DESIGNING WITH ADA* FOR SATELLITE SIMULATION: A CASE STUDY

W. W. Agresti, V. E. Church, D. N. Card, P. L. Lo
Computer Sciences Corporation**

ABSTRACT

A FORTRAN-oriented and an Ada-oriented design for the same system are compared to learn whether an essentially different design was produced using Ada. The designs were produced by an experiment that involves the parallel development of software for a spacecraft dynamics simulator. Design differences are identified in the use of abstractions, system structure, and simulator operations. Although the designs were significantly different, this result may be influenced by some special characteristics discussed in the paper.

INTRODUCTION

Some early experiences using Ada for scientific applications (e.g., [1]) showed that the design of the Ada system "looked like a FORTRAN design." As part of an experiment on the effectiveness of Ada, the experiment planners identified the following factors that were believed to be prerequisites for obtaining a new design, one that would take full advantage of Ada features:

- The opportunity to set aside previous designs for the system and work directly from system requirements
- Training in design methods that exploit Ada's capabilities
- The encouragement to explore these new design methods

The purpose of this paper is to address the following question:

> When these prerequisites were satisfied, was a different design produced?

The experiment in progress is being conducted by the Software Engineering Laboratory (SEL) [2] of the National Aeronautics and Space Administration's Goddard Space Flight Center (NASA/GSFC). NASA/GSFC and Computer Sciences Corporation (CSC) are cosponsors of the experiment, which is supported by personnel from all

*Ada is a registered trademark of the U.S. Government (Ada Joint Program Office).

**Authors' Address: Computer Sciences Corporation, System Sciences Division, 8728 Colesville Road, Silver Spring, Maryland 20910

three SEL participating organizations (NASA/GSFC, CSC, and the University of Maryland).

The objective of the overall experiment is to determine the effectiveness of Ada for flight dynamics software development at NASA/GSFC. ([2] describes the characteristics of this environment.) The experiment, begun in January 1985, consists of the parallel development, in FORTRAN and Ada, of the attitude dynamics simulator for the Gamma Ray Observatory (GRO) spacecraft. When completed, the system is expected to comprise approximately 40,000 source lines of code to execute on a DEC VAX-11/780 computer. Additional information about the experiment is presented in [3].

Although the FORTRAN and Ada development teams are proceeding in parallel, the FORTRAN team is further along, due, in part, to the time necessary to train the Ada team in the Ada language and design methods. Both teams have completed the critical design review. This paper reports on a preliminary review of the design processes and products of both teams in order to address the question of interest. The design problem is discussed, an overview of the designs is presented, design processes and products are compared, and the results and their implication for answering the question are summarized.

THE DESIGN PROBLEM

The purpose of the GRO dynamics simulator is to test and evaluate GRO flight software under conditions that simulate the expected in-flight environment as closely as possible [4]. The simulator is represented as a control problem in Figure 1. The right side of the figure models the onboard computer (OBC) flight software. The OBC Model uses sensor data provided by the Truth Model to determine the estimated attitude. Comparing the estimated attitude to the desired spacecraft attitude, the OBC determines the attitude error. Control laws are modeled within the OBC to generate attitude actuator commands that will reduce the attitude error.

The Truth Model, the left side of Figure 1, simulates the response of the attitude hardware. The Truth Model updates and interpolates the spacecraft ephemeris and environmental torques, integrates the spacecraft equations of motion, and generates the true attitude of GRO. The Truth Model produces sensor data corresponding to the attitude, for use by the OBC Model.

Both teams have the task of designing and developing software to simulate the attitude dynamics and control shown in Figure 1. An additional requirement on the FORTRAN team is to extract its Truth Model and integrate it with the Goddard GRO Simulator (GGS), a real-time simulator of the GRO OBC flight software.

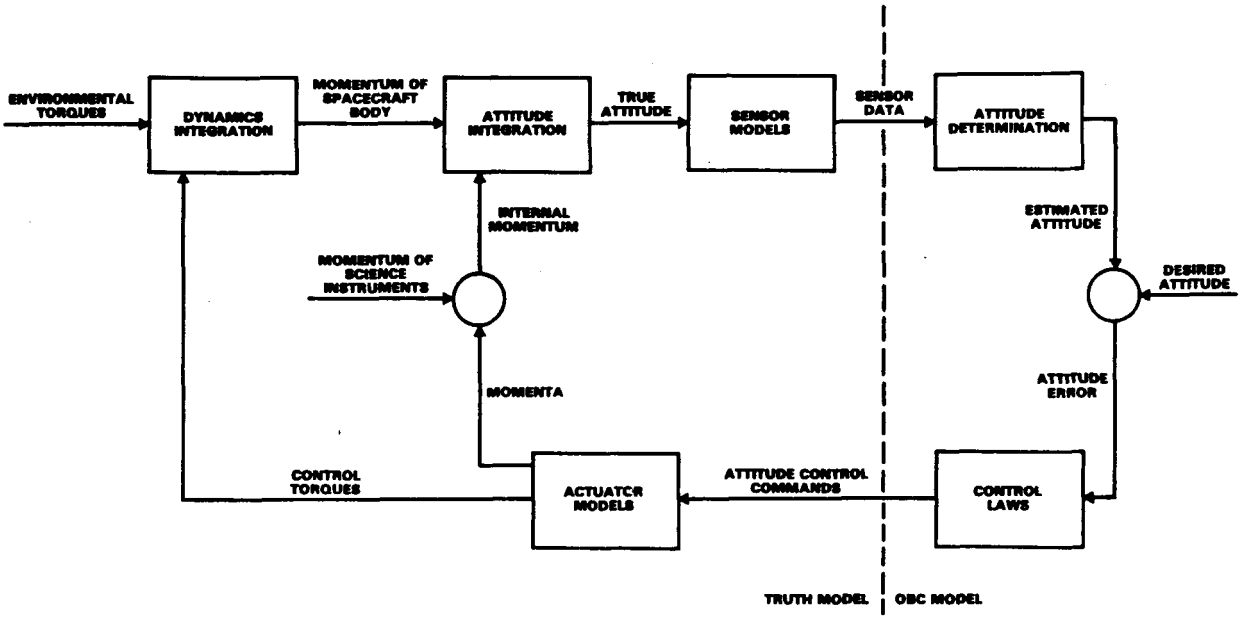

Figure 1. GRO Dynamics Simulator as a Control Problem

OVERVIEW OF THE DESIGNS

In this high-level look at each design, the overall system structure and the external and internal data flows are discussed. Some simple quantitative measures are extracted from each design.

System Structure

A top-level system diagram for each design is shown in Figures 2 and 3. To facilitate comparison, the identical system input and output objects are placed at the top and bottom, respectively, of each figure. The FORTRAN system consists of the five subsystems in the middle of Figure 2. The Ada system is the product of a design method (discussed below) that differs from the FORTRAN team method. So, although "subsystem" will be used to refer to the major Ada units, they are, in fact, Ada packages. Furthermore, the simulation support subsystem in Figure 3 is really a collection of three Ada packages for the simulation timer, parameters, and ground commands. The Ada system appears in Figure 3 as five subsystems only to invite comparison with FORTRAN regarding the high-level data flow.

The FORTRAN system is composed of three distinct programs: Profile, Postprocessor, and Simulator (Truth Model, OBC Model, and Simulation Control-I/O). As separate programs, each interacts with the user, as shown by the external data flows in Figure 2. The assignment of processing functions to each subsystem is shown in Figure 4 for both the FORTRAN and Ada systems.

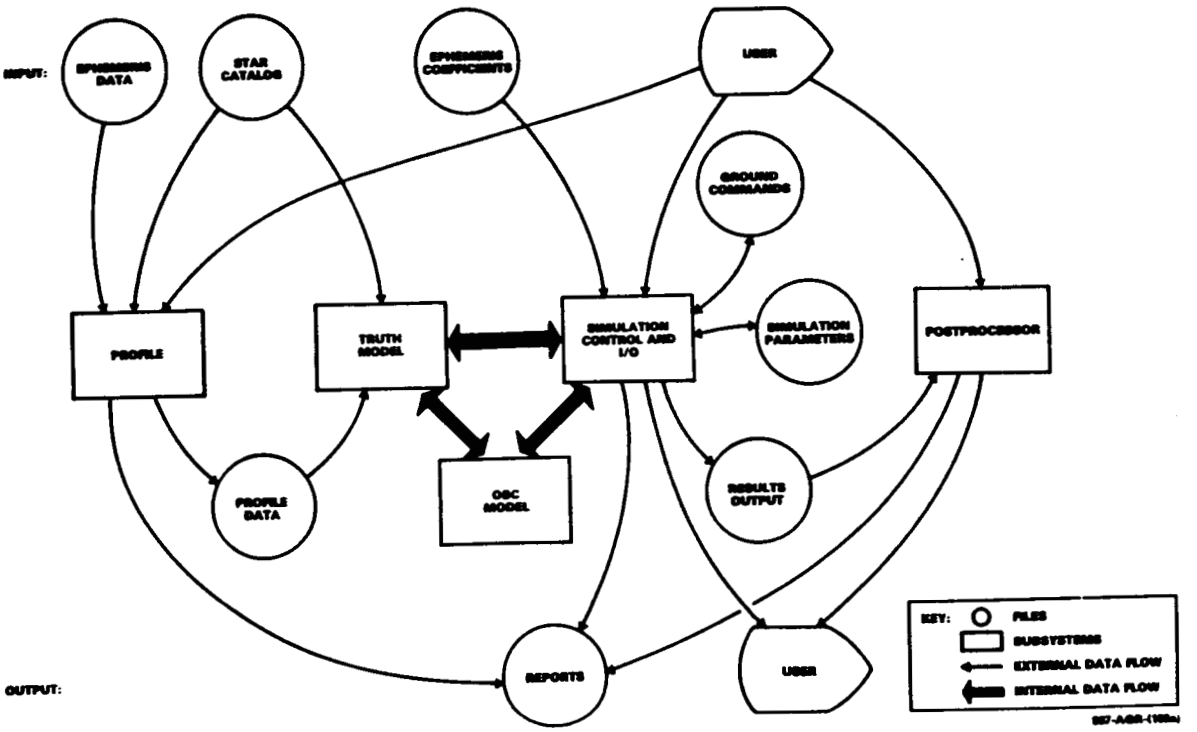

Figure 2. FORTRAN System Diagram

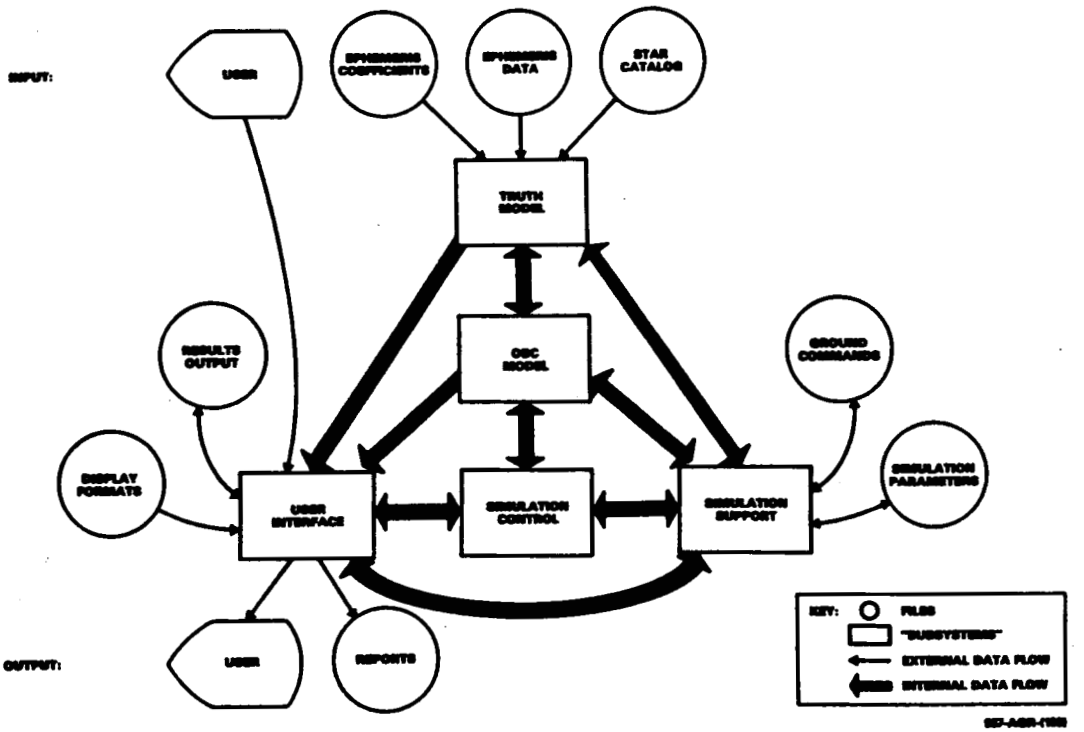

Figure 3. Ada System Diagram

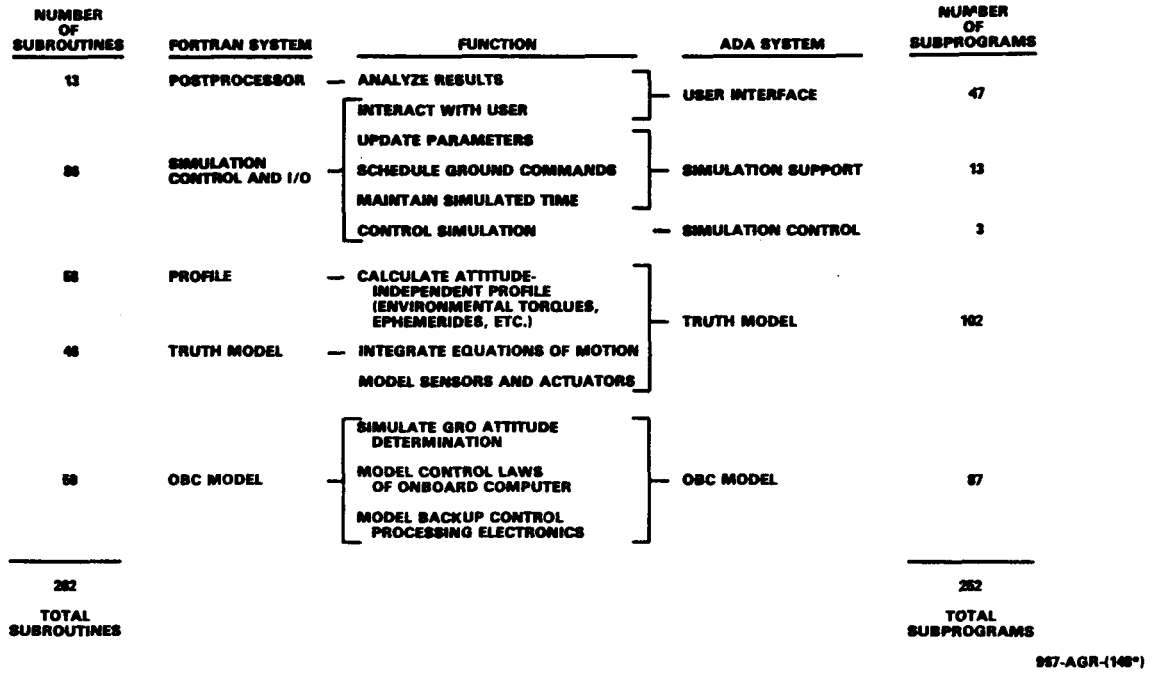

Figure 4. Allocation of Functions Among Subsystems

The Ada system is designed as a single program, with each subsystem performing the functions listed in Figure 4. The OBC Model is functionally similar to its FORTRAN counterpart. The Ada Truth Model incorporates the processing performed in the FORTRAN Profile in addition to the FORTRAN Truth Model. (The FORTRAN user has the option of choosing not to use Profile and having those calculations performed in the Truth Model, thereby mirroring the Ada design.) The Ada design pulls apart the simulation control functions from the User Interface; these processing elements are combined in the FORTRAN design. However, the User Interface in Ada includes the results processing that, in FORTRAN, is delegated to a separate program, the Postprocessor. Both designs have major units named Truth Model and OBC Model to reflect the underlying control problem illustrated in Figure 1.

External Data Flow

Both designs in Figures 2 and 3 show communication with nine external objects (files or devices). Eight of the nine are identical, the difference being the profile data file in FORTRAN and the display format file in Ada. The FORTRAN design requires the profile data file to decouple the Profile and Truth Model processing. The use of a display format file in the Ada design is motivated by reusability considerations. By keeping the detailed formats of menus and displays on an external file, the user interface is easier to reuse on a future simulator.

The number of external data flows is greater in the FORTRAN design, as shown in Table 1. Most of the additional data flows arise from the separation of the FORTRAN design into three programs, requiring more data flows to and from the user and distinct data flows to the profile data and results output files that decouple the programs. Also, as shown in Figure 2, the star catalog external file is required in both Profile and the Truth Model.

Table 1. Simple Quantitative Design Characteristics

CHARACTERISTIC	FORTRAN DESIGN	ADA DESIGN
SEPARATE PROGRAMS	3	1
TASKS	5 (IN SIMULATOR PROGRAM)	5
EXTERNAL ENTITIES	9	9
EXTERNAL DATA FLOWS	18	10
INTERNAL DATA FLOWS	3	9
SUBROUTINES/SUBPROGRAMS	262	252
PACKAGES	—	104

The Ada design (Figure 3) involves the minimum number of external data flows. The details of accessing each file are confined to a single subsystem.

Internal Data Flow

Table 1 shows that the Ada design has nine internal data flows, versus three for the FORTRAN design. Of course, no more internal data flows are possible in the FORTRAN case because Profile and the Postprocessor are separate programs. The three remaining subsystems in the FORTRAN design exchange data with one another via COMMON blocks. (Although the use of COMMON has been criticized, empirical results from the flight dynamics environment has shown it to be effective [5].)

Although the number of distinct data flows (connections) between subsystems is greater in Ada, fewer data items pass over these connections than in FORTRAN. An example will show how various Ada language features help to reduce the proliferation of data item names.

Both designs provide for the recording of simulation analysis results. In FORTRAN (Figure 2), these results pass from the

Truth Model and OBC Model via COMMON to the Simulation Control-I/O Subsystem, which writes them to the external results output file. In Ada (Figure 3), the internal data flows from the Truth Model, OBC Model, and Simulation Control carry results data to the User Interface, which writes them to the results output file.

In the FORTRAN design, the results data record comprises 43 distinct variable names. In Ada, the results are passed under a single identifier, Results_Data, when a procedure, Put_Results_Data, in the User Interface is called by the Truth Model, OBC Model, or Simulation Control. This reduction in the number of identifiers is possible because of the use of Ada's variant record feature. In the example, Results_Data can be either an executed ground command, parameter update, error message, or analysis result. In Ada, the user can declare Results_Data as type RESULT, defined as a record type with a variant part as follows:

```
type RESULT_KIND is (Error_Msg, Log_Command, Results,
                     Parameters);

type RESULT (Kind:  RESULT_KIND:=Results) is
  record
    case Kind is
      when Error_Msg | Log_Command =>
        Result_Line:  STRING (1...80);
      when Results | Parameters =>
        Result_Rec:  PARAM_RESULT;
    end case;
  end record;
```

Because of such features, the count of data items is consistently lower over the Ada data flows than over the FORTRAN data flows.

COMPARING DESIGN PROCESSES

Differences in the design processes help to explain the differences in the delivered design products of the FORTRAN and Ada teams. Two aspects of the design process--critical design "drivers" and the use of design abstractions--will be examined.

Design Drivers

The design drivers--critical characteristics that strongly influence design decisions--are different for the two teams. The FORTRAN team was influenced by its real-time processing requirement, previous designs, and schedule concerns. The Ada team was influenced by its training in alternative design methods and the opportunity to apply those methods.

Although the basic requirements for each team are identical, the FORTRAN team has a real-time requirement, noted earlier, to integrate its Truth Model Subsystem with the Goddard GRO Simulator. To help ensure that the Truth Model will complete its

processing in time to meet this requirement, the FORTRAN design removes those computations that are not strongly attitude dependent from the Truth Model to a separate Profile Program. Then, instead of performing these calculations (such as environmental torque and magnetic field) each iteration, the Truth Model can simply read the necessary values from the Profile data set (as shown in Figure 2). This separation of the Profile calculations from the Truth Model is further encouraged by the previous designs of dynamics simulators in FORTRAN, which also had separate Profile Programs. The FORTRAN design also provides the option, for greater accuracy, of performing the Profile calculations within the Truth Model.

The Ada design, not required to meet the real-time constraint in this experiment, includes in its Truth Model the calculations performed in the FORTRAN Profile Program and FORTRAN Truth Model. It will be of interest later to test whether the real-time requirement can be met by the Ada design and by the FORTRAN design under the option of performing Profile calculations inside its Truth Model.

A strong driver of the FORTRAN design is the presence of a previous design, used successfully on past simulators. The partitioning into subsystems in Figure 2 is identical to that of previous simulators. With this legacy, the interfaces between subsystems--a frequent problem area with original designs--are clarified early in the project. With the interfaces relatively clear, the subsystems can be assigned to individuals or small subgroups for detailed design and implementation with the "design envelope" fairly well established.

The Ada design was intended to be an independent one, free of the influence of past simulator designs. The subsystems that evolved were the product of lengthy design discussions. The similarity of the Ada subsystems to those in FORTRAN owes more to both designs reflecting the underlying control problem of Figure 1, rather than the Ada design copying the FORTRAN design.

The schedule constraints on the teams were different. To help explain this difference, consider that the dynamics simulator is a routine element of the set of ground support software for a satellite mission. The entire complement of software has rigid schedule constraints derived from launch dates. FORTRAN has been used in the past and is being used now for the GRO attitude ground support software. In such an environment, it is natural that the FORTRAN team was perceived as building the real, operational software, even though the Ada product is also expected to pass acceptance testing and to perform in an operational environment.

The FORTRAN team generally had more schedule pressure than did the Ada team, and this difference affected the design products and methods. Both teams were charged with developing operational software, but the Ada team was also encouraged to try Ada-related

design methods as a way of understanding their usefulness in the flight dynamics environment. The FORTRAN team had more exclusively practical concerns of meeting the development schedule.

Design Abstractions

The use of abstraction was also different for each team. The FORTRAN design products provide evidence of the procedural abstraction carried forward from earlier designs. An individual subroutine may be thought of as a black box that will, for specified values of its input variables, produce the same specific output values every time it is invoked. The input and output quantities are transmitted via argument lists or COMMON. This procedural abstraction can also be used at higher levels in the system. For example, the Truth Model is a procedural abstraction possessing an identifiable function (computing the current attitude state of the spacecraft), specific input quantities (primarily parameter values and actuator commands), and specific output quantities (primarily sensor data reflecting the time attitude state).

The FORTRAN design also has elements of being object oriented. Functional processing at the lower levels is organized around objects in the problem domain such as specific sensors and actuators. For example, the Truth Model contains a sensor modeling component that calls seven routines: one for each sensor type. Anyone making a code modification due to a requirement change relating to the fine Sun sensor will find a subroutine, FSSMOD, described as modeling the fine Sun sensor. The use of COMMON also reflects an orientation to objects. For example, one COMMON block holds gyro parameters; another has FSS parameters; and so on.

Concurrent processes are used in the FORTRAN design to model the concurrency that exists in the operational use of the simulator. For example, an analyst may interrupt the processing to change the value of a parameter. System services of the DEC VAX-11/780 VMS operating system are used to implement the concurrent processes. Both the object-oriented features and the use of concurrency are characteristics of past FORTRAN simulators, demonstrating that reuse of design is the operative high-level approach in the FORTRAN design.

The Ada design process was significantly different from that of the FORTRAN team. The differences begin to emerge even before the design phase of the project.

The functional specifications and requirements document [4] for the GRO dynamics simulator is influenced by the design legacy of dynamics simulators developed within the organization. For example, the document is organized by major subsystem because that particular partitioning into subsystems (Figure 2) has persisted through several simulator project teams. In effect, the highest level design is completed during the requirements analysis phase.

This encroachment of design on requirements actually provides a welcome headstart to a team who will be following that design and taking maximum advantage of the existing code based on that design. While such a document fit in well with the projected work of the FORTRAN team, it was not as helpful to the Ada team, who wanted to produce an independent design, uninfluenced by previous simulator designs.

A way out of this dilemma--the influence of the previous design present in the requirements--was to recast the requirements in a different form. The Ada team developed a specification for the dynamics simulator using the Composite Specification Model (CSM) [6], which represents a system from the functional, dynamic, and contextual views. Recasting the system requirements using CSM served other purposes as well: It provided a testbed for the CSM as a specification tool, and it allowed the Ada team, who was relatively inexperienced in the application area, to analyze the system requirements in a systematic manner. The result of this exercise was a specification document [7] and a better understanding of the needs of the system. For example, included in [7] are PDL-like process specifications describing the required functional processing. The specification succeeded in removing the inherited design from the system requirements and served as a starting point for the Ada design.

The Ada language itself influenced the design team because the team members knew that useful design abstractions could be represented in Ada. The team had been exposed to object-oriented design, the process abstraction methodology, and other approaches during their training program, which included the development in Ada of a 5700-line training exercise [3]. The principal design abstractions used by the team were the state machine abstraction and the representation of the system according to the orthogonal views of a seniority hierarchy and a parent-child hierarchy [8]. The state machines are conveniently implemented as Ada packages consisting of internal state data and a group of related procedures that operate on that state data. The Ada design product reflects this approach; the design includes 104 packages and 69 sets of state variables.

An instance of the seniority hierarchy is shown in Figure 5. The team's design approach is to build the system as layers of virtual machines [9]. For example, Figure 5 shows that the OBC package is senior to the Truth Model package. The arc between the two packages shows that OBC uses operations (subprograms) of the Truth Model. Arcs do not go from a package to one that is above it. In this way, each diagram expresses the relative seniority of the packages [10]. The orthogonal parent-child (or inclusion) hierarchy provides for a package (like one of those in Figure 5) to be represented on a separate diagram in terms of its constituent elements; for example, subprograms, other packages, and state data.

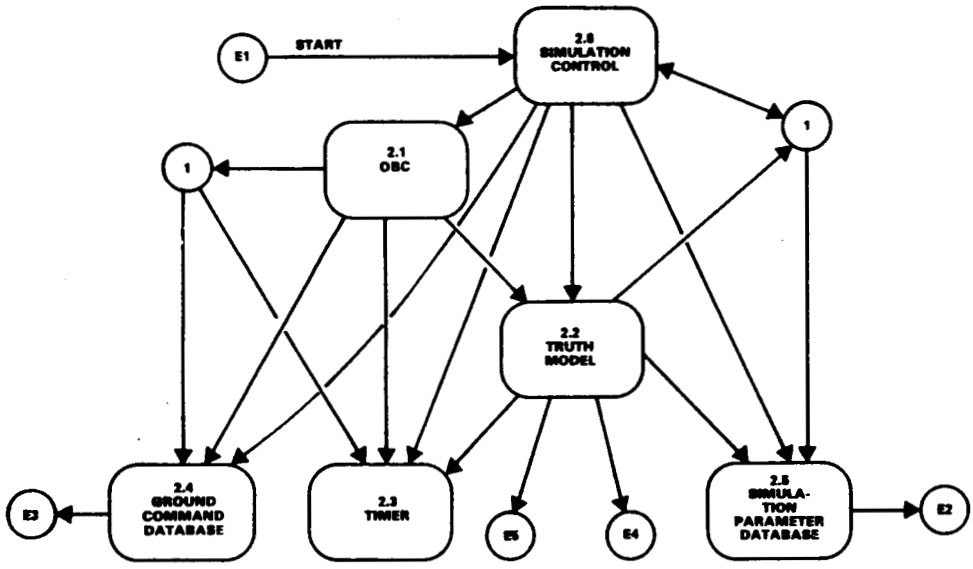

Figure 5. Ada Design: Seniority Hierarchy of Packages

In summary, the Ada team was able to use effective design abstractions because they were confident that these abstractions could be preserved in an Ada implementation.

COMPARING DESIGN PRODUCTS

The design documents were examined to determine any significant differences. Some differences were noted earlier: the FORTRAN design involving three programs; the different assignment of functional processing to subsystems; and the data flow. Review of the design documents revealed two more fundamental differences in the basic operation of each simulator, as specified by the designs. These key differences can be shown by tracing the operation of each simulator.

Figure 6 shows the logical relationships among the five tasks that constitute the FORTRAN simulator program (i.e., excluding Profile and the Postprocessor). The task called GROSS in Figure 6 is the main process started by the user via a RUN command. GROSS remains an active process throughout the simulation run, displaying a menu of user options at the user's terminal and remaining ready to respond to a user request.

The SIMCON process, created by GROSS, controls the simulation. As suggested by the control loop in Figure 1, the simulation involves iterating over the Truth Model and the OBC Model. SIMCON directs this iteration. SIMCON wakes up the Truth Model (TM) process, which computes the attitude state and deposits the corresponding sensor data into a global COMMON section. When TM is finished, it goes into hibernation, setting an event flag that signals SIMCON to wake up the OBC process. OBC obtains the

current sensor data left by TM, models the control laws, and generates actuator commands that are placed in a global COMMON section for access by TM on the next iteration. Its work finished, OBC hibernates, signaling SIMCON to wake up SIMOUT to write an analysis record to capture the results of this iteration. When SIMOUT hibernates, SIMCON wakes up TM to begin the next iteration.

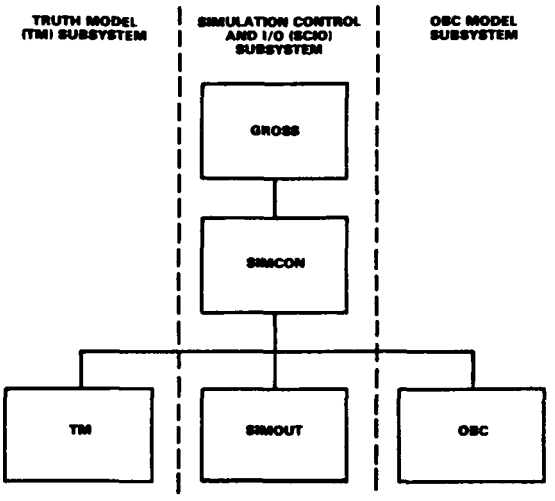

Figure 6. FORTRAN Design: Hierarchy of Execution Tasks

The FORTRAN user can set the cycle time, which is the amount of time that the simulation clock is incremented. The cycle time determines when events occur in the simulation, for example, when thrusters fire, when new sensor data are generated, and when the spacecraft attitude state is updated. The FORTRAN design thus involves iterating over the three processes (TM, OBC, and SIMOUT), with the user-settable cycle time determining when events occur.

Figure 5 shows an excerpt from the Ada design corresponding to the simulator operation. The notation in the figure needs some explanation. The rounded rectangles are Ada packages. Circles denote off-page connectors, with the labels E1, E2, etc., referring to external files and the label 1 denoting package number 1 from a different diagram. Arcs show the direction of a subprogram call from a subprogram in the calling package to a subprogram in the called package. More detail on the design notation is presented in [10].

The placement of packages on design diagrams such as Figure 5 shows the seniority hierarchy described earlier. Thus, in Figure 5, the Simulation Control package is senior to other packages on the diagram; that is, it uses services provided by these other packages and they do not use its services. The three

packages at the lowest level (which together constitute the simulation support subsystem of Figure 3) are junior to the packages higher in the diagram and as such are not the origin for any arcs that terminate at higher level packages.

This more detailed examination of the operation of each simulator revealed two clear differences in the Ada design: the passive role of the Truth Model and the separate timing of the OBC and the Truth Model.

The Ada design represented by Figure 5 shows that, unlike the FORTRAN design, the OBC and the Truth Model are not at the same level. The OBC calls the Truth Model to obtain sensor data when the data are needed. The Truth Model is passive; it performs processing and generates sensor data only when directed to do so.

Both the OBC and the Truth Model are junior to Simulation Control in Figure 5, an arrangement that appears to mimic the FORTRAN design. However, the Ada design notebook [11], which provides details of the actual calls made by Simulation Control, shows the Ada design to be quite different. Recall that the cycle time in FORTRAN affected both the OBC and the Truth Model. In the Ada design, the timing of the OBC and the Truth Model is separate: the Truth Model cycle time is under user control; OBC timing is not. The Ada team chose to model faithfully the spacecraft OBC flight software, whose timing is not under user control. Because timing and event scheduling are central elements in any simulation, this difference is of a fundamental nature and demonstrates that the Ada team was able to go back to basic system requirements for their analysis.

CONCLUSIONS

The comparison of FORTRAN and Ada designs has revealed significant differences in both the design processes and products. In this experiment, the Ada design has been shown to be different to a significant degree from the FORTRAN design. This result differs from that reported in [1] for another monitored Ada development project in a different environment.

The results have implications for other organizations contemplating the use of Ada. This experiment led to a design that exploits Ada's features for expressing design abstractions. However, this result was supported by (1) the use of a specification method, CSM, to counteract the influence of design-laden requirements; (2) the explicit allowance for the Ada team to pursue new design methods, not requiring the team to take the less costly route of reusing the existing design; and (3) training in alternative design methods.

ACKNOWLEDGMENTS

The Ada experiment is managed by F. McGarry and R. Nelson of NASA/GSFC and actively supported by representatives from all SEL

participating organizations (NASA/GSFC, CSC, and the University of Maryland), especially V. Basili, G. Page, E. Katz, and C. Brophy. The authors thank J. Garrick, S. DeLong, G. Coon, D. Shank, and E. Seidewitz for their assistance.

REFERENCES

1. V. R. Basili et al., "Characterization of an Ada Software Development," Computer, September 1985, vol. 18, no. 9, pp. 53-65

2. Software Engineering Laboratory, SEL-81-104, The Software Engineering Laboratory, D. N. Card, F. E. McGarry, G. Page, et al., February 1982

3. W. W. Agresti, "Measuring Ada as a Software Development Technology in the Software Engineering Laboratory (SEL)," Proceedings, Tenth Annual Software Engineering Workshop, NASA/GSFC, December 1985

4. Computer Sciences Corporation, CSC/SD-85/6106, Gamma Ray Observatory (GRO) Dynamics Simulator Requirements and Mathematical Specifications, G. Coon, April 1985

5. D. N. Card, V. E. Church, and W. W. Agresti, "An Empirical Study of Software Design Practices," IEEE Transactions on Software Engineering, February 1986, vol. SE-12, no. 2, pp. 264-271

6. W. W. Agresti, "An Approach for Developing Specification Measures," Proceedings, Ninth Annual Software Engineering Workshop, NASA/GSFC, November 1984

7. Computer Sciences Corporation, CSC/TM-85/6108, Specification of the Gamma Ray Observatory (GRO) Dynamics Simulator in Ada (GRODY), W. W. Agresti, E. Brinker, P. Lo, et al., November 1985

8. V. Rajlich, "Paradigms for Design and Implementation in Ada," Communications of the ACM, July 1985, vol. 28, no. 7, pp. 718-727

9. E. W. Dijkstra, "The Structure of 'THE'-Multiprogramming System," Communications of the ACM, May 1968, vol. 11,, no. 5, pp. 341-346

10. E. Seidewitz and M. Stark, "Toward a General Object-Oriented Software Development Method," Proceedings, First International Symposium on Ada for the NASA Space Station, Houston, Texas, June 1986

11. Computer Sciences Corporation, CSC/SD-86/6013, GRO Dynamics Simulator in Ada (GRODY) Detailed Design Notebook, W. Agresti, E. Brinker, P. Lo, et al., March 1986