The DaveMLTranslator: An Interface for DAVE-ML Aerodynamic Models

Melissa A. Hill[*]
Unisys Corporation, NASA Langley Research Center, Hampton, VA, 23681

E. Bruce Jackson[†]
NASA Langley Research Center, Hampton, VA, 23681

It can take weeks or months to incorporate a new aerodynamic model into a vehicle simulation and validate the performance of the model. The Dynamic Aerospace Vehicle Exchange Markup Language (DAVE-ML) has been proposed as a means to reduce the time required to accomplish this task by defining a standard format for typical components of a flight dynamic model. The purpose of this paper is to describe an object-oriented C++ implementation of a class that interfaces a vehicle subsystem model specified in DAVE-ML and a vehicle simulation. Using the *DaveMLTranslator* class, aerodynamic or other subsystem models can be automatically imported and verified at run-time, significantly reducing the elapsed time between receipt of a DAVE-ML model and its integration into a simulation environment. The translator performs variable initializations, data table lookups, and mathematical calculations for the aerodynamic build-up, and executes any embedded static check-cases for verification. The implementation is efficient, enabling real-time execution. Simple interface code for the model inputs and outputs is the only requirement to integrate the *DaveMLTranslator* as a vehicle aerodynamic model. The translator makes use of existing table-lookup utilities from the Langley Standard Real-Time Simulation in C++ (LaSRS++). The design and operation of the translator class is described and comparisons with existing, conventional, C++ aerodynamic models of the same vehicle are given.

Nomenclature

α angle of attack, degrees
β angle of sideslip, degrees
ϕ roll attitude angle, degrees
θ pitch attitude angle, degrees
φ yaw attitude angle, degrees

I. Introduction

It can take weeks or months to incorporate a new aerodynamic model into a vehicle simulation and validate the performance of the model. The Dynamic Aerospace Vehicle Exchange Markup Language (DAVE-ML) has been proposed as a means to reduce the time required to accomplish this task by defining a standard format for typical components of a flight dynamic model. The purpose of this paper is to describe an object-oriented C++ implementation of a class that interfaces a vehicle subsystem model specified in DAVE-ML and a vehicle simulation. Using the *DaveMLTranslator* class, aerodynamic or other static subsystem models can be automatically imported and verified at run-time, significantly reducing the elapsed time between receipt of a DAVE-ML model and its integration into a simulation environment. The translator performs variable initializations, data table lookups, and mathematical calculations for the aerodynamic build-up, and executes any embedded static check-cases for verification. A focus on efficiency during the implementation of the translator resulted in the capability to execute in real-time. Simple interface code for the model inputs and outputs is the only requirement to integrate the DaveMLTranslator as a vehicle aerodynamic model. The translator makes use of existing table-lookup utilities from

---

[*] Software Engineer, Simulation Development and Analysis Branch, M/S 169.
[†] Aerospace Technologist, Dynamic Systems and Control Branch, M/S 308, AIAA Associate Fellow.

the Langley Standard Real-Time Simulation in C++ (LaSRS++). The design and operation of the translator class are described and comparisons with existing, conventional, C++ aerodynamic models of the same vehicle are given.

LaSRS++ is a C++-based simulation framework developed by and in use at NASA's Langley Research Center[1]. The framework is used in both non-realtime-desktop and pilot-in-the-loop hard-real-time modes for flight simulation studies. LaSRS++ provides a reusable object-oriented environment to support single-vehicle and multiple-vehicle aerospace simulations in a variety of motion-base and fixed-base cockpits. Langley uses these facilities to develop and analyze crew-vehicle systems interactions, control laws, conventional and revolutionary aerodynamic vehicle configurations, and provide pilot and astronaut familiarization capability.

DAVE-ML is an eXtensible Markup Language (XML)-based grammar for describing simple-to-complex aerodynamic models for aerospace vehicles. DAVE-ML describes nonlinear aerodynamic tables and functions of unlimited dimensions, uncertainty bounds and functions, build-up equations, data provenance, and embedded check-case/verification data in a single text file. It has been jointly developed by NASA and the AIAA Modeling and Simulation Technical Committee and is being considered for adoption as an AIAA/ANSI standard[2,3,4].

II.   Motivation

The traditional method to re-host a flight dynamic model at Langley involves, at the very least, reformatting tabular aerodynamic databases from the original format to the LaSRS++ table source format. Usually the source code for the aerodynamic model is structured in a way that it is not compatible with the LaSRS++ framework and must be rewritten to some extent, another manual operation. Finally, the check-case data is usually a separate, sometimes paper, document, requiring some effort to generate check plots to verify proper reimplementation at the Langley facility. A process similar to this takes place each time a dynamic model is shared between simulation facilities, and indeed, every time the model is re-hosted into a dynamic analysis tool.

DAVE-ML was developed to obviate the need for manual processes, which sometimes takes weeks of effort. By defining a standard format for typical components of a flight dynamic model, the implementation of externally developed models can be done by automated processes. The emergence of XML, MathML (a mathematical relationship XML grammar) and UNICODE character encoding standards has provided a means to this end; using well-designed parsers makes this automation straightforward and nearly instantaneous.

III.   Description of the Translator

The DAVE-ML translator class (*DaveMLTranslator*) loads, at run-time, a DAVE-ML description of an aerodynamic model into a LaSRS++ simulation. Any included check-case data sets are verified after the file is parsed into memory-resident calculation structures, providing immediate verification of the model implementation, before beginning real-time operation.

Demonstrations of this capability used a subsonic aerodynamic model of the F-16A aircraft as described in the paper by Garza and Morelli[5], and an aerodynamic model of the HL-20 lifting body. The HL-20 model is a fairly complex full-envelope subsonic and supersonic hybrid aerodynamic model[‡] that is representative of a high-fidelity engineering simulation[6]. In both cases, the simulation characteristics of the DAVE-ML implementation were virtually identical to the traditional aerodynamic models constructed using FORTRAN or C/C++ source files. These comparisons were done both quantitatively (time-history comparison matches) and qualitatively (using informal piloted evaluations).

IV.   Class Design

A. LaSRS++ Overview

In the LaSRS++ framework, the *VehicleSystem* class interfaces vehicle models with subsystem models to decouple the two[7,8]. In the F-16A, the *F16aAeroSystem* is the interface between the *F16a* and the *F16aAero* classes (Fig. 1). The *DaveMLTranslator* was designed to replace a subsystem model (in this case, *F16aAero*), so that it can be included in any vehicle without disrupting the existing aircraft design. The corresponding *VehicleSystem*, (in this case the *F16aAeroSystem*) must be modified to interface with the *DaveMLTranslator*.

The design of the *DaveMLTranslator* class had the following requirements:
1. Implement the *DaveMLTranslator* to be independent of the vehicle model.
2. Automatically import and verify the subsystem model at run-time.
3. During import, identify and resolve order dependencies in the subsystem model.

[‡] Containing three-dimensional table lookups of coefficients of polynomial describing functions

4. Maximize the efficiency of the translator to enable real-time execution (at 50 Hz).
5. Use a generic interface so that the *DaveMLTranslator* can be used for any subsystem model specified in the DAVE-ML grammar.
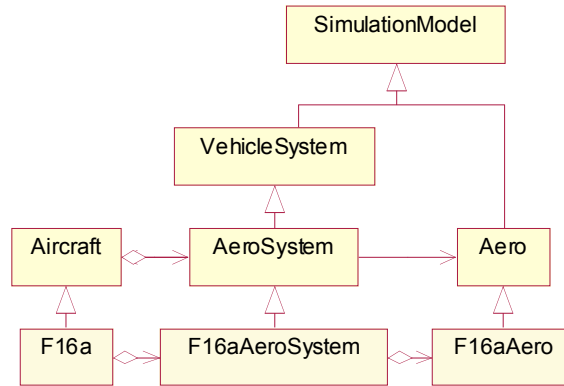

Figure 1. The LaSRS++ *VehicleSystem* Interface.

The *DaveMLTranslator* was designed as a stand-alone class, relying only on an existing library for performing data table lookups, and a new library implemented to handle the Mathematical Markup Language (MathML) elements of DAVE-ML. Neither the table lookup library nor the MathML library depends on the vehicle model, making the combination of the *DaveMLTranslator* and the libraries independent of the vehicle model to which the DAVE-ML model applies, satisfying requirement 1.

To minimize execution time during simulation operations, the bulk of the work in the *DaveMLTranslator* class is performed in the constructor, a member method that initializes the object. The DAVE-ML model, specified in plain text format, is parsed one time, completing costly string comparisons and file parsing during the load time of the simulation models, when real-time execution is not yet critical. The following steps are implemented in the *DaveMLTranslator* constructor:
1. Loading the DAVE-ML model from the text file and validating the model format
2. Handling elements specific to the DAVE-ML grammar
3. Resolving order dependencies in the model
4. Executing model check-cases

B. Loading and Validating the DAVE-ML Model
The XML Document Object Model (DOM) represents an XML document as a tree, with elements forming the nodes of the tree. The DOM provides a standard way of accessing and manipulating XML documents. Various XML parsing libraries have methods to facilitate traversing the document tree and pulling information from the element nodes. For the *DaveMLTranslator*, **libxml2**[9], the XML C Parser and Toolkit of Gnome, was chosen. **libxml2** provides the ability to parse the document and validate it against a Document Type Definition (DTD). The validation ensures that the model is well formed according to the grammar. Parsing of the DAVE-ML model into a node tree and validation of the DAVE-ML model is completed with two function calls to the **libxml2** library. The calls return a pointer to the root node of the newly created document tree. If the DAVE-ML model is not valid according to the DTD, the program exits with an error message.

C. Element Handling
Once the node tree has been created, preorder traversal is performed, starting from the root node (a `DAVEFunc` element). Element "handler" methods are called for each child node. The handler methods are specific to DAVE-ML elements, and they implement functionality based on the element definition in the DAVE-ML DTD. The execution of handler methods for each node proceeds until a leaf is reached, at which point control moves to the next sibling node in the tree. Once the traversal is complete, the DAVE-ML model has been imported, satisfying the first part of requirement 2.

During traversal of the tree nodes, C++ structures are used to store information about DAVE-ML elements. The structures loosely map to the DAVE-ML elements, though there is some overlap where multiple element types are

3
American Institute of Aeronautics and Astronautics

stored within a single structure. As handler methods are called for each node in the tree, new structures are dynamically allocated and populated with the element details. The structure pointers are stored in standard template library containers in the *DaveMLTranslator*.

The two primary structures are the *VariableDef* and the *Function*. These structures correspond to the DAVE-ML elements with the same names, and are the only structures that need to be updated during simulation execution; all of the other structures used in the *DaveMLTranslator* class exist as supporting players. The *DAVEFuncElement* structure was created as the base structure for the *VariableDef* and *Function*; it contains their common data members and methods. Each structure has data members that map to the attributes and sub-elements of the corresponding DAVE-ML element. The majority of the data members are intrinsic types; some are DAVE-ML specific structures (e.g., *IndependentVarRef*), and some are object pointers specific to the data table lookup library (e.g., *DependentVariable*) (Fig. 2).
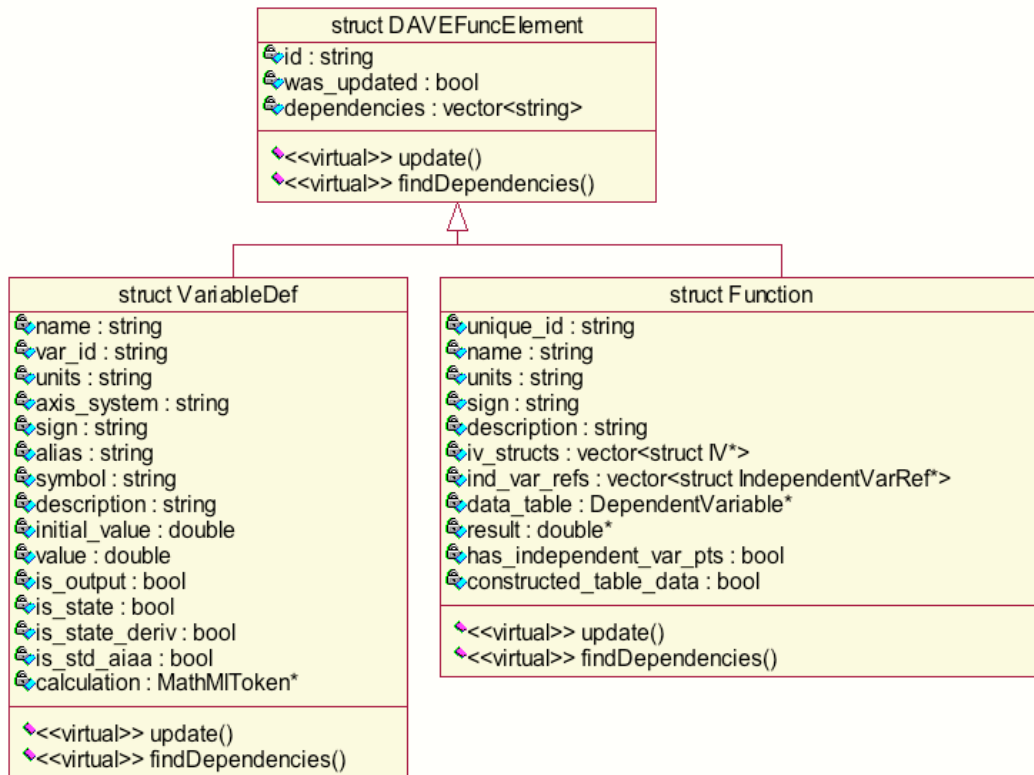

Figure 2. The *DAVEFuncElement*, *VariableDef*, and *Function* Structures

D. Mathematical Markup Language

A special case in the DAVE-ML grammar is the `calculation` element, which can be a child element of the `variableDef`. The `calculation` uses elements defined in the Mathematical Markup Language (MathML) to describe equations. To simplify the *DaveMLTranslator* implementation and to maximize code reuse for future applications using MathML, the MathML elements and handlers were designed within a separate class hierarchy. MathML is an extensive grammar; for the *DaveMLTranslator*, only the subset of MathML encountered in the F-16A and HL-20 aerodynamic models was implemented (Fig. 3).
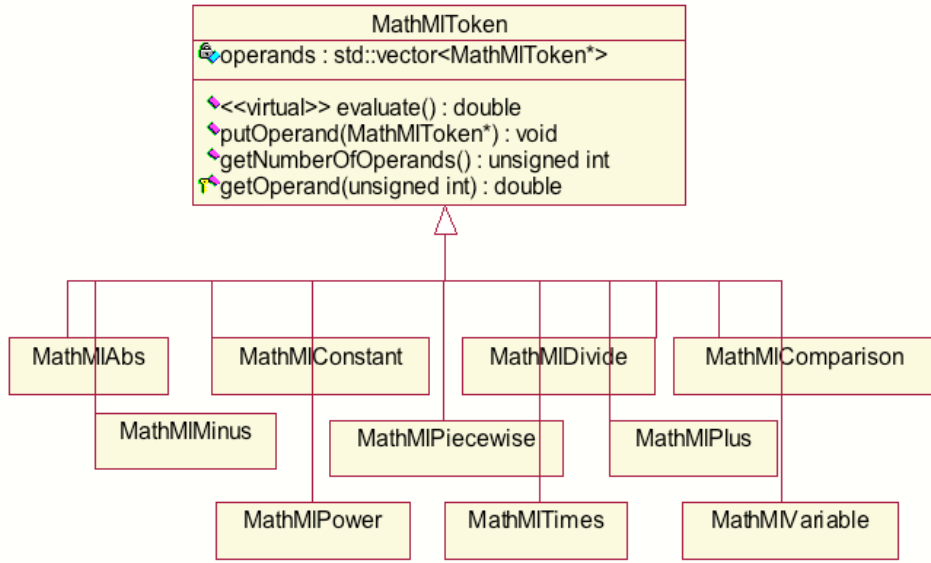
American Institute of Aeronautics and Astronautics

Figure 3. The MathML Class Hierarchy

MathML uses prefix notation, in which each operator precedes its operands. In prefix notation, the term $b/2V_t$ is expressed as "$/b(*2V_t)$". In the MathML handler method in the *DaveMLTranslator*, a MathML expression is contained within a single instance of a *MathMlToken*. The first operator is the entry point for the expression; in this case it is a *MathMlDivide* object. All participating operands and operators are instances of the appropriate derivative class of *MathMlToken*, and are pushed onto the *operands* member of the parent token (e.g., the *MathMlDivide* object), using the *putOperand* method (Fig. 4). Expressions are evaluated by recursively calling the *evaluate* method for each operator and operand while traversing the *operands* vectors. The *MathMlVariable* and *MathMlConstant* classes are the end conditions of the recursion; they return a value rather than returning the result of another call to *evaluate*.
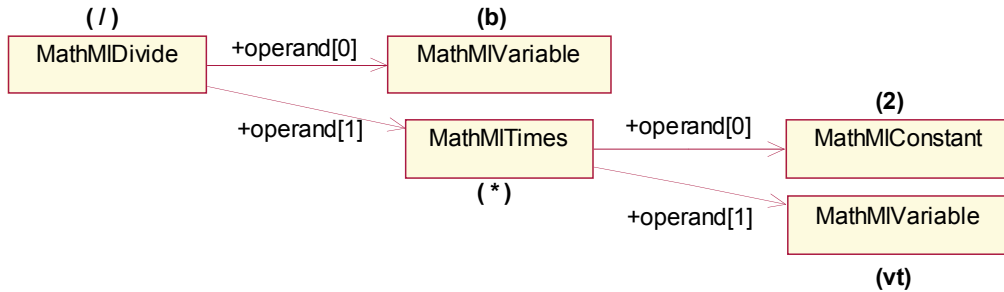

Figure 4. The Expression / b (* 2 vt) as Implemented with the MathML Class Hierarchy

E. Resolving Order Dependencies

In the DAVE-ML description of a subsystem model, it is possible to use variables in calculations or function table lookups before those variables are defined. If the *DaveMLTranslator* parsed and evaluated a model exactly as it was defined, multiple passes might be required during simulation execution to resolve the dependencies. To reduce the per-frame update of a DAVE-ML model to a single pass, a sorting method was implemented. The *sort* method is called once during initialization of the *DaveMLTranslator*. The DAVE-ML variableDef element can only depend on other variables within a calculation, and the function elements depend on independent variables,

expressed as `independentVarRef` or `independentVarPts` elements. The base structure *DAVEFuncElement* has a *dependencies* vector for storing the unique IDs of the dependencies.

Dependencies in `variableDef` elements are stored when the `calculation` elements are handled. The *findDependencies* method in the *Function* structure iterates through the list of independent variables, adding them to the *dependencies* vector. Once all of the dependencies have been found, the *sort* method in the *DaveMLTranslator* iterates through all of the *VariableDef* and *Function* structures, checking whether the dependencies precede the structure in the standard template library list of *DAVEFuncElements* to be updated. If they do not, the list is reordered to resolve the conflict. In the case where there are circular dependencies that cannot be resolved, an option is set in the *DaveMLTranslator* to perform multiple iterations of the *update* method during execution. The implementation of the *sort* method satisfies requirement 3, and partially satisfies requirement 4, improving the efficiency of the model by reducing the number of passes through the *update* method during simulation execution.

F. Updating the Model

To update a dynamic model in the *DaveMLTranslator*, the *update* method is called for each *DAVEFuncElement* structure in the sorted list (i.e., all *VariableDef*s and *Function*s). Only the *VariableDef* structures that contain calculations require action; the *evaluate* method is called on the calculation, and the result is stored in the *value* data member of the *VariableDef* structure. In the *update* method for the *Function* structure, independent variable values are set, and a data table lookup is performed. The result is stored in the *result* data member of the *Function* structure. Once all of the *VariableDef* and *Function* structures have been updated, the evaluation of the dynamic model is complete.

G. Running Check-Cases

In a DAVE-ML model, it is possible to define any number of static check-cases for model verification. The `staticShot` element contains values for check-case inputs and outputs, and optionally for internal variables. The *StaticShot* structure was defined to store the data for each check-case (Fig. 5). When the DAVE-ML model tree is traversed, the `staticShot` elements are parsed and stored in the same manner as the other elements. Once the entire model has been parsed and sorted, any check-cases are executed. For each defined check-case, the input values are set, the DAVE-ML model is updated, and the resulting internal values and outputs are compared with the expected values. Any discrepancies are reported via error messages to the terminal. Once all of the check-cases have been executed, the imported model has been verified, satisfying the second part of requirement 2.
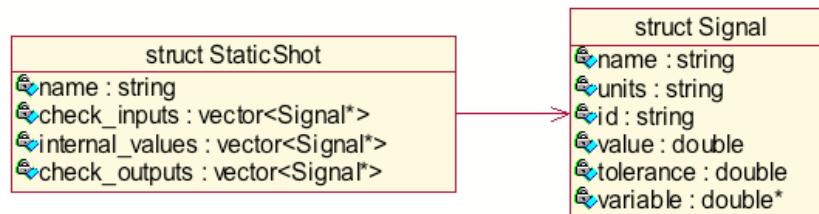

Figure 5. The *StaticShot* structure for DAVE-ML Check-Case Execution

V. Integration with LaSRS++

Models specified in the DAVE-ML grammar contain inputs, internal values, and outputs. The interface of the *DaveMLTranslator* was designed so that *VehicleSystems* using the translator only require access to the inputs and outputs for successful execution.

The *DaveMLTranslator* has a simple interface with only three public methods: *getVariablePointer*, *initialize*, and *update* (Fig. 6). The calling system uses the *getVariablePointer* method to access inputs to and outputs from the DAVE-ML model. The *initialize* method resets variable values, and the *update* method executes the dynamic model, performing table lookups and evaluating calculations to get the resulting outputs.
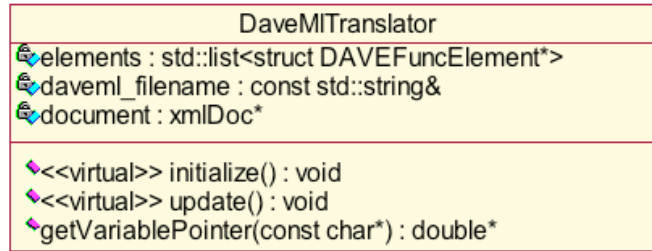
Figure 6. The public interface of the *DaveMLTranslator*

To use the translator, the programmer must inspect the DAVE-ML model, and make note of the unique identifiers (the DAVE-ML `varID` attributes) for all of the inputs and outputs; input/output (I/O) variables are generally indicated as such by the model author with a comment. In the system that will interface with the translator, in this case *F16aAeroSystem*, the user declares a variable pointer for each input and output. Calls to the *getVariablePointer* method are made once for each input and output, passing the unique identifier for each variable as an argument (Fig. 7). The *DaveMLTranslator* returns a pointer to the internal representation of the input or output. Then during each frame of execution, the *F16aAeroSystem* can set the values of the inputs and get the values of the outputs directly from the local pointers, without additional method calls. The programmer of the class interfacing with the *DaveMLTranslator* is responsible for ensuring that the unique identifiers and the units of the input variables match what is specified in the DAVE-ML model. With this simple interface, the only model-specific details are contained within the interface class (*F16aAeroSystem*), ensuring that the *DaveMLTranslator* can be used as implemented with any DAVE-ML model, satisfying requirement 5.
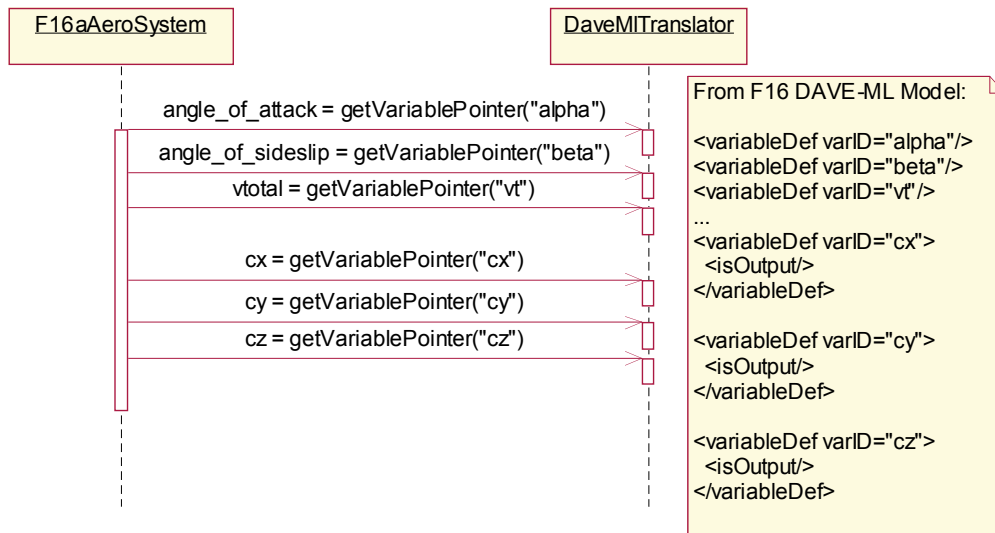

Figure 7. The *DaveMLTranslator* Interface for Accessing Model Inputs and Outputs

During each iteration of simulation execution, the *F16aAeroSystem* sets the inputs to the *DaveMLTranslator* by calling accessor methods in the *F16a* model to retrieve the current vehicle values. The *F16aAeroSystem* then calls the *update* method in the *DaveMLTranslator*, and processes its outputs (Fig. 8).
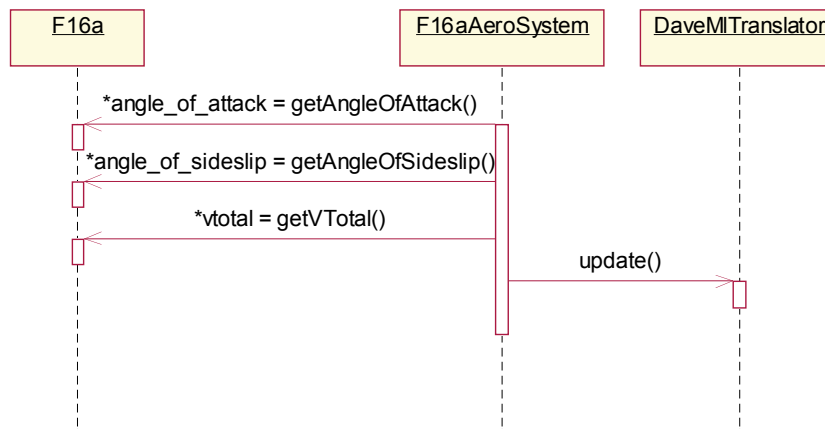
American Institute of Aeronautics and Astronautics

Figure 8. The *DaveMLTranslator* Interface during Simulation Execution

VI. Time History Comparisons

The performance of the *DaveMLTranslator* as a vehicle dynamic model was evaluated by comparing it with the existing hand-coded C++ implementations for the F-16A and the HL-20 vehicles at NASA's Langley Research Center. The vehicle simulation was executed in a non-real-time desktop mode, with a graphical user interface providing displays and a cockpit interface. A mathematical pilot was used to execute sequential doublets in the pitch, roll, and yaw axes. In both the F-16A (Fig. 9, 10) and the HL-20 (Fig. 11, 12), the aerodynamic model executed with the *DaveMLTranslator* compares favorably with the original, compiled C++ aerodynamic models.

Figure 10 shows some small (on the order of 1%) differences in the vehicle state variables; this is due to small differences in the two aerodynamic models. The LaSRS++ and DAVE-ML F-16 aerodynamic models share the same wind tunnel data but were coded independently by different parties: the LaSRS++ model was coded in C++ based on an earlier FORTRAN model; the DAVE-ML F-16 model was based on a Matlab realization of the same wind tunnel data by Garza and Morelli[5].

Figure 12 shows virtually no difference between the LaSRS++ and DAVE-ML realizations of the HL-20 aero model; these were both based on the same set of build-up equations and data tables found in reference 6. The HL-20 model is a very non-linear model based primarily on third-order polynomial fits to wind tunnel data, where the coefficients of the polynomials are from linearly interpolated tables as a function of flight condition. Thus, the DAVE-ML representation shows the capability to model fairly complex aerodynamic relationships.
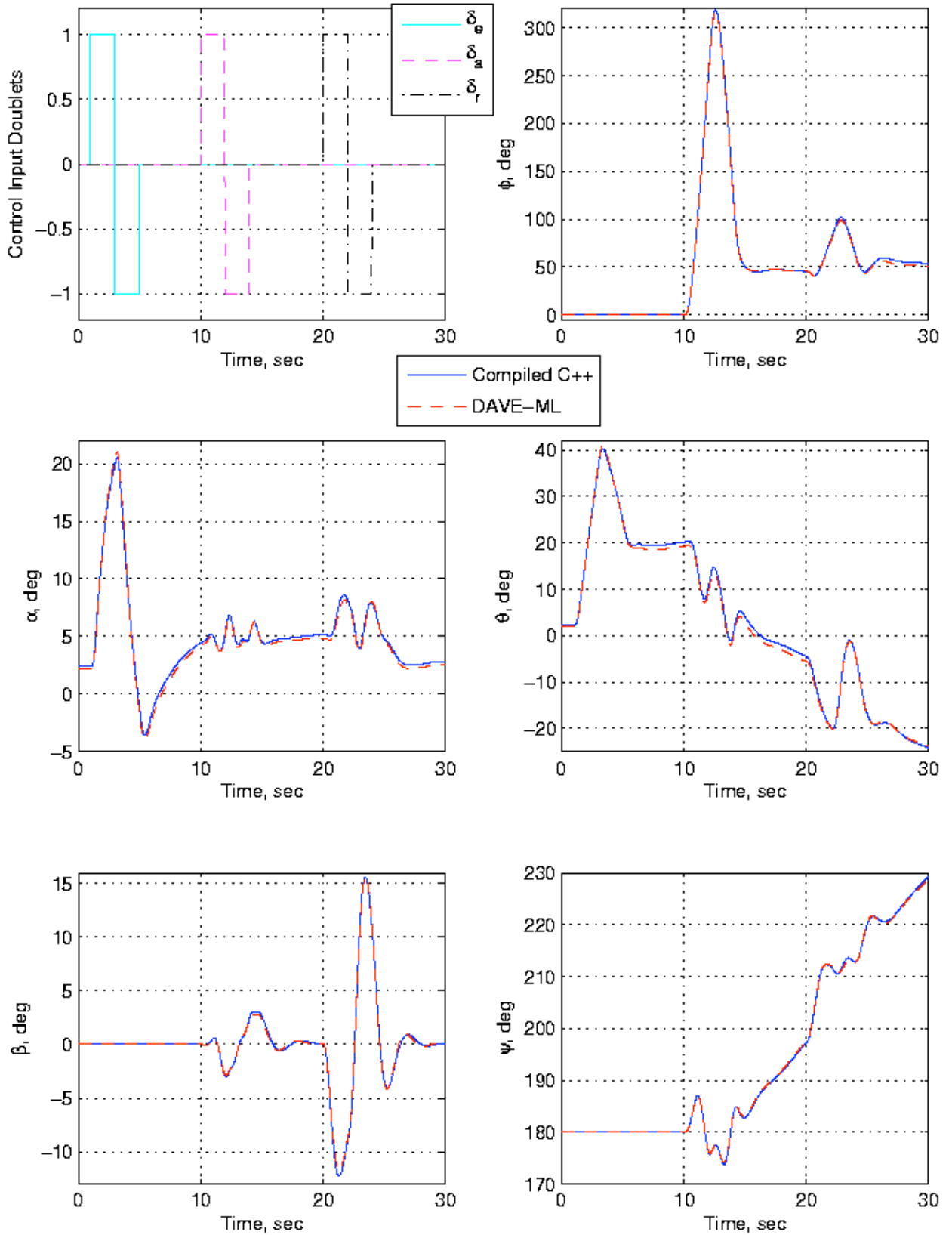
American Institute of Aeronautics and Astronautics

Figure 9. F-16A Time History Comparison with Compiled C++ and DAVE-ML Aerodynamic Models

Figure 10.   F-16A Time-History Deltas with Compiled C++ and DAVE-ML Aerodynamic Models

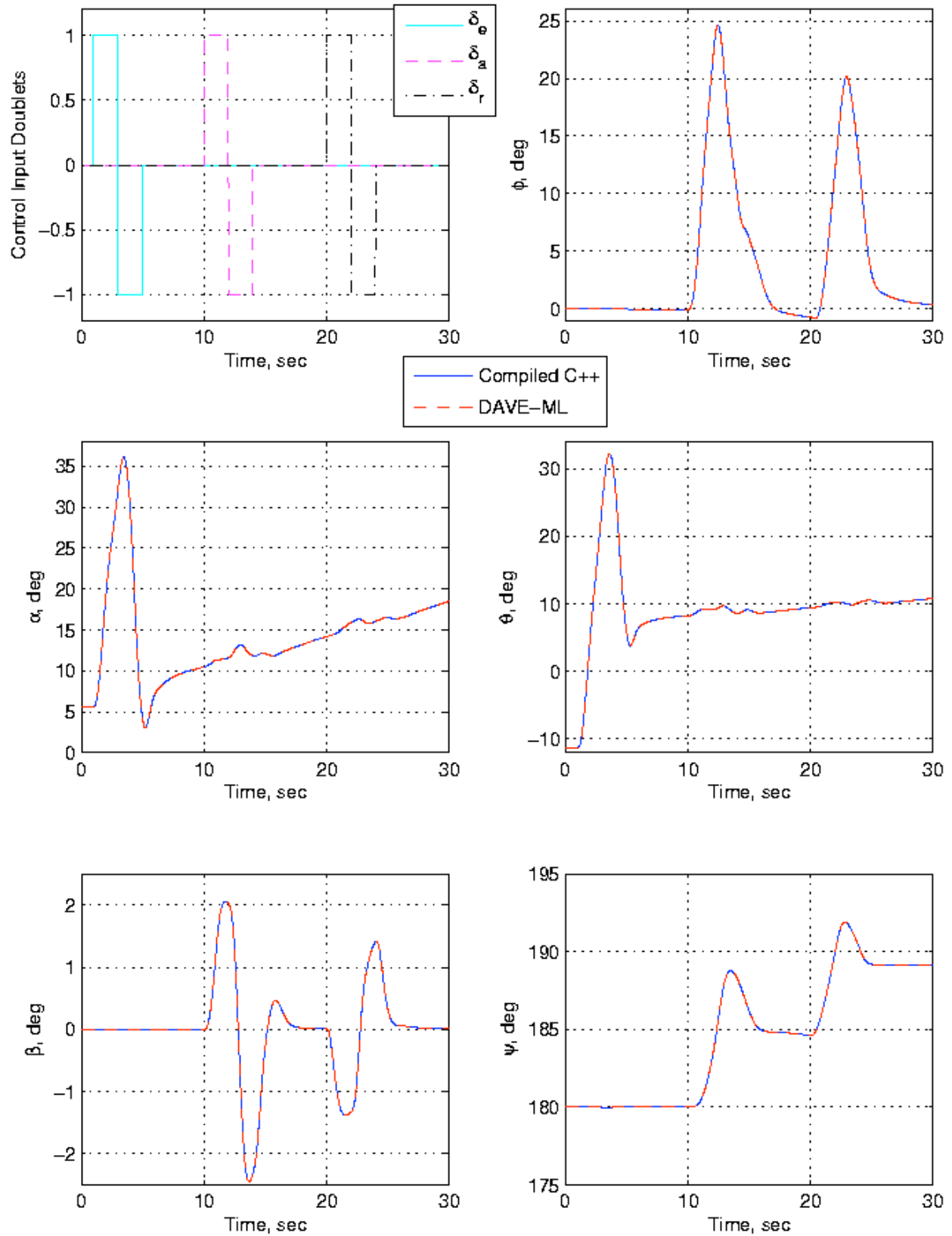American Institute of Aeronautics and Astronautics

Figure 11.   HL-20 Time History Comparison with Compiled C++ and DAVE-ML Aerodynamic Models
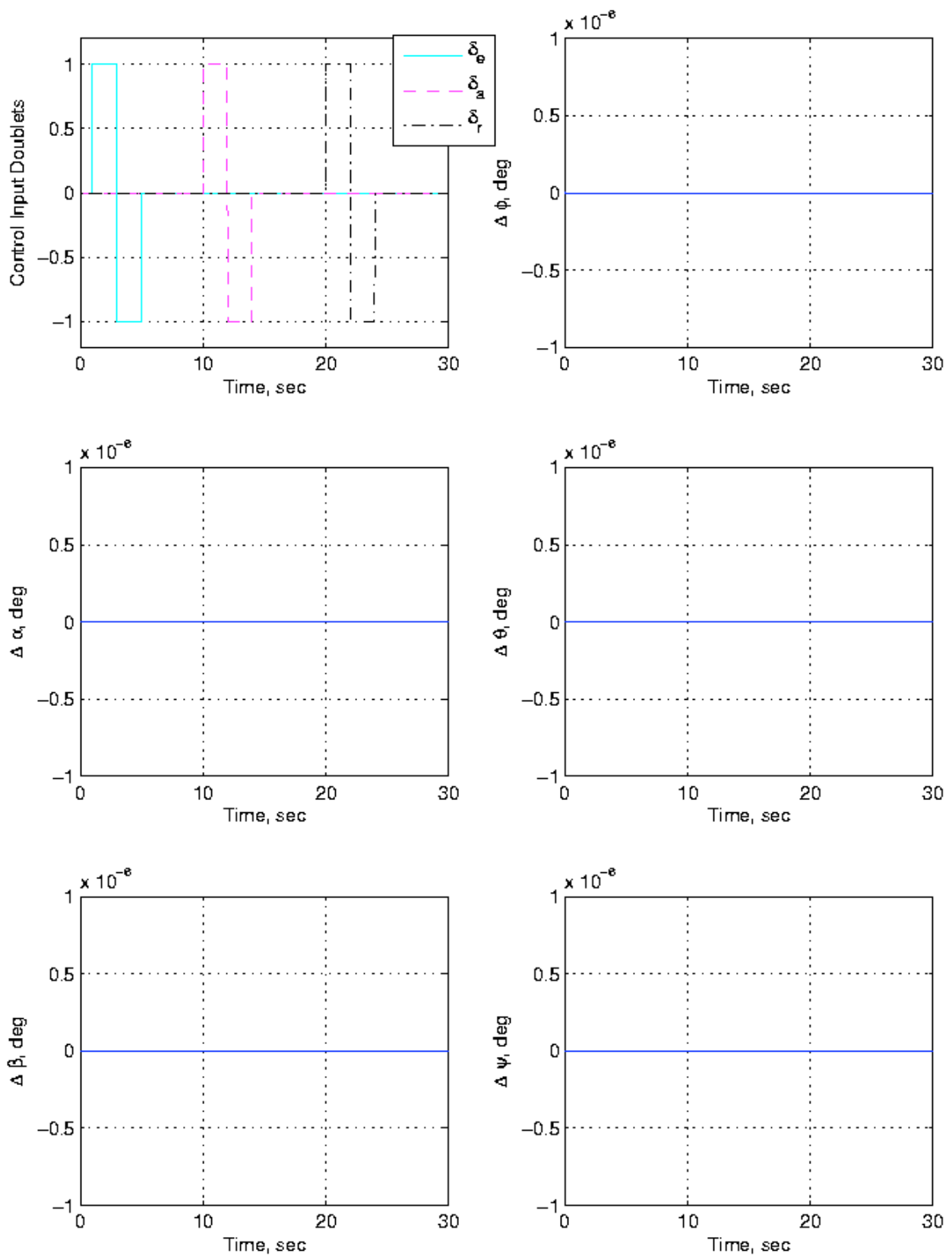
Figure 12.   HL-20 Time History Deltas with Compiled C++ and DAVE-ML Aerodynamic Models

VII.   Piloted Evaluation – F-16A and HL-20

An informal piloted evaluation of the *DaveMLTranslator* was conducted in the Generic Flight Deck (GFD) cockpit at NASA Langley Research Center. The F-16A and the HL-20 were flown using DAVE-ML aerodynamic models, loaded at run-time. The DAVE-ML models executed successfully within a real-time frame of 20 milliseconds (50 Hz), and were modified and re-loaded for subsequent runs to exercise the automatic loading capabilities of the *DaveMLTranslator.* Both vehicles performed as expected, and received favorable comments from the evaluation pilot who could not detect any qualitative difference between the compiled and interpreted models.

VIII.   Execution Speed Comparisons

Table 1 shows timing data for the F-16A and HL-20 vehicles using compiled C++ aerodynamic models and DAVE-ML aerodynamic models. The data were obtained on an SGI Origin 300 platform running the IRIX 6.5 operating system. Execution of interpreted DAVE-ML models instead of compiled C++ models increases the computational time by 5 to 75%, depending on model complexity.

Table 1.   Timing Data for Compiled C++ and DAVE-ML Aerodynamic Models

	Frame Time Used, milliseconds		
	Minimum	Mean	Maximum
F-16A Compiled C++	0.225	0.234	0.248
F-16A DAVE-ML	0.236	0.245	0.266
HL-20 Compiled C++	0.223	0.231	0.301
HL-20 DAVE-ML	0.392	0.402	0.472

IX.   Conclusion

The *DaveMLTranslator* was successfully implemented in the C++ programming language to automatically import and verify, at run-time, vehicle subsystem models specified in the DAVE-ML grammar of XML. The capability to import such models at run-time facilitates the comparison of multiple model variations, as well as reducing the time to modify the model and evaluate the effects. The translator uses a simple, generic interface, enabling its use for multiple models within a single simulation, and greatly reducing the elapsed time between delivery of a subsystem model and the integration of that model into a simulation environment. The *DaveMLTranslator* and its supporting libraries are independent of the LaSRS++ simulation framework, and may potentially be exported and integrated into any simulation framework capable of working with C++ components. As a result, the development of this capability has the potential to positively affect any future project that uses DAVE-ML to define subsystem models.

References

[1]Leslie, R., Geyer, D., Cunningham, K., Madden, M., Kenney, P., and Glaab, P., "LaSRS++: An Object-Oriented Framework for Real-Time Simulation of Aircraft," AIAA-98-4529, *AIAA Modeling and Simulation Technology Conference*, Boston, MA, 1998.

[2]Jackson, E., and Hildreth, B., "Flight Dynamic Model Exchange using XML," AIAA-2002-4482, *AIAA Modeling and Simulation Conference*, Monterey, CA, 2002.

[3]Jackson, E., Hildreth, B., York, B., and Cleveland, W., "Evaluation of a Candidate Flight Dynamics Model Simulation Standard Exchange Format," AIAA-2004-5038, *AIAA Modeling and Simulation Conference*, Providence, RI, 2004.

[4]Jackson, E., and Hildreth, B., "Progress Toward a Format Standard for Flight Dynamics Models," *SISO Simulation Interoperability Workshop*, Orlando, FL, 2006.

[5]Garza, F., and Morelli, E., "A Collection of Nonlinear Aircraft Simulations in MATLAB," NASA TM-2003-212145, 2003.

[6]Jackson, E., Cruz, C., and Ragsdale, W., "Real-Time Simulation Model of the HL-20 Lifting Body," NASA TM-107580, 1992.

[7]Cunningham, K., "Use of the Mediator Design Pattern in the LaSRS++ Framework," AIAA-1999-4336, *AIAA Modeling and Simulation Conference*, Portland, OR, 1999.

[8]Madden, M., and Neuhaus, J., "A Design for Composing and Extending Vehicle Models," AIAA-2003-5458, *AIAA Modeling and Simulation Conference*, Austin, TX, 2003.

[9]Veillard, Daniel, "The XML C parser and toolkit of Gnome," URL: http://xmlsoft.org [cited 1 June 2007].