Simulation Environment for Orion Launch Abort System Control Design Studies

J. Dana McMinn*
NASA Langley Research Center, Hampton, VA 23681

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

David M. Christhilf ‡
Lockheed-Martin Engineering Services Company, Hampton, VA 23681

Abstract

The development and use of an interactive environment to perform control system design and analysis of the proposed Crew Exploration Vehicle Launch Abort System is described. The environment, built using a commercial dynamic systems design package, includes use of an open-source configuration control software tool and a collaborative wiki to coordinate between the simulation developers, control law developers and users. A method for switching between multiple candidate control laws and vehicle configurations is described. Aerodynamic models, especially in a development program, change rapidly, so a means for automating the implementation of new aerodynamic models is described.

Nomenclature

ACM	=	Attitude Control Motor
API	=	Application Programming Interface
CEV	=	Crew Exploration Vehicle (Orion)
CM	=	Crew Module
G&C	=	Guidance and Control
LAS	=	Launch Abort System (abort motor, tower, control motors, etc.)
LAV	=	Launch Abort Vehicle (LAS with CM)
LES	=	Launch Escape System (Apollo)
SAREC	=	Simulation Architecture for Evaluating Controls
SVN	=	Subversion (version control software)

I. Introduction

THE Orion Crew Module (CM), as shown in Fig. 1, is a conical frustum-shaped capsule similar to the Project Apollo Command Module but approximately 28% larger to accommodate as many as six crew members.[1] The vehicle is under development by the Lockheed Martin Corporation as a component of the Manned Exploration Initiative known as Project Constellation.

For crew safety during launch, the CEV is required to have an abort capability throughout the launch ascent whereby the crew can be transported away from a malfunctioning booster in a trajectory that is amenable to parachute landing (typically on water). The Apollo program utilized a solid-rocket Launch Escape System (LES) that was ballasted with lead to provide aerodynamic stability during the initial launch abort and further relied on passive, timer-based, systems to reorient and stabilize the capsule for parachute deployment. In an attempt to save weight and add capability, the Launch Abort Vehicle (LAV) is being designed to use an active Attitude Control Motor (ACM) located in the Launch Abort System (LAS) tower to both stabilize the vehicle in the initial abort configuration and then to reorient the vehicle for parachute deployment.

* Aerospace Technologist, Dynamic Systems and Control Branch, M/S 308, AIAA Senior Member.
† Aerospace Technologist, Dynamic Systems and Control Branch, M/S 308, AIAA Associate Fellow.
‡ Research Engineer Staff, Langley Program Office, c/o NASA-LaRC M/S 308, AIAA Member.

To assess the physics and challenges of the ascent abort problem, a multi-center team of NASA researchers began collaboration in the fall of 2005. The first priority was the development of a simulation of the Apollo LES, which could then be scaled-up to CEV size and used as an initial study configuration. The CEV design would not emerge until contractor selection in the late summer of 2006, which was still some ten months away. With the Apollo-based simulation the government began the design of an active controller for the LAS to replace the passive Apollo system. During the run-up to contractor selection the government evolved its own candidate CEV LAS designs, which provided valuable aerodynamic data, mass, and propulsion system performance estimates. As data became available, the baseline simulation was updated and the control design modified to continue to meet prescribed performance requirements.

By the time of contract award, the government was concluding the design of its third CEV revision. This third design did not receive much scrutiny by the team because the contractor design was available. Using comparative analyses and simulations, over the past six months, the government and contractor have been working together and the control designs have been resolved into a single design for government analysis. This design continues to evolve as new test data becomes available.

While the standard NASA CEV simulation is written entirely in C and FORTRAN, the original government LAS controller design and subsequent comparative studies have been performed in a CEV LAS guidance and control (G&C) simulation built using an off-the-shelf dynamic system analysis package coupled with open-source support tools. This simulation was required to support a rapidly-changing baseline vehicle configuration; these changes included mass properties, abort rocket thrust levels and nozzle locations, reaction control system thrust levels, attitude control motor nozzle configurations and thrust levels, and aerodynamic database modifications. Derivative studies of alternative configurations are also supported with the same set of tools.

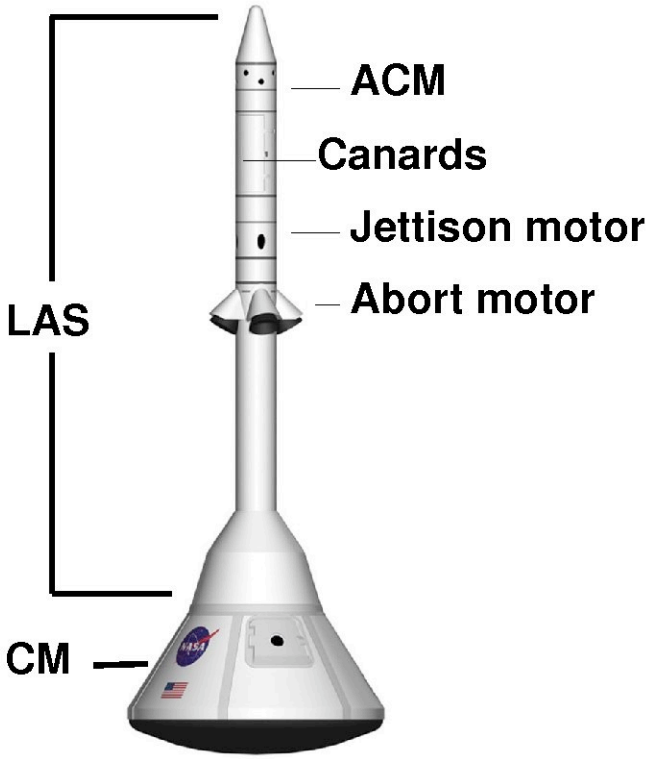

Figure 1. Elements comprising the Orion Launch Abort Vehicle.

II. Simulation Tools

This paper describes the tools and simulation environment in use by the NASA Langley CEV LAS G&C team. In addition to a brief description of the tools, special features and capabilities of the tool are pointed out where they make a significant improvement to the effectiveness of the team.

A. Simulation Environment

The CEV LAS G&C simulation is based on a version of the SAREC environment[2], itself based on the MathWorks® Simulink® commercial simulation and dynamic analysis tool. SAREC was developed by one of the authors at NASA Langley for control law development for a sub-scale blended-wing-body flight test vehicle.

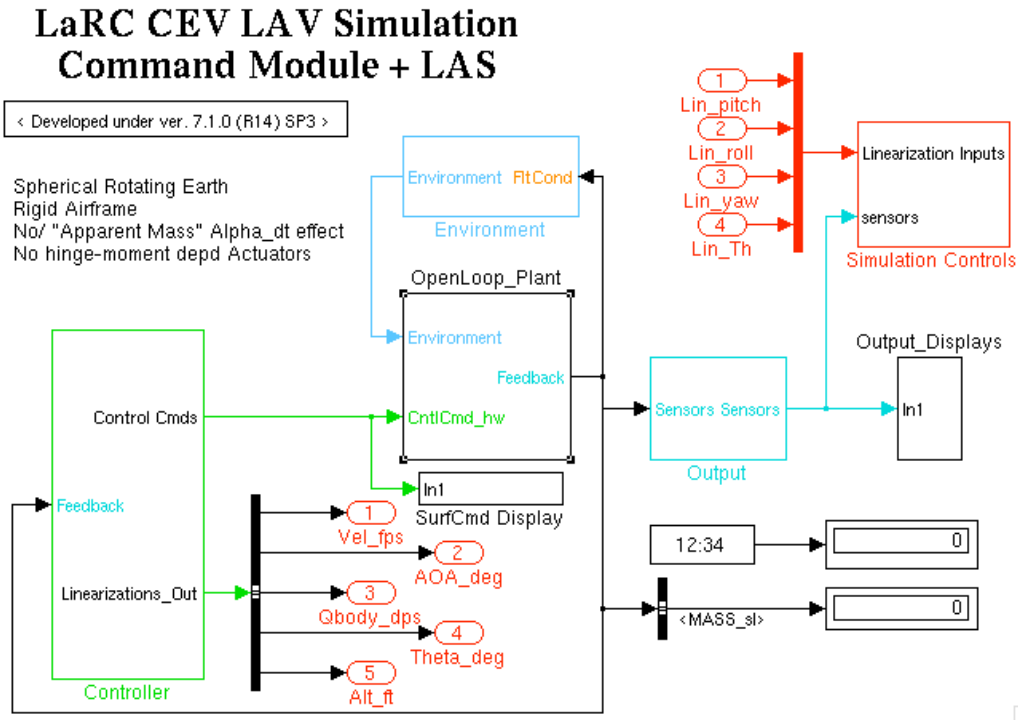

Figure 2. Top level diagram for the G&C simulation.

The G&C simulation is a full six degree-of-freedom, non-linear, spherical rotating Earth, rigid-body simulation environment utilizing quaternion attitude propagation. Figure 2 shows the top-level diagram of the simulation. The main blocks are the plant, controller and environment. Figure 3 shows the structure below the plant block. The diagonal layout is a feature of SAREC that follows the information flow from upper left to lower right with a minimum of information flowing in the reverse direction. The plant block is driven with information from the environment model and the control system. All outputs of the sub-blocks are captured in the main signal bus for recording during a simulation.

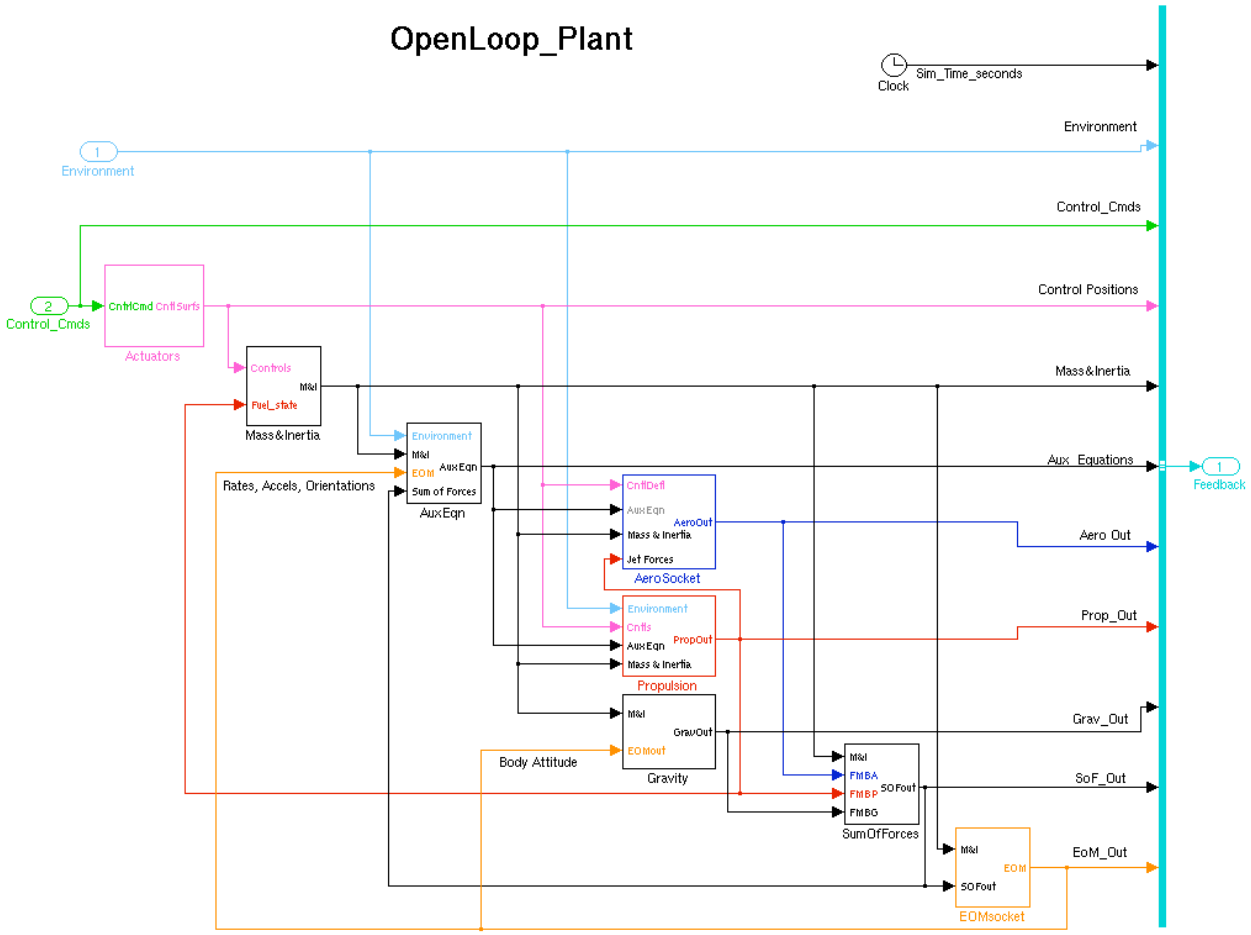

Figure 3. Open-loop plant subsystem block.

The mass and inertia block, shown in Fig. 4, has been built to accommodate changes with no hard-coded numbers that pertain to the vehicle. The mass, inertia and center of mass of each modeled part of the vehicle are parameters that are defined in a main set-up script that is executed prior to a simulation run. At each simulation step the composite mass and inertia properties are computed. Similarly, the propulsion block has each jet or rocket location, orientation, specific impulse, exit area and, if not throttleable, the thrust level defined in the main set-up file. In this way no block diagram editing is needed to change a parameter. Edits are only needed to add more parts or propulsive jets. It should be noted that to accommodate the stage separation there are two mass sub-blocks as depicted in Fig. 4. The upper block in the figure represents the CM only and the lower block represents the CEV capsule plus the LAS, known as the LAV. These two blocks execute as 'enabled subsystems' so when not enabled they do not execute and their outputs are zeroed. To represent different vehicle configurations, the same simulation diagram is used with different set-up files to define the different mass or jet parameters.

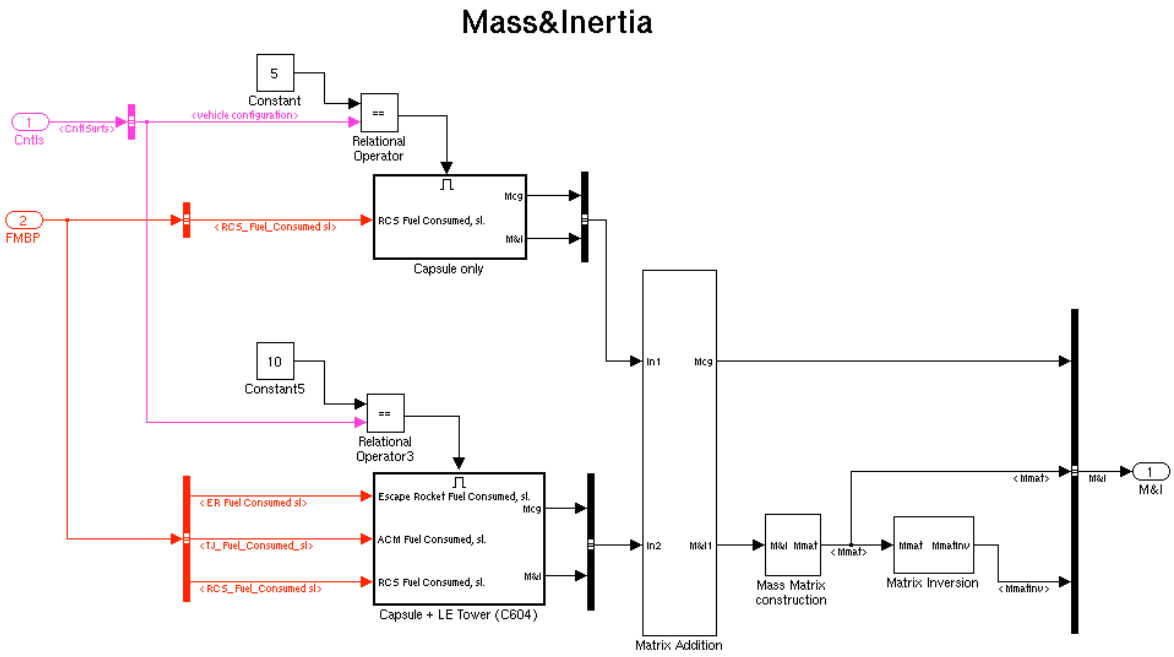

Figure 4. Mass and inertia diagram depicting the enable triggers used for staging.

The aerodynamics block is linked in from an aerodynamics library model. The aerodynamic table interpolation and build-up equations are currently provided by the CEV aerosciences team in the form of C Application Programming Interface (API) which reads text-based data files. The G&C simulation framework accommodates this by calling the C code as an external Simulink® "s-function."

Configurable subsystem template blocks have been used to allow the user to select one of several aerodynamics models at run time. Figure 5 shows the top-level of the aerodynamics library. The left-most block is the master and the two blocks on the right are the possible selections. The master is linked to the simulation structure. In general, the choice of which block a master represents can be made with a mouse-click on the block or programmatically. The G&C simulation uses the programmatic approach within the set-up script, where the appropriate supporting data files are also selected. In the case of Fig. 5, the last two inputs to the master are not common to the blocks the master can represent. These inputs are zeroed if not used by the chosen block that the master represents.

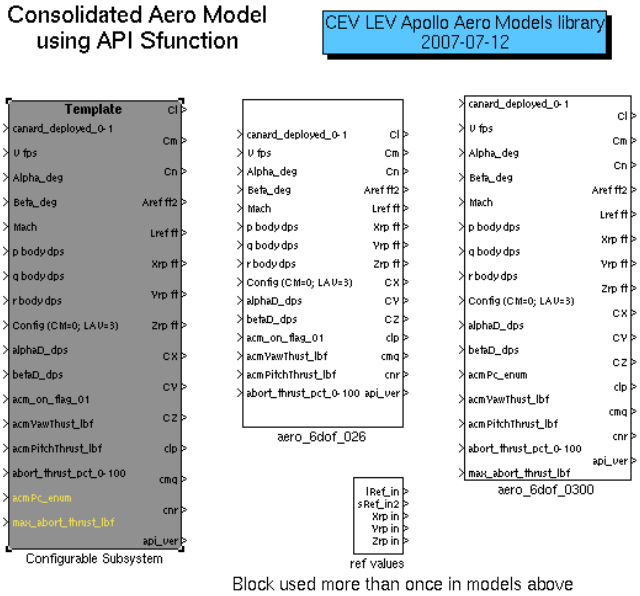

Figure 5. Aerodynamics library showing template and available options for aerodynamics subsystems.

The control system is also implemented as a library block that is linked into the simulation. The linked block is just under the controller block shown in Fig. 2. Like the aerodynamics library, the controller library was originally developed sequentially without the users having a choice of controller versions. When the configurable subsystem model was implemented for the aerodynamics module, a similar update was made for the controller. Now multiple versions of the controller exist in the control system library and the selection is made at run time. To avoid confusion, the block annotation property of the configurable subsystem in the simulation is set such that the block choice is displayed in the diagram below the block name. Further annotations can be made by turning on the 'display tag' property (enabled from within the Simulink® Block Properties -> Block Annotation menu) and then setting the tag string within the setup script.

The environment block, in Fig. 2, also uses a configurable subsystem to choose among available atmosphere models in an atmosphere library. Currently, only a US 1976 standard atmosphere is provided as a functioning model within the library. However, there is another block choice in which all the atmospheric terms (density, pressure, temperature, speed of sound, and dynamic viscosity) are parametrically defined in look-up tables as functions of altitude. This choice is convenient when comparing to an external simulation that employs a different atmosphere model. In such cases, the time history data of those terms and altitude from the external simulation are manipulated into a suitable matrix that satisfies the parameters. G&C simulations can then be compared to external simulations with identical atmosphere models.

B. Version Control

From the outset of the project it was recognized that the G&C simulation development effort was going to require configuration control to manage and facilitate the work among the developers as well as to provide a means to keep a record of files exchanged with other organizations across the country. Subversion[3] (SVN), an open-source, multi-platform source configuration management tool, was chosen for software and documentation version control. Through the use of a shared repository, team members could contribute, retrieve and track changes to vehicle simulation components, auto-coded models, check-case data and project documentation despite using different workstation hardware platforms. Both command-line and graphical user interfaces are available for Subversion for most computer platforms.

Employing SVN as the version control package, project files are developed within a 'trunk' directory tree. Whenever any changed file is checked in to the repository, the overall repository revision number increments by one. This revision number applies to every file in the repository and provides a way to refer to a version, or instance, of the simulation. In effect, the revision number corresponds to the state of the project at an instance in time. Furthermore, SVN has the ability to efficiently (in terms of speed and storage requirements) associate a 'tag' with a revision number. The team uses tags to more easily identify and record versions that correspond to significant versions of the G&C simulation- for example, the original Apollo-based simulation in final form. All revisions committed after the Apollo tag pertain to the CEV variants.

Another feature of SVN is the ability to spawn a development branch from the trunk. A new branch can be spawned at any time, and multiple branches can co-exist. The appeal of the branch feature is that it isolates development of new features from the trunk where users may continue to operate and develop as normal. And, it affords those working on the branch the ability to update and collaboratively develop just as though they were operating on the trunk. When features of the branch are deemed ready, they can be merged back into the trunk. The team has used branches to incorporate new subsystem models while leaving working, tested versions on the trunk available for other team members.

SVN also has the ability to refer to files in other branches. For example, a user wanting to assess the new aerodynamic model before the branch has been merged back into the trunk can use the SVN `switch` command to replace his local copy of the aerodynamics directory with one from the development branch. This switch remains in effect until cancelled (reversed).

Branches have also provided a means to more easily evaluate alternative vehicle and control system configurations. Without version controlled branches one would simply start with a copy of the original simulation and begin changing it to suit the needs of the new configuration. The drawback of this approach is that the new and old simulations are unconnected, but likely contain much common code that would gradually diverge. Updates or

fixes made in one place would have to be manually propagated to the other. Keeping an alternative configuration as a branch allows an automatic means of propagating changes from the branch to the trunk or vice versa.

Some additional features of the SVN tool are retrieval of old files, differencing files and logging of user comments. At any time a user can retrieve from the repository any file (or set of files) current or past. The user can specify the files by revision number or date; in effect, having nearly infinite 'undo' capability. These files can then be compared with the other versions using the SVN `diff` command. This feature is useful when de-bugging the simulation or assessing what changes might be lingering in a local copy prior to a check-in. When a check-in is performed, SVN allows the user to specify a description of the change(s). This has been extremely useful to the team by providing a narrative history of changes.

The repository is currently (July 2007) at revision number 985, contains approximately one thousand files, and requires about 260 MB of storage with all branches, tags and information to undo or return to any revision.

C. Aerodynamic Model Import

As the CEV project progressed, a variety of vehicle configurations were considered and analyzed. These configurations came from various sources and in a variety of formats. The aerodynamic database went through several iterations of format before becoming standardized, including text-based, binary-based, and spreadsheet-based formats; the original Apollo data was processed either manually or via optical character recognition tools. The wide variety of formats were all converted, via scripts and other custom-built utility programs, into skeletal DAVE-ML[4] aerodynamic model files which could then be imported into the Simulink® simulation tool. Reliance on the standard Unix 'make' utility, with its *makefile* scripts, provided a reliable and repeatable automation process to reduce human error. The DAVE-ML format also provided a convenient method to record information about the origins and subsequent modifications to each database, as well as allowed use of standard tools to inspect the data tables.

DAVE-ML, a human- and machine-readable text file format based on XML, is being proposed as an AIAA standard format for aerodynamic models. One text file is used to describe model I/O, multidimensional data tables, buildup equations (using MathML syntax), check case data, data uncertainty boundaries and model provenance. Thus, one file can be (with appropriate automation) realized as code or Simulink® block diagrams through batch scripts or can be read into memory structures at run-time with no intermediate code compilation[5].

With the adoption by the CEV project of a text-based aero table format with a standard C API aero model, the need to re-host a variety of aerodynamic models has been reduced. It is anticipated that as the configuration stabilizes later in the project, a DAVE-ML version of the aero model will be revisited.

D. Control System Model Development

Early in the CEV LAS development process, NASA Langley engineers developed a control law to provide active control of the CEV vehicle during launch abort (prior to CEV contract award) to explore the feasibility and benefits of an active control system. This control law was developed using a commercial dynamic system modeling tool (MathWorks® Simulink®) in which the CEV G&C simulation was also written. However, the need to share this control law with other users within a C-based simulation environment required the translation of the block diagram algebra into C code. Another MathWorks® product, Real-Time Workshop®, was utilized to automatically generate the equivalent C-code for this purpose.

As the design matured, repeated changes to the control law suggested that the 'autocode' process should be automated. To facilitate this, the conventional Unix `make` utility was brought to bear. A *makefile* was developed that:

1. determines if the control law model has changed
2. runs Matlab in batch mode to autocode the control law into C
3. compiles the resulting C source into a standalone utility program
4. runs the standalone utility program against a set of checkcase inputs
5. compares the resulting outputs with the outputs of the same checkcase in Matlab
6. reports on success or failure of the comparison

The *makefile* also provides rules to create a distribution tarball of the resulting source code, any required headers, the checkcase data and README files to facilitate sharing with other teams.

Once a CEV contractor was engaged, the contractor began to produce control laws. In order to analyze these control laws, which were written in C, these C routines were translated into equivalent Simulink® block diagrams by hand. To ensure proper operation of the Simulink® equivalent routine, each subroutine was compiled with a C-based s-function wrapper and linked into a special test model in Simulink®, allowing the C function to be exercised in parallel with the Simulink® equivalent. Differences between output signals from the native C and Simulink® equivalent routines were root-square-summed and compared against a tolerance (typically within 1/100 %) by an automatic unit test script; the complete equivalent Simulink® realization of the contractor-developed control law was then compared against checkcase time history data provided by the contractor to verify overall system operation.

E. Collaboration Tools

An interactive, collaborative, web-based common bulletin board wiki, based on the open-source Soks wiki[6], was implemented. This allowed posting of how-to lessons and team contact information, and sharing of an electronic library of documentation that could easily be edited by any team member with change tracking. Some unlearning of emailing methods was required to use this capability. Perhaps the most utilized wiki feature was the formatted revision log, an example of which is shown in Fig. 6. Automated scripts generate the web pages, providing easy access to the revision logs for the simulation allowing the team members to keep up with changes to the simulation.

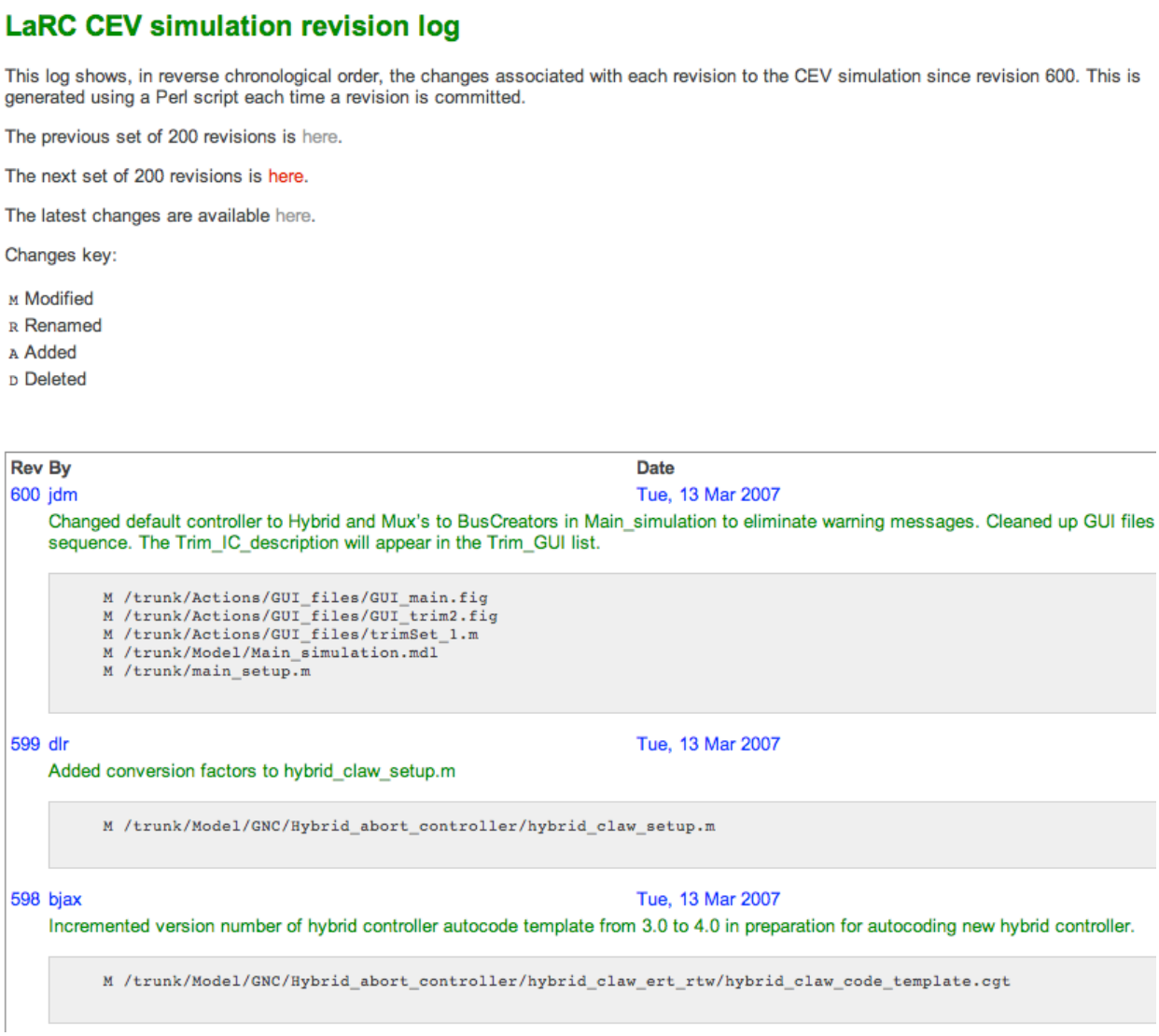

Figure 6. Wiki page displaying formatted SVN revision log information.

III. Lessons Learned

A. Version Control Simplifies Development and Coordination

Implementation of a version control system has made coordinated development and use of simulation tools easier by ensuring that everyone has access to the most recent copy of the simulation models and analysis tools. And, it eliminates the implementation of a 'sneaker net' (person-to-person) sharing of disks or thumb drives, which fosters an ambiguous, and likely undocumented, information dissemination chain. An open-source solution was found that satisfies most version control needs. It was also learned that frequent updates by each subscriber are a good idea.

B. Branches and Tags Benefit Parallel Development

Use of branches and tags helps segregate simulation development from production use; merges of development branches back into the production 'trunk' can be performed at an agreed-upon point of the project. It was discovered that keeping the branch fresh with frequent merges from the trunk of even minor changes is necessary to help ensure a successful merge of a branch back into the trunk. Ideally the branches should be updated after every modification of the trunk. In practice however, updating after every two or three trunk revisions is sufficient.

C. Wiki Pages Enhance Communications

Use of a collaborative internal wiki website has enhanced team communication and makes up for one missing feature of the Subversion version control system: keeping a log of merge points. A dedicated wiki page lists all known branches and tags for easy reference. Another wiki page is used to display recent revisions to the production 'trunk' of the common toolset; this page is automatically updated with each commit (Figure 6). We are pleased with the utility of this and heartily recommend this new technology for internal communications.

D. Automate Repetitive Tasks As Much As Possible

In the initial stages of the project, implementation of the initial (Apollo) aerodynamic database and autocoding of the first abort system control law was done manually. By the third or fourth revision, these tasks had become tedious. A good investment of resources was to automate (as much as possible) these tasks, and to provide README files and wiki pages with instructions so anyone could run them. Development and testing of the automated scripts and *makefiles* took some time, but this investment was regained several times over with continual inclusion of new aerodynamic models and control laws, despite adopting three different formats for aero data.

E. Importance Of Documentation and Unit Testing

Incorporation of one externally-produced subsystem model was hampered by a lack of understanding of some coordinate systems to which the models referred. This lack of understanding required extensive correspondence between the implementer and the developer to resolve. Good project documentation is always a good thing and is well worth the investment, but is often neglected by single developers.

Another aspect that made code reuse difficult was lack of unit tests. In addition to ensuring proper implementation of re-hosted code, the unit tests themselves (if well written) can serve as a starting point for documentation since each test set represents an example of how to use the code.

F. "Remove Before Flight" Tags Prevent Inadvertent Mutations

When working with a large multi-layer Simulink® block diagram it is difficult to keep track of temporary edits made to the diagram. These edits may have been made to turn off some feature or to change something for testing purposes, but remembering all the changes can be challenging. The danger is if a temporary change accidentally gets committed to the repository; everyone will receive the change the next time they update. To help keep track of those temporary edits a new Simulink® library palette has been created. The palette contains many of the more commonly used blocks and potentially could contain all the available blocks. The key feature of this palette is that the blocks are colored with a non-standard color and their tag property is set to “REMOVE BEFORE UPLOAD”. The block annotation property for each of these temporary blocks has been set to display the tag value below the block name. This provides a visual reminder to remove the blocks before checking the simulation into the repository. As a further safeguard, at initialization, the simulation automatically performs a `find_system` command to scan the entire block diagram (including linked files) for the existence of tags with the “REMOVE BEFORE UPLOAD” text. If such a tag is found, a warning message is displayed within the command window and

an auditory alert is made. This functionality is implemented with m-code inserted in the model initialization callback routine.

IV. Concluding Remarks

To date the CEV LAS G&C simulation has been significantly modified at least six times to keep up with the evolving CEV LAS design. Each new design brings with it a potentially new structure, as the aerodynamics database may bear no resemblance to the previous model and the mass properties may be more (or less) detailed than before. This report contained a brief description of the simulation environment used to build the initial CEV launch abort control system, as well as to perform subsequent analysis of the currently-selected control architecture. The environment was built using readily available commercial and open-source software to provide a flexible, responsive framework to support studies of a variety of CEV LAS configurations and control laws. Where appropriate, features that have been especially useful in improving the efficiency or accuracy of the simulation have been pointed out. To support effective information flow and documentation, an open source version control software package has been used. Communication among the team members has been further enhanced through the use of a wiki, again based on open source software. This report describes these tools and offers a lessons-learned narrative regarding the simulation development effort.

References

[1]NASA TM 2005-214062, "NASA's Exploration Systems Architecture Study," November 2005.

[2]Christhilf, David M., and Bacon, Barton J., "Simulink-Based Simulation Architecture for Evaluating Controls for Aerospace Vehicles (SAREC-ASV)." AIAA paper 2006-6726, presented at the AIAA Modeling and Simulation Technologies Conference and Exhibit, 21 August 2006, Keystone, Colorado.

[3]http://subversion.tigris.org [cited 16 July 2007].

[4]Jackson, E. Bruce, Hildreth, Bruce L., York, Brent W., and Cleveland, William, "Evaluation of a Candidate Flight Dynamics Model Simulation Standard Exchange Format." AIAA paper 2004-5038, presented at the AIAA Modeling and Simulation Technologies Conference and Exhibit, 17 August 2004, Providence, Rhode Island.

[5]Hill, Melissa A., and Jackson, E. Bruce, "The DaveMLTranslator: An Interface for DAVE-ML Aerodynamic Models." AIAA Paper 2007-6890, to be presented at the AIAA Modeling and Simulation Technologies Conference and Exhibit, August 2007, Hilton Head, South Carolina.

[6]http://www.soks.org [cited 16 July 2007].