Technical Report

CMU/SEI-90-TR-17
ESD-90-TR-217

Carnegie-Mellon University
Software Engineering Institute

Experiences Porting the Distributed Ada Real-Time Kernel

Brian Smith
Boeing Military Airplanes

James E. Tomayko
Software Engineering Institute

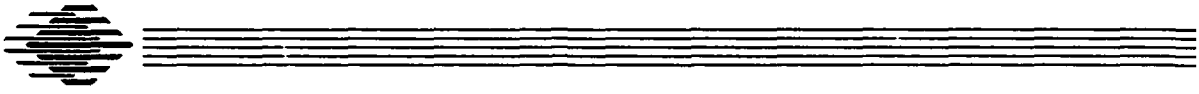
June 1990

The following statement of assurance is more than a statement required to comply with the federal law. This is a sincere statement by the university to assure that all people are included in the diversity which makes Carnegie Mellon an exciting place. Carnegie Mellon wishes to include people without regard to race, color, national origin, sex, handicap, religion, creed, ancestry, belief, age, veteran status or sexual orientation.

Carnegie Mellon University does not discriminate and Carnegie Mellon University is required not to discriminate in admissions and employment on the basis of race, color, national origin, sex or handicap in violation of Title VI of the Civil Rights Act of 1964, Title IX of the Educational Amendments of 1972 and Section 504 of the Rehabilitation Act of 1973 or other federal, state, or local laws or executive orders. In addition, Carnegie Mellon does not discriminate in admissions and employment on the basis of religion, creed, ancestry, belief, age, veteran status or sexual orientation in violation of any federal, state, or local laws or executive orders. Inquiries concerning application of this policy should be directed to the Provost, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-6684 or the Vice President for Enrollment, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-2056

Technical Report
CMU/SEI-90-TR-17
ESD-90-TR-217
June 1990

Experiences Porting the Distributed Ada Real-Time Kernel



Brian Smith

Boeing Military Airplanes

James E. Tomayko

Software Engineering Institute

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

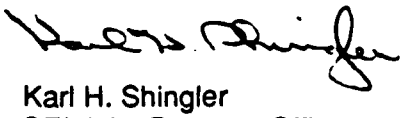
SEI Joint Program Office
ESD/AVS
Hanscom AFB, MA 01731

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER



Karl H. Shingler
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

During the time the work described in this report was being done, Dr. Tomayko was an associate professor of computer science at The Wichita State University, under contract to Boeing Military Airplanes, and was not part of the DARK Project.

Table of Contents

List of Figures

Experiences Porting the Distributed Ada Real-Time Kernel

Abstract: The Distributed Ada Real-Time Kernel (DARK) is a mechanism for supporting the execution of distributed real-time Ada applications in embedded computer systems. It provides a solution to scheduling and distributing tasks without modifying the Ada language or vendor-supplied runtime systems. An important test of the utility of the Kernel is whether or not it can be ported to different hardware architectures and still function effectively. As part of an independent research and development project, Boeing Military Airplanes and The Wichita State University became co-acceptors of a copy of DARK for the purpose of demonstrating a port to a 68000-based distributed architecture. This report describes the experiences in accomplishing the port.

1. Rationale

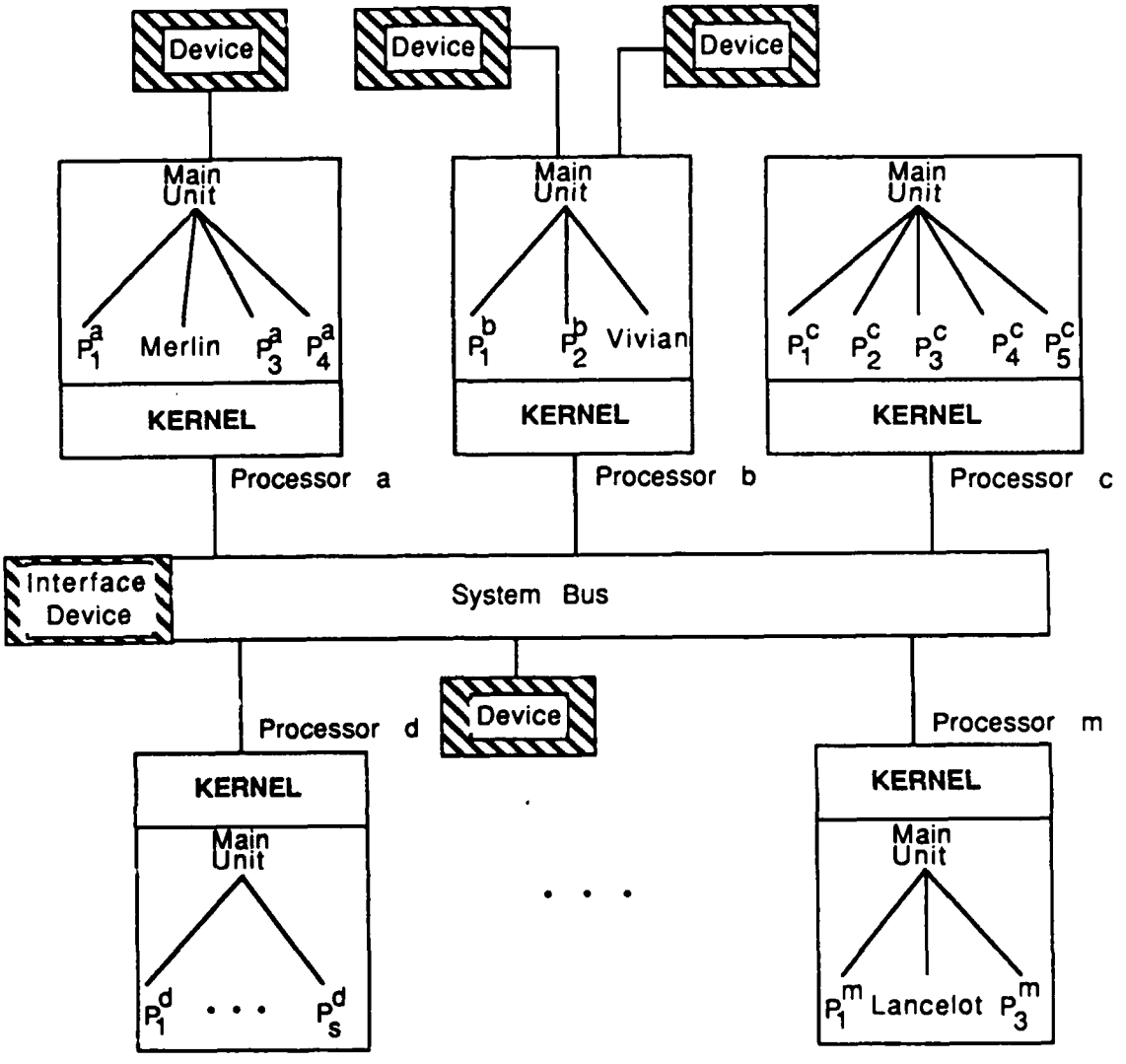
The Distributed Ada Real-Time Kernel (DARK) Project began as an attempt to prove that distributed real-time processing in Ada could be accomplished without tailoring the vendor-supplied runtime systems or the language itself. This is important because either solution to the problems of real-time systems would defeat portability and simplicity. Users want languages and systems to be functional, not feature-filled. They also want to solve scheduling and tasking requirements once, and not every time that they are faced with a new architecture or applications suite.

The requirements and design for DARK are detailed elsewhere, and it is expected that the reader of this report is familiar with the details of the process model and software architecture.[1] Essentially, the Kernel provides facilities for communication between processes on different processors, semaphore and scheduling management, control of time and interrupts, and provision for setting alarms. A view of the network on which DARK was originally implemented is shown in Figure 1-1. Note that non-Kernel devices such as sensors can be controlled by processes on Kernel processors, and that each processor can support multiple processes. Communication was implemented using the industry sector operations (ISO) seven-layer model; in this case messages are removed from queues, enclosed in packets, and sent along the network to the appropriate destination.

Creating the Kernel and demonstrating it on one testbed is clearly not enough to prove the efficacy of the approach. It is important to port the Kernel to different architectures both to examine its operation as a general solution to distributed processing problems and to enable users who are just beginning to deal with these issues to see a working solution. Accordingly, one of the goals of the DARK Project from the start has been technology transition.

---

[1]See especially Judy Bamberger et al., *Kernel Facilities Definition*, CMU/SEI-88-TR-16, July 1988. Other references listed on p. 21.



Key

P_i^q : Process #i running on processor q.

Main Unit: The Ada Main Unit running on the processor.

Merlin, Vivian, and Lancelot are named for use in examples.

Figure 1-1: Network on Which DARK Was Originally Implemented
(This figure originally appeared in the *Kernel Facilities Definition*, p. 18.)

One mechanism of transition is the port of the Kernel to a VAX system, which is more easily available to potential users and on which the operation of the Kernel can be studied without a lot of the problems encountered observing execution of software on more deeply embedded systems. This port was accomplished at the Software Engineering Institute by members of the DARK Project team. Another mechanism is the release of the Kernel to a set of Acceptor Sites, which then try to use DARK in their own projects. The remainder of this paper describes the experiences of the Boeing Military Airplanes-the Wichita State University Acceptor Site. The actual work was done at Boeing using independent research and development funds, with the Wichita State University supplying expertise under contract to the Boeing Project.

2. Porting DARK at Boeing Military Airplanes

Boeing has a long history of research in embedded distributed systems and is presently working on operating systems and hardware architectures based on both the PAVE PILLAR and PAVE PACE government initiatives. One result of this work was an operating system developed as an independent research and development project and described in papers by Higgins and Kroening.[2] This particular operating system was designed for the military standard 1750A 16-bit processor. As part of continuing research, the system is being ported to a 32-bit processor network. This is the network that is also the basis for Boeing's port of DARK. Figure 2-1 is a system view of the network.

Basically, the network consists of clusters of nodes, where each node is a processor capable of supporting multiple processes. As part of the operating system, a communications package enabling process-to-process message passing throughout the network is available (detailed in Kroening's paper). It was decided to follow two different paths in porting DARK to this architecture. One path is to demonstrate that DARK works with non-communicating processes first on a single processor and later on multiple processors. The second path is to use the Kernel as is, including the communications facilities available to the user, but to replace the existing DARK communications scheme with the already-developed Boeing communications package. The details of these two ports are in the next sections.

2.1. Porting DARK Without Communications

The first port of DARK at Boeing was to establish the functionality of the Kernel without communications, since from the start there was no intention of using the original DARK communications package. This test meant running several processes on a single processor and observing the results. Some positive and negative aspects of DARK were discovered as part of this port.

DARK has the advantage of being an extremely well-documented system. This is an immense aid in onsite modification. It is also, in general, fairly well designed. The use of Ada as a development language provides for easy readability and good data tracking. On the down side, there are some coding practices that the authors have used that complicate the dependency lists and increase the number of modules that require recompilation during updates. Chief among these is the use of generics.

Of the 23 generic packages defined in DARK, only two are truly generic. In the other 21 instances, DARK uses these generic packages as a means for allowing the user to define

---

[2]D. W. Higgins, *Implementing a Fault-Tolerant, Distributed Operating System in Ada*, The Wichita State University Computer Science Department, Technical Report WSU-CS-T-87-5, and J. E. Kroening, *Logical Addressing: Communications in a Distributed Architecture*, The Wichita State University Computer Science Department, Technical Report WSU-CS-T-87-6.

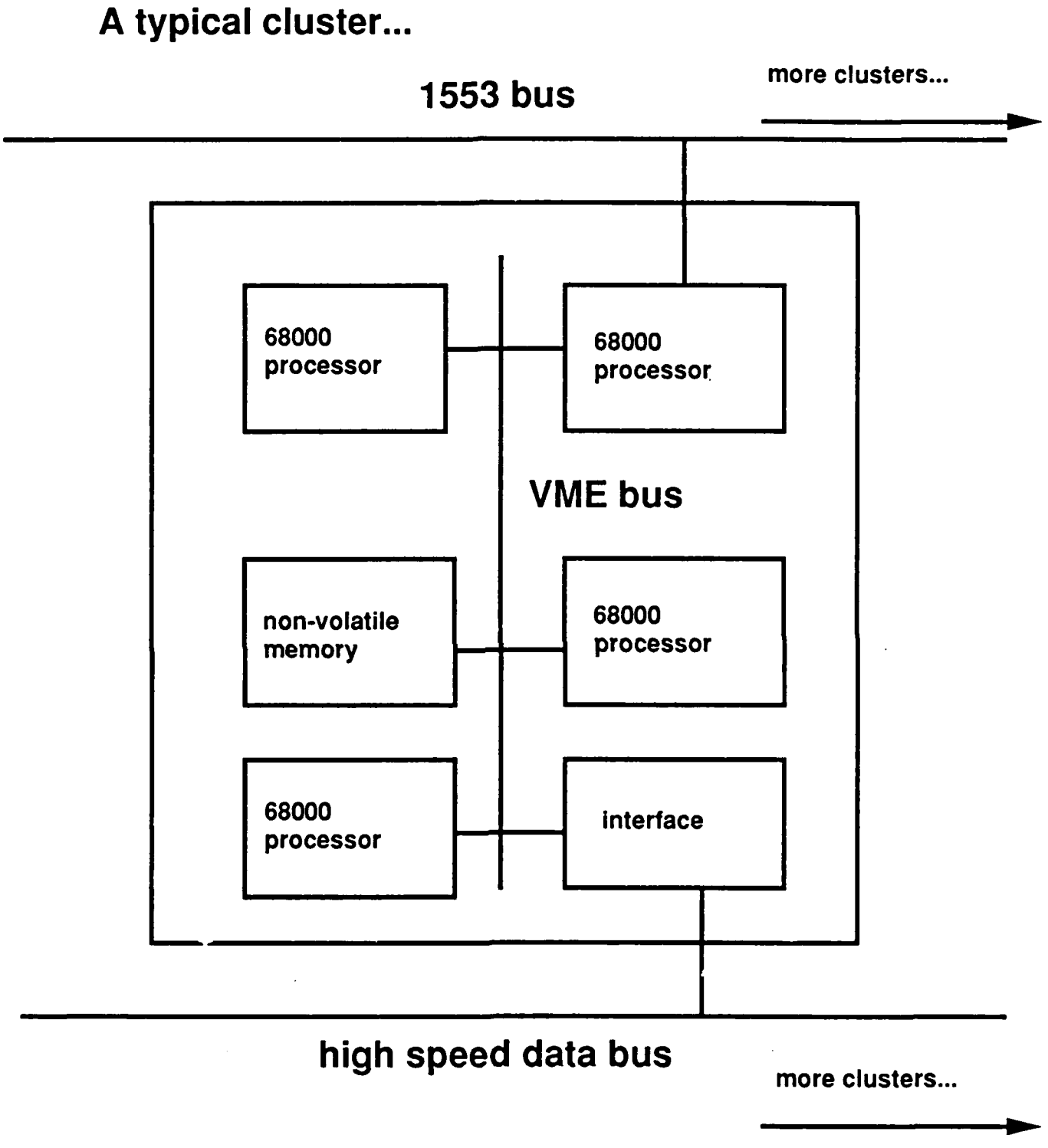


Figure 2-1: System View of the Network

constants. In porting this system, there were several occasions where changes were made to only the body of a generic package. In reinstantiating the generic, the specification of the instantiation is marked as having been updated even though only the body was changed. Ada dependency rules then force recompilation of all modules referencing this specification. By using generics in this fashion, the functionality of Ada in providing package specifications as useful module interface controls is violated.

Another area of difficulty was in the linkages to assembly code modules. In several instances the **Interface** and **Linkname** pragmas were defined in package specifications. These pragmas provide for interfacing between Ada and assembly modules and should only be placed in the package bodies. That way the package specification will not need to be modified should the linkname reference be changed.

The target hardware for the port consists of three clusters connected by a high-speed data bus, the system architecture as illustrated in Figure 2-1. Within each cluster are multiple PV682 central processing units (CPU's) and other elements such as nonvolatile memory and a 1553 communications board interconnected by a VME backplane bus. Three of the PV682 CPUs are available for user code; the fourth is used for the distributed system communications functions, including 1553, internode, and intercluster communications.

A processor node consists of the following:

- Motorola 68020 CPU
- Motorola 68881 Floating Point Coprocessor
- Advanced Micro Devices Am9513 System Timer Controller (5 timers)
- OKI MSM6242 RS/GS Real Time Clock/Calendar (not used)
- One Mb Ram
- Up to One Mb Rom
- Two Rs423/422 Serial Ports

The timers and serial ports do not match those used on the MVME133A board in the DARK testbed configuration. This affects the low-level software and interface protocols used by the DARK code. Additionally, each cluster consists of multiple CPUs on a VME backplane. Each CPU is capable of running application software. This in contrast to the DARK testbed, where only one CPU per two-processor node actually runs application software.

The rehost plan took into account the differences between the testbed architectures. The plan was divided into two major phases.

Phase 1 was a trim down phase. References to unsupported hardware and software elements were eliminated or modified. This included stripping out references to the ring communication architecture and modifying the timer/clock structure to reference the Boeing hardware. The end product of this phase was a subset of DARK that operates in a minimal fashion on a single PV682 CPU.

At the end of the Phase 1 modifications, the capabilities of DARK that remained were evaluated for use. Factors such as functional usefulness, efficiency, and code size were considered. Additional consideration was given to interfacing this core of DARK with other research and development efforts including internode communication, data security, and task fault tolerance. The nature of this Phase 2 effort is discussed in the last section of this paper.

The following directives were made for Phase 1 modifications:

- The target hardware was a subset of the final target hardware configuration. A single PV682 CPU was to be used for Phase 1 tests.
- All ring communications references were to be eliminated. Furthermore, local intraprocessor communications were not required.
- Unused data structures could be left "dangling." Phase 1 modifications were not to produce a final polished product, but something that could be quickly evaluated.

Original and selected intermediate versions of the DARK software were maintained during these modifications. This allowed for full design tracing and possible backtracking of software changes. Additionally, modifications are being tracked by an informal software problem report (SPR) system.

Details on these modifications are as follows:

2.1.1. Deletion of Debug Code

DARK contained a significant amount of idiosyncratic debug code that the team considered unnecessary. All references to the debug routines and any inline debug code were eliminated.

The following modules were modified:

```
-- Package specifications:

    - scheduler.Ada              - time_keeper.Ada

-- Package bodies

    - bio_body.Ada          - c_body.Ada
    - gam_body.Ada          - gpam_body.Ada
    - grm_body.Ada          - gtsm_body.Ada
    - nproc_body.Ada        - pe_body.Ada
    - pit_body.Ada          - scc_porta_body.Ada
    - sch_body.Ada          - tb_body.Ada
    - tk_body.Ada
```

The following modules (specs and bodies) are no longer referenced:

- csa_debug.Ada
- csa_debug_body.Ada
- debug.Ada
- dgg_debug.Ada
- dgg_debug_body.Ada
- kt_debug.Ada
- kt_debug_body.Ada
- nct_debug.Ada
- nct_debug_body.Ada
- ptb_debug.Ada
- ptb_debug_body.Ada

2.1.2. Reduction of Network to Single Processor

For expediency in getting a subset of DARK operational for evaluation, the network was reduced to a single processor. This simplified the removal of communication code, especially in areas where such code was hidden to the user. This simplification allowed for rapid modification with a minimum of worry about internal details. The drawback is several dangling code structures that will require future cleanup during Phase 2 modifications.

The main code change to accomplish the reduction was in package Generic_Processor_Management where much of the network startup code was eliminated. This change forced a simplified network configuration table (NCT) declaring a single Kernel process (KPROC) board. Additionally, any modules confined to the software tree rooted by package node process (NPROC) are not referenced. Package Low_Level_Hardware was also modified to provide stubouts for procedures requesting network identification switch readings.

2.1.3. Removal of Communication Code

Communication code falls into two categories: startup and interprocess communication. Most calls to the communications modules were simply commented out. This will provide a roadmap for rebuilding during Phase 2 modifications. The exception to this general rule is in package Generic_Processor_Management, where the communication calls were embedded in the network initialization code that was removed.

References to the following modules were eliminated during the modifications:

bus_io.Ada
bio_body.Ada
datagram_management.Ada
dgm_body.Ada
datagram_globals.Ada
generic_communication_management.Ada
gam_body.Ada

These modules were modified to remove communications references:

generic_process_table.Ada
gpam_body.Ada
gpm_body.Ada
grm_body.Ada
ipm_body.Ada
tk_body.Ada

2.1.4. Startup Syncronization Communication

The DARK architecture provided for a serial communications line between Kprocs, which was to be used for synchronizing the real-time clocks across the network. By assuming that only one processor was to be defined, we eliminated this communication line and its associated software drivers. As this was the only use of the serial communication port on the MVME133A, the serial port driver modification could be postponed indefinitely. The top-level time syncronization functions were rooted in package Generic_Processor_Management, which was modified to support these changes.

2.2. Intraprocess Communication

User-level communications were confined to the Generic Communication Management package. These modules could be eliminated from DARK simply by not referencing them. Embedded within DARK, however, were several references to lower level communication interfaces by the event scheduling, processor management, and process management algorithms. These algorithms were required to clean up messages and buffers left dangling when a message request timed out, a process was aborted, or the like. Given the complexities of the package dependency lists, it was easier to comment out all communication references wherever they occurred, rather than try to isolate and stub out the top level references of this lower tier of communication routines.

2.2.1. Updating of Clock/Timers to PV682 Hardware

The original DARK architecture used two hardware timers for time and event management. One timer ran the clock and the other was used for the event queue interrupts. This allowed for very fine granularity in time definition. The PV682 board has only one timer that could be used for these functions, timer #2. Timer #3 is theoretically available, but unfortunately it cannot be used with our emulator hardware.

A reasonable workaround is to quantize time to a larger value, on the order of 20 milliseconds, and convert to what is essentially a frame-based time event system. This was not entirely arbitrary or unreasonable, as typically real-time embedded software is frame based. This modification actually simplified the DARK timers and clock functions without any significant change to software above the level of the Time_keeper body software. This change eliminated packages Clock and Timer_Controller. The only significant addition to Time_keeper is code to initialize and reset the hardware timer and to increment the current time. The definitions of interrupts for DARK were updated as part of the timing replacement. All of the interrupts for the MZ8305 parallel i/o boards, the MVME133A serial i/o port, and the MVME133A timers were eliminated. The interrupt for the clock timer was redefined to match the hardware connection of the PV682 board.

The following modules were modified or created:

- **tk_body.Ada**
- **interrupt_names.Ada**
- **lltk.a68 (new)**

These modules are no longer used:

```
- clock.Ada                - c_body.Ada        - low_level_clock.a68
- timer_controller.Ada     - tc_body.Ada       - tc_body_machine_code.a68
- mz8305_definitions       - mz_body.Ada
```

2.2.2. Original Test Plans

This section describes the basic tests performed on DARK after Phase 1 modifications were completed. The test objective is to determine the efficiency and flexibility of the DARK scheduling algorithm and compare it to the default Telesoft Ada runtime functions.

2.2.2.1. Timeslicing

Timeslicing need not necessarily be considered at this time for embedded applications, as this software is typically run on a fixed-frame basis. Typically embedded software runs with a short time frame on the order of 20 to 50 milliseconds. True timeslicing between parallel processes would require that the slice time be on the order of one tenth or less of the frame time. This would produce a time slice on the order of 2 milliseconds or less. Any timeslicing test should be conducted with this order of slice size. Note that the default time slice size for DARK is one second.

2.2.2.2. Scheduling Overhead

An executive was defined, under DARK and standard Ada, to run a set of processes. These processes performed simple math functions in a loop to burn time. Execution time for these processes were compared under DARK and standard Ada for efficiency checks.

This test shall be repeated with blocking (wait) statements inserted into the process loops so that the context switching and scheduling algorithm overhead time may be computed. The processes were designed so that the blocking statement periodicity is on the same order as the frame time. With several processes queued up, this will keep the CPU busy and the time measurements accurate.

2.2.2.3. Process Priority

Functionality of the DARK process priority mechanism shall be tested. The tests shall verify that:

- High-priority processes are always scheduled in favor of lower priority processes when they become unblocked.
- Processes of equal priority with wait are cycled through without starvation.
- Low-priority processes become active when high-priority processes are blocked.

2.2.2.4. Kernel Size

Executable test code packages for equivalent processes running under DARK and standard Ada shall be compared for size. Excessive Kernel size could eliminate DARK use even if all other factors are favorable.

2.2.3. Phase 1 Evaluation Results

The following program design language (PDL) describes the software test driver used for the initial DARK tests on the uniprocessor configuration. Five parallel tasks were defined of the same format.

```
Define tasks:    Process_one      : high priority
                 Process_two      : high priority
                 Process_three    : high priority
                 Process_four     : high priority
                 dawdle           : low priority

Task structure:  Begin
                   start_tag (for debugger)
                   while (count_2 < 700) loop
                     while (count_1 < 15000) loop
                       increment count_1
                     end loop
                     midpoint_tag (for debugger)
                     wait/delay for 20 milliseconds (optional)
                   end loop
                   end_tag (for debugger)
                 End
```

The results can be summarized as follows:

2.2.3.1. Time Slicing

An initial attempt at time slicing was made; however, it was not successful. This attempt was made with three processes of equal priority that ran counting loops. These processes always executed sequentially, even when time slicing was enabled. This indicated some problem operationally with DARK, or the modifications, that was not determined as of this writing.

2.2.3.2. Scheduling Overhead

DARK requires an average of .67 milliseconds to perform a full end-to-end context switch resulting from a Wait call. This measurement is the total time introduced in switching from process A to process B, assuming process B is ready to run. Measurements from an equivalent process using delay statements under the standard Ada runtime yielded a full context switch time of .32 milliseconds, less than half of DARK's switch time. This time measurement also (unavoidably) included the clock/event interrupt handler execution time, so the true context switch time is really somewhat better than this.

A direct comparison of task encapsulation overhead time between DARK and standard Ada is not readily available, as the Ada compiler generated different format assembly code for the two test cases. In this particular case, a more efficient set of assembly code for the test procedures was generated under DARK (some variables were stored solely in registers, rather than being written out to memory). However, in general, this would not be true.

The total process elaboration times for the test software were measured. Use of DARK added 23 milliseconds to the 447 millisecond startup time required by the Ada runtime without DARK. This represents a five per cent increase.

2.2.3.3. Process Priority

Equal priority processes execute in a first-defined, first-executed order when blocking or other preemption is not used. When priority has been defined, DARK executes according to the ordered priority specified, highest priority process (lowest priority number) to lowest priority process (highest priority number). DARK switches to a lower priority process when a higher priority process is blocked. However, it did not automatically switch back once the higher priority process becomes unblocked. The lower priority process continues to run until it is blocked, at which time the higher priority process resumed.

2.2.3.4. Kernel Size

The use of DARK without the communication code adds approximately 70 Kilobytes to the size of the executable module as compared to the standard Ada runtime without DARK.

In summary, the Phase 1 port was not satisfactory from a performance standpoint as compared to the Ada tasking model for single processor operation. As a base for a distributed processor network, this performance may be acceptable.

2.3. Modifications to DARK to Use the Boeing Communications Package

This section describes the modifications to the Distributed Ada Real-Time Kernel (DARK) needed to use the Boeing communications package as part of the Phase 2 plan. The intent was to build on the Phase 1 version of the DARK software while minimizing the changes to the Boeing communications package. This effort was never completed due to the expiration of funding, so this section is a discussion of the design of the modifications needed.

2.3.1. System Architectural Considerations

DARK defines the hardware associated with the distributed network in a user-prepared procedure as "make_NCT." Each communicating piece of hardware in the system architecture (processors, radars, weapons, etc.) is given a logical name (type string); a physical address (bus address); is specified as a Kernel device or not (processors usually are, radars, etc., are not); is specified as needed to run or not; is given an allocated process ID; has its initialization order specified, and sets a flag when initialization is complete. An example of this

is on page 40 of the Kernel User's Manual.[3] The main unit of the software located on any kernel processor executes make_NCT as its first statement. Thus, each kernel processor has an identical view of the architecture of the system. Note that this view is unchangeable.

The DARK communications scheme uses the information in the network configuration table (NCT) in apparently only two ways: The actual bus address is associated with the hardware logical name and the processes on that hardware are thus also associated with that address. Second, if the hardware is defined as a Kernel device, then messages sent to it are embedded in a datagram following the ISO model. Otherwise, the data are sent in raw form.[4]

The Boeing architectural model is described in the package RR_Architectural_Profile, which enumerates the configuration of the nodes in a cluster, and the number of clusters, etc. according to the architecture illustrated in Figure 2-1. This package is "withed" to the communications package and thus forms the basis for the communications package message addressing schemes.

Relative to the system architecture, the changes to DARK can be limited to declaring ALL devices as non-Kernel devices. Thus the datagrams will not be made and the data will be in the form needed by the Boeing communications procedures. The bus addresses should be correctly specified, though it is worth testing to see if they are needed once the Boeing communications procedures are used. If not, then the addresses can be set to some bogus, but benign, numbers and will cease to be impediments to fault tolerance.

If the RR_Architecture_Profile is elaborated normally, then the hardware picture that will be used by the Boeing communications procedures will be available.

2.3.2. Modifications to DARK Communications Procedures

2.3.2.1. Send_Message

The DARK send_message procedure sends a message in the blind and does not block waiting for an acknowledgement. Its parameters are the name of the receiver (which is defined by the user in the main unit as part of processor_a_comm_area), a message tag that can be used by the receiver to decode the message, the length of the message in bytes, and the address of the beginning of the text of the message.

The corresponding Boeing procedure is Put. Its parameters are the logical address of the receiver (an integer constant assigned by calling declare_logical_addressee), the address of the beginning of the message, the number of words, a priority (range 1-256, with 1 being high), and a result (an address of where to put the result).

---

[3]Judy Bamberger et al., *Kernel User's Manual, Version 3.0*, and *Appendix A: Ada Code*, CMU/SEI-UG-1, December 1989.

[4]*Kernel User's Manual*, p. 75.

The Put procedure can be embedded in send_message. The four parameters of send_message can be matched up with the Put parameters as follows:

	DARK	Boeing
receiver:	unique string name	unique integer
message tag:	none needed if non-Ker.	none needed
address:	first byte of message	same
num. of words:	integer	same
result:	none	flag in a certain address

As can be seen, the parameters in send_message give all the information needed for Put, except that the name of the receiver has to be associated with its logical address. The simplest method of doing this is to assign unique string names for the DARK processes using numbers as characters. Then, inside send_message, the function T'VALUE can be used to convert the string parameter to its integer value for use in Put. *Note that this requires careful matching when the processes are assigned names and the logical addressees are declared.*

2.3.2.2. Send_Message_and_Wait

As this procedure may only be used for sending messages to Kernel processes, it is recommended that it not be used.

2.3.2.3. Receive_Message

DARK has a fairly complex receive_message, in that it does functions similar to Boeing's Get, Get_and_Wait, and Get_Cyclic. The procedure can be set for infinite wait, a wait until a certain epoch time, or a wait for a specified elapsed time. These three forms roughly approximate the three Boeing forms of Get. However, these three Boeing procedures are considered "active side services," and receive should be a "passive side service." Unfortunately, the underlying concept of the passive receive is distinctly different. Since DARK assumes that incoming messages will simply be added to a queue to wait for service, there is no protection from overwriting. Boeing's receive can check a number of possible logical addressees for incoming messages, and it avoids overwriting by passively waiting for a message to arrive, and then acting.

DARK's receive has the following parameters:

Sender	out	tells the caller whom the message came from
Message_tag	out	aids in decoding datagram
Message_length	out	

```
Message_buffer   in      where to put the message

Buffer_size      in      size of receiving buffer
```

Additional parameters can be used to indicate resumption priority, a messages lost flag, etc.

The key parameters can be integrated with the Boeing receive in the following way:

```
sender           from the parameter 'activation reason'

message_tag      null, not a Kernel device

message_length   object'size

message_buffer   any defined variable (here is where it is important
                 to match types on both ends)

buffer_size      defined variable'size
```

Everything else can be ignored at this point.

```
Example:

Let us say that process 1234 wants to receive messages
from process 5678.

In the body of process 1234:

        ...
                communications.Declare_Logical Addressee
                                (5678, message_in'address);

        ...

This associates the logical address 5678 with the actual
physical address message_in'address, where message_in is
declared as an object of the desired type.

Later, process 1234 checks to see if there is communication:

        ...
          Receive_message (sender, tag, length, message_in'address,
                                        message_in'size, ...)

        ...

Embedded inside the modified Receive_message is:
```

```
...
  communications.Receive (5678, 10.0, who_called)

...
```

Then the who_called is converted to a string and assigned to sender, and so on.

Note that the additional code needed to use the DARK communications primitives is isolated at the application level, and does not require too much to change in the DARK or Boeing code.

3. Conclusion

Porting DARK to the Boeing-specified architecture proved that the design features of DARK are so well encapsulated that it is possible to make major changes, such as the substitution of the communications package, without destroying the process model that DARK implements. However, if the software were to be used in a genuine real-time environment, it would have to be modified to have some fault tolerance. This may be accomplished by distributing the network configuration table, or using the information in it in a different way.

References

Judy Bamberger et al., *Kernel Facilities Definition*, CMU/SEI-88-TR-16, ESD-88-TR-17, ADA198933, July 1988.

Judy Bamberger et al., *Distributed Ada Real-Time Kernel*, CMU/SEI-88-TR-17, ESD-88-TR-18, ADA199482, August 1988.

Judy Bamberger et al., *Kernel Architecture Manual*, CMU/SEI-89-TR-19, ESD-89-TR-27, ADA219295, December 1989.

Judy Bamberger et al., *Version Description and Installation Guide*, CMU/SEI-89-TR-20, ESD-89-TR-28, ADA192292, December 1989.

Judy Bamberger et al., *DARK Porting and Extension Guide, Kernel Version 3.0*, CMU/SEI-89-TR-40, ESD-89-TR-40, ADA219291, December 1989.

Judy Bamberger et al., *Kernel User's Manual, Version 3.0*, and *Appendix A: Ada Code*, CMU/SEI-UG-1, December 1989.

D. W. Higgins, *Implementing a Fault-Tolerant, Distributed Operating System in Ada*, The Wichita State University Computer Science Department, Technical Report WSU-CS-T-87-5, 1987.

J. E. Kroening, *Logical Addressing: Communications in a Distributed Architecture*, The Wichita State University Computer Science Department, Technical Report WSU-CS-T-87-6, 1987.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS NONE
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE DISTRIBUTION UNLIMITED
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CMU/SEI-90-TR-17	5. MONITORING ORGANIZATION REPORT NUMBER(S) ESD-90-TR-217
6a. NAME OF PERFORMING ORGANIZATION SOFTWARE ENGINEERING INST.	6b. OFFICE SYMBOL (If applicable) SEI
7a. NAME OF MONITORING ORGANIZATION SEI JOINT PROGRAM OFFICE	
6c. ADDRESS (City, State and ZIP Code) CARNEGIE-MELLON UNIVERSITY PITTSBURGH, PA 15213	7b. ADDRESS (City, State and ZIP Code) ESD/XRS1 HANSCOM AIR FORCE BASE HANSCOM, MA 01731
8a. NAME OF FUNDING/SPONSORING ORGANIZATION SEI JOINT PROGRAM OFFICE	8b. OFFICE SYMBOL (If applicable) ESD/XRS1
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F1962890C0003	
8c. ADDRESS (City, State and ZIP Code) CARNEGIE-MELLON UNIVERSITY PITTSBURGH, PA 15213	10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
63752F	N/A	N/A	N/A

11. TITLE (Include Security Classification)
EXPERIENCES PORTING THE DISTRIBUTED ADA REAL-TIME KERNEL

12. PERSONAL AUTHOR(S)
Brian Smith and James E. Tomayko

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
FINAL	FROM _____ TO _____	June 1990	28

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	DARK Real-Time Distributed Ada Real-Time Kernel Processing Real-Time

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The Distributed Ada Real-Time Kernel (DARK) is a mechanism for supporting the execution of distributed real-time Ada applications in embedded computer systems. It provides a solution to scheduling and distributing tasks without modifying the Ada language or vendor-supplied runtime systems. An important test of the utility of the Kernel is whether or not it can be ported to different hardware architectures and still function effectively. As part of an independent research and development project, Boeing Military Airplanes and The Wichita State University became co-acceptors of a copy of DARK for the purpose of demonstrating a port to a 68000-based distributed architecture. This report describes the experiences in accomplishing the port.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ OTIC USERS ☐	UNCLASSIFIED, UNLIMITED DISTRIBUTION

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
KARL H. SHINGLER	412 268-7630	SEI JPO

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

SECURITY CLASSIFICATION OF THIS PAGE