# REENGINEERING AN AVIONICS SOFTWARE SYSTEM

Raghava G. Gowda, Ph.D.
Department of Computer Science, University of Dayton
Barbara Eldridge
Michael Bohler
Avionics Directorate, Wright-Patterson Air Force Base

## Abstract

This paper deals with the experiences in reengineering an existing system and developing an object-oriented design for the current system. The subsystem under study is the Terrain Following Algorithm (TFA) or the Vertical Steering subsystem, a component of the Guidance system of the Operational Flight Program (OFP). The OFP is a major subsystem of the Integrated Test Bed (ITB) facility used to support the development, test, and evaluation of advanced avionics systems in the Avionics Directorate at Wright-Patterson Air Force Base.

The techniques used in this reengineering process included Structure Charts, Object-Oriented Structured Design (OOSD) notation, and Visibility Diagrams for Ada packages. For developing an object-oriented design, the Object Modeling Technique (OMT) by Rumbaugh et al. was used. Some of the CASE tools used for reengineering were Software Through Pictures™, Rational Rose™, and OMTool™.

[The research was sponsored by the 1994 Summer Faculty Research Program of the Air Force Office of Scientific Research, Bolling Air Force Base, DC and Armstrong Laboratory. Research Site: Systems Group (WL/AAAS-3), Avionics Directorate, Wright-Patterson Air Force Base.]

## An Overview of the System

The subsystem selected for object-oriented modeling was the Guidance subsystem which consists of the Terrain-Aided Flight Algorithm (TFA). This system has been modified/expanded and implemented in the Ada package VERTICAL_STEERING.

The Integrated Test Bed (ITB) facility provides a real-time simulation of military aircraft performing an operational mission. The simulation generates the interface signals between the aircraft sensor suite and the avionics system so that the avionics equipment are subjected to a data signal environment which is nearly identical to actual flight. A simulated cockpit is also a part of the simulation for realistic evaluation of the avionics system. The Operational Flight Program (OFP) is a major component of the ITB. The OFP implemented in Ada supports a number of mission modes. The mission modes supported by avionics functions are bundled into Loadable Program Units (LPU) for implementation purposes. The OFP is supported by the Ada Avionics Real-Time Software (AARTS) operating system. It performs task loading, scheduling, running, reconfiguration, and unloading of tasks.

## Subsystem being modeled: Terrain Following Algorithm (TFA) or Vertical Steering implemented in JOVIAL

The purpose of the TFA is to implement a Terrain Following/Terrain Avoidance (TF/TA) algorithm using digital terrain elevation data



CH35797-95/0000-0685 $1.00©1995 IEEE

(DTED) from the Integrated Terrain Access/Retrieval System (ITARS). It also implements the Sandia Inertial Terrain Aided Navigation (SITAN) algorithm using terrain elevation data from ITARS. Using these algorithms the TFA generates in real-time lateral and vertical flight commands suitable for use in directing the flight cue on the Head-Up Display (HUD) or as input to an autopilot system. The system also generates a corrected aircraft position using the SITAN algorithm. The system was originally implemented in JOVIAL and was later converted to Ada.

**Reengineering Process:**

**1. Study Documents and Code**

After documenting the TFA algorithm as implemented in the JOVIAL language, the current system implemented in the Ada language was studied. A detailed listing of Ada files was obtained and the TFA algorithm implemented in the Guidance subsystem was documented. It was found that the Ada version had modified the JOVIAL version substantially. The VERTICAL_STEERING package implements some aspects of the Terrain Following algorithm, a component of TFA.

The next step was to document essential aspects of the Ada packages and their relationships with other packages. The details of the package body were documented using Structured English and detailed diagrams. Visibility Diagrams show visibility of other packages.

**2. Document Design**

Structure Charts were used to document the current design. Process details were documented using Structured English from the body of the VERTICAL_STTERING package. It could be documented in a CASE tool. Indentation helps to highlight control structures. Action Diagrams may be used with the Structured English.

**3. Track Data Elements**

Structured English captures process details. Ideally it should contain data details also. From the documents (packages, programs and other design documents) data can be systematically tracked and documented.

This method of tracking data elements can be very laborious. Reverse Engineering tools capable of tracking data in the entire system should be used for this purpose. The basic task is to determine whether the data element is essential for the process or not. Most of the flags used in parameter lists are dependent on programming style and other constraints. They could be simplified or eliminated. The derived data elements, in general, should not be part of the data structure as they can be easily computed from the basic data.

**4. Draw Visibility Diagrams for Ada Packages**

As the data used in a particular package could be derived from multiple packages, a Visibility Diagram may be drawn using the package specification. A Visibility Diagram for VERTICAL_STEERING is shown in Figure 1. The objective is to isolate data elements which are essential for a particular process. Once we specify essential process logic and data we may encapsulate them into objects.

The Visibility Diagram shows that sixteen packages are visible for the VERTICAL_STEERING. Some of them are generic packages such as por_reals, por_integers which deal with portable reals and portable integers. Attention was given to those packages which have been used (either to access data or a procedure) in the package body of VERTICAL_STEERING.

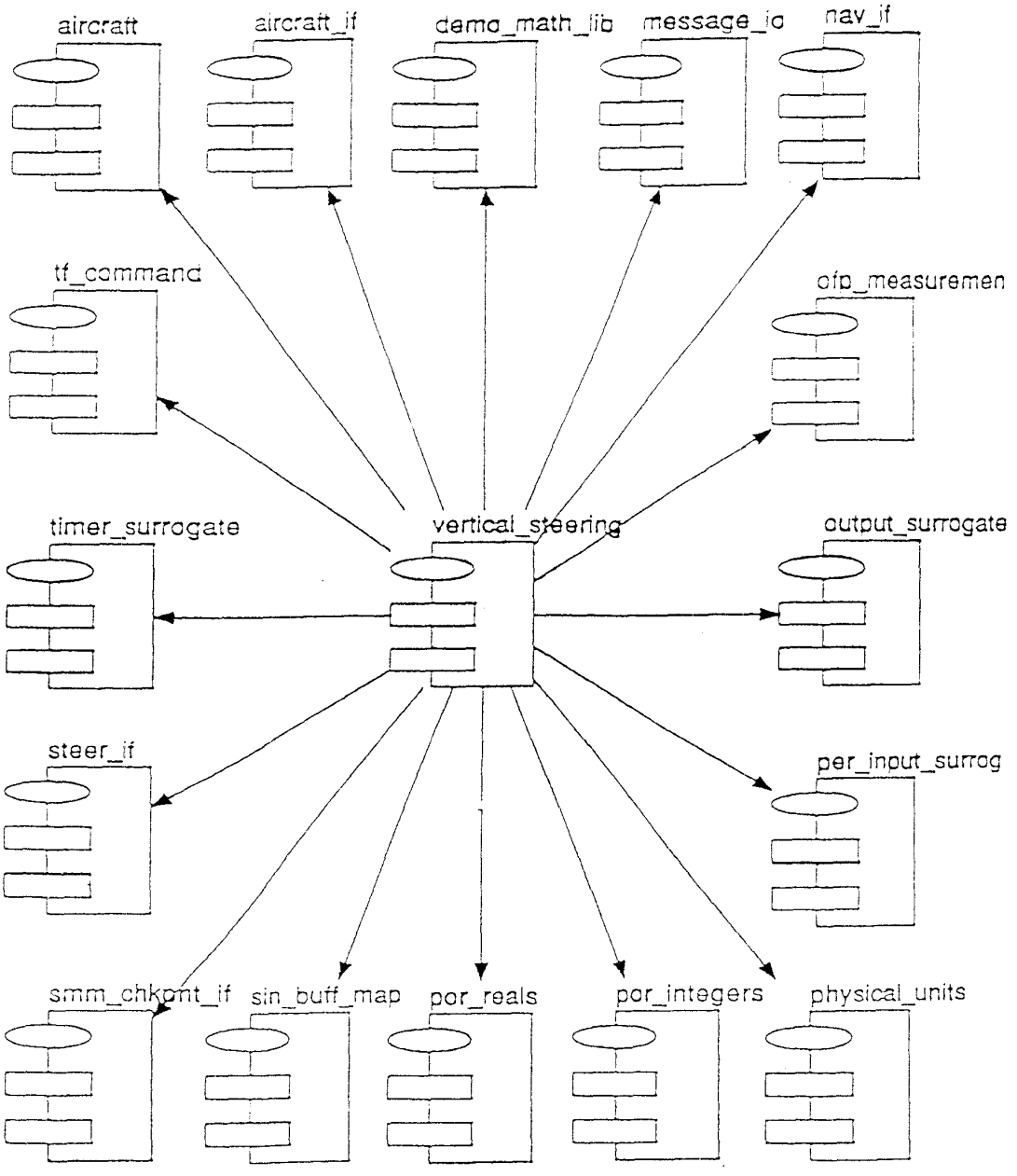


**Figure1** Visibility Diagram for VERTICAL_STEERING

## 5. Draw Detailed Diagrams

A detailed diagram for each process can be drawn using notations similar to Object-Oriented Structured Design (OOSD). It includes package interfaces, procedures, functions and tasks of the package, data declared within the package, and possibly its interface with other packages. Figure 2 shows a detailed diagram for VERTICAL_STEERING.

In an ideal case, the approach for deriving an overall model for the system will use the bottom-up approach. First, objects and classes are identified, and classes are grouped in to subsystems so that cohesion of subsystem components is maximized and coupling between subsystems is minimized. Sometimes the top-down approach may be needed because of the size of the system and multiple resource requirements. We have used both the top-down and bottom-up approaches in arriving at an overall object model. The system documentation provided hardware and system software requirements, and Ada code provided a bottom-up view of the system. Partitioning the system into subsystems is a very crucial activity. A poor partition will adversely affect the resource requirements, performance, and reusability of subsystems.

## 6. Develop Object Model for the System

After documenting the aspects related to VERTICAL_STEERING, the HORIZONTAL_STEERING and other packages of the Guidance system were examined. Finally an Object Model for the Operational Flight Program was developed using the Object Modeling Technique (OMT). The designer of the OFP used object-oriented concepts (Abstract Data Types) in the design of Ada packages. The designer was consulted in developing the overall model. The Object Model closely resembles the current architecture of the OFP. OMTool™ was used for drawing the Object Model shown in Figure 3.

Data dictionary entries are an essential aspect of any development or reengineering effort. The Structured English and Data Details can be used to define objects/classes in the data dictionary. OMT also provides for a Dynamic Model and a Functional Model. Basically the Functional Model uses Data Flow Diagrams to capture process details. Since we have already documented the process details in the reengineering process, we may not need the Functional Model. The Dynamic Model uses State Transition Diagrams to represent control flow of the system. The Dynamic Model should be used to represent relationships among classes/objects. A collaboration graph or object-interaction can also be used to represent interactions among subsystems/classes. The Object Model, Functional Model, and Dynamic Model show three distinct orthogonal views of the system. The Object Model shows inheritance, association and aggregation relationships between classes/objects. The Dynamic Model shows state transitions. The Functional Model shows the transformation of inputs to outputs. Consistency in naming processes and data should be observed across these models as it is essential to provide a coherent view.

The major classes of the Operation Flight Program are the Pilot/Vehicle Interface, Guidance, ITARS, Navigation, Sensor Management, and Steering. Though the current system also has a Mission Manager function, we consider that to be a part of the Avionics Operating System (AARTS).

## Conclusions

In the current implementation of the OFP, most of the data declarations are done in the interface packages with generic packages. These packages contain detailed record descriptions of

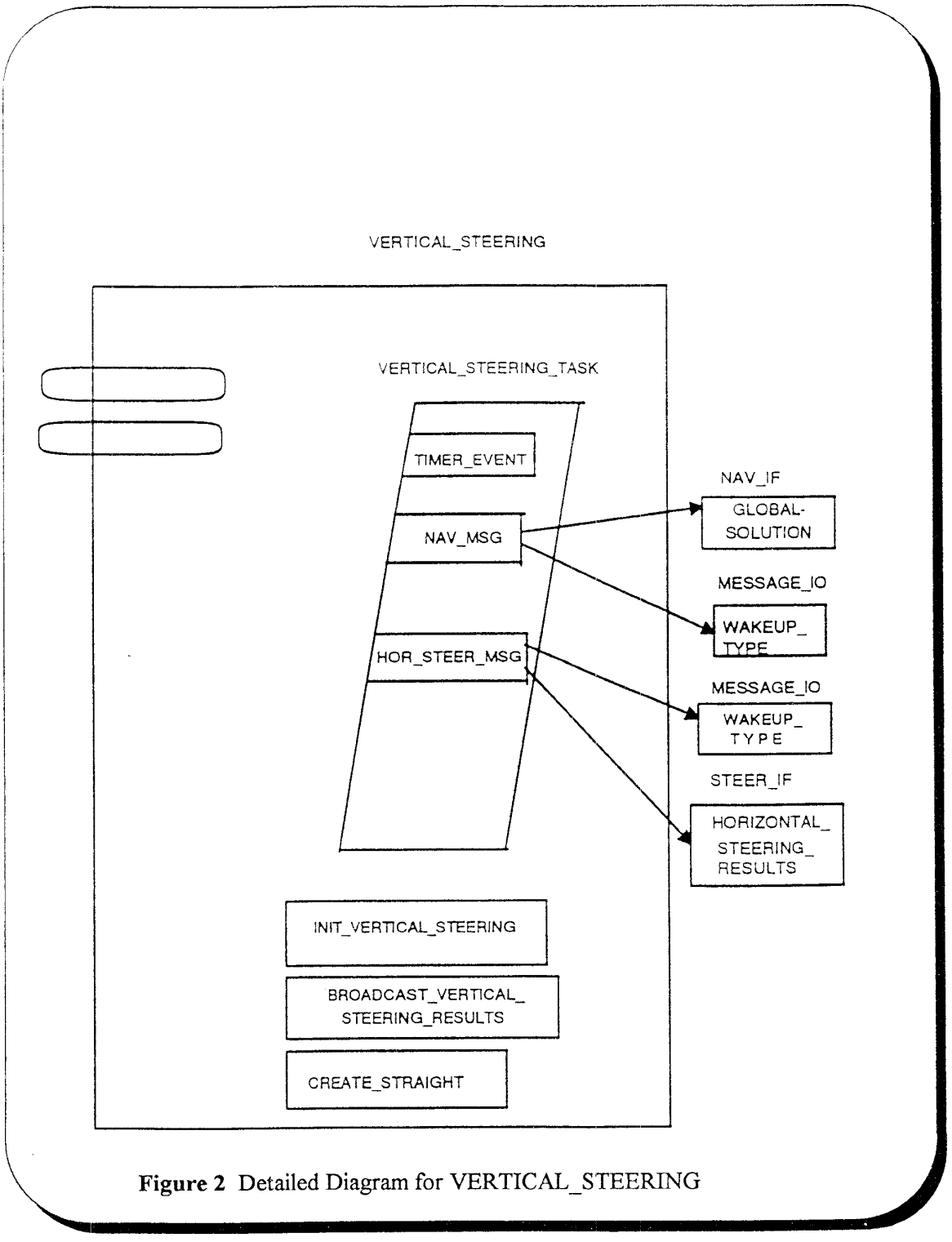


**Figure 2** Detailed Diagram for VERTICAL_STEERING

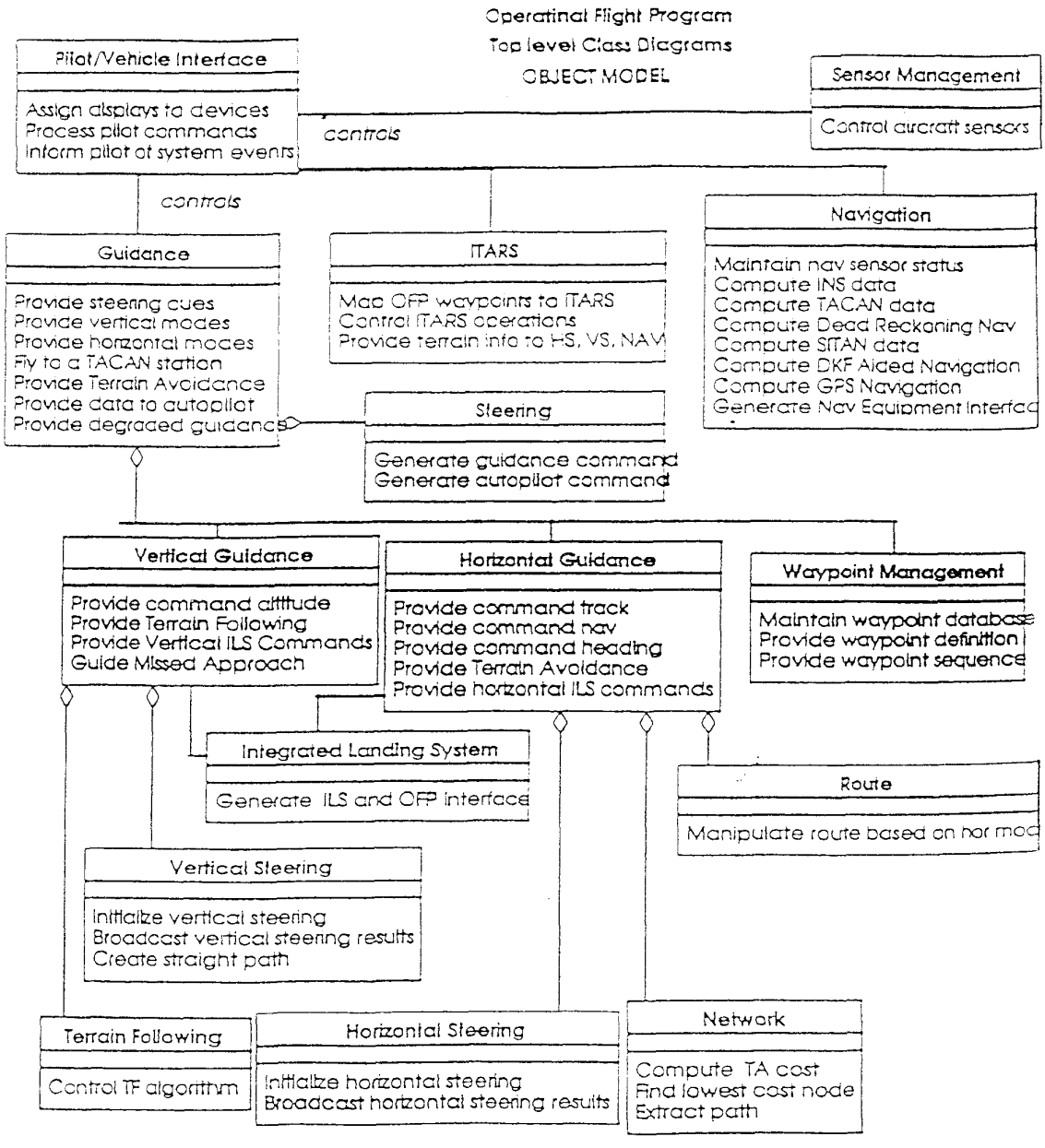


**Figure 3** Object Model for the Operational Flight Program

major components of the system such as the aircraft and navigation. They are instantiated and manipulated by packages of various subsystems. The record templates of the interface packages provide attributes of object classes. These data attributes can be partitioned and encapsulated into objects/classes by providing appropriate processes. The next important task is to identify the relationships among objects/classes such as inheritance, association, and aggregation.

Ideally, all the declarations of a package should be confined to the specification part of the package. In the current implementation, data elements used in a package are mainly subsets of interface definitions and other packages. This makes it difficult to verify the completeness and correctness of the code. It necessitates an understanding of all the packages which define records in order to understand a specific package. If a package imports a number of data elements which are components of many records, it just adds to the complexity.

The following guidelines may be kept in mind when redesigning the system:

1. The application package should contain data which are necessary for supporting its procedures, functions, or tasks. Exposure to unnecessary data introduces ambiguity in the process logic.

2. Some of the flags used in packages are intended to synchronize processes of other packages. Defining the role of a package in the overall structure will lead to a better understanding of the system and control over the flags. The nature of control mechanisms will vary in the object-oriented system Instead of hierarchic control, the objects should interact in a client-server relationship or use of inheritance.

The current design might have been motivated by a number of factors. One of the major factors is data communication between various subsystems through the PI Bus which needs a central store of updated information. Another factor is one of the strong feature of the Ada language - the generic package. With generic packages, one set of code can be reused with different sets of data types. The third factor relates to different formats of data needed by different subsystems due to hardware and software constraints. INPUT and OUTPUT SURROGATES packages are of this type. The input and output data are formatted in one package instead of distributing the tasks to different packages.

CASE (Computer Aided Software Engineering) tools have a significant role in the reengineering process. One of the facilities of CASE tools applicable for most of the systems is the Structure Chart facility used in the Structured Design. Structure Charts show hierarchic relationships between the system and subsystems. In, out, and inout parameters can be shown in the diagrams. The subsystem, process logic and data descriptions should be recorded in the data dictionary. Most of the CASE tools have this facility. To document Ada systems, Visibility Diagrams by Booch and Object-oriented Structured Design (OOSD) by Wasserman et al. are quite useful. The notations by Nielsen can also be used for this purpose. Nielsen's notations are used in Detailed Diagrams (Figure 2). Some of these notations are implemented in the CASE tools described earlier. All the techniques used in our study are not available in any one CASE tool. Selection of a methodology and appropriate CASE tools are of vital importance as they dictate software development process.

Forward Engineering is done using an object-oriented approach. There are a number of object-oriented analysis and design methodologies such as Object-oriented Design by Booch, Responsibility Driven Design by Wirfs-Brock et al., Object-oriented Analysis and Design by Rumbaugh et al., and by Coad and Yourdon. Some of the hybrid methodologies

such as FUSION contain features of multiple object-oriented methods. A methodology which is consistent with the reverse engineering approach needs to be used for the new development. We have used the Object Modeling Technique (OMT) by Rumbaugh et al. for developing an object-oriented model. Leading CASE tools such as Software Through Pictures™ and team*work*™ incorporate this methodology.

A systematic reengineering process will enable us to produce a reusable object-oriented system. Documents related to various aspects of the system would be generated during the process (not after-the-fact). The documents generated will address logical aspects. The system can be easily adapted to changing hardware and software configurations, and minimize future maintenance and modification costs and dependency on a specific vendor or contractor. Some of the subsystems such as the Pilot/Vehicle Interface, Sensor Management, and ITARS are dependent on the type of aircraft, hardware, system software, and communication facilities. Others such as Guidance and Navigation are basically algorithmic-oriented and are useful in any environment. If the laboratory decides to reengineer the existing system the algorithmic-oriented may be tried first. Adhering to a particular methodology and selection of CASE tools are crucial in the success of such an effort.

**References**


[BAR92]Barnhart, D., Benning, S., Blair, J., Bohler, M., Eldridge, B., Fanning, F.J., Goldman, P., Koenig, W., Morales, R., Powers, P., Rubertus, G., Schelling, E., Wilgus, J., Williams, D.,Woodyard, J., PAVE PILLAR IN-HOUSE, Final Report, Systems Section, WL/AAAS-2, WPAFB, Report Number WL-TR-92-1128, October 1992.

[HAL92]Software Reuse and Reverse Engineering in Practice, Edited by Hall, P.A.V., Chapman & Hall, 1992.

[NIE92]Object-Oriented Design with Ada - Maximizing Reusability for Real-Time Systems, Nielsen, Kjell., Bantam Books, New York, 1992.

[WHA92]Whalen, Phillip W., Stoudt, Amy C., Miedler, Michael J., The Integrated Test Bed (ITB) System Integration Project, Final Report for the Period July 1988 to July 1992, TRW Avionics & Surveillance Group, Sponsored by the Avionics Directorate, L/AAAS-2, Wright Laboratory, WPAFB, Report Number WL-TR-92-1119, September 1992.

Computer Program Product Specification for the Terrain-Aided Flight Algorithm (TFA) System Type C5 (STA0001), Released by Powers, Phillip H., Avionics Directorate, WL/AAAS-2, Wright Laboratory, WPAFB, August, 1991.

ADA Avionics Real-Time Software (AARTS) Program, Final Report for the Period March 1987 - October 1991, TRW Avionics & Surveillance Group, Sponsored by the Avionics Directorate, WL/AAAS-2, Wright Laboratory, WPAFB, Report Number WL-TR-91-1135, February 1992.