ADVANTAGES OF AN OBJECT-ORIENTED DESIGN APPROACH TO THE SIMULATION OF LEADSHIP EFFECTS

Jerome M. Weiss, Section Head (Aero/Propulsion)
Ruth E. Korba, Systems Engineer
CAE-Link Corporation
Binghamton, NY

ABSTRACT

The B-2 Aircrew Training Device (ATD) employs object-oriented design (OOD) accompanied by the Ada programming language. The task of choosing objects to simulate vortex, bow wave, and engine exhaust effects of a leadship on the B-2 is presented from an OOD perspective. The B-2 software architecture of the leadship effects model created from an OOD approach is analyzed and compared to previously used software architecture at CAE-Link. These comparisons are made against architectures used in other military trainers. The trainers are evaluated in terms of maintainability and reusability. Conclusions are drawn as to which architectures are most efficient from a data concurrency/subjective evaluation and future applications perspective.

INTRODUCTION

The task of training a pilot to successfully navigate air disturbances caused by a preceding aircraft has been a goal of many high-fidelity simulators. The design of what is called a leadship or othership effects model has been approached differently throughout the years at CAE-Link, with the most recent model being that of the B-2 Aircrew Training Device (ATD). The B-2 ATD was the first to use an object-oriented design approach. Object-oriented design (OOD) is the organization of software into layers of objects, where the higher layers of objects usually relate to separately compilable software units. Its purpose is to emphasize maintainability and reusability of the trainer software.

This paper first introduces the B-2 ATD software architecture of the leadship effects model. The current B-2 leadship effects model, as well as its development, is explained from an OOD perspective. Next, the leadship effects software architectures for previous CAE-Link military trainers are discussed. These architectures are then evaluated in terms of maintainability, reusability, model fidelity, and complexity of software. This investigation also presents possible areas for improving the B-2 architecture. Such improvements could make the design more object-oriented and more efficient. The improvements are based not only on this study, but on lessons learned throughout the B-2 ATD design and test processes. Conclusions are presented concerning the effectiveness of these architectures as they relate to maintainability and transportability.

B-2 ATD OOD OTHERSHIP SUBSYSTEM

An othership subsystem was developed to satisfy the B-2 ATD training requirement for realistic aerial refueling and base escape characteristics. This subsystem supplies the vortex/downwash, bow wave, and engine exhaust characteristics of a vehicle preceding the B-2. The preceding vehicle (leadship or othership) could be a KC-135, KC-10, or another B-2. Such characteristics require the othership subsystem to determine the relative position between the leadship and the B-2 (lagship). This subsystem defines the leadship's and lagship's physical geometry. The leadship's effects are transmitted to the lagship by computing the delta forces and moments due to the leadship's presence. Different vortex/downwash models are used to compute these delta forces and moments depending on whether a base escape or aerial refueling maneuver is performed. The accomplishment of the B-2 ATD training requirements mandated that these models be of high fidelity. The details of this subsystem are presented in Ref. 1.

It was originally planned that the aircraft manufacturer would supply delta aerodynamic coefficient data describing the leadship effects on the B-2. Unfortunately, this data was not available from the aircraft manufacturer. Therefore, CAE-Link needed to take another approach. CAE-Link applied an OOD software representation in Ada code and current software engineering concepts, including generic modeling techniques. These concepts are discussed in the following paragraphs.

An OOD representation of a subsystem must satisfy two general requirements:

1. The subsystem should consist of self-contained objects that reflect logical/physical real-world entities. This supports the concepts of maintainability and reusability because all characteristics of the object are centralized. This allows ease of update, repair, and/or reuse by having these real-world entities modeled in one location.

2. The software implementation of each object will consist of a separately compilable unit or unit and its subunits. This allows model features to be selectable and therefore be more reusable. Troubleshooting design problems is made easier by having the objects coded in this manner. This also reduces the risk of accidentally affecting other software when these objects need to be modified.

The first step in our approach was to define the objects of the othership subsystem. Three object selection criteria were used:

1. The objects must be independent physical or logical entities. The things that describe the entities are called their attributes.
2. The objects must be such that changes to the B-2 ATD training requirements can be easily accommodated.
3. The level of decomposition of the objects must be governed by the real-world system operations, pure OOD theory, the data available to simulate the system, and the training requirements.

The othership subsystem was decomposed into three primary objects: leadship, air, and lagship. Each primary object satisfies the three OOD object definition criteria and translates into separately compilable Ada units. The Ada software configuration is depicted in Figure 1. This shows how each primary object was housed in separately compilable packages. The subsystem import and export interfaces with other subsystems were also made separate packages. The subsystem controller may contain computations but primarily contains logic code that calls Ada procedures within the various object definition packages. This subsystem controller determines if a leadship is close enough to a lagship to execute this code. The logic code then selects the correct procedures for aerial refueling or base escape training missions. Each object definition package contains Ada procedures relating to each primary object's attributes (see Figure 1). The details of the B-2 ATD software architecture can be found in Ref. 1.

Figure 2 is a block diagram of the B-2 ATD othership effects software architecture. This shows how the othership subsystem interfaces with the lagship forces and moments subsystem and gets leadship information from other subsystems.

Several software engineering principles were applied to each of the objects. These principles also support the goals of good OOD. The equations in-

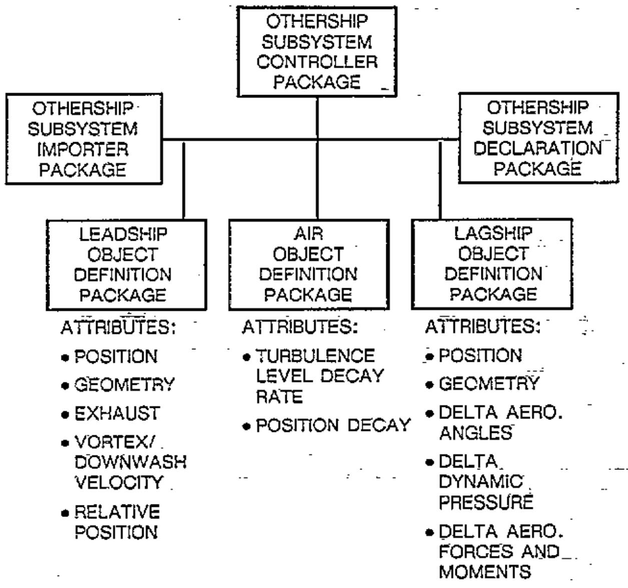

OTHERSHIP SUBSYSTEM CONTROLLER PACKAGE
OTHERSHIP SUBSYSTEM IMPORTER PACKAGE
OTHERSHIP SUBSYSTEM DECLARATION PACKAGE
LEADSHIP OBJECT DEFINITION PACKAGE
AIR OBJECT DEFINITION PACKAGE
LAGSHIP OBJECT DEFINITION PACKAGE
ATTRIBUTES:
• POSITION
• GEOMETRY
• EXHAUST
• VORTEX/ DOWNWASH VELOCITY
• RELATIVE POSITION
ATTRIBUTES:
• TURBULENCE LEVEL DECAY RATE
• POSITION DECAY
ATTRIBUTES:
• POSITION
• GEOMETRY
• DELTA AERO. ANGLES
• DELTA DYNAMIC PRESSURE
• DELTA AERO. FORCES AND MOMENTS

Figure 1 Othership Subsystem Ada Configuration

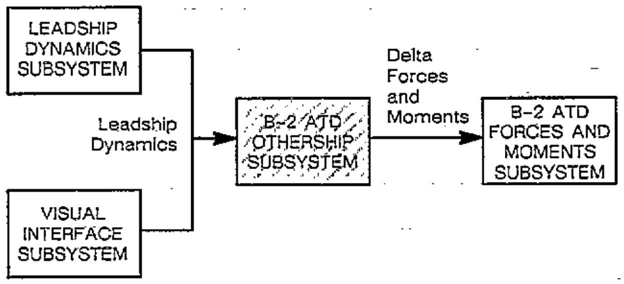

LEADSHIP DYNAMICS SUBSYSTEM
VISUAL INTERFACE SUBSYSTEM
Leadship Dynamics
B-2 ATD OTHERSHIP SUBSYSTEM
Delta Forces and Moments
B-2 ATD FORCES AND MOMENTS SUBSYSTEM

Figure 2 B-2 ATD Othership Effects Software Architecture

clude "adjustment constants" that allow instant investigation of problems with the pilot in the loop. The numerical definition of all application-dependent vehicle parameters is done in one location of the software. The definition of the vehicle parameters is accomplished in the declaration package. Equations were written and coded in generic rather than application-specific engineering notation. These principles allowed the coded equations to be independent of the vehicle being simulated, and increased their transportability and maintainability. The following two groups of equations are presented as examples of generic and application-specific engineering notations:

Generic Engineering Notation	Application-Specific Engineering Notation	
$L = qSC_L$	$L = 2400.0\ qC_L$,	where $S = 2400.0$
$Y = (\pi/4.0)b$	$Y = 74.6$,	where $b = 95.0$

With parameters b and S defined in one location, if an update becomes necessary, these parameters get modified only once. With the application-specific engineering notation, the parameters would be modified in each coded equation that uses them. The generic engineering notation also improves transportability because the equations and constants are more easily understandable by fellow engineers. These principles were applied to increase the software maintainability and reusability.

The specific use of the object definition criteria as they apply to this subsystem is discussed in the following paragraphs.

Leadship

This object models the preceding vehicle's geometry, vortex/downwash, and engine exhaust. All of this model's leadship information parameters and equations are housed within this object. The generic engineering notation was used within this object. This approach has the following advantages:

- The addition of a different leadship doesn't change the leadship equation. Their numerical values are defined in the declaration package.
- Problems identified during simulator testing or subjective evaluation related to the leadship can be addressed without risk of accidentally touching non-leadship software.
- Model features are represented by Ada procedures for each attribute. If a future trainer doesn't require a certain feature, that procedure is not included.

These advantages support maintainability and reusability. The attributes of the leadship object are listed below:

Leadship Velocity
Leadship Geometry
Leadship Position (Distance)
Leadship Engine Exhaust
Leadship-Generated Vortex/Downwash Velocity
Leadship/Lagship Relative Position

Air

This object models vortex/downwash decay factors due to the leadship/lagship relative position and atmospheric turbulence. All of this model's air information, parameters, and equations are housed within this object. The generic engineering notation was used within this object. This approach has the following advantages:

- Problems identified during simulator testing or subjective evaluation related to the air can be addressed without risk of accidentally touching non-air software.
- Model features are presented by Ada procedures for each attribute. If a future trainer doesn't require a certain feature, that procedure is not included.

These advantages support maintainability and reusability. The air object attributes are listed below:

Turbulence level decay rate
Position dependence decay relationship

Lagship

This object models the following vehicle's (lagship) reaction to the air disturbance due to a leadship. All of this model's lagship information, parameters, and equations are housed within this object. The generic engineering notation was used within this object. This approach has the following advantages:

- Transporting this model to different trainers would require no modification of the lagship equations – only the declaration package needs to be modified.
- Problems identified during simulator testing or subjective evaluation related to the lagship can be addressed without risk of accidentally touching non-lagship software. They can also be addressed directly because the various lagship attributes (features) are presented in different Ada procedures.
- Model attributes are presented by Ada procedures for each attribute. If future trainers don't require these attributes, the procedures are not included.

These advantages support maintainability and reusability. The attributes of the lagship object are listed below:

Lagship Geometry
Lagship Position (distance)
Lagship Delta Dynamic Pressure
Lagship Delta Aerodynamic Angles
Lagship Delta Aerodynamic Forces and Moments

The objects represent the key players in the real-world system of aerodynamic interference of a leadship on a lagship. The data available also fits the objects selected. For example, the leadship geometry data and vortex/downwash effects are given in terms of the leadship reference system (Reference 1) and therefore are housed in the leadship object. The leadship vortex/downwash and exhaust effects are applied to the lagship by computing a delta angle of attack and dynamic pressures in the lagship reference system. Therefore these items, as well as the delta forces and moments, are computed in the lagship object. The relative position attribute is placed in the leadship object. This is done because the exhaust and vortex/downwash data needs this parameter in the leadship axis system. Placing relative position in the leadship object also reduces an interface between primary objects.

PREVIOUS LINK OTHERSHIP EFFECTS SOFTWARE ARCHITECTURES

The problem of simulating leadship (othership) effects on a lagship is not unique to the B-2 ATD. It has been dealt with on at least five different trainers. Each previous solution used a different functional approach coded in Fortran. The level of fidelity in each approach reflected that project's training requirements. A brief description of the five projects' (A, B, C, F, and S) approaches, subsystem software architectures in block diagram form, and *general training requirements is presented below.* The block diagrams can be compared to Figure 2, B-2 Software Architecture. The highlighted blocks in Figures 2 through 7 compute the simulated leadship effects. During the comparison, note that a software subsystem is similar to a module.

Project A

This project training requirement was to instruct the pilot on formation flying. This approach included simulating leadship vortex/downwash and engine exhaust effects on the lagship. This training requirement mandated that this model be a high-fidelity model. This approach was also coded in the application-specific engineering notation. It consisted of two primary modules which interface with the lagship's forces and moments module. The software architecture for Project A is given in Figure 3. The first primary module (Module 1) computed the leadship's dynamics, relative position between the leadship and lagship, geometry parameters, and leadship exhaust effects on the lagship. The second module (Module 2) computed the vortex/downwash effects and summed the vortex/downwash with the exhaust effects and passed the total delta forces and moments to the lagship forces and moments module. Module 2 used geometry parameters, exhaust effects, relative position, and leadship dynamics computed in Module 1. No data was supplied by the airframe manufacturer for this model.

Project B

Project B's training requirement was to instruct the pilot of the receiver in an aerial refueling

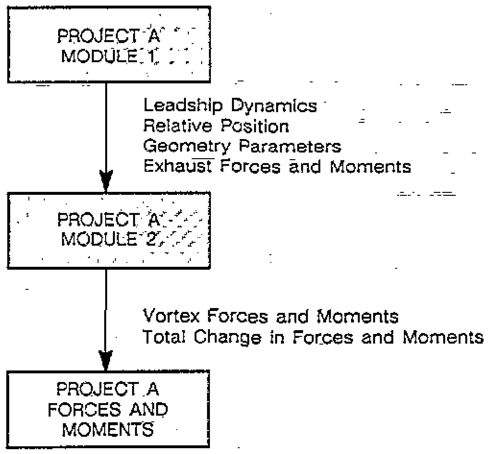
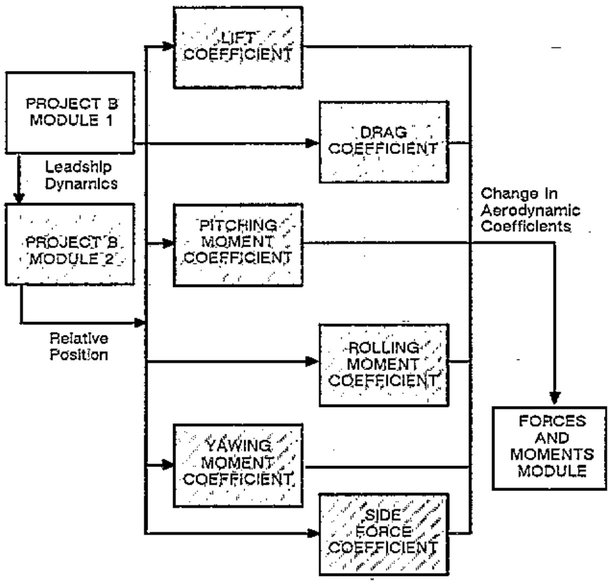

PROJECT A MODULE 1
Leadship Dynamics
Relative Position
Geometry Parameters
Exhaust Forces and Moments
PROJECT A MODULE 2
Vortex Forces and Moments
Total Change in Forces and Moments
PROJECT A FORCES AND MOMENTS

Figure 3 Project A Othership Effects Software Architecture

maneuver. This approach included simulating leadship vortex/downwash and bow wave effects on the lagship. This training requirement mandated that this model be a high-fidelity model. This approach was also coded in the application-specific engineering notation. It consisted of two primary modules, plus pieces of code distributed throughout six other existing modules. The software architecture for Project B is given in Figure 4. The first module (Module 1) contained the leadship dynamics. The second module (Module 2) computed the relative position of the leadship and lagship. The six existing modules were the six aerodynamic coefficient modules. The leadship effects on the lagship were handled as components of the total six aerodynamic coefficients. Totaled aerodynamic coefficients were passed to the forces and moments module, as is normally done in Link simulations. Several outputs of Module 1 were passed to Module 2 in the form of direction cosines. Module 2's relative position was passed to each coefficient module to be used to compute the aerodynamic coefficient component due to the leadship vortex/downwash and bow wave effects. The airframe manufacturer supplied data in the form of aerodynamic coefficient components.

Project C

This project had two training requirements. The first was to instruct pilots of the tanker in an aerial refueling maneuver. The second involved having a leadship in front of the aircraft. Therefore, the pilot could be either a leadship or a lagship. The training requirements were such that the ownship would only need to experience a general effect when in the proximity of an othership. Therefore, Project C was a high-quality, low-fidelity model containing translational wind effects. This approach included simulating the vortex/downwash and bow wave effects of



LIFT COEFFICIENT
PROJECT B MODULE 1
DRAG COEFFICIENT
Leadship Dynamics
PROJECT B MODULE 2
PITCHING MOMENT COEFFICIENT
Change In Aerodynamic Coefficients
Relative Position
ROLLING MOMENT COEFFICIENT
FORCES AND MOMENTS MODULE
YAWING MOMENT COEFFICIENT
SIDE FORCE COEFFICIENT

Figure 4 Project B Othership Effects Software Architecture

a leadship on a lagship. This model was coded in generic engineering notation with vehicle–specific data locally defined. The software architecture for Project C is given in Figure 5. It consisted of a single primary module which interfaced with the lagship equations of motion module. The primary module (Module 1) computed leadship dynamics, relative position, and delta translational wind components. These delta wind components were used to introduce the lagship effects into Project C's equations of motion. No data was supplied by the manufacturer for this model.

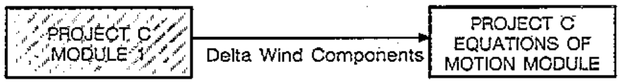


Figure 5 Project C Othership Effects Software Architecture

Project F

Project F's training requirement was to instruct the pilot of the receiver in an aerial refueling maneuver. The training requirements were such that the ownship would only need to experience a general effect when in the proximity of an othership. Therefore, this was a high–quality, low–fidelity model containing translational wind effects. This approach included simulating leadship vortex/downwash effects on the lagship. This approach was also coded in generic engineering notation with vehicle–specific notation locally defined. It consisted of two primary modules. The software architecture for Project F is given in Figure 6. The first module (Module 1) contained the leadship dynamics. The second module (Module 2) computed the relative position of the leadship and lagship. Both Modules 1 and 2 interfaced with the equations of motion module which computed changes in translational wind components. No airframe manufacturer data was supplied for this model.

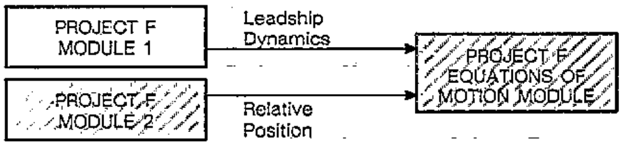
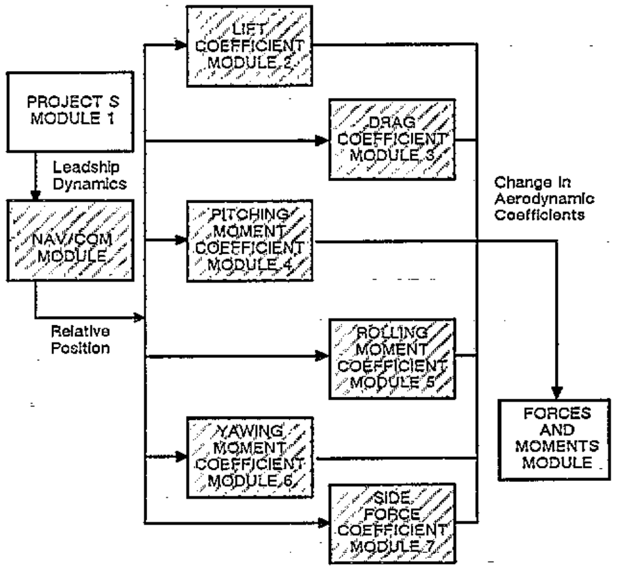


Figure 6 Project F Othership Effects Software Architecture

Project S

This project's training requirement was to instruct the pilot of the receiver in an aerial refueling maneuver. Although this project never implemented the leadship effects on the lagship, the planned approach is described in this paragraph. This approach would have simulated leadship vortex/downwash and bow wave effects on the lagship. It would have been coded in generic engineering notation. It would have consisted of seven primary modules, plus having code in one existing module. The software architecture for Project S is given in Figure 7. The first module (Module 1) would have contained the leadship dynamics. The existing navigation/communications module would have computed the relative position of the leadship and lagship. The other six primary modules (Modules 2 through 7) would have produced delta aerodynamic coefficients components due to the leadship and passed them to forces and moments to be incorporated with the other aerodynamic coefficients. No data was supplied by the airframe manufacturer.



LIFT COEFFICIENT MODULE 2
PROJECT S MODULE 1
DRAG COEFFICIENT MODULE 3
Leadship Dynamics
Change In Aerodynamic Coefficients
NAV/COM MODULE
PITCHING MOMENT COEFFICIENT MODULE 4
Relative Position
ROLLING MOMENT COEFFICIENT MODULE 5
FORCES AND MOMENTS MODULE
YAWING MOMENT COEFFICIENT MODULE 6
SIDE FORCE COEFFICIENT MODULE 7

Figure 7 Project S Othership Effects Software Architecture

SUBJECTIVE EVALUATION

The software architecture of the B–2 ATD project was evaluated along with Projects A through S. Each project was considered a possible candidate for a new program with personnel unfamiliar with the software. Assumptions were that the languages and the high–level operating systems' architectures would be the same. The projects were rated on reusability or transportability, ease of update or maintainability, complexity of software, and model fidelity. A rating of good, fair, or poor was assigned for the first three categories, and high or low was used for model fidelity. Following is a description of the ratings categories.

The reusability category rates the ease with which software can be removed from one simulator and incorporated into another. Application–specific designs of code decrease an architecture's reusability. Examples of this include vehicle dependency and computational machine dependency.

The maintainability category rates the ability to change data or features. Data embedded in code is difficult to find and change. The addition of a new feature, such as a new leadship, or the removal of a feature, such as the base escape feature, is also difficult to incorporate if features are not organized separately.

The complexity of software category is a rating of how easy each project's code is to understand by a new user, and also the ability of any user to troubleshoot problems. Separation of features is an important part of reducing the complexity of software.

Model fidelity is not a rating but a training requirement. Some projects did not require a high–fidelity model. This category is included for insight into each project's design.

The subjective evaluation of the given software architectures is presented in Figure 8.

PROJECT	Reusability or Transportability	Ease of Update or Maintenance	Complexity of Software	Model Fidelity
A	C	B	B	HIGH
B	C	B	A	HIGH
C	B	B	B	LOW
F	B	B	B	LOW
S	B	A	A	N/A
B–2	A	A	A	HIGH

Key:

A – Good
B – Fair
C – Poor

Figure 8 Evaluation of Othership Software Architectures

LESSONS LEARNED

During the evaluation of the previous architectures and throughout the testing process of the B–2 ATD othership subsystem, several lessons were learned.

Two primary lessons were learned during the development and testing of the OOD presentation of the othership subsystem. The first was that all leadship attributes should be housed together. The B–2 ATD is a full Weapon System Trainer (WST) which has training requirements that necessitate the interaction of additional air vehicles. Early in the program, it was decided to include the leadship dynamics within the subsystem that modeled those additional vehicles. The leadship attributes should have been added to that subsystem instead of being placed in the othership subsystem. This has the following advantages:

- The leadship computations depend on the leadship parameters computed by the leadship dynamics housed in a different subsystem, therefore reducing interfaces.
- The lagship relative position to the leadship affects the leadship dynamics in the aerial refueling maneuver.
- Reduces duplicated definition of leadship specific parameters.

The second lesson was that the use of Ada procedures to correlate to subsystem attributes has the following advantages:

- Increases transportability because model features can be subtracted or added with minimal work. The training requirements of a simulator should govern what features are necessary.
- This allows extra flexibility if computational resources become a problem. Model features that don't have direct training value may be eliminated with minimal effort.

Additionally, there were lessons learned during the evaluation of the various leadship effects software architectures. The higher–rated architectures tended to have two general characteristics. Both could be considered good software engineering techniques:

1. The leadship effects were contained within compilation units outside of any other ownship models. Such centralization allowed a user to easily debug problems. This enhanced maintainability.
2. The leadship effects were modeled in generic engineering notation. This enhanced transportability and minimized errors to parameters used in multiple locations.

CONCLUSIONS

The conclusions reached by this study are based on the lessons learned and the subjective evaluation of various othership software architectures.

First, othership effects that are modeled and coded in separate compilation units yield high maintainability. This allows for quick problem isolation and faster, safer modifications.

Second, othership effects that are modeled and coded in generic engineering notation yield high transportability. This allows for quick changes of particular vehicles without affecting the inner workings of the models/code. Generic engineering notation also increases maintainability if data updates occur because they can be done quickly and clearly.

Third, training devices which do not require high-fidelity othership effects have less of a need for high maintainability and transportability. A design that is less object-oriented in such cases is admissible because of the lower likelihood of data updates and reuse. It would only be reusable on another project that also had lower model fidelity requirements.

Lastly, the OOD architecture was the only architecture that provided high reusability and maintainability for a high-fidelity model.

REFERENCES

1. Weiss, Jerome M., "Othership System, Software Detailed Design Document", Link SDDD BD100-0101-02, April 1990.

ABOUT THE AUTHORS

Jerome Weiss is currently leading the Flight Simulation effort on the B-2 Aircrew Training Device. His responsibilities include the aerodynamic, propulsion, and motion simulations. Previously Mr. Weiss led the aerodynamic simulation effort for the Link C-135B and B-52H trainers. Prior to joining CAE-Link in 1980, he was employed by Cessna Aircraft Company, Boeing Military Airplane Company, and the Air Force Flight Test Center. Mr. Weiss holds an MS in Mechanical Engineering from California State University at Fresno (1978) and a BS in Aerospace Engineering from the State University of New York at Buffalo (1973).

Ruth Korba joined CAE-Link Corporation in 1988. She has since worked as a Systems Engineer on the SR-71 and the B-2 simulator programs. On both programs, she was involved in aspects of propulsion and aerodynamic simulation. Miss Korba graduated from Syracuse University in 1988 with a BS in Aerospace Engineering.