Programming in a Proposed 9X Distributed Ada

REPORT 2

JOHNSON
GRANT
IN-61-CR
115102
P.35

Raymond S. Waldrop
Richard A. Volz
Texas A&M University

Stephen J. Goldsack
A. A. Holzbach-Valero
Imperial College, London, England

May 1991

Cooperative Agreement NCC 9-16
Research Activity No. SE.35

NASA Johnson Space Center
Engineering Directorate
Flight Data Systems Division

ricis

Research Institute for Computing and Information Systems
University of Houston-Clear Lake

INTERIM REPORT

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information Systems (RICIS) in 1986 to encourage the NASA Johnson Space Center (JSC) and local industry to actively support research in the computing and information sciences. As part of this endeavor, UHCL proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a continuing cooperative agreement with UHCL beginning in May 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The UHCL/RICIS mission is to conduct, coordinate, and disseminate research and professional level education in computing and information systems to serve the needs of the government, industry, community and academia. RICIS combines resources of UHCL and its gateway affiliates to research and develop materials, prototypes and publications on topics of mutual interest to its sponsors and researchers. Within UHCL, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business and Public Administration, Education, Human Sciences and Humanities, and Natural and Applied Sciences. RICIS also collaborates with industry in a companion program. This program is focused on serving the research and advanced development needs of industry.

Moreover, UHCL established relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research. For example, UHCL has entered into a special partnership with Texas A&M University to help oversee RICIS research and education programs, while other research organizations are involved via the "gateway" concept.

A major role of RICIS then is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. RICIS, working jointly with its sponsors, advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research and integrates technical results into the goals of UHCL, NASA/JSC and industry.

RICIS Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Raymond S. Waldrop and Richard A. Volz of Texas A&M University and A. A. Holzbacher-Valero and Stephen J. Goldsack of Imperial College, London, England. Dr. E.T. Dickerson served as RICIS research coordinator.

Funding was provided by the Engineering Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between the NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA research coordinator for this activity was Terry D. Humphrey of the Systems Software Section, Flight Data Systems Division, Engineering Directorate, NASA/JSC.

PROGRAMMING IN A PROPOSED 9X DISTRIBUTED ADA

STATUS REPORT 2

by

Raymond S. Waldrop[1]
Richard A. Volz[2]
Stephen J. Goldsack[3]
A. A. Holzbacher-Valero[4]

1 Introduction

This is the second report from the joint project[5] of Texas A&M University and Imperial College to study the proposed Ada 9X constructs for distribution, now referred to as AdaPT. The previous report covered the selection of an example to be used as a basis for this study. The goals for this time period were to revise the chosen example scenario and to begin studying about how the proposed constructs might be implemented.

2 Scenario Specification

The example scenario chosen for this project is the Submarine Combat Information Center (CIC) developed by IBM for the Navy. It is believed to be representative of the kinds of real-time applications that can be found in NASA applications and was accepted by the NASA project manager as the vehicle of study. The specification provided by IBM was preliminary and had several deficiencies relative to our purposes. These deficiences included the following:

- the specifications of the actual operations performed by the time critical processing functions were somewhat vague,
- some timing specifications were incomplete,
- some additional features which would further illustrate the capabilities of the AdaPT extensions were needed.

[1]Texas A&M University
[2]Texas A&M University
[3]Imperial College, London, England
[4]Imperial College, London, England
[5]This work is supported by NASA subcontract #074 Cooperative Agreement NCC-9-16.

To address these problems, we have made some changes to the scenario specification. Some of the more important changes include:

- **Addition of a system database management function.** We felt that the amount of interaction with the system database merited a separate function to oversee this interaction.
- **Addition of a fourth processing unit to the standard resources.** We felt that the amount of interaction with the system database would require the resources of a separate processing unit in order to provide timely service.
- **Addition of an operator console interface function.** We felt that, to be complete, the scenario specification should give some idea about how the operator interacts with the system.
- **Removal of the Time Synchronization function.** We felt that this function did not offer any pertinent research issues and, in any event, since we will only be constructing a simulation, the time synchronization issue is not that important.

Appendix A reflects these changes.

3 Scenario Implementation Strategy

To implement the CIC scenario in AdaPT, we decided upon the following strategy:

- **Publics** would be created to provide all types needed for communication among the CIC functions,
- **Partitions** would be used to implement all major CIC functions, and
- **Nodes** would be used to group the functions onto physical processors.

As part of this strategy, we decided that each dataflow presented in the specification would become a parameter of a procedure call, a function call, or an entry call, and that examples of all three would be used. Since the actual computational workings of the scenario were not of research interest, only the structural specifications of the functions as implemented in AdaPT were seen as necessary. Thus, the internal workings of the functions would be modeled by delays and loops to simulate computation between communications. Finally, after the basic scenario implementation is established, fault-tolerant features will be added.

While designing the implementation of the CIC scenario, we wanted to study all of the various options that are available to a programmer working with AdaPT. For this reason we deliberately selected an implementation which is not consistent throughout in the way these features are used. In particular, we have studied various ways of using:

- **Conformant partitions.** Conformant partitions were used in implementing the Weapon Control function, but not in implementing the Sensor Control functions. The reasoning behind this decision was that the Weapon Control function should provide a sufficient example of the use of conformant partitions, and using conformant partitions for the two Sensor Control functions would add unnecessary complexity to the implementation.
- **Remote procedure calls and remote entry calls.** The AdaPT extensions allow the use of both remote procedure calls (RPC's) and remote entry calls. The semantics of the two forms of remote calls are sufficiently different to justify the use of both forms within our implementation.

4 Status of implementation

The principle purpose for implementing the CIC scenario is to demonstrate how the AdaPT constructs interact with the program structure. Since our purpose is to examine the general structure of the scenario implementation, it is not necessary to actually implement a working system which could be put into service on a naval platform and perform correctly. With this in mind, we plan to implement the scenario only to the extent necessary to model such aspects as communication, synchronization, configuration, and fault tolerance. All other aspects of program operation will be replaced with dummy procedures. As mentioned previously, the general structure is adapted from the IBM specification, as are the parameters for data transfer and transfer rates. The present state of the implementation is as follows:

- The specifications for the partitions implementing the various CIC functions are complete, as are the node specifications and the node bodies. The current version of these specifications is included with this report.
- The bodies of the partitions are partially complete. When all bodies are complete, they will be forwarded as a supplement to this report.
- Fault-tolerance aspects of the implementation will be dealt with once the bodies are complete.

Appendix B contains the specifications for the program.

5 AdaPT Issues

While considering ways that the AdaPT constructs might be translated to Ada 83, it was observed that the **partition** construct could reasonably be modeled as an abstract data type. Although this gives a useful method of modeling **partitions**, it does not at all address the configuration aspects of the **node** construct. Another report, "Transforming AdaPT to Ada," that addresses these issues is being readied and will be submitted shortly.

During the joint group1 and group2 conference held in Orlando, Florida, March 11-13 of this year, the Ada 9X mapping team presented a proposed mapping for the new language. Dr. Volz was present at that workshop, and our research teams have been considering the impact of the proposed mapping on the AdaPT constructs.

A Appendix A : CIC Specification

Several real-time system scenarios were developed for the Navy by IBM. The intent was to provide specifications for systems which could be implemented by researchers studying real-time systems and needing a realistic system for testing. This document adapts the IBM specification for a submarine combat information center. The system is described as on a submarine but is typical of command and control or surveillance systems on submarines, surface ships, land installations, or the space station.

A.1 Combat Information Center Overview

The combat information center provides a focus for all collected tactical information and platform status. The intent is to provide human operators with the information they will need to make decisions. To this end, the information center includes a number of workstations for the use of system operators. Each workstation can access and display any or all of the system data. This information may include everything going on within hundreds of miles. All surface, air, and sub-surface targets may be plotted against true geographical coordinates and topography. A full track history is kept for each target. All displays are kept current[1]. In addition to the workstations, several computers are used as common computational resources for the system. These computers are used to prepare and coordinate system information for display by the workstations.

The amount of data collected, its format (text, audio, or video), and the number of active operators determine the data processing load. Note that an "operator" could be a background processor applying classification or correlation algorithms to the arriving reports.

A.2 CIC Physical Resources

The CIC may include up to twenty display workstations. All the workstations are retrieving data from the main track file. The track file contains the current information on the position and identity of all ships, aircraft, and submarines of interest to the CIC's own platform. Information within the track file is updated many times a second[2]. A single function called Track Update fields the individual changes and prepares the track file for distribution to the workstations.

Each workstation combines the current track information with data from the system database for display. Each display is either static or interactive. On a static display, the new reports are shown on a fixed background. On an interactive display, an operator is actively pulling more information from the system database as new tracks arrive. These are just two illustrative types of displays; many others are possible.

[1] J. L. McClane, "The Ticonderoga Story," U. S. Naval Institute Proceeding, May 1985.

[2] David T. Marlowe, "Requirements for a High Performance Transport Protocol for Use on Naval Platforms," Report No. ANSI X3S3.3 HSP-8, July 23, 1989.

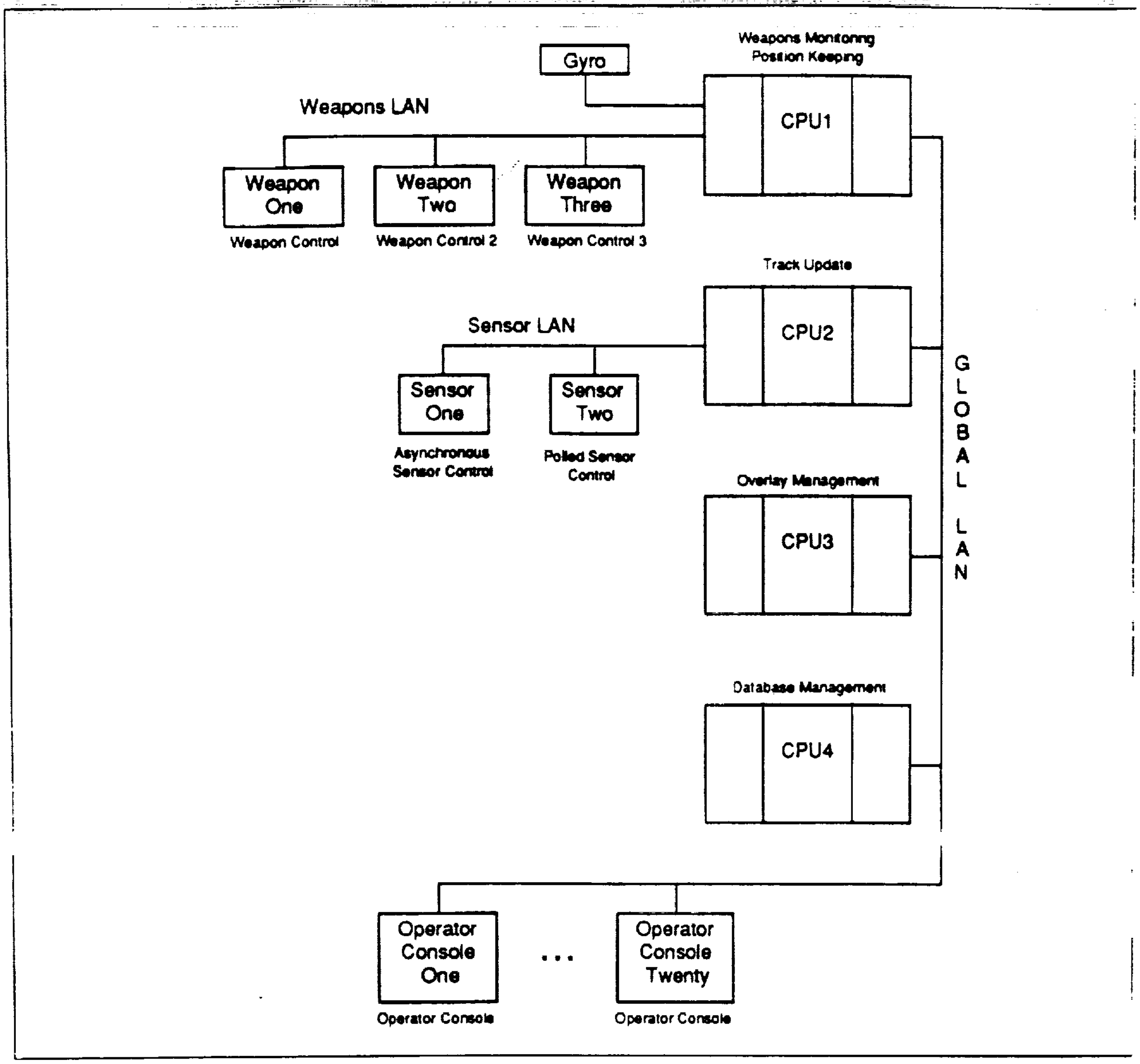

Figure 1: CIC Processing Resources

Figure 1 shows the nominal configuration of standard computer resources for this scenario. The operators' display workstations are attached to the global LAN. The other systems are attached to the standard processors by local networks. These local networks are attached to the standard processors according to the manner in which the CIC functions are distributed among the standard processors. No redundant equipment is shown in the figure. In a real system the external connections would be duplicated and any function that communicates with the other system would be reconfigurable. These duplication and reconfiguration considerations are part of the system's fault-tolerance capabilities, and will be instituted when fault-tolerance is added to the system.

A.3 Data Processing for the CIC

The critical processing functions of interest to our research include:

- Track Update,
- Overlay Management,
- Weapons Monitoring,
- Database Management,
- Operator Console,
- Weapon Control,
- Sensor Control, and
- Position Keeping.

The CIC System Database is controlled by the Database Management function. These modules and the calls and dataflows among them are shown in Figure 2.

The processing and I/O requirements are specified in the following terms:

- *Ns* - the number of different sensor suites.
- *Nc* - the number of operators steering a cursor. Moving the cursor around the screen generates a query for text information. The results appear in a pop-up window.
- *Np* - the number of operators paging. A page selection extracts an image from the database. The operator examines the image and (based on what is found) selects another page.
- *Rp* - the rate an operator flips through pages.

The following sections describe the inputs, outputs, and processing of each of the major modules.

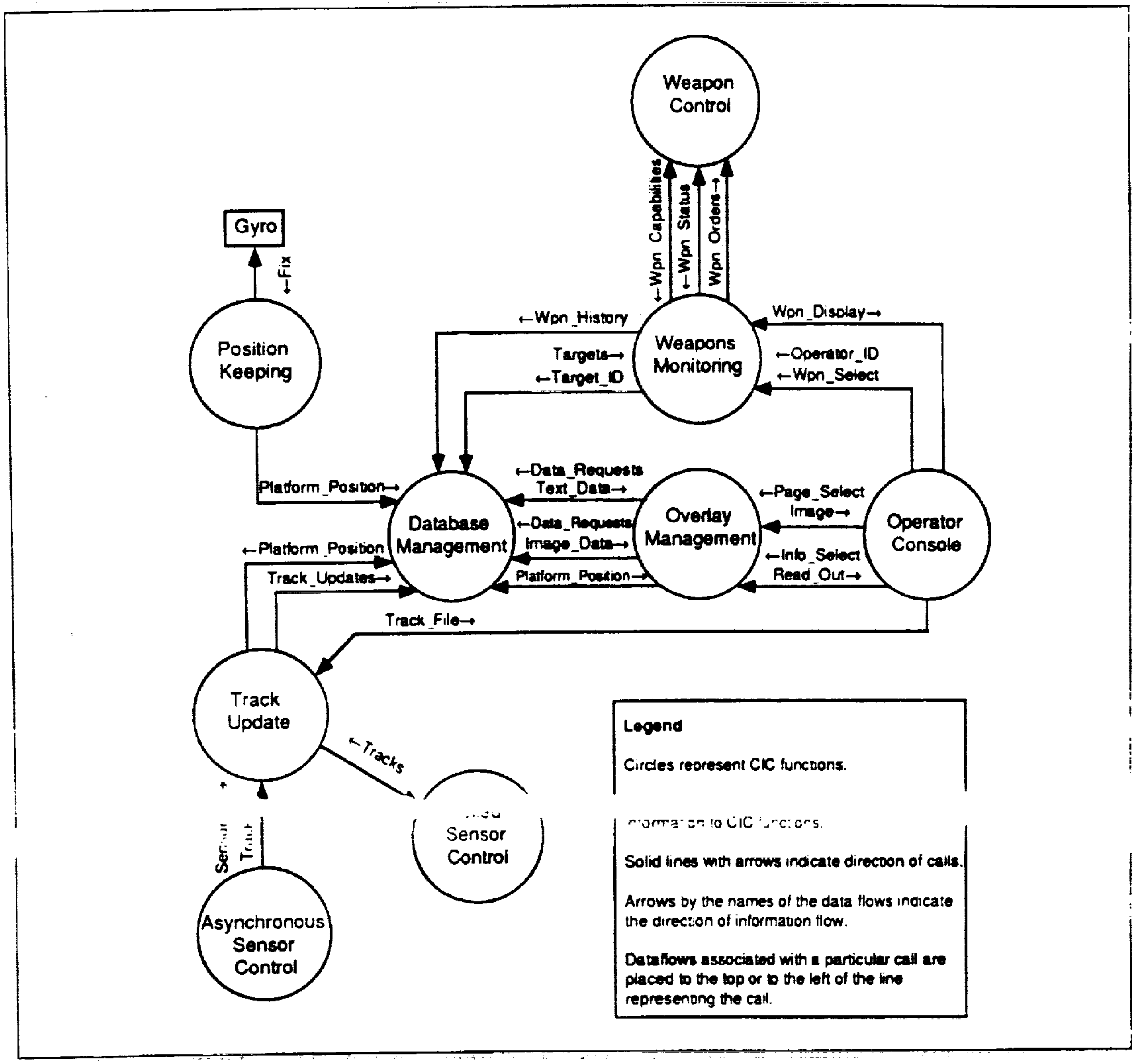

Figure 2: CIC Interfaces

A.3.1 Track Update

The Track Update function (TU) manages the collection and distribution of data on targets of interest.

Input

- Tracks - Track parameters from Asynchronous Sensor and Polled Sensor.
- Platform_Position - Location, velocity, and attitude vectors from Database Management.
- Sensor_ID - Identity of the Asynchronous Sensor which sent Tracks. Not needed for Polled Sensor Control.

Processing

TU shall collect tracks from the different sensor suites, reconcile different sensors tracking the same target, and apply these updates to the track file. TR shall distribute the updated track file on a regular basis. The processing will require $50 * Ns$ KIPS

Timing Requirements

Tracks will be kept current enough to support the Overlay Management and Weapons Monitoring functions.

Output

- Track_File - All current target tracks. These are supplied to the Operator Console functions.
- Track_Updates - Current position of targets of interest. These are supplied to Database Management

A.3.2 Polled Sensor Control

The Polled Sensor Control function (PSC) is responsible for the actual control of a sensor suite. This function will reside on the embedded processor of the sensor suite it controls. This function reports its information to the Track Update function only when the Track Update function requests it.

Input

The only input to this function comes directly from the physical devices it controls, and is therefore not of direct interest at this level of specification.

Processing

This function is responsible for taking the information in its raw form from the actual sensing devices and converting it into a from usable by Track Update.

Timing Requirements

This function was not in the original IBM specification, so no values were given for the timing requirements that must be met. Since our main concerns in this research are the structure, communication, and configuration of the scenario, we will merely provide parameters to the module implementing this function. These parameters can then be given values which result in appropriate behavior of this subsystem.

Output

- Tracks - Track parameters from this sensor suite. Supplied to Track Update.

A.3.3 Asynchronous Sensor Control

The Asynchronous Sensor Control function (ASC) is responsible for the actual control of a sensor suite. This function will reside on the embedded processor of the sensor suite it controls. This function periodically calls the Track Update function to report its results.

Input

The only input to this function comes directly from the physical devices it controls, and is therefore not of direct interest at this level of specification.

Processing

This function is responsible for taking the information in its raw form from the actual sensing devices and converting it into a from usable by Track Update.

Timing Requirements

This function was not in the original IBM specification, so no values were given for the timing requirements that must be met. Since our main concerns in this research are the structure, communication, and configuration of the scenario, we will merely provide parameters to the module implementing this function. These parameters can then be given values which result in appropriate behavior of this subsystem.

Output

- Tracks - Track parameters from this sensor suite. These are supplied to Track Update.
- Sensor_ID - Identity of the instance of this function. Sent with Tracks to Track Update so that function will know where Tracks came from.

A.3.4 Overlay Management

The Overlay Management function (OM) extracts both geographic and intelligence data from the system database for the displays.

Input Data

- Platform_Position - Location, velocity, and attitude vectors from Database Management.
- Text_Data - Information for a textual readout. Supplied by Database Management.
- Image_Data - Information for a page display. Supplied by Database Management.

Data Requests

- Page_Select - Operator's choice of new image. Supplied by Operator Console.
- Info_Select - Operator's choice of new textual readout. Supplied by Operator Console.

Processing

OM shall prepare, retrieve and format system data. Preparing a text readout will require 10 to 100 KI. Preparing a new page will require 20 to 40 KI.

Timing Requirements

Readouts shall be returned to the display within 0.1 seconds. Page images shall be returned within 0.5 seconds. New tracks shall be less than 3 seconds old.

Output

- Readout - Pop-up text for the track picked by the display cursor. Supplied to Operator Console.
- Image - Pixel data formatted for display. Supplied to Operator Console.
- Data_Requests - Queries to the system database for data needed to fulfill requests by Operator Console.

A.3.5 Database Management

The Database Management function (DM) is responsible for maintaining the CIC system database. Although this function did not appear in the original IBM specification, we felt that the amount of traffic to and from the system database warranted a separate function to control access to that data.

Input Data

- Platform_Position - Current position from the Position Keeping function.
- Track_Updates - Changes in track information from Track Update.

- Wpn_History - Running commentary from Weapons Monitoring on weapon status.

Data Requests

- Data_Requests - Queries from Overlay Management for information.
- Target_ID - Identification of a target about which Weapons Monitoring needs more information.

Processing and Timing Requirements

This function was not in the original IBM specification, so no values were given for the amount of processing required or the timing requirements that must be met. Since our main concerns in this research are the structure, communication, and configuration of the scenario, we will merely provide parameters to the module implementing this function. These parameters can then be given values which result in appropriate behavior of this subsystem.

Output

- Platform_Position - The current position of the platform. Supplied to Track Update and Overlay Management.
- Text_Data - Raw textual data for readouts. Supplied to Overlay Management for processing.
- Image_Data - Raw image data for display. Supplied to Overlay Management for processing.
- Targets - Current target positions and characteristics. Supplied to Weapons Monitoring.

A.3.6 Operator Console

The Operator Console function (OC) facilitates the operator's interaction with the rest of the CIC system. To this end, it accepts information requests from the operator, passes on operator commands, and coordinates the information being sent to the operator.

Input

- Track_File - All current target tracks from Track Update.
- Wpn_Display - Display of weapon status and tracks. Supplied by Weapons Monitoring.
- Readout - Pop-up text for the track picked by the display cursor. Supplied by Overlay Management.
- Image - Pixel data formatted for display. Supplied by Overlay Management.

Processing

This function is responsible for merging information from Track Update, Weapons Monitoring, and Overlay Management into a coherent form for presentation to an operator. It must provide the operator with a means of requesting further information and issuing commands. Additionally, this function must translate the operator's commands into requests that can be sent to the appropriate CIC functions.

Timing Requirements

This function was not in the original IBM specification, so no values were given for the timing requirements that must be met. Since our main concerns in this research are the structure, communication, and configuration of the scenario, we will merely provide parameters to the module implementing this function. These parameters can then be given values which result in appropriate behavior of this subsystem.

Output

- Page_Select - Operator's choice of new image. Supplied to Overlay Management.
- Info_Select - Operator's choice of new textual readout. Supplied to Overlay Management.
- Wpn_Select - Operator's choice of weapon configuration. Supplied to Weapons Monitoring.
- Operator_ID - Identity of the operator sending Wpn_Select information.

A.3.7 Weapons Monitoring

The Weapons Monitoring function (WM) provides the data and timing needed by the weapons.

Input

- Wpn_Select - Operator's choice of weapon configuration. Supplied by Operator Console.
- Wpn_Status - Current state of each weapon as reported by Weapon Control.
- Targets - Current position of each target. Supplied by Database Management.
- Operator_ID - Identity of the operator sending Wpn_Select information.

Processing

WM shall pass configuration and aiming commands to the active weapons as directed by the operator's choices. WM shall provide current track data on operator selected targets. WM shall report the status of any active weapons to both the operators' workstations and the system database.

Timing Requirements

Updated tracks shall reach the weapons within 0.5 seconds of being reported by the sensors. Status shall be displayed within 1.0 second of being reported by the weapons.

Output

- Wpn_Display - Display of weapon status and tracks. Supplied to Operator Console.
- Wpn_Orders - Steering, Positioning, and configuration commands. Supplied to Weapon Control.
- Wpn_History - Current and projected state of each weapon. Sent to Database Mangement.
- Target_ID - Identification of a target about which more information is desired.

A.3.8 Weapon Control

The Weapon Control function (WC) is responsible for the actual control of an individual weapon system. This function will reside on the embedded processor of the weapon system to be controlled.

Input

- Wpn_Orders - Steering, positioning, and configuration commands from Weapons Monitoring.

Processing

The Weapon Control function is responsible for translating the operator's commands into the control signals for the weapons systems. Additionally, this function must maintain a record of the current status of the system, and provide this information to Weapons Monitoring.

Timing Requirements

This function was not in the original IBM specification, so no values were given for the timing requirements that must be met. Since our main concerns in this research are the structure, communication, and configuration of the scenario, we will merely provide parameters to the module implementing this function. These parameters can then be given values which result in appropriate behavior of this subsystem.

Output

- Wpn_Status - Current status of this weapon system. Used by Weapons Monitoring.
- Wpn_Capabilities - Provides Weapons Monitoring with information about the capabilities of this particular weapon.

A.3.9 Position Keeping

The Position Keeping function (PK) broadcasts the current location of the platform and hydrophones.

Input

- Fix - Navigational fix obtained from the gyro.

Processing

PK shall compute attitude vectors for the platform and weapons. The resulting position data shall be sent to the database.

Timing Requirements Position information shall reach the weapons within 200 milliseconds of the fix being taken.

Output

- Platform_Position - Location, velocity, and attitude vectors to be stored by Database Management.

A.4 Data Flows for CIC

Table A.4 shows the real time data flow rates and sizes. In addition, the system will have to support occasional file transfers of up to 100 Megabytes in length.

Table 1: The data flows and their rates.

Signal	Description	Rate(Hz)	Size(Bytes)
Info_Select	Choice of new textual readout data.	$2 * Nc$	50
Data_Request	Queries from Overlay Management for data.	$2 * Nc + Rp * Np$	100
Targets	Current target position and characteristics.	0.5	1K
Wpn_Orders	Weapons settings and steering commands.	1	32
Wpn_Display	Current weapons status.	> 1	64
Image_Data	Image retrieved from the database.	$Rp * Np$	1M
Text_Data	Textual data retrieved from the database.	$2 * Nc$	100
Wpn_History	Running commentary on weapon status.	0.1	1K
Wpn_Status	Current weapon state.	1	32
Wpn_Select	Weapon display choices.	Aperiodic	50
Page_Select	Choice of new image.	$Rp * Np$	50
Fix	Input from gyroscope.	16	24
GMT	Greenwich Mean Time from external clock.	1	24
Image	Pixel data ready for display.	$Rp * Np$	1M
Platform_Position	Latitude, longitude, pointing and velocity vectors for the platform.	< 16	32
Readout	Text for cursor readout display.	$2 * Nc$	50
Time	Greenwich Mean Time.	TBD	8
Track_File	The entire current track file.	0.5	1M
Track_Updates	Changes in the tracks from each sensor suite.	Ns	1K
Tracks	Updates to the tracks from each sensor suite.		

B Appendix B : CIC Interface Specifications

```
-- CIC interfaces, version 2.4
-- 9-May-91
-- This document gives the interfaces for the various partitions implementing
-- the functions of the CIC scenario.  The number of expected instantiations
-- of each partition is given.  Also, for each task entry call, the expected
-- rendezvous frequency and the parameter sizes are given according to the
-- information in the IBM specification in Table 2.

-- This document also gives node specs for the processing resources of the
-- CIC scenario.  The corresponding node bodies are filled in to the extent
-- necessary to indicate the desired configuration behavior.
-- The following terms are used in specifying the frequencies of use of the
-- various entry calls:
--      Ns - the number of different sensor suites
--      Nc - the number of operators steering a cursor
--      Np - the number of operators paging
--      Rp - the reate an operator flips through pages

-- Changes since 2.4
-- * change Transfer_Initiating_Sensor_Control to Asynchronous_Sensor_Control.
-- Changes since 2.3
-- * Public Track_TypeDefs
--      * changed TRACK_ID_TYPE to private.
-- * Partition Database_Management
--      * changed Position_Keeping_int.Accept_Position to procedure
--        Accept_Position.
--      * removed task Position_Keeping_int.
--      * changed Overlay_Management_int.Send_Position to procedure
--        Send_Position. (This makes it more general; There should have been a
--        similar entry call in Track_Update_int, but there was not.  Now
--        there is no need.)
--      * changed Overlay_Management_int.Send_Text to procedure Send_Text.
--      * changed Overlay_Management_int.Send_Image to procedure Send_Image.
--      * removed Overlay_Management_int.
-- * renamed Vocal_Sensor_Control to Transfer_Initiating_Sensor_Control.
-- * switched names for CPU2 and CPU3 to bring them into conformance with
--   the Processing Resources diagram in the CIC specification document.
-- Changes since 2.2
-- * Partition Weapon_Control
--      * changed the procedure Send_Capabilities into the function
--        Weapon_Capabilities.
--      * changed the procedure Send_Status into the function Weapon_Status.
--      * changed the procedure Accept_Orders into the entry New_Orders of
--        a new task Commands.
--      * added an entry Priority_Orders for sending orders that would override
--        any orders currently being carried out (e.g. an "abort firing" order)
-- Changes since 2.1
-- * added pragma Distiguished to node CPU4.
-- * added the common frame types.
-- * added two conformant partition types derived from Weapon_Control.
-- * added a procedure to the Weapon_Control partition so each instance can
--   be queried as to the capabilities of the weapon system it controls.
-- Changes since 2.0:
-- * Add publics containing all mentioned types.  Meaningful type definitions
--   will come later (perhaps as the simulator is built).
-- * Changed the name of the Configuration_Interface tasks to CFI.  This
--   change was made to reduce the problems with name length.
-- Changes since 1.2:
-- * Operator_Console partition was changed so that it actively calls other
--   partitions for needed data, e.g. it calls the Track_Update partition for
--   the track file as needed.  This eliminates problems with the other
--   partitions, e.g. Track_Update, having to keep track of all active
--   Operator_Console partitions.
-- * Changed the Weapon_Control partition so that the Accept_Orders entry is
--   replaced by a procedure call.  This included the elimination of a visible
--   Weapons_Monitoring interface task inside this partition.
-- * Removed Position_Keeping interface task from Track_Updates partition
--   because Track Updates is supposed to get the position from the CIC database
-- * Renamed the Sensor_Control partition as Polled_Sensor_Control
-- * Added a Vocal_Sensor_Control partition to allow for sensors which
--   automatically report their data to the Track_Updates function.
-- * Added configuration code, including partition Configuration_Interface
--   tasks and nodes.
------------------------------------------------------------------------
package FRAME_TYPEDEFS is
   type FRAME_32 is array(1..8) of INTEGER;
   type FRAME_64 is array(1..2) of FRAME_32;
   type FRAME_1024 is array(1..32) of FRAME_32;
   type FRAME_1M is array(1..32768) of FRAME_32;

   type FRAME_50 is
      record
         FIELD1 : array(1..12) of INTEGER;
         FIELD2 : array(1..2) of BYTE;
      end record;
   type FRAME_100 is array(1..2) of FRAME_50;
end FRAME_TYPEDEFS;
```

```
---------------------------------------------------------------
-- Note: all type definitions are assuming 32-bit integers.

with FRAME_TYPEDEFS;
public POSITION_TYPEDEFS is
   -- 32 bytes
   type POSITION_TYPE is FRAME_TYPEDEFS.FRAME_32;
end POSITION_TYPEDEFS;

---------------------------------------------------------------
with FRAME_TYPEDEFS;
public WEAPON_TYPEDEFS is
   -- Size: 32 bytes
   -- Variables: Wpn_Orders
   type ORDERS_TYPE is FRAME_TYPEDEFS.FRAME_32;
   -- Size: 32 bytes
   -- Variables: Wpn_Status
   type WPN_STATUS_TYPE is FRAME_TYPEDEFS.FRAME_32;
   -- Size: 32 bytes
   -- Variables: Wpn_Select
   type WPN_SELECT_TYPE is FRAME_TYPEDEFS.FRAME_32;
   -- Size: 64 bytes
   -- Variables: Wpn_Display
   type WPN_INFO_TYPE is FRAME_TYPEDEFS.FRAME_64;
   -- Size: 100 bytes (again, just a shot in the dark)
   -- Variables: Weapon_Capabilities
   type WPN_CAP_TABLE is FRAME_TYPEDEFS.FRAME_100;
   -- Size: 1K
   -- Variables: Wpn_History
   type WPN_HIST_TYPE is FRAME_TYPEDEFS.FRAME_1024;
   -- Size: 1K
   -- Variables: Targets
   type TARGETS_TYPE is FRAME_TYPEDEFS.FRAME_1024;
end WEAPON_TYPEDEFS;

---------------------------------------------------------------
with FRAME_TYPEDEFS;
public TRACK_TYPEDEFS is
   -- Size: 32 bytes
   --      This seems to be their most-used size, so it seems
   --      reasonable to use it and assume it contains additional
   --      information about the sensor (besides a simple number).
   -- Variables: Sensor_ID
   type SENSOR_ID_TYPE is private;
   -- Size: 32 bytes (See above.)
   -- Variables: Target_ID
   type TRACK_ID_TYPE is private;
   -- Size: 1K
   -- Variables: Track_Updates
   type TRACK_INFO_TYPE is FRAME_TYPEDEFS.FRAME_1024;
   -- Size: 1K
   --          This size not provided in specs, so Track_Updates is used
   --          as a guide.
   -- Variables: Tracks
   type TRACKS_TYPE is FRAME_TYPEDEFS.FRAME_1024;
   -- Size: 1M
   -- Variables: Track_File
   type TRACK_FILE_TYPE is FRAME_TYPEDEFS.FRAME_1M;

   function NEXT_ID(OLD_ID : SENSOR_ID_TYPE) return SENSOR_ID_TYPE;

private
   type SENSOR_ID_TYPE is FRAME_TYPEDEFS.FRAME_32;
   type TRACK_ID_TYPE is FRAME_TYPEDEFS.FRAME_32;
end TRACK_TYPEDEFS;

public body TRACK_TYPEDEFS is
   function NEXT_ID(OLD_ID : SENSOR_ID_TYPE) return SENSOR_ID_TYPE is
   begin
      -- code to compute the next ID given the current ID
   end NEXT_ID;
end TRACK_TYPEDEFS;

------------------------------------------------------------------
with FRAME_TYPEDEFS;
public OVERLAY_TYPEDEFS is
   -- Size: 100 bytes
   -- Variables: Data_Request
   type REQUEST_TYPE is FRAME_TYPEDEFS.FRAME_100;
   -- Size: 50 bytes
   -- Variables: Page_Select
   type PAGE_ID_TYPE is FRAME_TYPEDEFS.FRAME_50;
   -- Size: 50 bytes
   -- Variables: Info_Select
   type READOUT_ID_TYPE is FRAME_TYPEDEFS.FRAME_50;
   -- Size: 100 bytes
   -- Variables: Text_Data
   type RAW_TEXT_TYPE is FRAME_TYPEDEFS.FRAME_100;
   -- Size: 50 bytes
   -- Variables: Readout
   type READOUT_TYPE is FRAME_TYPEDEFS.FRAME_50;
   -- Size: 1M
   -- Variables: Image_Data
   type RAW_IMAGE_TYPE is FRAME_TYPEDEFS.FRAME_1M;
   -- Size: 1M
   -- Variables: Image
   type IMAGE_TYPE is FRAME_TYPEDEFS.FRAME_1M;
end OVERLAY_TYPEDEFS;

------------------------------------------------------------------

with FRAME_TYPEDEFS;
public OPERATOR_TYPEDEFS is
   -- Size: 32 bytes  (Same arguments as for Sensor_ID_Type.)
   -- Variables: Operator_ID
   type OPERATOR_ID_TYPE is private;

   function NEXT_ID(OLD_ID : OPERATOR_ID_TYPE) return OPERATOR_ID_TYPE;

private
   type OPERATOR_ID_TYPE is FRAME_TYPEDEFS.FRAME_32;
end OPERATOR_TYPEDEFS;

public body OPERATOR_TYPEDEFS is
   function NEXT_ID(OLD_ID : OPERATOR_ID_TYPE) return OPERATOR_ID_TYPE is
   begin
      -- code to return next ID given the current ID
   end NEXT_ID;
end OPERATOR_TYPEDEFS;

------------------------------------------------------------------
-- This partition implements the Database Management function DM.
-- This function was not in the original IBM spec, but we felt that it was
-- reasonable to include such a function given the amount of interaction
-- with the system database.
-- There will be one instantiation of this partition.
with POSITION_TYPEDEFS, WEAPON_TYPEDEFS, TRACK_TYPEDEFS, OVERLAY_TYPEDEFS;
use  POSITION_TYPEDEFS, WEAPON_TYPEDEFS, TRACK_TYPEDEFS, OVERLAY_TYPEDEFS;
partition DATABASE_MANAGEMENT is
   -- Each function that must call the DATABASE_MANAGEMENT partition is
   -- served by a separate interface task. Depending on the implementation,
   -- this may yield more predictable behavior than procedure and
   -- function calls because using tasks explicitly specifies a
   -- separate thread of control, whereas procedures and functions
   -- implicitly create separate threads when they are called
-- by remote entities.  (The RPC mechanism may or may not have to create
-- a new thread from scratch, while a task should aready be in
-- existence.  The task therefore seems to offer a greater chance of
-- predictable behavior.)
-- This procedure allows the current position of the platform to be
-- updated.
-- Frequency: <16Hz
-- Parameter size: Platform_Position => 32 bytes
-- NOTE: we are using the term Platform_Position to refer to the dataflow
--          called Position in the IBM specification.  This is to help
--          prevent confusion between data giving the platform's position
--          and data giving target positions.
procedure ACCEPT_POSITION(PLATFORM_POSITION : in POSITION_TYPE);
-- This procedure returns the current platform position.
    -- Parameter size: Platform_Position => 32 bytes
procedure SEND_POSITION(PLATFORM_POSITION : out POSITION_TYPE);

-- This is the Weapons Monitoring interface task.
task WEAPONS_MONITORING_INT is
    -- Frequency: 0.1Hz
    -- Parameter size: Wpn_History => 1K
    entry ACCEPT_WPN_DATA(WPN_HISTORY : in WPN_HIST_TYPE);
    -- Frequency: 0.5Hz
    -- Parameter size: Targets => 1K
    -- NOTE: we are not satisfied with the IBM specification's definition of
    --          the Targets dataflow.  It seems to be intended to give a
    --          weapon controller information about the target it is currently
    --          working with, yet the dataflow has a plural name.  We have
    --          thought that the name might imply that information might be
    --          passed about a group or class of targets.  If so, another entry
    --          could be provided to allow for group information to supplement
    --          the current entry which provides information about a single
    --          target.
    entry SEND_TARGETS(TARGET_ID : in TRACK_ID_TYPE;
                       TARGETS : out TARGETS_TYPE);
end WEAPONS_MONITORING_INT;
-- This is the Track Update interface task.
task TRACK_UPDATE_INT is
    -- Frequency: Ns
    -- Parameter size: Track_Updates => 1K
    entry ACCEPT_TRACKS(TRACK_UPDATES : in TRACK_INFO_TYPE);
end TRACK_UPDATE_INT;
-- This is the Overlay Management interface section.

    -- Although the IBM specification has a single dataflow called
```

```
      -- DataRequest, this dataflow carries requests for both text data and
      -- image data. We have left the name the same in both the following
      -- entries, but the dataflow might as well be split into two separate
      -- dataflows called Text_Requests and Image_Requests respectively.
      -- Frequency: 2*Nc (c.f. the frequency for the Text_Data dataflow)
      -- Parameter size: Data_Request => 100 bytes, Text_Data => 100 bytes
   procedure SEND_TEXT (DATA_REQUEST : in REQUEST_TYPE;
                        TEXT_DATA : out RAW_TEXT_TYPE);
      -- Frequency: Rp*Np (c.f. the frequency for the Image_Data dataflow)
      -- Parameter size: Data_Request => 100 bytes, Image_Data => 1M
   procedure SEND_IMAGE(DATA_REQUEST : in REQUEST_TYPE;
                        IMAGE_DATA : out RAW_IMAGE_TYPE);
end DATABASE_MANAGEMENT;
------------------------------------------------------------------------
-- This partition implements the Overlay Management function OM.
-- There will be one instantiation of this partition.
with OVERLAY_TYPEDEFS;
use  OVERLAY_TYPEDEFS;
with DATABASE_MANAGEMENT;
use  DATABASE_MANAGEMENT;
partition OVERLAY_MANAGEMENT is

   -- This task is used for configuring the Overlay Manager to use a particular
   -- instance of the Database_Management.
   task CFI is
      entry ASSIGN_DATABASE(DATABASE : in DATABASE_MANAGEMENT);
   end CFI;

   -- Procedure calls are used here to eliminate the queueing and blocking
   -- of requests that are associated with the use of task entry calls.

      -- Caller: Operator_Console
      -- Frequency: Rp*Np
      -- Parameter size: Page_Select => 50 bytes
      --                       Image => 1M
   procedure GET_IMAGE(PAGE_SELECT : in PAGE_ID_TYPE;
                       IMAGE : out IMAGE_TYPE);
      -- Caller: Operator_Console
      -- Frequency: 2*Nc
      -- Parameter size: Info_Select => 50 bytes
      --                       Readout => 50 bytes
   procedure GET_READOUT(INFO_SELECT : in READOUT_ID_TYPE;
                         READOUT : out READOUT_TYPE);
end OVERLAY_MANAGEMENT;
------------------------------------------------------------------------
```

```
-- This partition implements the Weapon Control function WC.
-- It must be polled by the Weapons Monitoring function.  It
-- is intended to reside on the embedded processor of a weapon system
-- There will be multiple instantiations of this parition, each residing
-- on the embedded processor of one of the weapons systems.
-- For our simulation, one instance of this partition should be sufficient.
with WEAPON_TYPEDEFS;
use  WEAPON_TYPEDEFS;
partition WEAPON_CONTROL is

   task COMMANDS is
      -- Caller: Weapons Monitoring
      -- Frequency: 1Hz
      -- Parameter size: Wpn_Orders => 32 bytes
      entry NEW_ORDERS(WPN_ORDERS : in ORDERS_TYPE);
      -- Caller: Weapons Monitoring
      -- Frequency: aperiodic (used only for special situations)
      -- Parameter size: Wpn_Orders => 32 bytes
      entry PRIORITY_ORDERS(WPN_ORDERS : in ORDERS_TYPE);
   end COMMANDS;
      -- Caller: Weapons Monitoring
      -- Frequency: 1Hz
      -- Parameter size: Wpn_Status => 32 bytes
   function WPN_STATUS return WPN_STATUS_TYPE);

      -- Caller: Weapons Monitoring
      -- Frequency: aperiodic (when Weapons Monitoring is informed of this
      --                          controler)
   function WPN_CAPABILITIES return WPN_CAP_TABLE;
end WEAPON_CONTROL;

----------------------------------------------------------------
with WEAPON_TYPEDEFS;
use  WEAPON_TYPEDEFS;
partition WEAPON_CONTROL2 is new WEAPON_CONTROL;

with WEAPON_TYPEDEFS;
use WEAPON_TYPEDEFS;
PARTITON WEAPON_CONTROL3 is new WEAPON_CONROL;
----------------------------------------------------------------
-- This partition implements the Weapons Monitoring function WM.
-- There will be one instantiation of this partition.
with WEAPON_TYPEDEFS;
use  WEAPON_TYPEDEFS;
```

```
with WEAPON_CONTROL, WEAPON_CONROL2, WEAPON_CONTROL3, DATABASE_MANAGEMENT;
use  WEAPON_CONTROL, WEAPON_CONROL2, WEAPON_CONTROL3, DATABASE_MANAGEMENT;
partition WEAPONS_MONITORING is

   -- This task is used by the node to inform the Weapons Monitoring function
   -- of available Weapon Control partitions as well as the system database.
   task CFI is
      entry ACCEPT_NEW_WEAPON(WEAPON : in WEAPON_CONTROL);
      entry REMOVE_WEAPON(WEAPON : in WEAPON_CONTROL);
      entry ACCEPT_DATABASE(DATABASE : in DATABASE_MANAGEMENT);
   end CFI;

      -- Caller: Operator_Console
      -- Frequency: Aperiodic
      -- Parameter size: Wpn_Select => 50 bytes
   procedure ACCEPT_WPN_CONFIG(WPN_SELECT : in WPN_SELECT_TYPE;
                               OPERATOR_ID : in OPERATOR_ID_TYPE);

      -- Caller: Operator_Console
      -- Frequency: >1Hz
      -- Parameter size: Wpn_Display => 64 bytes
   procedure REPORT_WPN_STATE(WPN_DISPLAY : out WPN_INFO_TYPE);

end WEAPONS_MONITORING;

------------------------------------------------------------
-- This partition implements the Sensor Control function SC.  This partition
-- must be polled by the Track_Update function when data is desired.
-- There will be multiple instatiations of this partition, each residing
-- on the embedded processor of one of the sensor systems.
-- For our simulation, one instance of this partition should be sufficient.
with TRACK_TYPEDEFS;
use  TRACK_TYPEDEFS;
partition POLLED_SENSOR_CONTROL is
      -- Caller: Track Update
      -- Frequency: not given
      -- Parameter size: Tracks => not given
   procedure SEND_DATA(TRACKS : out TRACKS_TYPE);
end POLLED_SENSOR_CONTROL;
------------------------------------------------------------
-- This partition implements the Track Update function TU.
-- There will be one instantiation of this partition.
with TRACK_TYPEDEFS;
use  TRACK_TYPEDEFS;
with POLLED_SENSOR_CONTROL, DATABASE_MANAGEMENT;
```

```
use  POLLED_SENSOR_CONTROL, DATABASE_MANAGEMENT;
partition TRACK_UPDATE is
begin

    -- This task is used by the node to inform the Track Update function about
    -- the sensors it will be working with.  Because of the semantics of AdaPT,
    -- the Track_Update partition cannot be given pointers to
    -- Asynchronous_Sensor_Control partitions, since those partitions
    -- must be given a pointer to Track_Update in order to make their reports.
    -- The node also uses this task to give Track_Update a pointer to the
    -- system database.
    task CFI is
        entry ACCEPT_POLLED_SENSOR(SENSOR : in POLLED_SENSOR_CONTROL;
                                   SENSOR_ID : in SENSOR_ID_TYPE);
        entry REMOVE_POLLED_SENSOR(SENSOR : in POLLED_SENSOR_CONTROL);
        entry ACCEPT_ASYNCHRONOUS_SENSOR_ID(SENSOR_ID : in SENSOR_ID_TYPE);
        entry REMOVE_ASYNCHRONOUS_SENSOR_ID(SENSOR_ID : in SENSOR_ID_TYPE);
        entry ACCEPT_DATABASE(DATABASE : in DATABASE_MANAGEMENT);
    end CFI;

    -- This is the Operator_Console interface task.
    -- A task was used so that requests for track files can be interleaved
    -- with updates to the master track file.
    task OPERATOR_CONSOLE_INT is
        -- Frequency: 0.5Hz
        -- Parameter size: Track_File => 1M
        entry EXPORT_TRACK_FILE(TRACK_FILE : out TRACK_FILE_TYPE);
    end OPERATOR_CONSOLE_INT;

        -- Caller: Asynchronous_Sensor_Control;
        -- Frequency: not given
        -- Parameter size: Tracks => not given
        --                 Sensor => ??
    procedure ACCEPT_SENSOR_DATA(TRACKS : in TRACKS_TYPE;
                                 SENSOR_ID : SENSOR_ID_TYPE);

end TRACK_UPDATE;

---------------------------------------------------------------
-- This partition implements the Sensor Control function SC.  It sends its
-- data to the Track_Update partition as needed.
-- There will be multiple instatiations of this partition, each residing
-- on the embedded processor of one of the sensor systems.
-- For our simulation, one instance of this partition should be sufficient.
with TRACK_TYPEDEFS;
```

```
use   TRACK_TYPEDEFS;
with TRACK_UPDATE;
use   TRACK_UPDATE;
partition ASYNCHRONOUS_SENSOR_CONTROL is

    -- This task is used by the node to set up pointers to the Track Update
    -- function.
    task CFI is
        entry ACCEPT_TRACK_UPDATE(UPDATER : in TRACK_UPDATE);
        entry ACCEPT_SENSOR_ID(SENSOR_ID : in SENSOR_ID_TYPE);
    end CFI;

end ASYNCHRONOUS_SENSOR_CONTROL;
---------------------------------------------------------------------
-- This partition implements the Operator Console function OC.
-- There will be multiple instantiations of this partition, each
-- residing on one of the operators' workstations.
-- To conform to the IBM specification we will need 20 of these partitions.
-- The body of this partition should contain calls to other partitions to
-- obtain the data needed by this function.  The decision was made to have
-- this partition actively seek the necessary information because of the
-- difficulties involved in having all partitions communicating with the
-- operators keep track of the current status of all instances of this
-- partition.

with OVERLAY_TYPEDEFS, TRACK_TYPEDEFS, WEAPON_TYPEDEFS, OPERATOR_TYPEDEFS;
with OVERLAY_MANAGEMENT, TRACK_UPDATE, WEAPONS_MANAGEMENT;
use OVERLAY_TYPEDEFS, TRACK_TYPEDEFS, WEAPON_TYPEDEFS, OPERATOR_TYPEDEFS;
use OVERLAY_MANAGEMENT, TRACK_UPDATE, WEAPONS_MANAGEMENT;
partition OPERATOR_CONSOLE is

    -- The node uses this task to give pointers to the partitions which must
    -- be called.
    task CFI is
        entry ACCEPT_OVERLAYER(OVERLAY_MANAGER : in OVERLAY_MANAGEMENT);
        entry ACCEPT_UPDATER(UPDATER : in TRACK_UPDATE);
        entry ACCEPT_WPN_MONITOR(WPN_MONITOR : in WEAPONS_MANAGEMENT);
        entry ACCEPT_OPERATOR_ID(OPERATOR_ID : in OPERATOR_ID_TYPE);
    end CFI;

end OPERATOR_CONSOLE;

---------------------------------------------------------------------
-- This partition implements the Position Keeping function PK.
-- It is not called by anything else.
```

```
with POSITION_TYPEDEFS;
use  POSITION_TYPEDEFS;
partition POSITION_KEEPING is
   -- This task is used by the node to set up a pointer to the system
   -- database.
   task CFI is
      entry ACCEPT_DATABASE(DATABASE : in DATABASE_MANAGEMENT);
   end CFI;
end POSITION_KEEPING;

-----------------------------------------------------------------
                 ---------------------------------
                 -- node definitions below: --
                 ---------------------------------
-----------------------------------------------------------------

-- with convention:
--      Partitions to be created locally
--      Partitions which must be known about
--      Nodes which must be known about or created by this node

with WEAPON_CONTROL;
use  WEAPON_CONTROL;
node EMBEDDED_WEAPON_CPU is
   function MY_CONTROLLER return WEAPON_CONTROL;
end  EMBEDDED_WEAPON_CPU;

node body EMBEDDED_WEAPON_CPU is
   CONTROLLER : WEAPON_CONTROL := new WEAPON_CONTROL'PARTITION;

   function MY_CONTROLLER return WEAPON_CONTROL is
   begin
      return CONTROLLER;
   end MY_CONTROLLER;
end EMBEDDED_WEAPON_CPU;

-----------------------------------------------------------------
with POSITION_KEEPING, WEAPONS_MANAGEMENT;
use  POSITION_KEEPING, WEAPONS_MANAGEMENT;
with DATABASE_MANAGEMENT, WEAPON_CONTROL;
use  DATABASE_MANAGEMENT, WEAPON_CONTROL;
with EMBEDDED_WEAPON_CPU;
use  EMBEDDED_WEAPON_CPU;
node CPU1(NUMBER_OF_WEAPONS : INTEGER; DATABASE : DATABASE_MANAGEMENT) is
   function MY_POSITIONER return POSITION_KEEPING;
```

```
    function MY_WPN_MONITOR return WEAPONS_MONITORING;
end  CPU1;

node body CPU1(NUMBER_OF_WEAPONS : INTEGER; DATABASE : DATABASE_MANAGEMENT) is
    POSITIONER : POSITION_KEEPING := new POSITION_KEEPING'PARTITION;
    WEAPON_MONITOR : WEAPONS_MONITORING := new WEAPONS_MONITORING'PARTITION;
    EMBEDDED_WEAPONS : array[1..NUMBER_OF_WEAPONS] of EMBEDDED_WEAPON_CPU
                     := (others => EMBEDDED_WEAPON_CPU'NODE);

    function MY_POSITIONER return POSITION_KEEPING is
    begin
        return POSITIONER;
    end MY_POSITIONER;

    function MY_WPN_MONITOR return WEAPONS_MONITORING is
    begin
        return WEAPON_MONITOR;
    end MY_WPN_MONITOR;
begin
    -- Set up the Weapons_Monitoring partition with its initial values.
    PASS_WEAPONS_CONTROLLERS:
    for I in 1..NUMBER_OF_WEAPONS loop
        WEAPON_MONITOR.CFI.ACCEPT_WEAPON
                (EMBEDDED_WEAPONS[I].MY_CONTROLLER);
    end loop PASS_WEAPONS_CONTROLLERS;
    WEAPON_MONITOR.CFI.ACCEPT_DATABASE
                (PROC4.SYSTEM_DATABASE);
end CPU1;

-----------------------------------------------------------------
with ASYNCHRONOUS_SENSOR_CONTROL;
use  ASYNCHRONOUS_SENSOR_CONTROL;
with TRACK_UPDATE;
use  TRACK_UPDATE;
node ASYNCHRONOUS_SENSOR_CPU(UPDATER : TRACK_UPDATE) is
    function MY_SENSOR return SENSOR;
end  ASYNCHRONOUS_SENSOR_CPU;

node body ASYNCHRONOUS_SENSOR_CPU(UPDATER : TRACK_UPDATE) is
    SENSOR : ASYNCHRONOUS_SENSOR_CONTROL
           := new ASYNCHRONOUS_SENSOR_CONTROL'PARTITION;

    function MY_SENSOR return SENSOR is
    begin
        return SENSOR;
```

```
   end MY_SENSOR;
begin
   SENSOR.CFI.ACCEPT_UPDATER(UPDATER);
end ASYNCHRONOUS_SENSOR_CPU;

-------------------------------------------------------------

with POLLED_SENSOR_CONTROL;
use  POLLED_SENSOR_CONTROL;
node POLLED_SENSOR_CPU is
   function MY_SENSOR return POLLED_SENSOR_CONTROL;
end POLLED_SENSOR;

node body POLLED_SENSOR_CPU is
   SENSOR : POLLED_SENSOR_CONTROL := new POLLED_SENSOR_CONTROL'PARTITION;

   function MY_SENSOR return POLLED_SENSOR_CONTROL is
   begin
      return SENSOR;
   end MY_SENSOR;
end POLLED_SENSOR_CPU;
--updated through here

-------------------------------------------------------------

with TRACK_TYPEDEFS;
use  TRACK_TYPEDEFS;
with TRACK_UPDATE;
use  TRACK_UPDATE;
with ASYNCHRONOUS_SENSOR_CONTROL, POLLED_SENSOR_CONTROL,
       DATABASE_MANAGEMENT;
use  ASYNCHRONOUS_SENSOR_CONTROL, POLLED_SENSOR_CONTROL,
       DATABASE_MANAGEMENT;
with ASYNCHRONOUS_SENSOR_CPU, POLLED_SENSOR_CPU;
use  ASYNCHRONOUS_SENSOR_CPU, POLLED_SENSOR_CPU;
node CPU2(NUM_INITIATORS, NUM_POLLED : INTEGER;
             DATABASE : DATABASE_MANAGEMENT) is
   function MY_UPDATER return TRACK_UPDATE;
end  CPU2;

node body CPU2(NUM_INITIATORS, NUM_POLLED : INTEGER;
                  DATABASE : DATABASE_MANAGEMENT) is
   UPDATER : TRACK_UPDATE := new TRACK_UPDATE'PARTITION;
   INITIATORS : array[1..NUM_INITIATORS] of ASYNCHRONOUS_SENSOR_CPU
           := (others => new ASYNCHRONOUS_SENSOR_CPU'NODE);
   POLLED_SENSOR_SYSTEMS : array[1..NUM_POLLED] of POLLED_SENSOR_CPU
           := (others => new POLLED_SENSOR_CPU'NODE);
   ID : SENSOR_ID;
   ASYNCHRONOUS_SENSOR : ASYNCHRONOUS_SENSOR_CONTROL;
   POLLED_SENSOR : POLLED_SENSOR_CONTROL;

   function MY_UPDATER return TRACK_UPDATE is
   begin
      return UPDATER;
   end MY_UPDATER;

begin
   UPDATER.CFI.ACCEPT_DATABASE(DATABASE);
   INITIATOR_SETUP:
   for I in 1..NUM_INITIATORS loop
      ASYNCHRONOUS_SENSOR := INITIATORS[I].MY_SENSOR;
      ASYNCHRONOUS_SENSOR.CFI.ACCEPT_ID(ID);
      ASYNCHRONOUS_SENSOR.CFI.ACCEPT_UPDATER(UPDATER);
      UPDATER.CFI.ACCEPT_ASYNCHRONOUS_SENSOR_ID(ID);
      ID := TRACK_TYPEDEFS.NEXT_ID(ID);
   end loop INITIATOR_SETUP;
   POLLED_SETUP:
   for I in 1..NUM_POLLED loop
      POLLED_SENSOR := POLLED_SENSOR_SYSTEMS[I].MY_SENSOR;
      POLLED_SENSOR.CFI.ACCEPT_ID(ID);
      POLLED_SENSOR.CFI.ACCEPT_UPDATER(UPDATER);
      UPDATER.CFI.ACCEPT_POLLED_SENSOR(POLLED_SENSOR,ID);
      ID := TRACK_TYPEDEFS.NEXT_ID(ID);
   end loop POLLED_SETUP;
end CPU2;

-----------------------------------------------------------------

with OVERLAY_MANAGEMENT;
use  OVERLAY_MANAGEMENT;
with DATABASE_MANAGEMENT;
use  DATABASE_MAGAGEMENT;
node CPU3(DATABASE : DATABASE_MANAGEMENT) is
   function MY_OVERLAYER return OVERLAY_MANAGEMENT;
end  CPU3;
node body CPU3(DATABASE : DATABASE_MANAGEMENT) is
   OVERLAYER : OVERLAY_MANAGEMENT := new OVERLAY_MANAGEMENT'PARTITION;
   function MY_OVERLAYER return OVERLAY_MANAGEMENT is
   begin
      return OVERLAYER;
   end MY_OVERLAYER;
begin
```

```
   OVERLAYER.CFI.ACCEPT_DATABASE(DATABASE);
end CPU3;

------------------------------------------------------------
with OPERATOR_CONSOLE;
use  OPERATOR_CONSOLE;
node OPERATOR_WORKSTATION is
   function MY_CONSOLE return OPERATOR_CONSOLE;
end  OPERATOR_WORKSTATION;

node body OPERATOR_WORKSTATION is
   CONSOLE : OPERATOR_CONSOLE := OPERATOR_CONSOLE'PARTITION;

   function MY_CONSOLE return OPERATOR_CONSOLE is
   begin
      return CONSOLE;
   end MY_CONSOLE;
end OPERATOR_WORKSTATION;

------------------------------------------------------------
with DATABASE_MANAGEMENT;
use  DATABASE_MANAGEMENT;
with TRACK_UPDATE, WEAPONS_MONITORING, OVERLAY_MANAGEMENT, OPERATOR_CONSOLE;
use  TRACK_UPDATE, WEAPONS_MONITORING, OVERLAY_MANAGEMENT, OPERATOR_CONSOLE;
with CPU1, CPU2, CPU3, OPERATOR_WORKSTATION;
use  CPU1, CPU2, CPU3, OPERATOR_WORKSTATION;
node CPU4 is
pragma DISTINGUISHED(CPU4);
end;

with OPERATOR_TYPEDEFS;
node body CPU4 is
   DATABASE : DATABASE_MANAGEMENT := new DATABASE_MANAGEMENT'PARTITION);
   PROC2 : CPU2 := new CPU2'NODE(1,1,DATABASE);
   PROC1 : CPU1 := new CPU1'NODE(2,DATABASE);
   PROC3 : CPU3 := new CPU3'NODE(DATABASE);
   OPERATORS : array[1.20] of OPERATOR_WORKSTATION
         := (others => new OPERATOR_WORKSTATION'NODE);

   UPDATER : TRACK_UPDATE;
   OVERLAYER : OVERLAY_MANAGEMENT;
   WPN_MONITOR : WEAPONS_MONITORING;
   CONSOLE : OPERATOR_CONSOLE;
   ID : OPERATOR_ID;
```

```
begin
    UPDATER := PROC2.MY_UPDATER;
    OVERLAYER := PROC3.MY_OVERLAYER;
    WPN_MONITOR := PROC1.MY_WPN_MONITOR;
    CONSOLE : OPERATOR_CONSOLE;

    OVERLAYER.CFI.ACCEPT_DATABASE(DATABASE);
    CONSOLE_SETUP:
    for I in 1..20 loop
        CONSOLE := OPERATORS[I].MY_CONSOLE;
        CONSOLE.CFI.ACCEPT_DATABASE(DATABASE);
        CONSOLE.CFI.ACCEPT_UPDATER(UPDATER);
        CONSOLE.CFI.ACCEPT_WPN_MONITOR(WPN_MONITOR);
        CONSOLE.CFI.ACCEPT_OPERATOR_ID(ID);
        ID := OPERATOR_TYPEDEFS.NEXT_ID(ID);
    end loop CONSOLE_SETUP;
end CPU4;
```