AN OPEN-SOURCE, EXTENSIBLE SPACECRAFT SIMULATION AND MODELING ENVIRONMENT FRAMEWORK

Andrew J. Turner[†] and Christopher D. Hall[‡]

An Open-Source, extensible spacecraft simulation and modeling (Open-SESSAME) framework was developed with the aim of providing to researchers the ability to quickly test satellite algorithms while allowing them the ability to view and extend the underlying code. The software is distributed under the GPL (General Public License) and the package's extensibility allows users to implement their own components into the libraries, investigate new algorithms, or tie in existing software or hardware components for algorithm and flight component testing. This paper presents the purpose behind the development of the framework, the software design architecture and implementation, and a roadmap of the future for the software package.

INTRODUCTION

There are many spacecraft software packages, both commercial and open-source. Commercial examples include Analytical Graphic's Satellite Toolkit (STK),[1] Princeton Satellite System's MultiSatSim,[2] and NASA JPL's DARTS Shell.[3,4] These packages are either proprietary, expensive, expandable only through a request to the manufacturer, or dependent on other commercial software packages. Example open-source or free of cost software are SaVi,[5] ORSA,[6] or WinOrbit.[7] While these codes are typically available to the user, their current architectures are not designed to fit a broad class of spacecraft simulation or allow for easy extension by new users. The common shortcomings of available satellite modeling software impose limitations on users that can limit research efforts. Where the current solutions excel, there is still a need for a software package to interface with these packages and extend their functionality.

The Open-Source, Extensible Spacecraft Simulation And Modeling Environment (Open-SESSAME) framework is a project aimed to develop the framework for a cross-platform, open-source package of software that does not have these limitations. The framework provides a useful spacecraft simulation modeling tool to engineers and students while complementing current tools and packages. Software components should be reusable from one model to the next and not require users to reimplement the code each time they build a simulation. The Open-SESSAME framework is also designed to allow for easy, infinite extensiblity without requiring the user to learn a new scripting language or to redesign the framework's architecture. The framework gives a basis in which students and researchers alike can quickly and easily test various control algorithms, stability conditions, and sensor and actuator components.

The rest of this paper presents some of the benefits of developing a simulation framework under an open-source license. An outline is given of how Open-SESSAME has met the projects goals through the use of intelligent architecture design and available tools. The collection of libraries and their ability to interconnect is discussed, as well as examples of how users can develop various simulation

[†]Former Graduate Research Assistant, Aerospace and Ocean Engineering, Virginia Polytechnic Institutde and State University, Blacksburg, Virginia 24061. Currently with Realtime Technologies, Inc., Royal Oak, Michigan 48067. aturner@engineer.com. Member AIAA, Member AAS.

[‡]Professor, Virginia Polytechnic Institutde and State University, Blacksburg, Virginia 24061. cdhall@vt.edu. Associate Fellow AIAA, Member AAS.

applications. Readers of this paper should understand the design strategy behind Open-SESSAME, know how to use the framework components for their own applications, and be able to extend the functionality and contribute to the online community.

ROLE OF OPEN-SOURCE

Open-source software is released to the public under a license which ensures that modifications to the code, as well as the code itself, remains freely available to all users. A well-known example of a successful open-source project is the Linux operating system. Examples of other open-source software projects include advanced mathematical libraries, graphical user interfaces, and visualization tools. Most of these tools are hosted on public repositories that assist in storing the software and providing useful development tools and forums for developer and user discussion.

This sharing of software offers great benefits to academic users because useful tools are kept free of cost and are maintained by a community of developers throughout the world. Software bugs are fixed and new functionality is constantly being added and resubmitted to the community at large. Projects that are released under an open-source license tend to live beyond their original creators and beginnings, helping to ensure that useful ideas and developments continue to be used.

The philosophy behind open-software can be summed up as follows:[8]

- The freedom to run the program, for any purpose.
- The freedom to study how the program works, and adapt it to your needs.
- The freedom to redistribute copies so you can help your neighbor.
- The freedom to improve the program, and release your improvements to the public, so that the whole community benefits.

For these reasons, a satellite simulation framework, such as Open-SESSAME, developed alongside other open-source projects has the benefit of being available to more engineers who may use and improve it. The framework is released under the General Public License (GPL),[9] which is available at www.gpl.org. The code is currently available to download for free, and users are encourage to use and extend the software and contribute back to the community.

SCOPE OF FRAMEWORK

Open-SESSAME is a design architecture for building spacecraft simulation and modeling applications. The framework does not dictate how a user must employ the libraries. For example, the math library includes integration routines that can be used independently of satellite modeling. However, the design of the libraries and their interconnection is based on a regime of target applications that are typical in satellite simulation.

The current target domain of Open-SESSAME is in dynamics modeling and simulation. These applications can include attitude or orbit modeling, coupled orbit and attitude simulation, hardware-in-the-loop testing and verification, space environment assessment, or control algorithm validation. However, it would be possible to add software for modeling power, structure, or thermal modeling of a spacecraft.

The architecture is also designed to allow for easy extension without requiring users to recode the library components or change existing applications. The next section discusses the current Open-SESSAME components and also presents possible extension points where it is beneficial to future users to add components as necessary. Furthermore, the framework could be extended to include

visualization and a graphical user interface depending on the application. Since Open-SESSAME is not designed to require users to replace their current tools, extensions can be added to integrate Open-SESSAME application input and output with current engineering tools.

FRAMEWORK ARCHITECTURE

The Open-SESSAME framework is composed of a number of libraries that provide specific functionality. These libraries are comprised as follows:

1. *Math*: Useful math operations and tools such as matrix, vectors, integrators, interpolators, and conversions.
2. *Utilities*: Assorted miscellaneous tools, and classes for dealing with time.
3. *Rotation*: Collection of coordinate transformation representations and their conversions.
4. *Attitude*: Spacecraft attitude dynamics equations, reference frames, and state representations.
5. *Orbit*: Spacecraft orbit dynamics equations, reference frames, and state representations.
6. *Dynamics*: Algorithms for propagating the dynamics of spacecraft orbit and attitude, coupled or uncoupled in varying degrees.
7. *Environment*: Models of central bodies and space environment disturbances.
8. *Data Handling*: Data and system level file handling for saving and loading spacecraft applications or models, as well as integrating with external applications.
9. *Communications*: Utilities for setting up networks and communications between simulation applications and hardware components.

Interdependency between libraries is kept to a minimum. High coupling of libraries would create difficulties when attempting to upgrade or maintain components within the libraries. Therefore, libraries are design around a target domain and references to generalized interfaces of external library components are used.

The Open-SESSAME framework is programmed in C++. This language was chosen due to its prevalence in engineering curriculums and applications. Furthermore, C++ is designed to use Object-Oriented Programming (OOP),[10] upon which the Open-SESSAME framework is laid out. By using OOP the framework is designed to encapsulate data and operation within classes.[11,12] Classes then use a specified interface to allow the user access to the data without worrying about the underlying implementation.

A specific example of object-oriented programming in the dynamics domain involves the use of rotations. Open-SESSAME includes a **Rotation**[§] class which is a generalized representation of a rotation using no specific representation. Internally, the **Rotation** class is implemented using a **Quaternion**. However, this internal storage opaque to the user, who only will use the public interface to query the **Rotation** for the information in whatever representation is required, whether it is a quaternion, direction cosine matrix, or euler angles. the **Rotation** class internally handles the conversion from a **Quaternion** to a **DirectionCosineMatrix** (see Figure 1). The conversion routines are also opaque to the user. It is important to note, however, that because the framework is open-source, users of the software are free to inspect, correct, or optimize the code as necessary.

The UML (Unified Modeling Language) diagrams in this paper are a common method of drawing OOP design architectures. Open arrowheads denote a derived, or "is-a" relationship, where the

[§]Boldface is used to denote a class name and italics is used for naming an instance of a class.

module from the base of the arrow inherits and extends the functionality of the module at the end of the arrow. Solid arrowheads are a composition, or "has-a" relationship. Blocks that are stacked on top of one another can be interchanged as necessary. Within a block, the first row is the class name, the second row (usually with '−', which denotes that the variable is private and not visible outside of the class) are private data members of the class, and the third row are member functions ('+' denotes public, and '#' denotes protected).

The next section discusses some of the important aspects of the libraries mentioned above. However, since Open-SESSAME is a living software project that is open to the public and constantly under development, readers are encouraged to obtain the most current documentation and software from the project repository.

Libraries

Rotation Library

The *Rotation Library* is a collection of kinematic representations and operations used to represent coordinate transformations. Attitude orientations or orbit relative reference frames require a transformation to describe their axes relative to a specified reference frame. For instance, a rotation describes the transformation required to determine the orbit reference frame from the Earth-Centered Inertial (ECI) reference frame: $\mathbf{R}^{oi}$.

The currently implemented representations are: **Quaternion**, **Modified Rodriguez Parameters**, and **Direction Cosine Matrix**. The functionality is included , but not specifically implemented in a seperate class, for *Euler Angles* and *Euler Axis & Angle*. As mentioned previously, new rotation representations can be added to the library as necessary.

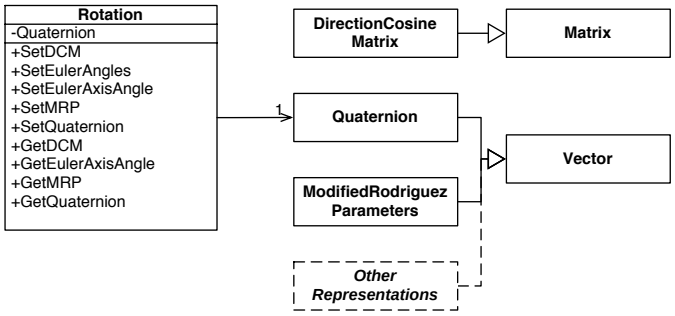

Figure 1: Rotation Library UML diagram.

Attitude Library

The *Attitude Library* is the collection of classes and tools that provide for analyzing, modeling, and simulating the attitude of a spacecraft. This collection includes state representations, kinematic and dynamic equations of motion, and the general spaceraft attitude.

The **AttitudeState** class represents the actual attitude measurement of a spacecraft at an instant in time. It contains a reference to both the appropriate rotation and relative reference frame of the rotation. Therefore, the **AttitudeState** class encapsulates this combined data into a single, succinct class with methods.

The kinematic and dynamic equations of motion are the physical algorithms that describe the motion of the spacecraft due to torques, disturbances, and any other desired modeling characteristic.

$$\dot{\mathbf{x}} = f(t, \mathbf{x}, pOrbit, pAttitude, Parameters, DisturbanceFunction) \tag{1}$$

These functions follow a generalized prototype, **odeFunctor**, which allows any kinematic and kinetic equation to be used, as long as it follows this function prototype. This prototype is shown in Equation 1. The state, $\mathbf{x}$, is the vector of states values at time t. The $pOrbit$ and $pAttitude$ inputs are instances of the current **Orbit** and **Attitude** classes at the evaluation time, $Parameters$ are any constants, and $DisturbanceFunction$ is a reference to the disturbance function (such as torque disturbances).

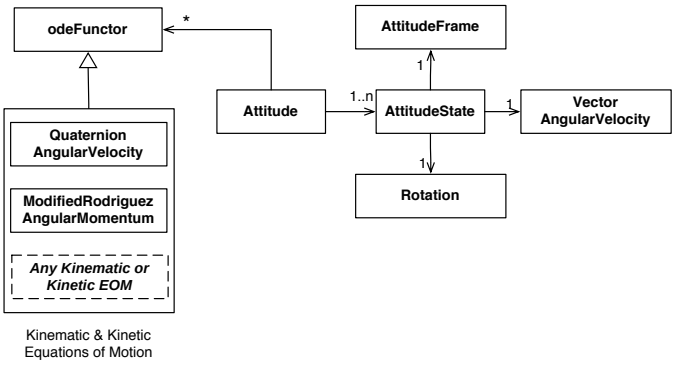

Figure 2: Attitude library UML diagram.

Orbit Library

The *Orbit Library* includes all the functionality to represent and simulate a spacecraft orbit. This toolkit includes, similar to the *Attitude Toolkit*, state representations, kinematic and dynamic equations of motion, and the general spacraft orbit.

The orbit state is representated by a conglomeration of classes. **OrbitStateRepresentation** is an abstract interface definition for storing and converting the state parameters of an orbit. The two primary representations are **Keplerian** and **PositionVelocity**. Each class stores a vector of its respective parameters and provides conversion functions for changing between the parameter types. Future representations could include canonical units, Delauney, or Poincaré variables.

State representations are accompanied by **OrbitFrame**, which stores the information regarding the frame from which the **OrbitStateRepresentation** is measured. Example frames could be Earth-Centered Inertial, Moon-Centered Moon-Fixed, or other pertinent representations. By associating a frame with a state representation, consistency is promoted to prevent comparing, for example, position vectors in ECI versus ECEF frames. Therefore, the **OrbitState** class contains both an **OrbitStateRepresentation** and an **OrbitFrame**. This concise class ensures that orbit state information that is passed through functions has a representation in a specified frame.

The **Orbit** class, much like the **Attitude** class, is a general encapsulation of the current orbit state, orbit history, associated environment, and reference to the current propagator and dynamic equations.

Dynamics Library

The *Dynamics Library* provides the functionality necessary to simplify the entire simulation process by encapsulating the simultaneous operation of many of the other toolkits, such as *Integration*, *Orbit*, *Attitude*, and *Environment*. Furthermore, it also is useful for coupling orbit and attitude dynamics. There are several existing schemes, as well as extension points for any new algorithms, for varying degrees of coupling. Currently, there are independent, weak (attitude dependent on orbit, or orbit dependent on attiude), strong (orbit and attitude interdependent), and joined (fully coupled dynamic equations) propagation schemes. A **Propagator** can also be used when the attitude or orbit is available from an external source (*e.g.* file, hardware, other software package).

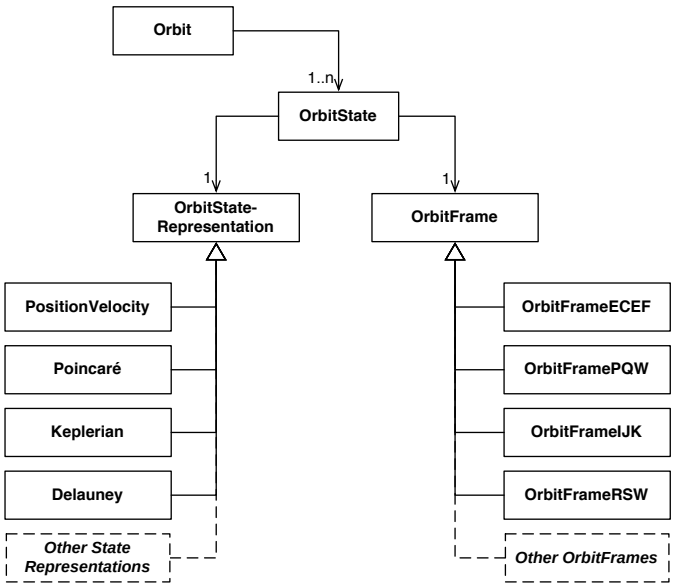

Figure 3: Orbit library UML diagram.

The **Propagator** class provides a defined interface to the library of propagators. The two derived classes, **NumericPropagator** and **AnalyticPropagator**, each implement the respective method of propagation. Specifically, a **NumericPropagator** requires an **Orbit** and/or **Attitude** class with dynamic equations or populated history. When the user assigns an orbit with a dynamics equation, the propagator will integrate the orbit, according to the propagation scheme, which is the same for attitude.

If the user does not include an orbit or attitude object, then the class will not attempt to integrate it. If a orbit or attitude is supplied (via external methods such as from file or hardware), then the other motion may use the assigned orbit or attitude history to calculate the dynamics due to coupling.

An **EnckeCombinedPropagator** is a unique scheme that applies Encke corrections to the orbit propagation during attitude propagation.[13] The user may inherit from the **NumericPropagator** or **AnalyticPropagator** to implement new propagation schemes.

Environment Library

The *Environment Library* is the collection of central bodies, external force and torque disturbance functions, and methods of calculating the effect of the environment on a spacecraft. The primary class, **Environment**, encapsulates all of the environment data that is usually referenced by the **Orbit** and **Attitude** objects.

The user can create an instance of a **CentralBody**, a representation of the Earth, Moon, or whichever celestial body about which the spacecraft is situated. This **CentralBody** object contains information regarding the radius, atmosphere, angular velocity, mass, and any other data that is pertinent to spacecraft modeling. The **EarthCentralBody** and other planets and moons are derived from the **CentralBody** class and, therefore, have the general functionality of the **CentralBody**.

The **Environment** object contains a reference to the central body and a list of the applied disturbance functions. These disturbance functions have a generalized but specified interface (time, orbit state, and attitude state) that is used to calculate the specific torque or force on the spacecraft during the integration of the dynamics. The user creates these functions, assigns constants that may

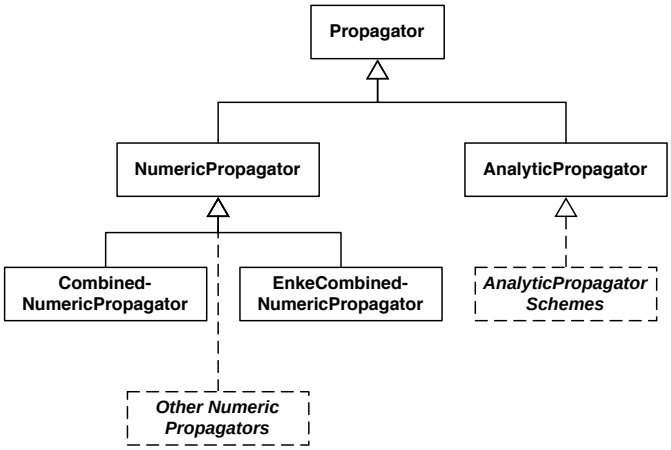

Figure 4: Dynamics library UML diagram.

be used (spacecraft mass, altitude, reflectivity), and stores them in the *Environment* instance that is used for the spacecraft. In each integration step, the attitude or orbit dynamics may call this function, at the instantaneous state, to obtain the torques and forces upon the spacecraft.

Data Handling Library

The *Data Handling* library is a collection of classes and functions for interacting with large sets of data as well as the external system environment. The **History** class and associated subclasses are used for storing the states of the spacecraft's orbit, attitude, or other parameters during simulation. The **Converter** collection of classes is used for saving and restoring the spacecraft states and parameters from a variety of forms, such as comma-separated value ASCII, MatLab, Satellite ToolKit (STK), or XML. Lastly, the communications software is included that allows an Open-SESSAME application to connect to networked machines to retrieve state, send state, or control multiple simulations or external software packages.

To succinctly store any number of states of the spacecraft, the **History** class provides a dynamically resizable vector of times, and derived classes add state variables that can be stored at associated times. For example, **OrbitStateHistory** and **AttitudeStateHistory** each respectively store the **OrbitState** and **AttitudeState** of the spacecraft after integration. These can then be retrieved, stored, or erased.

BUILDING APPLICATIONS

The previous section described each of the libraries in detail. These components are then used to build up an application depending on the desired simulation problem. Users can design applications with the Open-SESSAME framework by using the provided online documentation and examples, or even by discussing the desired application in the online forums. The next sections describe how a user could use Open-SESSAME to build and run an application.

Documentation

Users should begin using Open-SESSAME by obtaining the current version of the libraries and documentation from the public repository. The included API (Application Program Interface) doc-

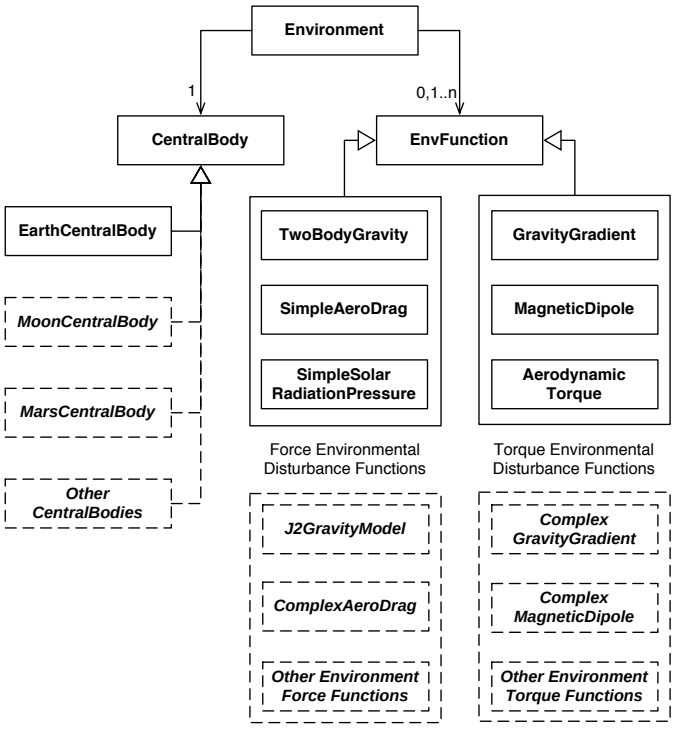

Figure 5: Environment library UML diagram.

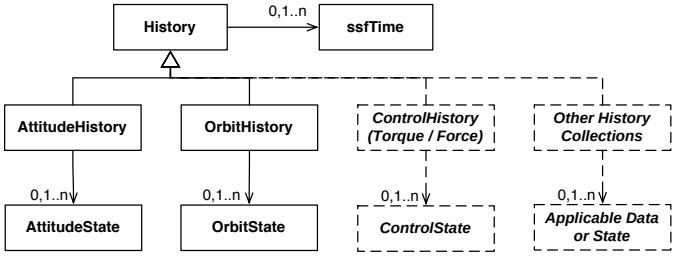

Figure 6: Data Handling library UML diagram.

umentation is created through an automatic document generation tool, Doxygen[¶]. This open-source, free program produces produces hyperlinked html documentation as shown in Figure 7 directly from the comments in the software. By keeping the comments within the software itself, it is easier to

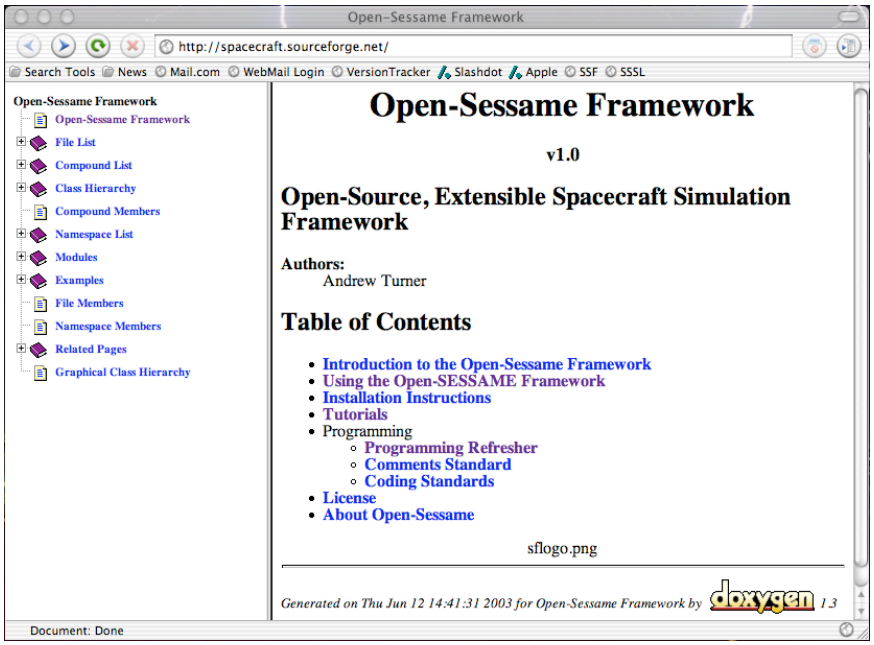

Figure 7: Example Doxygen API document html output.

maintain since changes to the code can be reflected in the documentation in the same location. This method is much preferred when compared to the standard maintaining the code and documentation as seperate entities. Doxygen can also compile Latex, XML, and Rich-Text Formatting (RTF) documentation from the same inline comments. For documenting formulas, Doxygen parses out Latex formatted equations from the code comments and compiles them into symbolic format for reading in the documentation. Furthermore, because Doxygen is open-source, the code is free to change, update, and maintain.

The documentation includes cross-linked and grouped information. For users to develop an attitude simulation, for example, they would access the *Attitude Library* under the *Modules* section in the sidebar. The library module gives a description of how attitude simulation works, and other pertinent modules for the user to inspect and include in a simulation. As users click through the hyperlinks, they are led through the functions, classes, and interfaces they need to use to develop a simulation application.

The Open-SESSAME documentation also includes a programming primer for users who have limited experience with programming, or have not touched on a number of difficult concepts. The primer covers a brief discussion and explanation of the major programming concepts employed in Open-SESSAME, and also suggests references for further reading if necessary. These lessons vary from variable definition to C++ classes to function pointers.

Using the API documentation, the SourceForge forums and notes, users can begin to use Open-SESSAME framework components in their code. The next section presents several specific examples of how an Open-SESSAME application would be built and look in code.

[¶]http://www.doxygen.org

Attitude or Orbit Modeling

A very standard, and relatively trivial, modeling exercise is that of independent attitude or orbit modeling. Since with Open-SESSAME, the implementation is relatively similar, and in the interest of brevity, this section will present only an attitude simulation. For more examples, and verification of the simulations, refer to Andrew Turner's Masters Thesis.[14]

An attitude simulation is a stand-alone integration of the spacecraft attitude dynamics equation with a possible disturbance torque function. The following steps could be followed to simulate a spacecraft's attitude:

1. Code the attitude dynamics equation
2. Code the disturbance torque function
3. Assign function parameters
4. Create an initial attitude state
5. Create and initialize an integrator
6. Integrate the equations
7. Graph or output the state history

These steps are shown in the code snippets below and the module interconnections are illustrated in Figure 8. It is not important at this point to understand the specifics of each of the lines of code and how to interface the matrix libraries and so forth. It is the goal of this section to merely present a typical simulation and how an application is built. As users read the online documentation and work with Open-SESSAME, building applications like will be relatively simple.

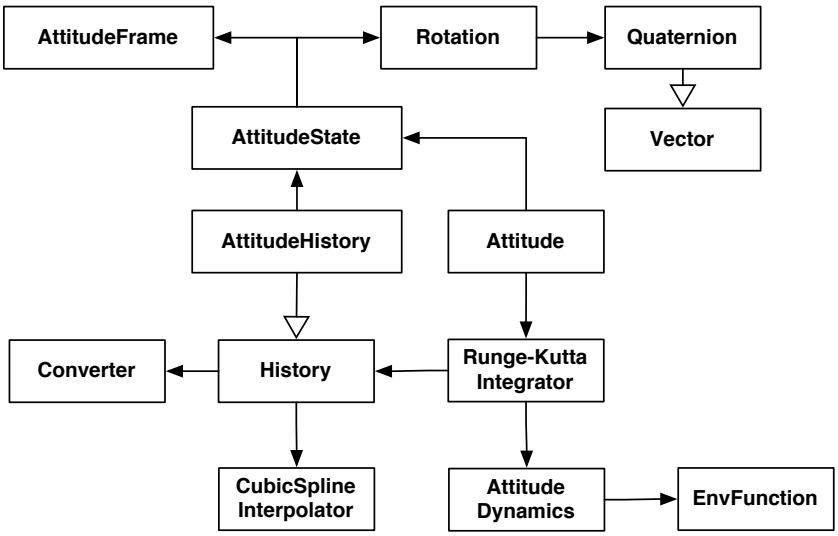

Figure 8: Attitude integration using Open-SESSAME.

Code the attitude dynamics equation

The dynamics equation is the right-hand side time derivative of the attitude dynamics. The function requires the current integration time, integrating state, spacecraft attitude and orbit (if applicable), a matrix of parameters, and a function pointer to the disturbance function.

```
static Vector AttituteDynamics(const ssfTime &_time,
                                    const Vector& _integratingState,
                                    Orbit *_Orbit, Attitude *_Attitude,
                                    const Matrix &_parameters,
                                    const Functor &_forceFunctorPtr)
{
   // initialize the variables, static to save memory allocation
    static Vector stateDot(7);
    static Matrix bMOI(3,3);
    static Matrix qtemp(4,3);
    static Matrix Tmoment(3,1);
    static Vector qIn(4);
    static Vector qDot(4);
    static Vector wIn(3);
    static Vector wDot(3);
    // get the state variables from the input integrating state
    qIn = _integratingState(_(1, 4));
    wIn = _integratingState(_(5,7));
    // normalize the quaternion
    qIn /= norm2(qIn);

    // calculate qDot
    qtemp(_(1,3),_(1,3)) = skew(qIn(_(1,3))) + qIn(4) * eye(3);
    qtemp(4, _(1,3)) = -(~qIn(_(1,3)));
    qDot = 0.5 * qtemp * wIn;

    // get the moments of inertia from the input parameters
    bMOI = _parameters(_(1,3),_(1,3));
    // calculate the disturbance torques
    Tmoment(_,_) = (_forceFunctorPtr.Call(_time, _Orbit->GetStateObject(),
                                _Attitude->GetStateObject()))(_);
    // calculate omegaDot
    wDot = (bMOI.inverse() * (Tmoment - skew(wIn) * (bMOI * wIn)));

    // setup the time derivate return state vector
    stateDot(_(1,3)) = qDot;
    stateDot(_(5,7)) = wDot;

    return stateDot;
}
```

Code the disturbance torque function

The disturbance function would contain any modeled torque disturbances. This example models the gravity gradient disturbance torques.

```
Vector GravityGradientTorque(const ssfTime &_currentTime,
            const OrbitState  &_currentOrbitState,
            const AttitudeState &_currentAttitudeState,
            const EnvFuncParamaterType &_parameterList)
{
    static Matrix MOI(3,3);
    MOI = *(reinterpret_cast<Matrix*>(_parameterList[0]));
    static Vector o3(3);
```

```
    o3 = (_currentAttitudeState.GetRotation2Orbital(_currentOrbitState))
                        .GetDCM()(_,3);
    static Vector Position(3);
    Position = (_currentOrbitState.GetStateRepresentation()
                        ->GetPositionVelocity())(_(1,3));
    return 3 * *(reinterpret_cast<double*>(_parameterList[1]))
                    / (pow(norm2(Position),3))   * skew(o3) * MOI * o3;
}
```

Assign function parameters

Function parameters are used to pass information into the dynamics equation. In this example, the moments of inertia are used in the right-hand side. The disturbance torque function is set to the gravity gradient modeling function from above.

```
    Matrix I(3,3); //I=[100 0 0;0 200 0;0 0 150];
        I(1,1) = 100;
        I(2,2) = 200;
        I(3,3) = 150;

    SpecificFunctor AttitudeTorquesFunctor(&GravityGradientTorque);
```

Create an initial attitude state

The initial attitude is setup with a quaternion corresponding to no rotation and an angular velocity of 0.1 m/s about the x-axis.

```
    AttitudeState myAttitudeState;
    myAttitudeState.SetRotation(Rotation(Quaternion(0,0,0,1)));
    Vector initAngVelVector(3);
        initAngVelVector(1) = 0.1;
    myAttitudeState.SetAngularVelocity(initAngVelVector);
```

Create and initialize an integrator

The Runge-Kutta integrator is setup with 1000 steps for 20 seconds.

```
    RungeKuttaIntegrator myIntegrator;
    myIntegrator.SetNumSteps(1000);

    // Integration times
    vector<ssfTime> integrationTimes;
    ssfTime begin(0);
    ssfTime end(begin + 20);
    integrationTimes.push_back(begin);
    integrationTimes.push_back(end);
```

Integrate the equations

The integration function is called within a "tick()" and "tock()" to calculate the operating time.

```
    cout << "PropTime = " << begin.GetSeconds() << " s -> "
                << end.GetSeconds() << " s" << endl;
    cout << "Attitude State: " << ~myAttitudeState.GetState() << endl;
```

```
tick();
Matrix history = myIntegrator.Integrate(
                         integrationTimes, // seconds
                         &AttituteDynamics,
                         myAttitudeState.GetState(),
                         NULL,
                         NULL,
                         I,
                         AttitudeTorquesFunctor
                     );
cout << "finished propagating in " << tock() << " seconds." << endl;
```

Graph or output the state history

Figure 9 shows an example attitude plot from Open-SESSAME using gnuplot. The formatting is plain, and allows the user to change the format as needed.

```
cout << history;
Matrix plotting = history(_,_(1,5));
```

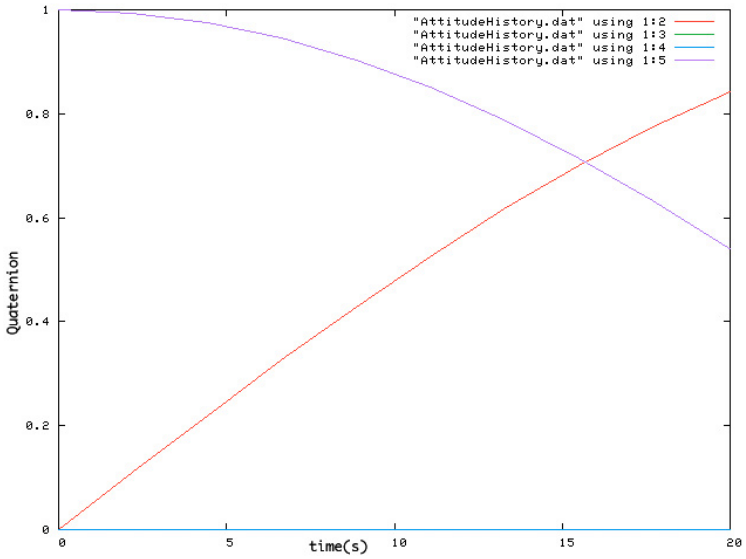

Figure 9: Example gnuplot output from Open-SESSAME of an attitude integration.

The entire simulation is only 55 lines of code, most of which are reusable for multiple iterations of propagating the simulation forward through time. Also, the dynamics equation is about 30 lines of code, but again most of this is setting up the variables that must be defined. The next section discusses coupled simulation, which, due to the design of Open-SESSAME, does not require many more lines of a code for a much more complicated simulation.

Coupled Simulation

Coupled simulation involves integrating the orbit and attitude dynamic equations with some dependence of dynamic or disturbance functions on the state of the other dynamic. As discussed previously, there are several different schemes for propagating coupled equations. However, each

scheme would employ the same, following method. This implementation assumes the user is using the orbit and attitude dynamics, distubance functions, function parameters, and initial state as mentioned above. The entire application architecture is shown in Figure 10.

1. Create and populate the environment
2. Define orbit and attitude conversion functions
3. Create and initialize the propagator
4. Propagate the equations
5. Graph or output the state history

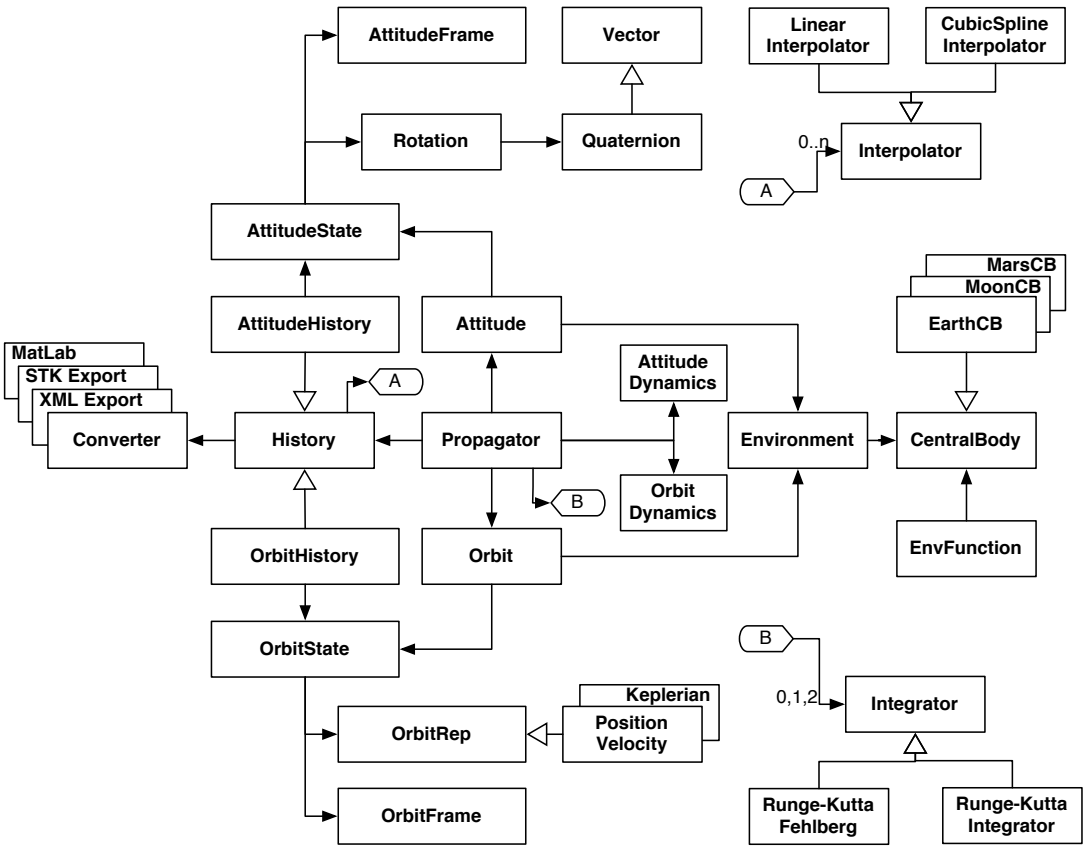

Figure 10: UML Diagram of Spacecraft simulation and control software components.

The environment object is used to store the central body and disturbance functions. The disturbance functions can take references (pointers) to data that can change dynamically with time, or even dependent on a changing central body (such as when moving from the Earth's sphere of influence to the Sun's SOI). The code for creating an environment in Open-SESSAME is as follows.

```
Environment* pEarthEnv = new Environment;
EarthCentralBody *pCBEarth = new EarthCentralBody;
pEarthEnv->SetCentralBody(pCBEarth);
```

```
// Add Gravity force function
cout << "Filling Parameters" << endl;
EnvFunction TwoBodyGravity(&GravityForceFunction);
pEarthEnv->AddForceFunction(TwoBodyGravity);

// Add Drag Force Function
EnvFunction DragForce(&DragForceFunction);
double *BC = new double(200);
DragForce.AddParameter(reinterpret_cast<void*>(BC), 1);
double *rho = new double(1.13 * pow(10., -12.)); // kg/m^3
DragForce.AddParameter(reinterpret_cast<void*>(rho), 2);
pEarthEnv->AddForceFunction(DragForce);

myOrbit->SetEnvironment(pEarthEnv);
myAttitude->SetEnvironment(pEarthEnv);
```

Another important point of discussion is the use of a propagator as an object. The attitude and orbit dynamics can use different integrators with different stepsizes and meshpoints. Therefore, it is useful to encapsulate the algorithm that maintains syncronicity between the orbit and attitude and also allows users to test various methods of multi-rate coupled propagation without changing the external interface to a *Propagator* object. The propagator then uses the *conversion functions* to convert the integrating state of the dynamics equations into a known storable format for the history objects.

Once an Open-SESSAME application like this example is setup, the program can continually propagate the satellite state forward and query for the new state as necessary. This allows users to quickly reuse components of their simulations, or other users' simulations available from the web, in new simulations. Seperate attitude and orbit models can be easily and quickly integrated together through the use of a propagator to build a more advanced model. This is useful for building *simulation servers* that can serve state information for hardware-in-the-loop simulations like those described in the next section.

Hardware-in-the-Loop Simulation

Figure 11 shows the application architecture and interaction between flight software and an Open-SESSAME simulation. The *Simulation Server* is setup as described in the previous section, and ready to propagate the simulated spacecraft state. The sensor stubs interact with the flight software like the hardware drivers, but instead of accessing hardware on a satellite bus, the code stubs query the simulation application for the current state information. The sensor stub then converts this simulated state information into the sensor's expected output, and adds any simulated measurement errors. The same occurs as control commands are fed to the actuator stubs. These stubs add noise and error; then they send an applied torque or force to the simulator server. The server uses this control information in propagating the state. Communication occurs through a socket connection, which allows the simulation server to reside and operate on a separate machine if necessary.

OPEN-SESSAME APPLICATIONS

Open-SESSAME was developed as part of the efforts of the Space Systems Simulation Laboratory (SSSL)[‖] at Virginia Tech. It is currently being used as the simulation server for developing and testing the hardware and flight algorithms for the Virginia Tech nanosatellite, HokieSat.[15]

Open-SESSAME components will also be used in an orbit simulation server for the two air-bearing

[‖] http://www.aoe.vt.edu/research/groups/sssl/

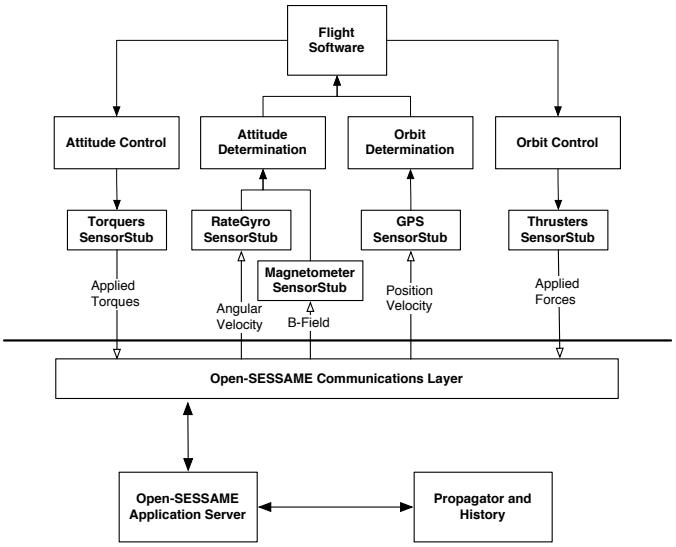

Figure 11: Hardware-in-the-loop integration with Open-SESSAME

spacecraft simulator tables.[16] The simulator tables will simulate the attitude of a satellite and use various control hardware and algorithms to model different mission scenarios. An Open-SESSAME simulation server will receive orbit control commands and provide system time and state information to the satellites and simulate formation flying. This application can be tied into other simulators as desired.

During the fall term at Virginia Tech, Open-SESSAME will be used as a means of teaching and demonstrating spacecraft modeling and simulation in an advanced spacecraft dynamics course. Students will implement new extensions to Open-SESSAME and develop novel applications as part of a semester long project. This course could serve as a model for other universities to employ open-source software such as Open-SESSAME in teaching students how to meld analysis and theory with software implementation, while also contributing to a larger body of work that can be shared amongst researchers and industry.

CONCLUSIONS

The Open-Source, Extensible Spacecraft Simulation And Modeling Environment framework has grown from small, one-person project into an extensive collection of tools and components that can be used in a wide-range of spacecraft simulations. The provided libraries allow users to quickly begin using the framework for developing applications for modeling and analysis, and also provide easy extensibilty for any specialized analysis that is required.

Furthermore, Open-SESSAME is implemented to provide for a good basis in beginning to learn the techniques and methods necessary to simulate satellites. Students and scientists can browse the documentation to learn how the Open-SESSAME components are implemented and refer to cited references for more in-depth discussion of advanced concepts. These concepts can then be implemented by using existing examples and tested.

There is no limit to the application of Open-SESSAME in the satellite industry. As an open-source project, the package is available to anyone with an internet connection and computer. The framework software is cross-platform and therefore does not restrict users to a particular operating system. Open-SESSAME is not dependent on any small group of users, project, or proprietary

product that could restrict its use or growth. Therefore, Open-SESSAME promises to be a tool that can continue to grow and extend.

For more information regarding Open-SESSAME, including the source code, documentation, forums, and team contact information, visit the SourceForge site at http://spacecraft.sourceforge.net.

References

[1] Analytical Graphics, Inc., *Satellite Toolkit Website*. February 10, 2003. <http://www.stk.com>.

[2] *MultiSatSim User's Guide v1.1.3*. 33 Witherspoon St. Princeton, NJ 08542: Princeton Satellite Systems, May 2002.

[3] J. Biesiadecki, D. Henriquez, and A. Jain, "A Reusable, Real-Time Spacecraft Dynamics Simulator," in *6th Digital Avionics Systems Conference*, (Irvine, CA), October 1997.

[4] *NASA JPL Darts Website*. February 10, 2003. <http://dshell.jpl.nasa.gov>.

[5] *SaVi Sourceforge Website*. May 5, 2003. <http://sourceforge.net/projects/savi>.

[6] *ORSA Sourceforge Website*. May 5, 2003. <http://orsa.sourceforge.net>.

[7] *WinOrbit Homepage*. May 5, 2003. <http://www.sat-net.com/winorbit>.

[8] R. Stallman, *The Free Software Definition*. 59 Temple Place - Suite 330, Boston, MA 02111: Free Software Foundation, Inc., 2003.

[9] *The GNU General Public License*. 59 Temple Place - Suite 330, Boston, MA 02111: Free Software Foundation, Inc., 2 ed., Available at http://www.gnu.org/licenses/gpl.html. 1991.

[10] G. Booch, *Object-Oriented Analysis and Design*. Boston, MA: Addison-Wesley, 1994.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.

[12] B. Stroustrup, *The C++ Programming Language*. Boston, MA: Addison-Wesley, 3rd ed., 1997.

[13] J. Woodburn and S. Tanygin, "Efficient Numerical Integration of Coupled Orbit and Attitude Trajectories Using An Encke Type Correction Algorithm," in *2001 Astrodynamics Specialist Conference*, (Quebec City, Canada), AAS 01-428, July 30 - August 2 2001.

[14] A. J. Turner, "An Open-Source, Extensible Spacecraft Simulation And Modeling Environment Framework," Master's thesis, Virginia Polytechnic Institute and State University, July, 2003.

[15] K. Makovec, A. Turner, and C. Hall, "Design and Implementation of a Nanosatellite Attitude Determination and Control System," *2001 AAS/AIAA Astrodynamics Specialists Conference*, July 30 - August 2 2001.

[16] J. L. Schwartz and C. D. Hall, "Comparison of System Identification Techniques for a Spherical Air-Bearing Spacecraft Simulator," *AAS/AIAA Astrodynamics Specialists Conference*, August 3-7 2003. AAS 03-611.