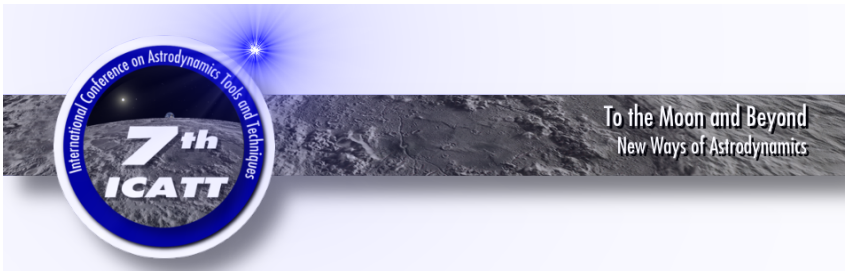

Contribution ID: **17** Type: **Oral presentation at the conference**

Basilisk: A Flexible, Scalable and Modular Astrodynamics Simulation Framework

The plethora of spacecraft simulation software tools is an indispensable part of modern spacecraft design processes. The continual increase in complexity of spacecraft missions and maneuver design, dynamical and kinematic design verification and post-launch telemetry analysis all heavily rely on software simulation tools. This simulation ability provides engineers with the tools to increase the quality of design and testing, by reducing cost and duration of development. For example proposed changes to a mission's configuration, parameter tuning or in-flight anomalies may be explored via Monte Carlo simulation. Additionally, hardware in the loop testing (HWIL) allows for verification and validation of the spacecraft hardware and software systems in a controlled laboratory environment. Such HWIL testing can expose technical faults and system integration problems saving considerable project financial and personnel resources before launching the system to space.

Basilisk is an astrodynamics framework that simulates complex spacecraft systems in the space environment. The types of missions, which Basilisk can be used to simulate, lay on a spectrum with earth orbiting cubesats at one end and interplanetary probes and spacecraft constellations at the other. The hallmark of the Basilisk framework is its highly modular system architecture. Modular design has been the guiding principle throughout Basilisk's development. The result is that Basilisk implements only two core system components, the Basilisk message exchange and Basilisk simulation controller. These two components are the only components required to begin building a Basilisk simulation scenario.

While many simulation tools possess overlapping features with Basilisk, none others possess the unique characteristics of Basilisk. These characteristics are that Basilisk is a highly modular, Python user-friendly, open-source simulation framework that provides sufficiently accurate (fidelity is configurable) coupled vehicle position and attitude dynamics, along with optional structural flexing, imbalanced momentum exchange device and fuel slosh dynamics, with at least a 365 times speedup (one mission year in one compute time day). Furthermore, Basilisk is equally well employed during early mission design phases as it is later on during detailed design phases and further in post-launch telemetry analysis and spacecraft command sequence validation.

Beyond the two required Basilisk components, the user constructs a simulation scenario by including any number of Basilisk Modules relevant to their specific mission. A Basilisk Module is a stand-alone code. Modules typically implement a specific model (E.g. an actuator, sensor, and dynamics model) or self-contained logic (E.g. translating a control torque to a RW command voltage). The two core Basilisk components and most modules are written in C++ to allow for object-oriented development and fast execution speed. However, Basilisk Modules can also be developed using Python, for easy and rapid prototyping, C (to allow flight software modules to be easily ported directly to flight targets) and Fortran (to accommodate legacy space environment models).

Whereas Basilisk modules are developed in a number of computing languages, Basilisk users interact and develop simulation scenarios using the Python programming language. The Python interface to the C/C++/Fortran layer relies on the Simplified Wrapper and Interface Generator (SWIG) library. SWIG is a cross-platform, open-source library that generates Python wrappers (amongst many other languages) for compiled C/C++ code. With the Python user layer and Basilisk's cross-platform development, (currently developed for MacOS, Windows, and Linux systems) the modularity results in no compile time or run time dependencies between one module and another. This modularity provides the engineer with the ability rapidly develop and reconfigure their simulation scenario in the Python language.

This modularity is achieved by the unique implementation of the Basilisk message exchange. A Basilisk module may read input messages and write output messages to the Basilisk messaging exchange. Decoupling of

the data flow between modules is achieved via the message exchange. The message exchange acts as a message broker for a Basilisk simulation. Modules read and write data structures referred to as 'messages' to the message exchange. A message is defined as having a name and payload data structure (typically a C/C++ struct). The Basilisk messaging exchange manages the trafficking of messages and employs a publisher-subscriber message passing nomenclature. A single module may read and write any number of messages. A module that requires data output from another modules subscribes to the message by registering the 'subscription', to the specific message name, with the messaging exchange. Conversely, a module that desires to output data for other modules registers the 'publication' of that payload data associated with a specific message name. The messaging exchange then maintains the messages read and written by all modules and the network of publishing and subscribing modules.

This paper will be presented in three main sections. The first section will survey the current context of state-of-the-art commercial-off-the-shelf (COTS) and government-off-the-shelf (GOTS) spacecraft simulation software and provide a context for the Basilisk framework. Each of these COTS/GOTS systems posses' unique feature sets. These unique feature sets predispose each tool excel when applied to different classes of spacecraft system analysis, by different engineers teams.

The second section will give a detailed account of the aforementioned novel Basilisk system architecture, which allows for the rapid development of a simulation for of a wide variety of complex spacecraft systems. The key architectural components to be discussed are the Basilisk message exchange, the Basilisk simulation controller and the anatomy of a typical Basilisk module. A number of archetypal Basilisk simulation configurations will be presented from the perspective of the end user engineer to illustrate the functionality of this design.

Finally this paper will discuss and provide demonstration of the following core Basilisk capabilities:
- Controlling simulation fidelity and speed of any module in a simulation by dynamically adjusting integration rates for each module during simulation execution.
- Basilisk multi-processing Monte Carlo tools.
- Data logging, analysis and visualization tools for large (multi-gigabyte) data sets.
- Execution of a Basilisk simulation distributed across multiple computing devices.
- 3D Visualization of a simulation in real-time or after the simulation has complete via playback.

Summary

Primary author: KENNEALLY, Patrick (University of Colorado, Boulder)

Co-author: Prof. SCHAUB, Hanspeter (University of Colorado, Boulder)

Presenter: KENNEALLY, Patrick (University of Colorado, Boulder)

Session Classification: Verification and Validation Methods #1

Track Classification: 12: Verification and Validation Methods