer-D team, and is employed to provide real-time, fault tolerance, and scalability services to the distributed AAW application. Ongoing work includes evolution of the infrastructure to address (1) quality of service assessment and (2) adaptive resource management.

Distributed implementation of dynamic path-based control systems (such as the AAW system described in section 2) present many challenges to technologists, particularly in the areas of (1) instrumentation, data analysis and visualization, (2) definition of mathematical models that are based on metrics and parameters obtainable via instrumentation and analysis tools, (3) real-time distributed system control and management services, (4) dynamic resource allocation services, and (5) embedded system benchmarks and processing and communication load simulations.

6. References

1. S. Cheng, J. Stankovic, and K. Ramamritham, "Dynamic Scheduling of Groups of Tasks With Precedence Constraints in Distributed Hard Real-Time Systems," *Proceedings of The IEEE Real-Time Systems Symposium*, 1986.

2. H. Chetto and M. Chetto, "Some results of the Earliest Deadline Scheduling Algorithm," *IEEE Transactions on Software Engineering*, 15(10):466-473, 1989.
3. Chintamaneni, X. Yuan, S. K. Tripathi, and A. K. Agrawala, "Scheduling Tasks in a Real-Time System," Technical Report, Department of Computer Science, University of Maryland at College Park, 1994.
4. R. I. Davis, K. W. Tindell, and A. Burns, "Scheduling Slack Time in Fixed-Priority Preemptive Systems," *Proceedings of The IEEE Real-Time Systems Symposium*, pp. 222-231, 1993.
5. R. Gupta and M. Spezialetti, "A Compact Task Graph Representation for Real-Time Scheduling," *Journal of Real-Time Systems*, July, 1996.
6. Gary Koob, "Quorum," *Proceedings of the DARPA ITO General PI Meeting*, pages A-59 to A-87, October 1996.
7. C. M. Krishna and K. G. Shin, "Scheduling Tasks with a Quick Recovery from Failure," *IEEE Transactions on Computers*, May 1986.
8. J. P. Lehoczky and S. Ramos-Thuel, "An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems," *Proceedings of The IEEE Real-Time Systems Symposium*, pp. 110-123, 1992.
9. J. P. Lehoczky, L. Sha, and J. K. Strosnider, "Enhanced Aperiodic Responsiveness in Hard-Real-Time Environments," *Proceedings of The IEEE Real-Time Systems Symposium*, 1987.
10. J. Y.-T. Leung and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks," *Performance Evaluation*, 2:237-250, 1982.
11. A. L. Liestman and R. H. Campbell, "A Fault-Tolerant Scheduling Problem," *IEEE Transactions on Software Engineering*, November, 1986.
12. C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, 20(1):46-61, 1973.
13. R. Mirchandaney, D. Towsley, and J. A. Stankovic, "Analysis of the effect of delays on load sharing," *IEEE Transactions on Computers*, vol. 38, pp. 1513-1525, November 1989.
14. A. K. Mok, "Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment," Ph.D. Dissertation, M.I.T., Cambridge, MA 02139.
15. M. W. Mutka and J. P. Li, "A Tool for Allocating Periodic Real-Time Tasks to a Set of Processors," *Journal of Systems and Software*, vol. 29, pp:135-148, 1995.
16. K. Ramamritham and J. A. Stankovic, "Dynamic Task Scheduling in Hard Real-Time Distributed Systems," *IEEE Software*, July, 1984.
17. K. Ramamritham, J. Stankovic and P. Shiah, "Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems," *IEEE Transactions on Parallel and Distributed Systems*, April 1990.
18. K. Ramamritham, J. A. Stankovic, and W. Zhao, "Distributed Scheduling of Tasks with Deadlines and Resource Requirements," *IEEE Transactions on Computers*, August 1989.

19. S. Ramos-Thuel and J. P. Lehoczky, "On-Line Scheduling of Hard Deadline Aperiodic Tasks in Fixed-Priority Systems," *Proceedings of The IEEE Real-Time Systems Symposium*, pp. 160-171, 1993.
20. B. Ravindran and L. R. Welch, "Exploiting Parallelism in High Performance Embedded System Scheduling," *Proceedings of The Third International Conference on High Performance Computing*, December 1996.
21. K. Schwan and H. Zhou, "Dynamic Scheduling of Hard Real-Time Tasks and Real-Time Threads," *IEEE Transactions on Software Engineering*, 18(8):736-748, 1992.
22. K. G. Shin and, C.-J. Hou, "Analytic Models of Adaptive Load Sharing Schemes in Distributed Real-Time Systems," *IEEE Transactions on Parallel and Distributed Systems*, July 1993.
23. K. G. Shin and, C.-J. Hou, "Design and Evaluation of Effective Load Sharing in Distributed Real-Time Systems," *IEEE Transactions on Parallel and Distributed Systems*, July 1994.
24. B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic Task Scheduling in Hard-Real-Time Systems," *Journal of Real-Time Systems*, vol. 1, no. 1, pp: 27-60, 1989.
25. M. Spuri and G. Buttazzo, "Scheduling Aperiodic Tasks in Dynamic Priority Systems," *Journal of Real-Time Systems*, March 1996.
26. J. A. Stankovic, K. Ramamritham, and S. Cheng, "Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems," *IEEE Transactions on Computers*, vol. C-34, no. 12, pp. 1130-1141, December, 1985.
27. J. A. Stankovic, M. Spuri, M. D. Natale and G. C. Buttazzo, "Implications of Classical Scheduling Results for Real-Time Systems," *IEEE Computer*, June 1995.
28. T-S. Tia, J. W.-S. Liu, and M. Shankar, "Algorithms and Optimality of Scheduling Soft Aperiodic Requests in Fixed-Priority Preemptive Systems," *Journal of Real-Time Systems*, January 1996.
29. J. P. C. Verhoosel, L. R. Welch, D. K. Hammer, A. D. Stoyenko, and E. J. Luit, "A Formal Deterministic Scheduling Model for Object-Based Hard Real-Time Executions," *Journal of Real-Time Systems*, 8(1), January 1995.
30. J. P. C. Verhoosel, L. R. Welch, D. K. Hammer, and E. J. Luit, "Incorporating Temporal Considerations during Assignment and Pre-run-time Scheduling of Objects and Processes," *Journal of Parallel and Distributed Computing*, vol. 36, pp. 13-31, 1996.
31. L. R. Welch, A. D. Stoyenko, and T. J. Marlowe, "Response Time Prediction for Distributed Periodic Processes Specified in CaRT-Spec," *Control Engineering Practice*, 3(5), May 1995, pp. 651-664.
32. G. Yu, "Identifying and Exploiting Concurrency in Object-Based Real-Time Systems," Ph.D. Thesis, Dept. of Computer & Information Science, New Jersey Institute of Technology, October, 1995.
33. G. Yu and L. R. Welch, "A Novel Approach to Off-Line Scheduling in Real-Time Systems," *Informatica*, Special Issue on Parallel and Distributed Real-Time Systems, vol. 19, pp. 71-82, 1995.