AIAA Modeling and Simulation Technologies Conference and Exhibit
18 - 21 August 2008, Honolulu, Hawaii

AIAA 2008-6344

Modeling and Simulation of Space Vehicle Operations in the National Airspace System

Victor H. L. Cheng*, Gregory D. Sweriduk†, and Jason Kwan‡
Optimal Synthesis Inc., Los Altos, CA 94022

NASA's Future ATM Concepts Evaluation Tool (FACET) provides an extensive set of modeling, simulation and analysis capabilities for studying air transportations in the National Airspace System. The Configurable Airspace Research and Analysis Tool (CARAT) has been developed to build on the FACET capabilities to develop an environment useful for studying the interaction between space vehicle operations in the airspace and the air traffic. CARAT introduces a flexible vehicle-model database that allows the user to easily add and configure space transportation vehicle models or air transportation vehicle models for integration with the FACET simulation. CARAT also provides capabilities for safety analyses, including trajectory analysis, debris modeling, and specific functions dealing with flight hazard areas in the airspace and on the surface: the Aircraft Hazard Area and Individual-Casualty Contour Analysis, respectively. The CARAT system was originally developed on the Windows platform, but has recently been updated to work with newer versions of FACET on the latter's preferred platforms: Linux and other Unix variants, including the Apple Mac OS X. The update also provides a cleaner integration with FACET and additional enhancements in the area of vehicle modeling, including a redesigned software architecture for implementing aerospace models. Models of several aerospace vehicle that have recently garnered increasing interest have been added to the CARAT model database, including models for studying Ares launch vehicle concepts motivated by NASA's Constellation Program, and several models of unmanned aerial vehicles.

I. Introduction

Previous articles have reported on the development of a flight safety analysis tool—*Configurable Airspace Research and Analysis Tool*[1, 2] (CARAT)—for studying the safety of space launch/return operations and their relationship with the air traffic. This tool combines the air traffic simulation provided by NASA's *Future ATM Concepts Evaluation Tool* (FACET)[3] with new functionality including the flexibility to set up new launch vehicle models. It includes flight safety analysis capabilities, developed according to proposed FAA rules for launch licensing[4,5,6], to determine hazard areas associated with space launches. These capabilities are useful for assessing the interruption on the air traffic caused by space launch operations, as well as the effect of potential debris fallout on the ground population. CARAT also features 3-dimensional (3D) computer graphics for visualization of air and space vehicle operations in the National Airspace System (NAS). The previous CARAT development effort included the preparation of an initial vehicle model database containing several generic launch- and return-vehicle models that could be configured for studying many of the potential launch vehicle concepts, though these models were not meant to represent any specific concept under consideration. That initial list of vehicle models was motivated by several innovative launch vehicle concepts including advanced expendable launch vehicles (ELVs) with traditional launch characteristics to reusable launch vehicle (RLV) concepts that can take off and land on standard runways.

The FACET program has been developed for different platforms that are mostly variants of the Unix operating system, including Linux and the Apple Mac OS X. Development of the previous CARAT software included porting the FACET software to the Windows platform, resulting in the original CARAT software developed on this platform. To facilitate integration with subsequent versions of FACET on the Unix-variant platforms, a decision was made to port CARAT to these platforms, and to update its design so that it would be easier to maintain for

* Principal Scientist, 95 First Street, Suite 240, Los Altos, CA 94022, AIAA Associate Fellow.
† Research Scientist, 95 First Street, Suite 240, Los Altos, CA 94022, AIAA Member.
‡ Research Engineer, 95 First Street, Suite 240, Los Altos, CA 94022.

compatibility with future versions of FACET. This paper reports on the efforts in extending the CARAT functions for integration with a more recent FACET implementation and the incorporation of enhanced functions. Section II provides an overview of the CARAT functions.

To minimize the effort needed to integrate CARAT with new updates of the FACET software in the future, a cleaner interface between FACET and CARAT was designed and created. The previous version of CARAT was created as an extension of FACET, and CARAT code was embedded directly into many different FACET modules. For the revision of CARAT reported in this paper, a much cleaner separation between the FACET and CARAT code is achieved. Almost all of the CARAT code that was previously embedded directly within FACET modules has been moved to separate CARAT modules and encapsulated within a larger `CARAT` class. Section III discusses the software architecture for the new CARAT-FACET integration.

Additional enhancements have been introduced to the software to include the capability to model Unmanned Aerial Vehicles (UAVs) for analysis in the FACET/CARAT environment. Since NASA Ames Research Center had some of these models implemented in their Pseudo Aircraft System (PAS) software, part of the effort was to implement in CARAT an interface to these PAS models. The enhanced architecture for more flexible vehicle modeling is discussed in Section IV. Furthermore, motivated by the NASA Constellation Program for space exploration, additional models have been developed for inclusion in the CARAT model database to allow the study of Ares/Orion launches, including modules to represent the Ares I Crew Launch Vehicle[7] (CLV), the Ares V Cargo Launch Vehicle[8] (CaLV), and Orion the Crew Exploration Vehicle (CEV). Section V describes the implementation of these models as well as a few UAV models. The implementation of the Ares launch vehicle models has been based on preliminary, sketchy data available in open literature, as detailed design of these vehicles has not even taken place yet.

II. Overview of CARAT Functions

The FACET system consists of two pieces: a Java-based FACET graphical user interface (GUI) serving as the front end, and a C-based FACET core program serving as the backbone. The two pieces are represented by the two leftmost blocks in Figure 1, where the communication between them is based on Java Native Interface (JNI). The CARAT augmentation consists of Java code directly integrated with the Java component of FACET. Figure 1 illustrates the four functional components of CARAT that supplement the FACET program, including three components to support quantitative analyses—Space Vehicle Models, Special Airspace Definitions, and Flight Safety Analyses—as well as a 3D graphical capability to support qualitative visualization of the airspace and traffic interaction.

CARAT allows the flexibility for new vehicle models implemented in Java and compiled into standard class files to be deposited in an external vehicle model database that allows for grouping the models in a standardized directory structure. When a user uses the CARAT GUI indicates the intention to add a "CARAT aircraft" to the simulation, CARAT would scan the model database to dynamically update the FACET menu with all vehicle models available for selection. Once a vehicle model is selected, CARAT would retrieve the model information and dynamically construct the GUI for the user to configure the vehicle model. When the user issues the command to run the simulation, the C simulation engine in FACET would continue to maintain and update the aircraft states for the air traffic, but the simulation loop would be intercepted by CARAT to allow the "CARAT aircraft," i.e., the user-selected launch vehicle models, to maintain and update their own state information.

CARAT also allows new airspace definitions beyond traditional special use airspaces (SUAs) to support the investigation of future space-launch operational concepts with tighter spatial and temporal bounds. As space transportation vehicles do not deliver the same level of reliability as

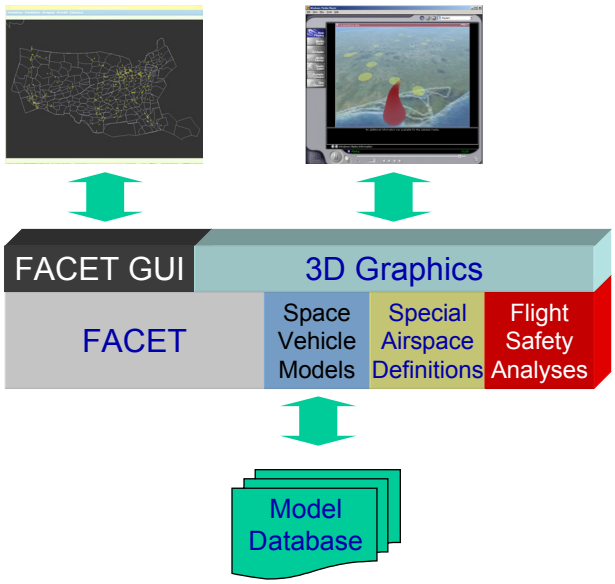

Figure 1. Functional Components of Configurable Airspace Research and Analysis Tool (CARAT).

commercial aircraft, it is necessary to define hazardous regions in the airspace to account for potential debris from space transportation operations. The extent of the potential debris needs to account for both operational failures and planned stage ejections. The FAA rules for licensing a launch site[4] and individual launches[5,6] contain debris-modeling requirements with various levels of details. CARAT includes functions compatible with these requirements for predicting the debris dispersion, which can be used in the definition of hazard volumes as functions of time. Ref. 2 introduced the notion of Dynamic Hazard Volume (DHV) to represent all the potential debris dispersions at any time t from all possible breakups since the launch time t_L, i.e., DHV = $\{D(\tau,t) | t_L \le \tau \le t\}$, where $D(t_0,t)$ denotes the hazard debris uncertainty field at time t caused by the vehicle breaking up at t_0.

Flight safety analyses of space vehicle operations in the NAS constitute an important objective of CARAT. Most of these analyses concern safety of the air traffic in the NAS, extending down to the population on the surface. The relevant flight safety analyses address the nominal flight profiles as well as potential debris resulting from failures or planned events. CARAT provides these analysis functions in the form of trajectory analysis, debris modeling, and functions dealing with flight hazard areas in the airspace and on the surface: the Aircraft Hazard Area and Individual-Casualty Contour Analysis, respectively. Background material for the definition of these hazard areas can be found in Refs. 5 and 6,

Lastly, CARAT provides 3D computer graphics functionality to support its functions through the popular OpenGL standard. To interface the Java code to an OpenGL library, which usually provides interface functions compatible with standard C calls, "OpenGL for Java" was used in the previous version[1] of CARAT to provide the bindings between the Java and C standards. The 3D graphics capability is used to visualize the vehicles, terrain, separation requirements, potential debris dispersions, and flight hazard areas. Figure 2 shows an example of a 3D graphical display of the space shuttle, while Figure 3 illustrates the notion of potential debris fallout for debris classes with different ballistic coefficients.

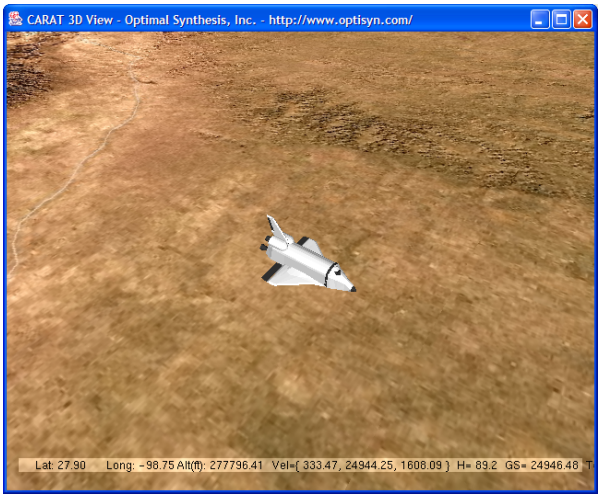

Figure 2. 3D Graphical Display of Space Shuttle

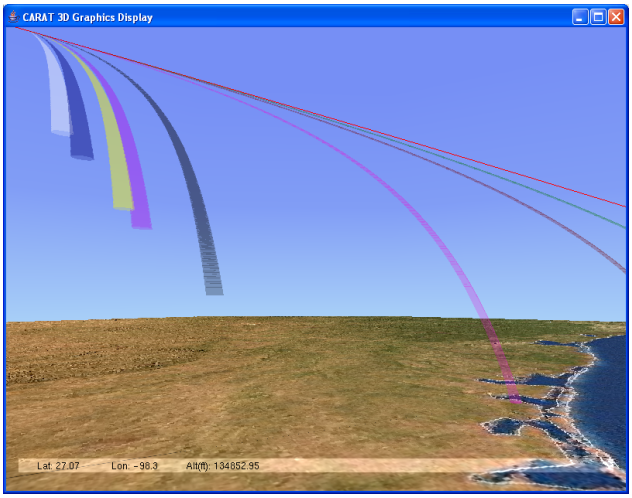

Figure 3. Graphical Display of Debris Dispersions of Different Debris Classes

III. Enhanced Design for FACET-CARAT Integration

To minimize the effort needed to integrate CARAT with new updates of the FACET software in the future, a cleaner interface between FACET and CARAT was designed and created. The original version of CARAT reported in Ref. 1 was created as an extension of FACET, and CARAT code was embedded directly into many different FACET modules in order to facilitate implementation. A much cleaner separation between the FACET and CARAT code is maintained for the revision of CARAT. Almost all of the CARAT code that was previously embedded directly within FACET modules has been moved to separate CARAT modules and encapsulated within a larger `CARAT` class. Figure 4 summarizes the changes in the object-oriented implementation of the FACET-CARAT interface.

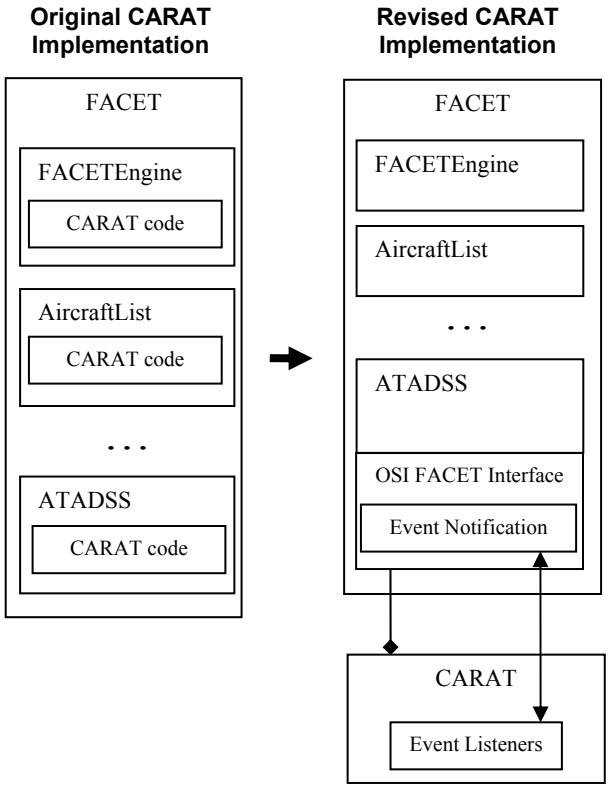

Figure 4. New Design of CARAT Integration with FACET.

In the previous CARAT implementation, access modifiers for many of the FACET package classes and methods were changed to allow CARAT access. FACET classes such as `FACETEngine` and `AircraftList` contained various CARAT classes as member variables, requiring us to write CARAT-specific code in order to achieve the same functionality. A new event-notification system allows us to be much less intrusive within the FACET code. FACET code modifications are limited to fulfilling the interface requirements, and can be become integrated into future distributions of the FACET software.

The main FACET class `ATADSS` now implements the `OSIFACETInterface` interface, and CARAT-specific code has been removed from all FACET modules. Previously, an instance of CARAT was instantiated from within the `ATADSS` class, which contained CARAT-specific code for registering certain events and creating the custom CARAT menu. Now, any Java code module that wishes to access or extend upon FACET capabilities can instantiate and initialize FACET through the help of a FACET-loader class. In our case, the CARAT application instantiates and "contains" an instance of FACET. Interaction between the CARAT and FACET modules is now completely limited to the `OSIFACETInterface` interface and the accompanying event notification system, ultimately making the integration of future versions of FACET and CARAT much easier. A UML diagram that illustrates the new CARAT-FACET architecture is shown in Figure 5, where it shows clearly the object classes of the CARAT software on the left-hand side and those of FACET on the right, with a simple connection in the center showing the `ATADSS` implementing the `OSIFACETInterface`.

In addition to a new software design, porting CARAT from the Windows platform to the Linux and Mac OS X platforms has revealed other problems due to incompatibilities between the previous software and more advanced software and hardware technologies. One such incompatibility lies in the implementation of the 3D graphical software. The 3D graphics in the original CARAT prototype were created using OpenGL in Java through the "OpenGL for Java" library (GL4Java), which mapped the OpenGL application program interface (API) to a set of Java packages using the Java Native Interface (JNI). Although GL4Java is available for the Windows, Linux, and Mac OS X platforms, the library is no longer being maintained, and the resulting software has stability issues with newer graphics hardware. To enhance compatibility and robustness of the CARAT software, a decision was made to migrate the CARAT software from GL4Java to Java bindings for Open GL (JOGL). JOGL is an "Open Source"

product with precompiled libraries available for the Solaris 8, Red Hat Linux 7.3, Mac OS X 10.3, and Windows 2000 and XP platforms.

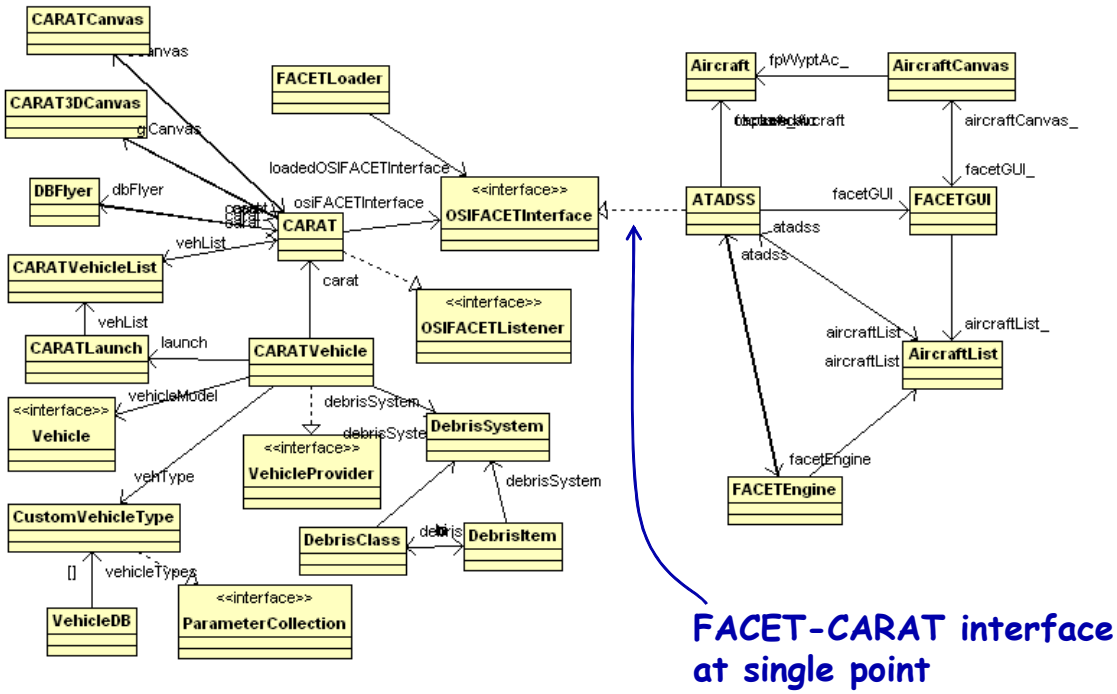

Figure 5. UML Diagram of Revised CARAT-FACET Integration.

IV. Enhanced Vehicle Modeling Architecture

This section describes the implementation of aerospace vehicle models in CARAT, including descriptions of the model code database, 3D graphical model, pre-built model classes, and treatment of staging where the number of vehicle bodies changes over the course of a mission.

The framework for implementing simulation vehicle models has been completely redesigned. The main goal is to simplify the process by which a user implements and inserts a vehicle model into a FACET/CARAT simulation. We also seek to create a much cleaner and more extensible internal implementation.

For the original version of CARAT, the implementation of a CARAT vehicle model required the user to extend upon a very large and complex `CARATAircraft` class. The `CARATAircraft` class was structured to contain default implementations for five basic model subcomponents: Equations of Motion, Mass Properties, Controls, Propulsion, and Aerodynamics. This is appropriate for vehicles with detailed models, but would cause low-fidelity aircraft models (e.g., those that compute their trajectories based on waypoint interpolation) to contain a great deal of unnecessary dynamics code. To implement even a simple model for insertion, a user would have to create subclasses of each of the subcomponents to override the default functionality. The `CARATAircraft` class also contained a great deal of CARAT system data which could potentially be overwritten by an author who is unfamiliar with the complexities of the class extended upon.

Under the new implementation, a new `Vehicle` Java interface defines a small set of methods required by all user models. Any Java class which implements that interface can be inserted into the simulation, and models need not extend upon any CARAT class. A model is only responsible for returning very basic state data such as position and velocity. As shown in Figure 6, the rest of the CARAT data has been moved up to a container `CARATVehicle` class, the contents of which are protected and not exposed to the user model. To facilitate the addition of more complex models, a user will also be able to extend upon abstract classes that implement the `Vehicle` interface. The `StandardVehicleModel` class contains dynamics for implementing a launch or return vehicle (with Equations of Motion, Controls, Aerodynamics, etc.), and the `GenericAircraft` class is a standard aircraft model for following waypoints. The design of this `GenericAircraft` class is motivated by the need to implement models that are compatible with their implementation in the Pseudo-Aircraft System (PAS), which has been used by NASA Ames Research Center to study operations in the NAS, including operations of unmanned aerial vehicles (UAVs).

NASA Ames has provided several of these PAS-based UAV models for inclusion in the CARAT model database. In general, if other forms of vehicle model implementation are desired, they would just need to be created to implement the `Vehicle` interface, similar to the way the `GenericAircraft` class is prepared.

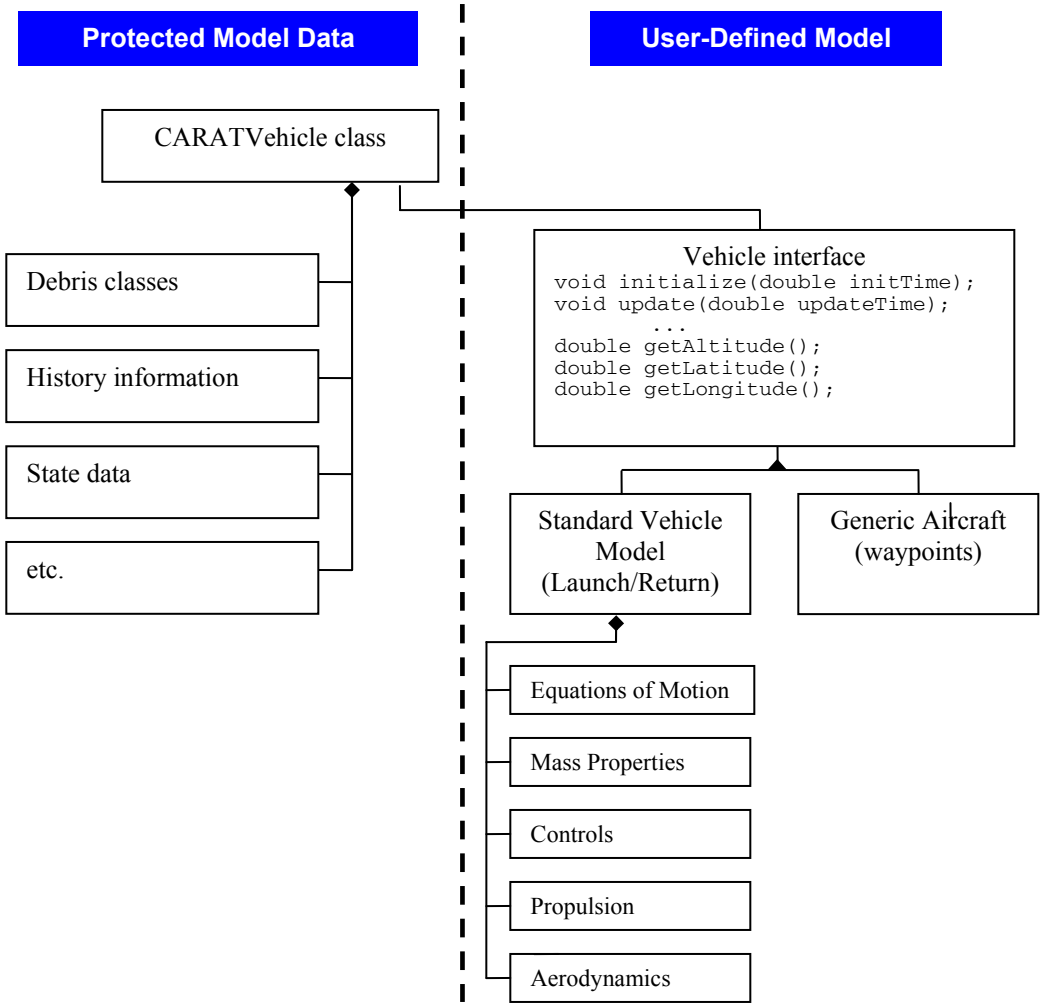

Figure 6. Architecture of Vehicle Model Implementation.

A. Vehicle Model Code Database Structure

To insert a custom vehicle model into a simulation, the vehicle must have its own directory in the model database as illustrated by Figure 7. The directory serves as a general-purpose repository for code and data files that a custom vehicle model is dependent upon. As shown in the figure, the current CARAT implementation has the vehicle models stored under the directory `vehicleDB`, which is located in a fixed location within the main FACET/CARAT directory structure.

At a minimum, each vehicle's directory should contain the following three files, where the wild-card symbol * represents the name given by the user for the vehicle model:

- *_info.txt — This file provides general information about the custom vehicle. It contains 3 lines of text that provide:
  - The name of the main Java class that represents the custom vehicle.
  - A name for the custom vehicle to be displayed in GUI menu items.
  - A short description of the vehicle.
- *.class — This file contains the byte code corresponding to the Java class named in the aforementioned *_info.txt file. This class must implement the `Vehicle` interface in order to be added to a simulation.

- *_param.txt — This file specifies the names, types and default values of the runtime parameters that the vehicle's model code is expecting.

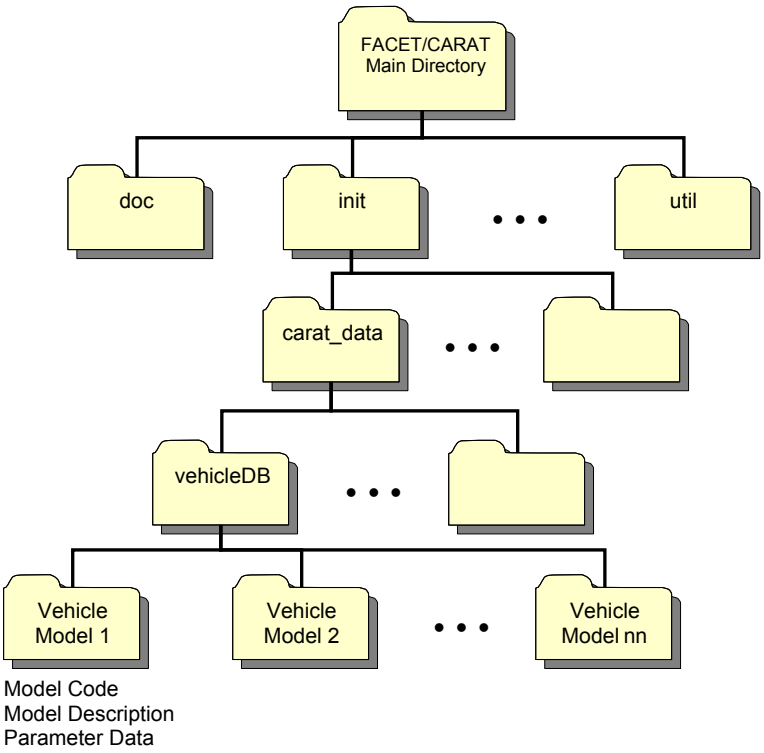

Figure 7. Vehicle Database File Structure.

As mentioned previously, the core module of the model implementation must implement the `Vehicle` Java interface. The name of this main model class must be specified in the *_info.txt file mentioned above. CARAT uses a custom class loader which looks for the specified class and support classes within the vehicle model's directory. Because the non-static directories within the model database are not a part of the classpath environment variable, the standard Java class loader will not find these files. If the model is dependent upon other supporting classes, such class files must also be present in the model's directory, and the class loader will automatically find them. It is possible to use the same concept to write a class loader to obtain model code from a database remotely over a network. In addition to these three required files, a vehicle model can also include a Debris Information File if the analyses of debris fallout are desired.

B. Implementation of New Vehicle Models

CARAT provides the `Vehicle` interface for the user to implement custom vehicle models. After a model's class files are loaded, an instance of the vehicle model is instantiated through the default constructor.

1. Run-Time Components

The CARAT vehicle model implements the `Vechicle` and `ParameterCollection` interfaces. The `Vehicle` interface provides methods for accessing the vehicle's state variables such as position, heading, and velocity. The `ParameterCollection` interface provides methods for obtaining the user-specified parameters for the model. Upon insertion into the simulation, the model loads the user-provided parameter values and performs error checks on those parameters prior to initialization. As shown in Figure 8, the model returns a `ParameterResponse` object which notifies the user if any of the model's expected parameters are not present or if the parameter values are found to be unacceptable. The `VehicleProvider` interface contains methods which allow the model to signal its landing and to register staging event messages for notifying the system when a new stage is to be added or an old one removed within the current launch operation. The `ModelRendererProvider` interface declares the methods for rendering the default vehicle models to the OpenGL view. The

`ModelRenderer` interface declares the methods for specifying custom user-defined rendering code for drawing a vehicle in the OpenGL view. The `EnvRendererProvider` interface declares the methods for rendering the 3D terrain.

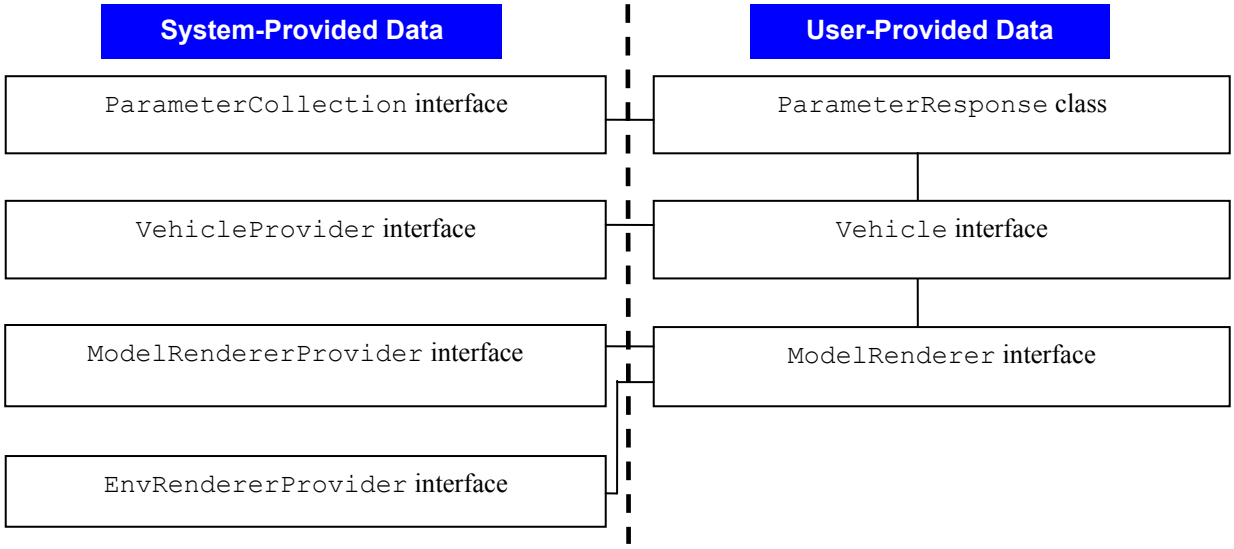

Figure 8. Vehicle Implementation Requirements.

2. Graphical Model Rendering

The `Vehicle` interface also provides a method that returns an object implementing the `ModelRenderer` Java interface. The returned object is often the model class itself. If implemented, the `ModelRenderer` interface allows the model to customize the rendering of the vehicle in OpenGL through JOGL routines, as shown in Figure 9 for the rendering of a couple of vehicles modeled after the Kistler concept.

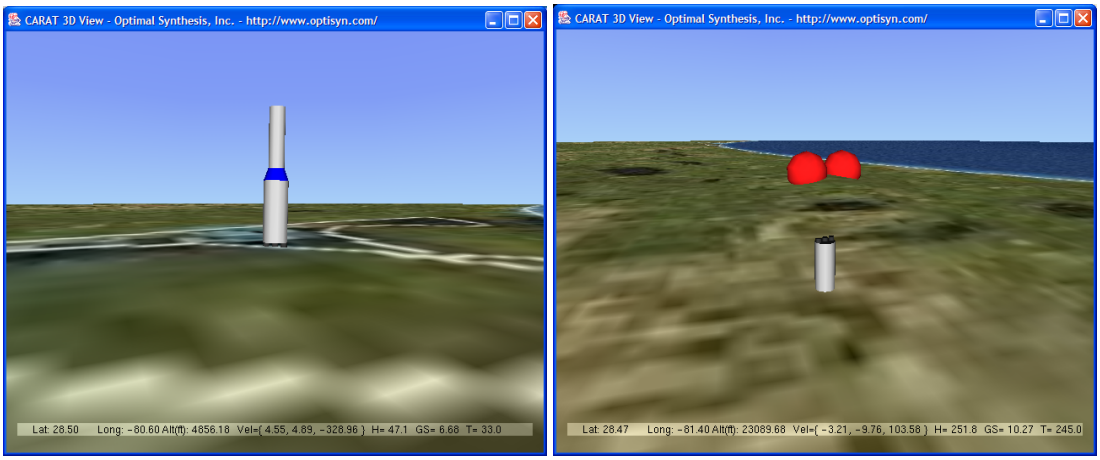

Figure 9. Sample 3D Rendering of Launch and Return Vehicles Motivated by Kistler K-1 Concept.

3. Implementing Vehicles Derived from Pre-Built Classes

If a user does not wish to generate a model from scratch by implementing the `Vehicle` interface, they may derive vehicle models from the provided `StandardVehicleModel` and `GenericAircraft` abstract classes shown in Figure 6. Both abstract classes extend upon another abstract class, `VehicleModel`, as detailed in Figure 10. The `VehicleModel` class maintains a collection of objects which implement the `ModelComponent` interface.

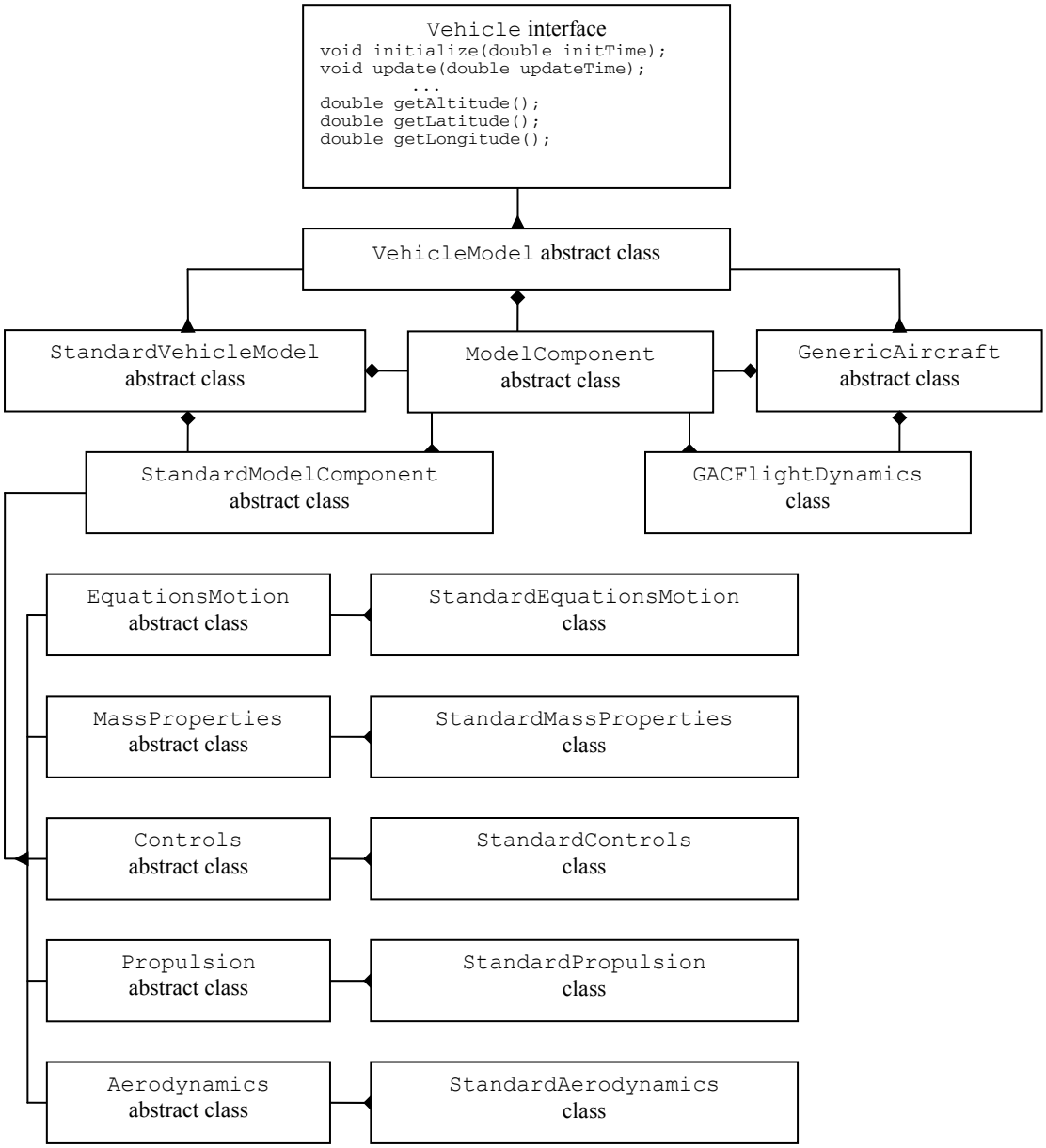

Figure 10. Details of the `VehicleModel` Subclasses `StandardVehicleModel` and `GenericAircraft`.

The abstract `VehicleModel` class by itself is not useful, because the individual model components and their connections are not well defined. Extending upon the `VehicleModel` class, the `GenericAircraft` abstract class can be used to generate aircraft models which follow an input list of waypoints according to a file defining the aircraft's performance characteristics. Both files are provided to the model through standard model parameters. A typical model extending upon the `GenericAircraft` class might simply provide code for rendering the aircraft on the 3D display.

Also extending upon the `VehicleModel` class, the abstract `StandardVehicleModel` class defines the individual model components and their outputs to match an extensible set of flight dynamics and control. A different set of flight dynamics and model components could easily be defined by extending the `VehicleModel` class in a similar manner. The `StandardVehicleModel` contains the abstract `StandardModelComponent` class, which extends upon the `ModelComponent` interface. Five ordered model components for Equations of Motion, Mass Properties, Controls, Propulsion, and Aerodynamics extend upon the `StandardModelComponent` abstract

class to define Java interfaces bearing the same name. These component interfaces are implemented by the classes `StandardEquationsMotion`, `StandardMassProperties`, `StandardControls`, `StandardPropulsion` and `StandardAerodynamics`.

4. Treatment of Staging and Launches

Most space launch operations involve staging. When vehicles are inserted into the simulation, they are created as part of a "launch." A launch is used to group a vehicle model together with child vehicle models which are spawned through the occurrence of staging events. During calls to a simulation, the vehicle model may create staging events which instruct the simulation to insert or remove vehicle models within the launch in order to simulate the occurrence of staging. The different types of staging events are outlined in Figure 11. There are two basic types of staging events which are created through the `StagingEventAdd` and `StagingEventRemove` classes. The third type, `StagingEventMulti`, is merely a collection of add and remove staging event objects which allow the system to handle multiple staging events for a launch as a batch.

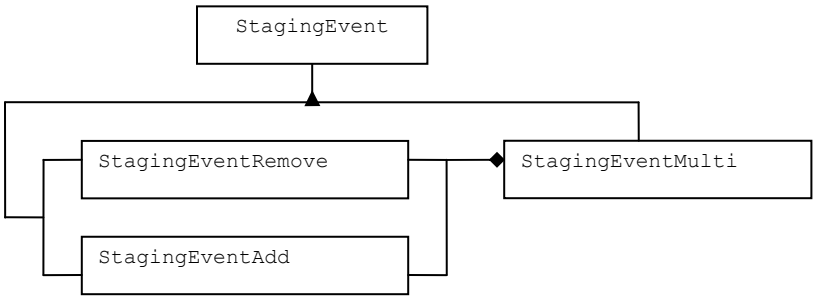

Figure 11. Staging Event Types.

A `StagingEventAdd` object is instantiated by passing the name of the vehicle model to be inserted into the launch. When the system handles the insertion staging event, it searches the vehicle model database for the model with the given name and loads the model files from the appropriate directory. The `StagingEventAdd` object can also be supplied with parameter name and value pairs which are passed onto the vehicle during the initialization of the model. These parameters are typically used to allow the new stage to inherit the current position and velocity of the parent vehicle.

A `StagingEventRemove` object is instantiated by passing the name of the vehicle to be removed within the launch. The `StagingEventRemove` event should be used to remove models which have been previously added to the launch through a `StagingEventAdd` event. If one wishes to replace the parent model with a new model for staging, a `StagingEventMulti` event should be employed.

A `StagingEventMulti` object should be used to simultaneously insert multiple additional vehicle models or to replace the current model (the model registering the staging event) with one or more new models. A parameter passed to the constructor of the `StagingEventMulti` object signals whether the current model should be removed during the handling of the staging event. The appropriate `StagingEventAdd` and `StagingEventRemove` objects are then created and added to the `StagingEventMulti` object.

V. Implementation of New Vehicle Models

Along with enhancement of the CARAT system, two sets of new vehicle models have been added to CARAT's model database. These include space launch vehicles motivated by NASA's Constellation Program, and several UAV models available from the PAS collection at NASA Ames Research Center.

A. Ares Launch Vehicle Models

New launch vehicle models have been created for the Ares I Crew Launch Vehicle (CLV) and for the Ares V Cargo Launch Vehicle (CaLV). Figure 12 contains an illustration of these vehicles as envisioned by NASA. Since these vehicles are still in the early state of being designed, realistic model data are not available. OSI has developed models for them based on early information available in the open literature. The CLV and CaLV vehicle classes each extends upon the `StandardVehicle` class so they implement extensions of the `StandardPropulsion`, `StandardAerodynamics`, `StandardControls`, and `StandardEquationsMotion` classes. The propulsion class is used to set the thrust and specific impulse of the launch vehicle. The aerodynamics class is used to interpolate the lift and drag coefficients based on altitude, Mach number, and angle of attack. These coefficients have been generated using the U.S. Air Force Digital Data Compendium (Datcom) program[9]. The controls class is used to control the throttle; it is used to specify when the thrusters are to be turned off. Finally, the equations of motion class are used to integrate the state variables (velocity, angle of attack) at each time step.

Figure 12. Illustration of Ares I (Right) and Ares V (Left) Launch Vehicles.

Using information on the vehicle concept available from open literature, OSI has put together rough simulation models for the complete launch vehicle, the separated first stage, and the separated upper stage. Additionally, a model of the CEV re-entry vehicle was created to allow simulation of re-entry scenarios. Models of the Orion CEV Earth departure stage and the separated upper stage are not modeled for the FACET/CARAT environment since this separation event occurs near the limit of the atmosphere and the vehicle burns up upon reentry. Separation of the first and upper stage is modeled after a specified burn time. With vehicle parameters estimated for the Ares I mass properties, propulsion, and aerodynamic models based on publicly available information, lift and drag coefficients for the Ares I airframe are generated as functions of altitude, Mach number, and angle of attack. Figure 13 contains the example of the lift coefficients plotted against Mach number with altitude as a parameter for the case of 5º angle of attack. Similar lift and drag coefficient models were created for the first stage, upper stage, and CEV return vehicles.

An aerodynamic model was similarly developed for the Ares V as a two-stage, vertically stacked rocket. The first stage consists of a liquid fueled core powered by 5 RS-68 engines. The first stage is assisted by two shuttle-derived reusable solid rocket boosters (RSRBs) to provide the launch thrust. The graphical models for the Ares I and Ares V were created using standard OpenGL functions. Figure 14 shows an example with the Ares V model. Similar models have been generated for the Ares I system as well as all the stages of the two launch vehicle systems. Graphics to show potential debris dispersion follow the conventions described in Section II.

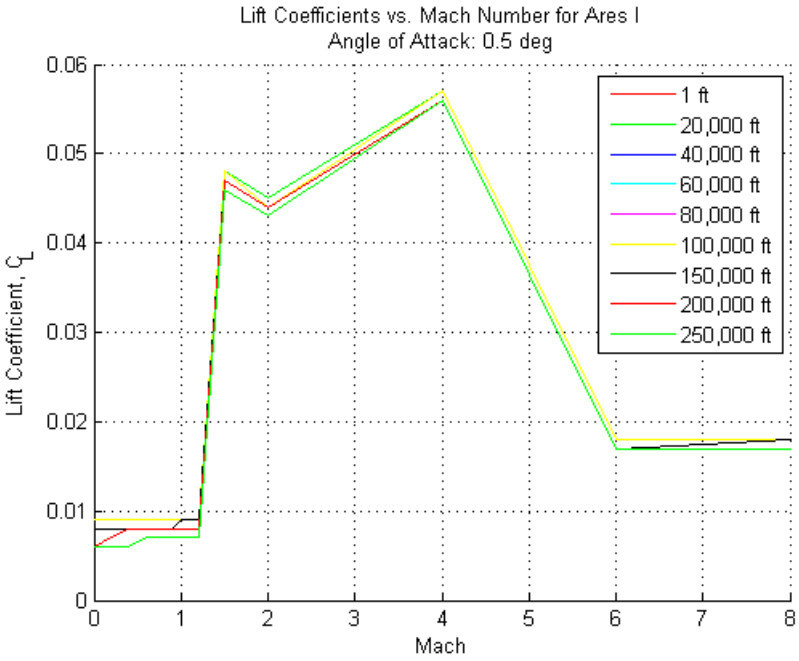

Figure 13. Ares I Lift Coefficients for 0.5º Angle of Attack.

Figure 14. 3D Graphical Model of Ares V Launch Vehicle.

B. Unmanned Aerial Vehicle Models

UAV models from the PAS library have been added to CARAT by creating a `PASAircraft` class as an extension of the `GenericAircraft` class discussed above in Figure 10. The UAV models extend the `PASAircraft` class with the subclasses shown in Figure 15. Models are created for the Helios, Global Hawk, Altair Predator, and Perseus B UAVs. Performance data for the UAV models are obtained by reading data files

provided by the PAS database. These include model reports files, ascent and descent ratio files, and engines files. Graphical 3D models for the Perseus B, Helios, Global Hawk, and the Altair Predator were created to provide visualization of the aircraft. Figure 16 shows a screenshot of the 3D UAV model of the Global Hawk in flight.

VI. Conclusion

The Future ATM Concepts Evaluation Tool (FACET) developed at NASA Ames Research Center provides an extensive set of modeling, simulation and analysis capabilities for studying air transportations in the National Airspace System. The Configurable Airspace Research and Analysis Tool (CARAT) was originally developed to build on the FACET capabilities to develop an environment useful for studying the interaction between space vehicle operations within the airspace and the air traffic. CARAT introduces a flexible vehicle-model database that allows the user to easily add and configure space transportation vehicle models or air transportation vehicle models for integration with the FACET simulation. The database enables dynamic reconfiguration of FACET's Java-based graphical user interface to reflect user addition or modification of the models. Vehicle models, their characteristics and 3-dimensional (3D) graphical visualization models can all be dynamically added to the model database without further need to modify the FACET program. CARAT also provides capabilities for safety analyses, including trajectory analysis, debris modeling, and specific functions dealing with flight hazard areas in the airspace and on the surface: the Aircraft Hazard Area and Individual-Casualty Contour Analysis, respectively.

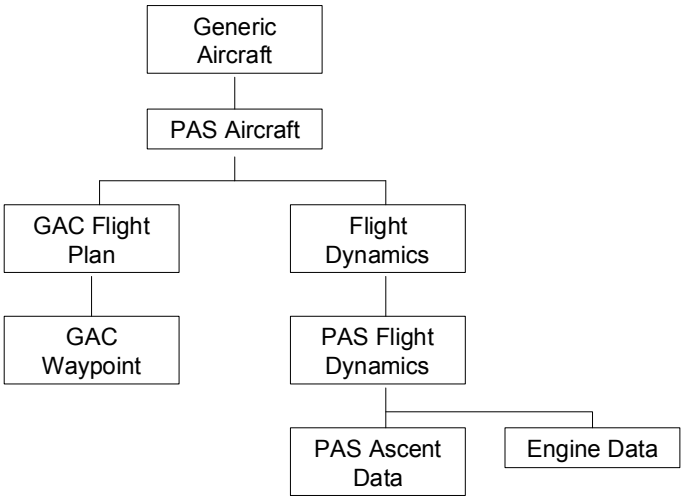

Figure 16. PAS Aircraft Classes.

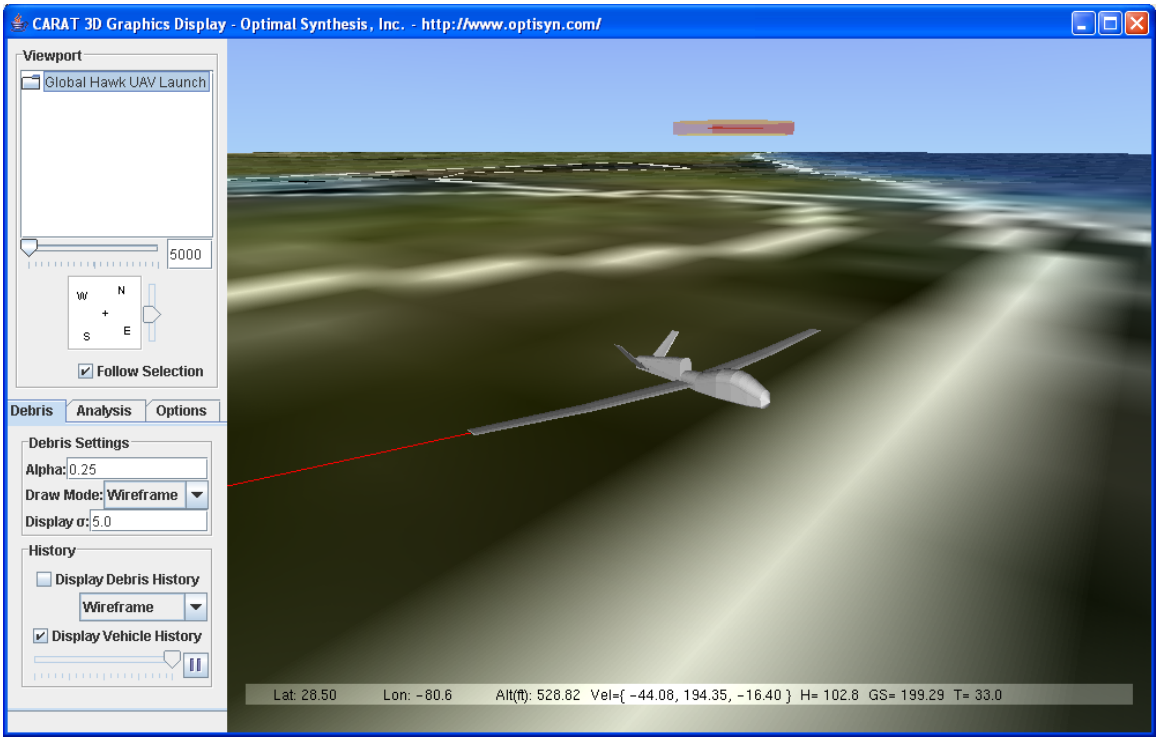

Figure 15. 3D Graphical Model of the Global Hawk UAV.

The CARAT system was originally developed on the Windows platform, but has been updated to work with newer versions of FACET on its preferred platforms: Linux and other Unix variants, including the Apple Mac OS X. The update also provides a cleaner integration with FACET and additional enhancements in the area of vehicle modeling, including redesigned software architecture for implementing aerospace models.

New models of aerospace vehicles that have garnered interest recently have been developed and added to the CARAT environment. These include the Ares I and Ares V launch vehicles identified by NASA's Constellation Program, and several Unmanned Aerial Vehicles (UAVs). Beyond launch vehicle designs being conceived for NASA's space exploration program, there appears to be an increase in interest in developing space launch vehicles for space tourism. After SpaceShipOne won the X-Prize on October 4, 2004, there has been increasing interest in building commercial sub-orbital spaceships and launch aircraft. Development of the new SpaceShipTwo (SS2) and White Knight Two (WK2) launch systems by the X-Prize winning team is an example, along with other ventures formed to compete in the space tourism market. Blue Origin is another example with a launch vehicle concept involving a vertical takeoff and landing spacecraft, similar to the previous DC-X concept. These launch vehicle concepts can all benefit from the FACET/CARAT technologies to help with planning their launch and return concepts, including simulation of failure conditions and other analyses required for FAA certification of commercial space launches.

Acknowledgments

Development of CARAT and its integration with FACET has been supported by NASA under Contract No. NNA05BF54C. The authors thank Drs. Shon R. Grabbe and Banavar Sridhar of NASA Ames Research Center for their support and contribution to the development of CARAT.

References

[1]Cheng, V. H. L., Menon, P. K., Sridhar, B., and Draper, C. H., "Computer Simulation and Analysis Tool for Air and Space Traffic Interaction Research," *Proceedings of the 21st IEEE/AIAA Digital Avionics Systems Conference*, Irvine, CA, October 29–31, 2002.

[2]Cheng, V. H. L., Diaz, G. M., and Sridhar, B., "Flight Safety Analysis Tool for Space Vehicle Operations in the National Airspace," *Proceedings of the 2003 AIAA Aircraft Technology, Integration, and Operations (ATIO) Technical Forum*, Denver, CO, November 17–19, 2003, AIAA Paper 2003-6793.

[3]Bilimoria, K., Sridhar, B., Chatterji, G. B., Sheth, K., and Grabbe, S., "FACET: Future ATM Concepts Evaluation Tool," *Air Traffic Control Quarterly*, Vol. 9, No. 1, 2001, pp. 1–20.

[4]Department of Transportation – Federal Aviation Administration "14 CFR Parts 401,417, and 420: Licensing and Safety Requirements for Operations of a Launch Site; Rule," *Federal Register*, pp. 62812–62898, October 19, 2000.

[5]Department of Transportation – Federal Aviation Administration "14 CFR Parts 413, 415, and 417: Licensing and Safety Requirements for Launch; Notice for Proposed Rulemaking; Proposed Rule," *Federal Register*, pp. 63922–64123, October 25, 2000.

[6]Department of Transportation – Federal Aviation Administration "14 CFR Parts 413, 415, and 417: Licensing and Safety Requirements for Launch; Proposed Rule," *Federal Register*, pp. 49456–49521, July 30, 2002.

[7]"Constellation Program: America's Fleet of Next-Generation Launch Vehicles, The Ares I Crew Launch Vehicle," *NASA Facts*, Pub 8-40598, NASA George C. Marshall Space Flight Center, Huntsville, AL, July 2006.

[8]"Constellation Program: America's Fleet of Next-Generation Launch Vehicles, The Ares V Cargo Launch Vehicle," *NASA Facts*, Pub 8-40599, NASA George C. Marshall Space Flight Center, Huntsville, AL, July 2006.

[9]*Missile DATCOM — 1997 Fortran 90 Revision*, US Air Force Research Laboratory, Air Vehicle Directorate, Wright-Patterson Air Force Base, February 1998.