IN SEARCH OF A REUSABLE Ada AERODYNAMIC MATH MODEL

By John Hicks and Lynn Stuckey

Boeing Military Airplanes
Simulation and Training Systems
Huntsville, Alabama

ABSTRACT

The aerodynamic math model is one of the more important areas of a simulator design. The simulation industry needs a standard to use, not only for the data structure of the aerodynamic model, but also for the aerodynamic coefficient build-up logic. If and when the Department of Defense acquires a highly reusable aerodynamic math model, implementation of this model will be more cost-effective and streamlined on future simulation contracts. This would allow future combat trainers to be fielded quicker at a lower cost to the taxpayer. The use of the Ada language directly supports the creation of a reusable aerodynamic math model. One of the major goals in the development of Ada was to promote software that could be applied to future designs, thus making the reinvention of previous work unnecessary. Reusable components do not automatically fall out from the use of Ada. Reusability requires careful specification and design. Ada provides features such as packages, generics, unconstrained array types, and others that allow easier construction of reusable components. This paper presents an approach toward the development of a reusable aerodynamic math model for fixed-wing aircraft in the Ada language.

INTRODUCTION

Reusability is defined simply as the ability to use again. This same basic thought provides the theme for this paper: to present the issues that are important in the development of an aerodynamic model (fixed-wing aircraft) that can be used again. What would be the potential of a model that is reusable for every aircraft, no matter what manufacturer made it or what simulator manufacturer produced the simulators? First, any simulators that would be needed (training and engineering) would be ready for testing against flight test aerodynamic data as soon as the data was available. Second, the engineers working on the aerodynamic model would have a greater chance of implementing the aerodynamic data correctly regardless of the aircraft manufacturer or the simulator manufacturer. What happens if a reusable model is not produced? The wheel is reinvented, continuing to waste the taxpayer's money.

The vehicle for this development is the Ada software language. The problem is not in developing the laws of aerodynamics, but simply in finding a method for efficiently implementing given laws in a way that can be adapted easily for any fixed-wing aircraft. Ada was developed for large computer programs to be used in embedded computer systems. The DoD and ANSI have adopted Ada as a software standard. It is this standardization that allows an attempt at the development of true reusable software. "The very high degree of standardization of Ada will enhance both the use of packages of standard software and the reuse of software created for other systems."(1) Ada also promotes sound software engineering, which is a must in the development of reusable software. Ada improves on many earlier languages in that the development of the language was guided by a strong concern for reliability, readability, and maintainability. As stated by Grady Booch "Unlike most other production high-order languages, such as FORTRAN, COBOL, or even Pascal, Ada not only embodies many modern software development principles but also enforces them."(2)

Specifically, this paper presents: 1) some of the problems that affect the design of a reusable model, 2) the basic structure of the aerodynamic model that needs to be addressed, 3) some of the Ada language features which are basic to reusability, 4) the issue of corporate design strategies, 5) a candidate approach towards the reusable model and 6) some of the benefits of developing an aerodynamics model that is reusable.

INDUSTRY-WIDE REUSABILITY?

The primary problem in trying to design a totally reusable aerodynamic

math model is a lack of standardization. The International Air Transport Association, IATA, has issued "Flight Simulator Design and Performance Data Requirements"(3) to address what data aircraft manufacturers should expect to provide to simulator manufacturers regarding simulator requirements. The authors would like to suggest that the Department of Defense issue a similar data requirements guide. This data requirements guide would need to be tailored for aircraft functionality and could not be considered as a contractual item in the near future. However, standardization of the design criteria would make the design of simulators much easier and less costly.

The simulator data requirements document is a good set of guidelines for aircraft manufacturers to use, but does it cast the aerodynamic data in concrete? No. It is unrealistic to attempt to do so. Although most parts of the aerodynamic model are common to most aircraft, there always seems to be some additional odd quirk required in the model to make the simulation realistic. A model could be developed and called "reusable", but the problem with this approach is that the model would become extremely bulky and changes would have to be made every time a new stability derivative is introduced. This points out that an absolutely reusable aerodynamic (aero) model is unrealistic at this time. So, what kind of approach is to be used?

Another approach to consider is that the aero data could be processed in an engineering simulator to correlate the current data into a form which is usable with a more general aero build-up logic. This software tool is sometimes referred to as a coefficient correlator. This approach not only makes sense, but until industry standards are set and in widespread use, has a much better chance of success. The coefficient correlator would be used before implementing the aero data into an "in-house", reusable aero math model. The modeling details of the correlator depend on the design of the reusable aero model.

STRUCTURE OF THE AERODYNAMIC MODEL

In order to point out where and how the intelligent use of Ada could help in the production of a reusable aerodynamic math model, some knowledge of the basics may be useful to the readers.

Aerodynamic models are used to simulate the forces and moments acting on the airframe due to unequal atmospheric pressures. These forces and moments represent the same forces and moments which contribute (along with the effects of propulsion and landing gear) to the action/reaction cause and effect relationship of the aircraft movements. The forces and moments acting on the aircraft are used with Newton's Second Law to generate linear and angular accelerations. See Figure 1. These accelerations are integrated first into velocities and rates and then into linear and angular displacements. The linear and angular displacements, generated in real time, represent the six-degree-of-freedom aircraft motion model.

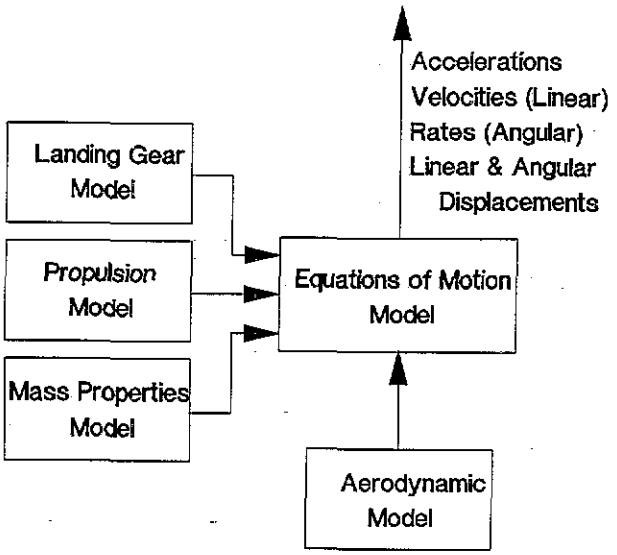

Figure 1

Two basic areas need to be discussed when explaining the simulation of aircraft aerodynamics: 1) aero coefficient data and 2) aero coefficient build-up logic.

Aerodynamic Data

The aero coefficients are actually a series of stability derivatives which, integrated, result in six "total" coefficients. These six coefficients represent non-dimensional aero forces and moments acting on the airframe. Once the coefficients are found, they are converted into the forces and moments used by the equations of motion to develop aircraft accelerations along and about the aircraft body axes. The aero coefficients are developed by several methods including the use of empirical formulas, wind tunnel data, flight test data and in some cases, expert opinion. In simulations these normally exist in the form of a multitude of data tables which are accessed to look up and interpolate the data. Occasionally constant values, straight line equations or simple curvilinear equations are used instead, but those are the exception rather than the rule.

The use of standard aero data naturally implies that standard aero data tables are required. Reference 3 provides two suggested standard data table formats, one from Boeing (which the authors prefer) and one from Aerospatiale. A sample table and further explanation of the table set-up can be found in Reference 3.

Another reason that the aero data and the associated build-up logic are difficult to standardize is because of the amount of dependency they have on each other. A simplified example of aerodynamic functional dependencies is shown in Figure 2. The numbers denote the severity of effect given functions might have on given coefficients with 3 being a major effect, 2 being a moderate

Coefficient / Function	Lift	Drag	Side	Roll	Yaw	Pitch
Pitch Rate	2	0	0	0	0	3
Roll Rate	0	0	2	3	3	0
Yaw Rate	0	0	2	3	3	0
Angle-of-Attack	3	2	1	1	1	3
Sideslip	0	1	3	3	3	1
Flap/Slat	3	2	0	0	0	3
Rudder	0	1	2	2	3	0
Aileron	1	1	1	3	1	0
Stabilator	2	1	0	0	0	3

Sample Aerodynamic Dependency Matrix

Figure 2

effect, 1 being a minor effect and 0 being a negligble effect. This matrix serves to give a designer a starting point for setting the structure of the design. The actual matrix of aerodynamic data tables required for a simulation is normally larger than this. Again this should serve only as a starting point because many aircraft manufacturers use several different ways of coming up with the same basic answer. Another use of a matrix such as this is to remind the designer to keep in mind actual functional dependencies.

The Aero Build-Up Logic

The next potential area of standardization is the aero build-up logic. The aero build-up logic is used to determine which aero data are to be looked up and used in the final aero equations. Build-up logic normally consists of a series of conditional control structures with nested calls to table interpolation routines to access the aerodynamic coefficient derivatives. This logic can get quite complex and simulation fidelity is lost if this logic is not interpreted correctly from the design criteria. The aero build-up logic will vary from company to company and, within company, from aircraft to aircraft.

The build-up logic is the section which makes the necessary inquiries about the aircraft configuration and external conditions which cause certain aero derivatives to be used rather than others. For example, many aircraft use flaps as high-lift devices. However, flaps are normally only extended at low airspeeds. Therefore, a stability derivative may be described with two mutually exclusive sets of data tables. One set of data might define a coefficient when the flaps are extended and the other set of data defining the same coefficient when the flaps are retracted. The first set of data would probably be dependent upon flap angle and angle of attack. The second set of data would probably be dependent upon Mach number, altitude and angle of attack. The build-up logic section of the aero math model makes the determination of which data tables are to be used and makes the function calls to search the data table for the proper data value per the dependent data values. The data value is then returned to the summation module for inclusion into the "total" coefficient.

Of all the areas requiring standardization, the build-up logic will probably be the most difficult because no matter how many different stability derivatives the training simulation engineer includes in the "reusable" model, the engineering simulation engineer will always need additional aerodynamic components to describe the aero math model more completely. The best approach towards designing an "in-house" reusable model is to set it up with room and capability for growth. This becomes difficult in the Ada language because requirements are supposed to be complete before any code is written. However, with the proper design methodology, it becomes simple to add parameters at appropriate points in the structure with little effect on the rest of the model.

LANGUAGE FACILITIES ADVISORY

The idea of reusable software components is not new, it is just that in the past there has been no convenient vehicle for the design of the

components. With the emergence of Ada as a DoD software standard there is now a vehicle available. "Ada is a much better language for supporting reusability in flight simulation than Fortran 77 because the language has been designed with reusability in mind."(4) Ada also directly supports the principles of sound software engineering: modifiable, efficient, reliable, and understandable software. These principles are required in a reusable software model. Ada supports these principles by introducing new features such as packages, information hiding, abstraction, strong typing, generics, and separate compilation. These features completed the vehicle for the development of reusable software components.

Certain features or facilities in Ada prove to be very useful in developing reusable software. Packages are perhaps the most important facility in Ada. They serve as the collections of logically related resources. Hence, they are the fundamental building blocks of any Ada system. Packages are the authors' basis for a structure. They are required in order to encapsulate objects, data, and types used in the reusable aerodynamic model. Packages provide the means of structuring the reusable components into logically related units. Packages also provide the capability to isolate the core aerodynamic functions common to all aircraft simulators from those that are device-specific. Packages are the basis for abstraction and information hiding. An abstraction is a simplified description of a system that emphasizes certain properties while hiding or suppressing other properties. Each of the elements of an abstraction can itself be an abstraction for details at a lower level. Abstractions are central to all intellectual activity and thus to the development of a reusable aerodynamic model. Abstraction is used to separate the physics, control, and data of the *problem, thus simplifying* the solution and resulting in a more understandable and modifiable system.

As stated by Bray and Pokrass, "Ada is strongly typed, which means that the programmer specifies the type of every object, that the type cannot be changed, and that an object can be used only in operations defined for its type."(5) This strong typing allows the compiler to check the consistency of the code and also enables the compiler to detect most errors normally left uncovered until integration. The use of types, especially enumeration types, is recommended in the development of the reusable aerodynamic model. Enumeration types are used to define or model the environment while enhancing the readability and clarity of the code. These types can be the basis for implementation-independent code generation. Strong typing forces a rethinking or different mindset about the location and usage of types. This will mean a tighter control of type declarations.

Generic units are templates of packages or subprograms. They are the key to the creation of certain reusable software components. Generics provide the ability to create templates of program units that can be written just once and then tailored to particular needs at instantiation. This feature is supported by the authors as a basis for the interpolation and zone routines needed by the aerodynamic build routines. The routines are basically the same except for execution rate dependencies. Templates could be written and then instantiated with a parameter of execution rate.

The Ada ability of separate compilation is another plus in designing a reusable software component. This feature allows decomposition of a body into smaller compilation units and the capability to defer creation of a stub for the body of a subprogram. This feature is also an advantageous tool for compiled abstraction, visibility control, and avoiding recompilation penalties. The SEPARATE clause defines dependency among program units, visibility is inherited from the parent, and type rules are enforced across the *program body*.

There are many advantages to using Ada as the vehicle to developing a reusable software component. The language features provided by Ada not only provide tools for reusable software, but also enforce sound software engineering principles. A final note though: the tendency to misuse features to define complex or unusual data structures merely to facilitate a "clever coding" technique must be avoided (i.e. coding artistry vs sound software engineering).

CORPORATE DESIGN STRATEGY

A truism about reusable software is that reuse must be designed in from the onset of development. Reusability does not simply happen, it must be planned and worked for. The planning involved with developing reusable software must begin at a managerial level. There must be a corporate design strategy. This strategy serves a dual purpose:

(1) it advocates the use of company resources toward the reusable software goal and

(2) it determines the concept and structure of the reusable components.

The first purpose, allocation of resources, is difficult to embrace. The design and development of reusable and nonreusable software have apparent differences. Since the reusable software must meet more than just the immediate needs of the model, requirements analysis and development of detailed specifications are more demanding than for application specific software design. This extended effort will increase both the cost of the reusable component (initially) and the time spent in the development cycle. It should also be recognized that the initial model of the reusable component will not be completely reusable. Thus an iterative process of refinements must be expected. Not only will a reusable component require time, and therefore money, but the resource of talent must also be invested. The design effort will require the participation of some of the company's best engineers. (If the commitment to investing time, money, and talent is not provided by the company, then a reusable software component is only a pipe dream.)

The second purpose of the corporate strategy is more complex and involved. It deals with the concept of reusability and its structure. There are generally three concepts of reusability. The first is the concept of a totally reusable piece of software. This means that the entire model is reusable and requires only minor modification to provide a suitable model. The second concept is one of a reusable frame/skeleton in which application dependent software is inserted in order to form the software model. The last of the concepts is the obvious combination of concepts one and two. In this concept the bulk of the software is reusable and the application dependent software is abstracted into plug-in units. This third concept makes the most sense and is very easy to implement in Ada.

The structure of the reusable component must be determined and put forth as a standard for the company. This structure should be documented in the form of a "structure report". The idea of a structure report is being pursued and its development is encouraged by the authors at this point. In terms of Ada this structure report simply defines, as a minimum:

(1) How you package.
(2) What is in a package.
(3) How do packages communicate.

This report will ease the two main problems incurred when implementing a reusable software model. These problems as stated by Mitchell D. Lubars in "Code Reusability in the Large Versus Code Reusability in the Small"(6) are as follows:

(1) The proper interaction between the reusable code and the rest of the program must be set up. As in performing an organ transplant, all the life lines and vessels must be made to line up correctly between the transplanted code and the host program.

(2) The reusable code may have to be modified to precisely fit the new application. The code may solve a slightly different case, or may be too general or specialized as originally coded. In either case, some changes may have to be made before the reusable code is suitable for use.

A side issue of corporate strategy deals with the formidable obstacle of programmer acceptance. There is a reluctance in most engineers to accept someone else's code as complete and good. This reluctance is especially true of aero engineers with flight models. Since the quality of the flight models is paramount on a flight simulator, aero engineers find difficulty depending on other engineer's code. This can only be overcome if the engineer is highly confident in the reusable software component. This confidence must be instilled at a managerial level. The corporate strategy is the first essential in a good reusable aerodynamic model.

A CANDIDATE APPROACH

The concept of reusability and the structure of the reusable aerodynamic model is very simple. The authors' concept of reusability is the combination concept with the aircraft-dependent data being abstracted away to plug-in components. The structure is based on the use of Ada packages to encapsulate and separate reusable and non-reusable components in the aerodynamic model. Figure 3 graphically describes the aerodynamic model

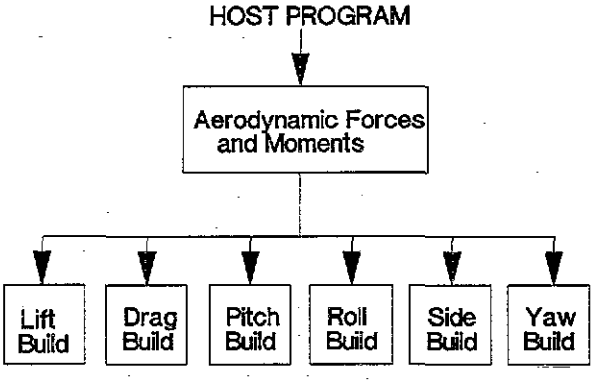

Figure 3

suggested by the authors. The Aero Forces and Moments object is a package containing a main procedure. This procedure is the only interface with the host program. The interface is accomplished via the procedures' parameter lists. The object is then abstracted into six sub-objects consisting of all the build models. These sub-objects then access generic interpolation and zone routines (Figure 4). The generic routines are instantiated for different execution rates. The routines access the aero data tables which are abstracted so to be easily replaced with each new aircraft's data. Thus the aerodynamic

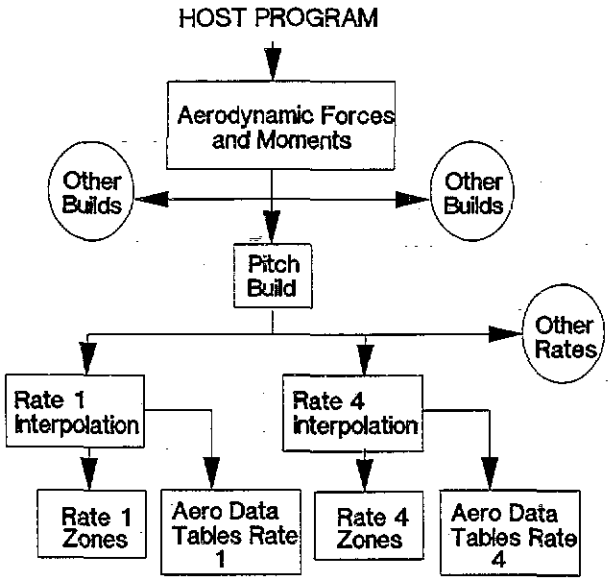

Figure 4

model's reusable units consist of the Aero Forces and Moments executive, the build routines, and the interpolation and zone generics. The application dependent data is abstracted away to become a separate package that is plugged into the model. The last part of the model is a package of types. Since only types are in this package, a datapool situation is avoided. This types package will be the basis for variables and logic in the reusable code. An example of its contents would include an enumeration type containing all possible coefficients incurred in fixed-wing aircraft. A subtype of this enumeration would contain the coefficients applicable to the particular application and would be modified with each use. The code is designed such that it operates on the contents of the subtype.

The reusable aerodynamic model must also answer the questions posed in a structure report. It is then the authors' contention that the candidate approach answers to the structure report are:

(1) *How you package:* Based on an object abstracted design and that all knowledge about the behavior of objects is strictly encapsulated within the objects themselves.

(2) What is in a package: Simply the objects noted in Figure 3 and 4, their procedural interfaces, and all pertinent information about the object.

(3) How packages communicate: Provided through procedure calls. All information passing is performed up and down the model and never side to side. This form of communication provides information hiding, alleviates the numerous problems of datapools, and furnishes the user with an accurate list of inputs and outputs.

BENEFITS OF REUSABILITY

The benefits of a reusable aerodynamic model are easily recognizable. The first benefit is in development cost. This benefit is a perceived benefit. The initial software development cost will be higher, to account for the time and generalized design required for reusable software. But with each reuse of the model the development cost is greatly reduced. The next benefit of a reusable aerodynamic model is in the area of software engineering. The standardization of the model and the software language promotes/delivers the goals of software engineering. These four goals are modifiability, efficiency, reliability, and understandability. The last obvious benefit is in the area of transportability. With standardization of aero data and models comes the ability to transport code from company to company. The standardization of Ada provides transportability from machine to machine. This benefit leads directly to the emergence of a software components library. The reusable models will encourage a transition to the use of standard software packages, saving effort otherwise expended in creating special-purpose software. This will allow engineers to concentrate on solving problems at a higher level of abstraction. Large software systems could be developed by combining existing software components, with new software being needed only as a glue to hold the components together.

CONCLUSION

The final analysis is that simulator software design will be more reliable and maintainable and could be developed quicker and cheaper after reusable components are available. This means that combat readiness could be improved by paying more attention to the details that support reusable software. The purpose of this paper was to remind the software community of the importance of a reusable model and to introduce a plausible approach at the design of a reusable aerodynamic math model.

REFERENCES

[1] K. Shumate, Understanding Ada, Harper & Row, New York, 1984 p.44.

[2] G. Booch, Software Engineering with Ada, Second Edition, Benjamin/Cummings, Menlo Park, California, 1987, p. XV.

[3] "FLIGHT SIMULATOR DESIGN AND PERFORMANCE DATA REQUIREMENTS", International Air Transport Association, 2nd Edition, 1986.

[4] D. T. Ross, J. B. Goodenough, and C. A. Irvine, "Software Engineering: Process, Principles, and Goals", Computer (May 1975), p. 65.

[5] G. Bray, and D. Pokrass, Understanding Ada: A Software Engineering Approach, John Wiley & Sons, New York, 1985, p. 8.

[6] M. D. Lubars, "Code Reusability In The Large Versus Code Reusability In The Small", ACM SIGSOFT Software Engineering Notes, vol. 11, no. 1 (January 1986) p. 21.

ABOUT THE AUTHORS

MR. JOHN HICKS is a systems engineer with the Simulation and Training Systems organization of Boeing Military Airplanes in Huntsville, Alabama. He has been responsible for implementation of design criteria, test and integration of the aerodynamic models in several Boeing S&TS simulator projects. Mr. Hicks has 10 years of experience with Boeing in the field of aerodynamics and holds a Bachelor of Science degree in Aeronautical Engineering from Tri-State University, Angola, Indiana.

MR. LYNN D. STUCKEY JR. is a software systems engineer with Boeing Military Airplanes in the Simulation and Training Systems division. He has been responsible for software design, code, test, and integration on several Boeing simulator projects. He is currently involved in research and development activities on Ada software development. Mr. Stuckey holds a Bachelor of Science degree in Electrical Engineering from the University of Alabama, in Huntsville.