Modelica Library for Improved Spacecraft Resource Budgeting

Niccolo Cymbalist Marc-Andre Lauriault Chahé Adourian (supervisor)
Space Technologies, Canadian Space Agency
John H. Chapman Space Centre 6767 Route de l'Aéroport Saint-Hubert, QC, Canada

Abstract

The *SpacecraftLib* library has been developed in Modelica for use in the domain of Systems Engineering for space systems with a special emphasis on modularity, usability and ease of modification and expansion. It is a multidisciplinary tool which combines all the relevant subsystems. Power, command and data handling, and mechanical models are integrated into a single Modelica *device* in order to model as completely as possible the behaviour of a physical onboard device. We will describe the tool, examine a case study and briefly analyze the results of a simulation.

Keywords: spacecraft simulation; resource budgeting; systems engineering; command network

1. Introduction

Modeling and simulation tools in Systems Engineering for spacecraft have the potential to improve the efficiency of the design process. One task in which simulation may be effectively utilized is to assist in the generation of requirements through improved resource budgeting. Budgeting in this context is defined as the process of characterizing the components which affect an overall system parameter while focusing on system level requirements and trade-offs [1].
A simulation tool for use in Systems Engineering as a whole, and specifically the resource budgeting exercise, must meet the following requirements:

- **Simple**. The model must have the capability of being rapidly and easily assembled and modified.
- **Multidisciplinary**. The behaviour and interactions of power, command and data handling, mechanical and thermal systems and link behaviour must be modeled together.
- **Appropriate level of detail**. The model must be accurate enough to define requirements.
- **Easily Customizable and Expandable**. The tool must be able to be extended for use in more advanced stages of the design process.

Currently available spacecraft simulation tools are generally either simple spreadsheet based models, mission design tools (*STK*)[2], complex custom built simulators or tools targeted at a specific subsystem, such as Attitude and Orbit Control Systems (AOCS)[3]. While these tools are all useful in their respective domains, they do not meet the needs of a resource budgeting tool for spacecraft Systems Engineering.

This paper introduces a Modelica library specifically designed for resource budgeting, capable of expanding to a full design tool, called *SpacecraftLib*. It can be used to build a ground station and multiple spacecraft models, each including power, payload, command and data handling subsystems and link models. The spacecraft models receive and react to time-tagged commands during the simulation and interact with commercial spacecraft modeling software for advanced functions.

The use of this tool will allow the user to optimize the level of complexity of the simulator at each stage of the design process, especially in the early stages of the design, by increasing or decreasing the complexity of the models built using *SpacecraftLib*. This will lead to a more effective and efficient design process.

This library is implemented in Modelica because we find it well suited to hybrid, multidisciplinary modeling due to its modularity and ease of use.

DOI: 10.3384/ecp09430037

2. The *SpacecraftLib* library

The *SpacecraftLib* library is divided into 4 main sections, each containing components used to model a different subsystem.

These components are easily assembled into *devices*, which behave as physical onboard devices would. They have mass and inertia, consume power, generate data and interact with the user or onboard computer via the command network.

- The *DataBudget* package includes components for processor, data flow, command network and communications equipment modeling.
- The *PowerBudget* package includes components used for power generation and distribution modeling. It also includes components used to model the power consumption characteristics of onboard devices.
- The *Mechanical* package includes a version of the Multibody library expanded to account for advanced gravity and magnetic fields modeling. It includes a modified *world* model, as well as reaction wheel, extendable solar array and thruster assemblies. It also includes controllers for each of the assemblies.
- The *OrbitalMechanics* package includes an ephemerides generator for the nine planets and the Moon based on external ephemeris tables.

We provide the user with an inheritable *device* template with components common to all devices: a command port and command decoder, power consumption characteristics and a state box. To model the behavior of a specific device, the user may *extend* the *device* template to a new class and add the desired components from the available subsystem packages (see Figure 1).

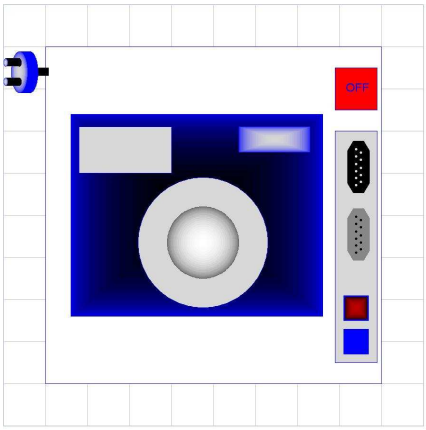


Figure 1. Camera device

The devices are in turn assembled into a complete spacecraft model which is initialized into an orbit. The model exchanges data with the ground station, generates and consumes power and responds to user commands.

SpacecraftLib is complemented by ephemeris tables and solar flux data from *STK* and a commercial spacecraft modeling library called *Spacecraft Control Toolbox*[4]. The latter was converted from MATLAB to C to interface with Modelica.

2.1 *DataBudget* section

SpacercraftLib provides components to model the processor unit(s), the command network, the flow of data between different devices and the ground station, and the communication equipment on the ground and onboard the spacecraft(s). The results of the simulation allow the user to accurately define the requirements of the command and data handling system onboard the spacecraft and on the ground.

2.1.1 Data flow modeling

In *SpacecraftLib* data is treated as a flow, behaving such that it may be generated, deleted, or compressed. Data modeling components include various data links, data storage units, data sources and sinks, and data processing units. These components are purely conceptual, designed to model the behavior of a data handling system instead of modeling the actual physical components of the system.

Data ports are *flow* connectors together with a control line. Data ports may be active or passive.

The active ports define the bit rates (in or out) while the passive ports simply accept this flow.

The passive port may send back information regarding the state of the receiving device via the control line. This allows the active device to stop removing (feeding) data when the receiving device is not able to give (accept) data. Link components are either variable bit rate links with a maximum bit rate set by the user or links that compress or expand the volume of data. The latter are used to model data compression and EDAC (error detection and correction) operations. Switch components include on/off switches and splitters.

2.1.2 Command network modeling

The effort to maximize the usability of the tool led to the creation of a flexible layered command bus with 'plug and play' behavior. It allows for rapid assembly or modification of a model by dropping in a new device, naming it, and connecting it to the bus. The command network model is portable and has potential applications outside the field of spacecraft engineering.

The command network contains two complimentary levels of hierarchy. One level is name based and one level is based on numerical indices. The name based level roughly corresponds to the physical location of the device to which the command is intended for, and is set by the user. The second level is based on numerical indices, and is automatically generated and hidden from the user. This level is used internally to map the network of devices so that each 'parent' device knows the location of all its 'child' devices and is able to forward the command to the appropriate device.

This combination of name and numeric based addressing separates the user from the numerical address system, allowing the user to specify the command destination using only the physical location of the device, as demonstrated in Figure 2.

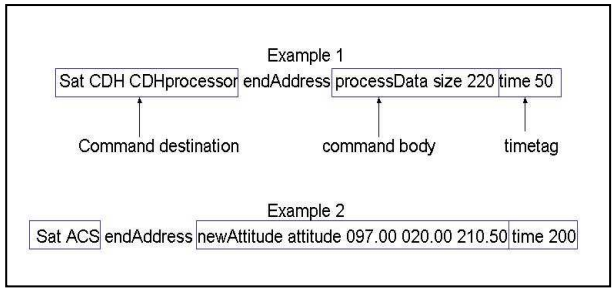
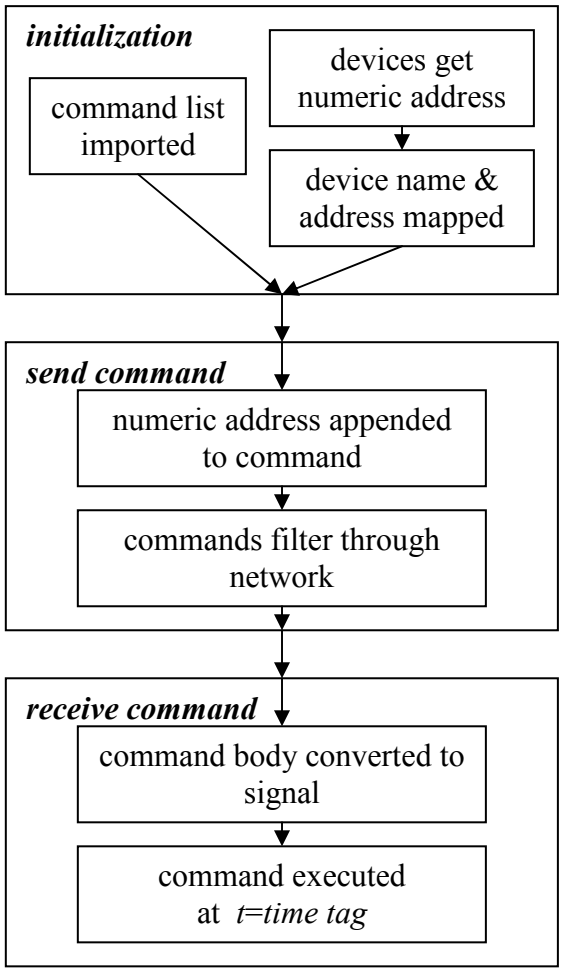


Figure 2. Command syntax examples

In order to achieve 'plug and play ' level usability, the Modelica code is complemented by a custom built C library. The Modelica model (Modelica side) and C library (C side) run concurrently throughout the simulation. Flowchart 1 illustrates the sequence of events in the command network model.



Flowchart 1. Sequence of events in command network model

The command network is composed of four components on the Modelica side (see Figure 3),

including the *commandInput, networkNode, device* and *satControls* components.

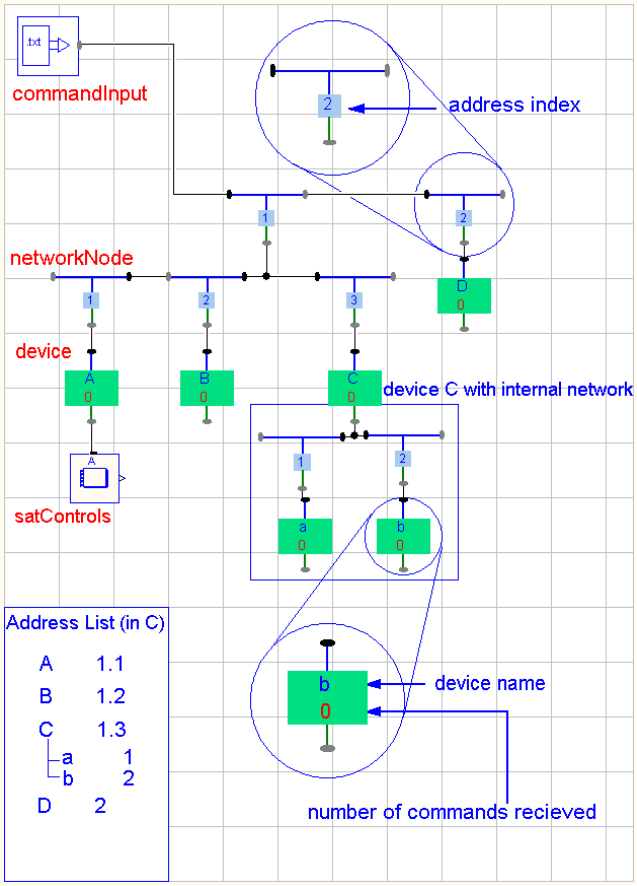


Figure 3. Command network example

1. The *commandInput* component imports the textual commands from the user, parses and translates them and stores them in a buffer. It retrieves the appropriate numerical address for each command from the automatically generated device list on the C side and appends it to the command (see *networkNode* description, next paragraph, for details on the address generation mechanism). *CommandInput* then sends the commands through the network in packets. The size and frequency of the packets are defined by the user.

2. The *networkNode* component is a node in the network between different levels. It contains a user assigned address index which is used to generate the addresses of devices under it (at initialization) and to filter the commands by their address indices (when the commands are sent during the simulation). The Modelica code automatically generates an address at initialization by propagating an empty array from the top to the lower levels of the network. At each node in the network, the empty array gets progressively populated, recording its path through the network from the command source (which may be either the *commandInput* component or a parent *device*) to each device.

3. The *device* component is at the receiving end of the command network. The user assigns a name to the *device* instance as a parameter. Upon initialization, the *device* receives a unique identifier (UID) from the C side and registers itself to the list of devices by passing its name, UID, address and parent device UID (if applicable) back to the C side. When a *device* receives a command or packet of commands, it either passes them on to a sub-device through its internal network or feeds the commands to the *satControls* component.

4. The *satControls* component receives the commands, extracts the time tag, translates them to the appropriate signals and places the commands in a queue. When simulation time matches the time tag the command is executed.

On the C side, there is a unique identifier generator and the address list which contains the device UID, address, name and parent/child relationship with other devices. The C code and Modelica model interface via three *external C* functions.

1. The *registerUID* function is called by the *device* component to obtain a unique identifier from the C side.

2. The *registerDevice* function is called by the *device* component to pass its name, address, UID and parent UID to the C side to be added to the device list (mapped).

3. *getDeviceAddress* is called by the *commandInput* and *device* components, to retrieve the device and sub device addresses, respectively.

2.1.3 Processor modeling

The processor model is used to test sequences of instructions in a process and to evaluate the processing load on the CPU. The main components are the processor model template and the individual process blocks (see Figure 4).

The processor model template is an extendable version of the *device* model, with the addition of data ports and additional parameters to

characterize the processing power and internal bitrates. The user may select from the available process blocks to build a new processor model, or implement custom process blocks. Standard processor options are also available.

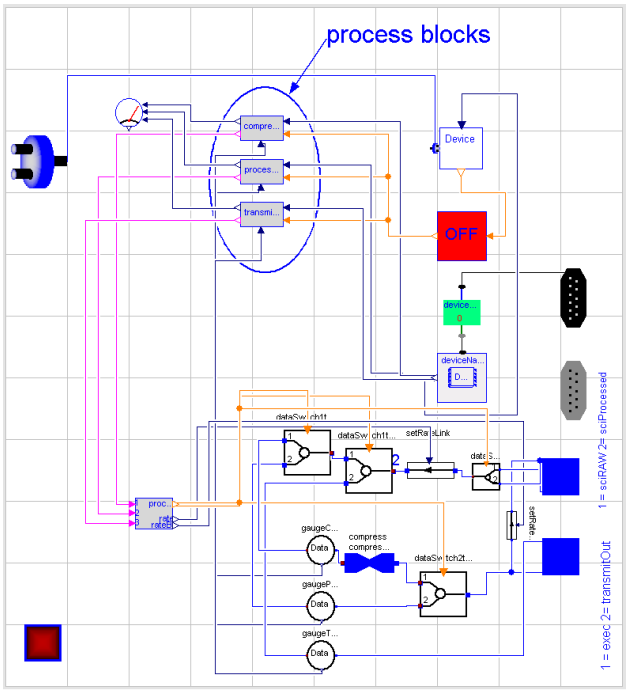


Figure 4. An example of a processor model

Each process block contains a set of instructions which are executed in sequence when the process block is activated. The processor block executes the instruction by sending signals to the 'dumb' components in the processor, such as the data link, compress and switch components.

2.1.4 Communications modeling

The communications modeling components include:

- Onboard communication equipment consisting of customizable TX and RX antennas. User defined parameters include antenna gain, transmission power and frequency.
- Ground station antenna, characterized by its location in latitude and longitude, horizon angle, gain and by the parameters affecting attenuation.
- The link analysis block uses Princeton Satellite functions to calculate the theoretical maximum uplink and downlink bitrates (Shannon limit) based on the ground station and onboard antenna parameters and the location of the spacecraft and ground station.

2.2 *PowerBudget* section

The power budget section contains the components related to the generation, storage, distribution and consumption of power which would be found in a typical photovoltaic cell based power subsystem. Customization is achieved through parameterizing the components and through redeclaration. For instance, the user may use one large solar array at one average temperature or multiple smaller solar arrays of different temperatures to account for temperature differences across a physical solar array.

- The solar array model is parameterized by the number and area of solar cells, maximum power current, maximum power voltage, array temperature and temperature coefficients. It generates power according to the angle of incidence of the panel and the intensity of the solar flux.
- The battery model takes into account the cell capacity, maximum depth of discharge, charge/discharge ratio, and maximum charge and discharge rate.
- The power distribution model interacts with the power consumption block in the *device* model to provide power according to the state of device: on, off or idle. The *device* has parameters to specify the power consumption at each state.

2.3 *Mechanics* section

In order to model the physical characteristics of each device as well as the spacecraft Attitude and Orbit Control System, the standard Modelica Multibody *world* model and *body* model were modified to incorporate more complex gravity and magnetic fields, as well as gravity gradient effects [5]. Various actuators including reaction wheels, thrusters, torque rods and their respective controllers are included in the *Mechanics* section.

2.3.1 *world* model

The extended *world* model includes the option to use a spherical harmonics gravity model instead of

point gravity. This component models the Earths magnetic field using the International Geomagnetic Reference Field (IGRF) and incorporates a default spherical earth visualization where the radius is set to the average Earth radius. The user may customize the precision of the gravity and magnetic field models by specifying the number of coefficients used for the gravity and magnetic field polynomials. The *world* model detects when the spacecraft comes in contact with the earth surface.

2.3.2 *complexBody* model

This model is derived from the standard *body* model, modified to allow the user to account for gravity gradient torque, if desired. The gravity gradient calculation function is in the *world* model.

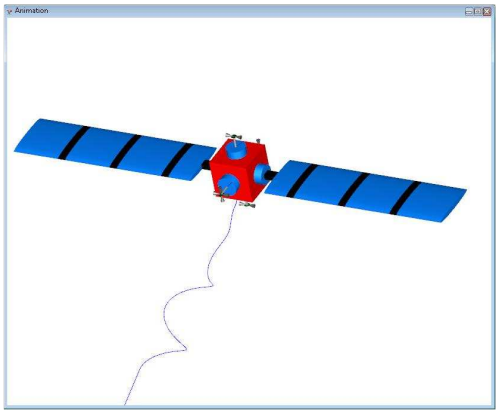


Figure 5. *Multibody* spacecraft visualization

Available actuators include reaction wheels, thrusters and magnetic torque rods. All are assembled from the standard Multibody library components and the modified *complexBody* model.

- Reaction wheels are generally mounted on each of the 3 axes. One spare wheel is mounted obliquely and ready to take over in case one of the 3 primary wheels fails. In order to accurately model any unique reaction wheel assembly, the user can specify the orientation of each wheel, shape, size and density of the rotor.
- Thrusters are modeled by exerting a frame force on the thruster nozzle. The user specifies the mass of the thruster assembly and the maximum force exerted by the thruster, as well as the position and orientation of the thruster nozzle.
- Magnetic torque rods are modeled by exerting a torque on the torque rod body. The component *magneticFieldSensor* is used to measure the magnitude and orientation of the magnetic field in order to apply the appropriate torque.

There are controller blocks for performing operations including detumbling (stabilizing the spacecraft into a certain attitude after separation from the launcher, see Figure 5) and momentum unloading (using the torque rods to decrease the momentum which has accumulated in the reaction wheels), as well as routine changes in attitude and orbit.

2.4 Orbital mechanics

The orbital mechanics section includes a precise ephemeredes model based on tables generated by *STK*. It generates the position in heliocentric equatorial coordinates for the nine planets and the Moon by interpolating the ephemeris tables using a Lagrange polynomial expansion.

3. Case study

To verify the behavior of *SpacecraftLib* , we built a generic spacecraft (called *Sat,* see Figure 6) with all the major subsystems. These include command and data handling (C&DH), power control and distribution unit (PCDU), Tracking, Telemetry and Control (TTC), Attitude Control System (ACS) and a payload consisting of 2 sensors. We orbited the spacecraft for 2 orbits, and uploaded a list of commands to be performed over the duration of its short ‘mission’.

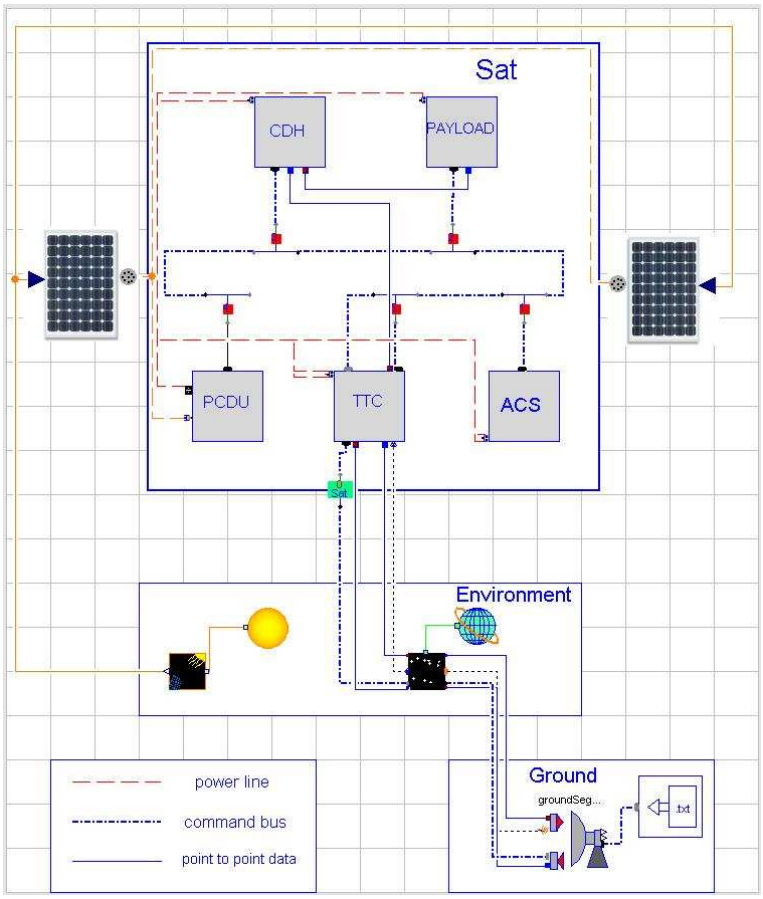


Figure 6. Model of complete spacecraft *'Sat'* and ground station

- C&DH. This subsystem is composed of 1 processor with a maximum processing power of 100 KIPS and internal bitrate of 10 Mbps, a science data memory unit, and an execution memory unit.
- PCDU. This subsystem includes the power controller and a battery, and is connected to 2 solar arrays.
- TTC. This subsystem includes a processor, buffer memory, a transmitting antenna and a receiving antenna.
- ACS. For this example, the attitude control system is a dummy system to model power consumption. Multibody components are not used and the orbital parameters for the duration of the simulation are imported from STK.

The ground station is located in St Hubert, Quebec, at the location of the Canadian Space Agency while the spacecraft orbits at an altitude of approximately 820 km in a near polar, circular orbit, with an inclination of 98.7 degrees.

We entered the commands into a .txt file before compiling the model and ran the simulation for 12000 seconds (about 2 orbits).

The commands included image capture and compression operations, device state changes (on, off and idle), attitude changes and data transmission operations. The model reacted to these commands at the appropriate times, exhibiting behaviour which would allow us to optimize the design.

3.1 Results and Analysis

We will examine the behaviour of the C&DH subsystem as well as the link behaviour.

In Figure 7, the upper plot shows the amount of data in the memory space allocated to unprocessed science data. Each spike corresponds to an unprocessed image entering the raw data memory, and immediately being removed, processed and placed in the memory allocated to processed data (not plotted).

The lower plot in Figure 7 shows the status of the 'shutter' on the camera sensors. At each pulse, the 'camera' takes an image and generates a certain amount of data, which is passed to the raw data memory.

In Figure 8, the upper plot shows the maximum theoretical bitrate possible according to the Shannon-Hartley theorem[1] and the actual transmission bit rate. The lower plot shows the amount of data in the TTC memory (buffer). As soon as there is a link, the data in the TTC memory is transmitted.

Certain facts which have an impact on the design are immediately evident. From the upper plot in Figure 7, we can conclude that for this operating profile, there must be at least 350 to 400 Mb of storage allocated to the raw data from the sensors.

From Figure 8 we see that the communications system is not optimized for this operating profile. At a downlink bitrate of 10 Mbps it takes only a fraction of the available link duration to download all the data. This is an indication that communication equipment capable of 10 Mbps downlink speeds is not necessary for this imaging rate, which will have secondary and tertiary effects on the power subsystem specifications, solar array size, and eventually on the total mass of the spacecraft. Moreover, the closer we get to the Shannon limit of the channel the more we must invest in terms of hardware and power for advanced coding techniques.

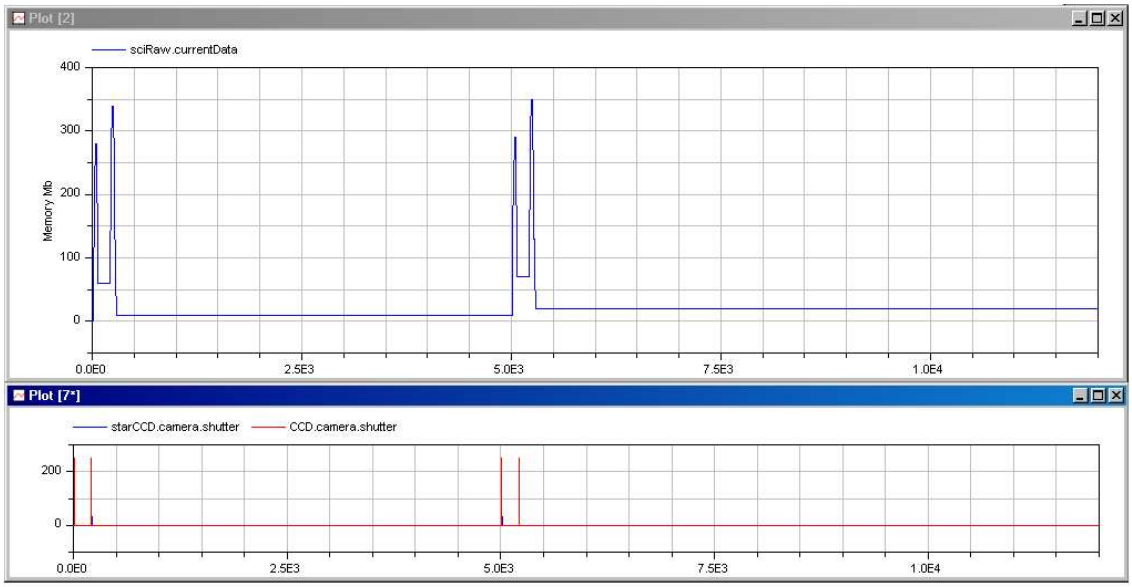


Figure 7. Science memory level (upper), camera state (lower)

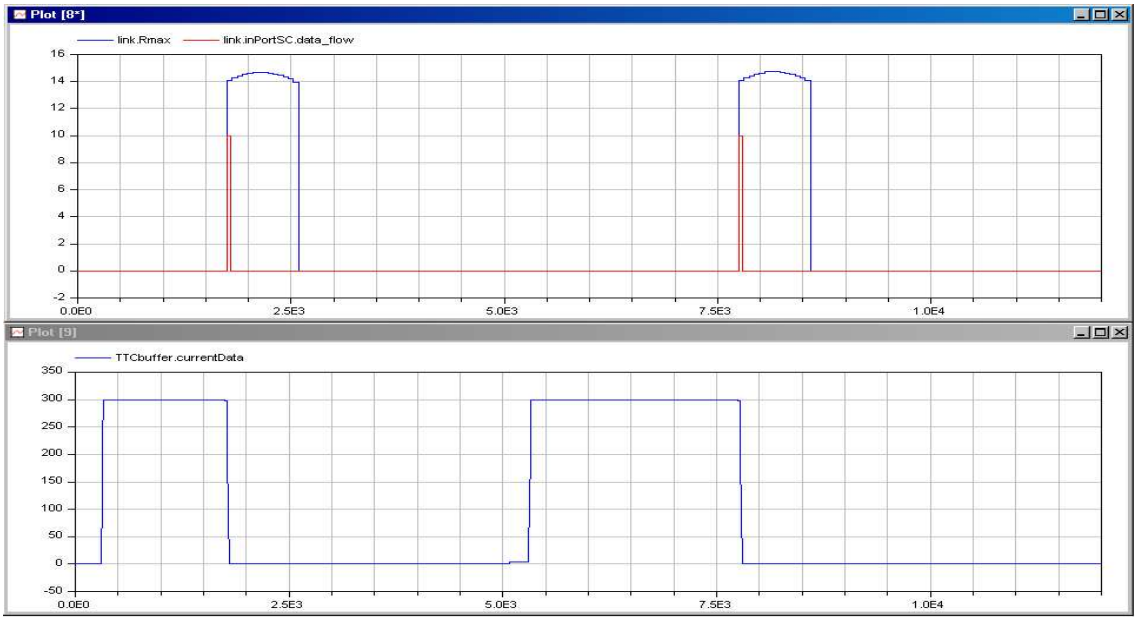


Figure 8. Maximum theoretical bitrate and actual bitrate (upper), TTC buffer level (lower)

4. Conclusion

Modelica is proving to be well suited to hybrid modeling of power, data and command systems as applied to a generic spacecraft modeling tool for Systems Engineering, when complemented by our C library. It provides a flexible graphical modeling environment that allows for fast and accurate estimates for use in resource budgeting. The precision of the estimates depends on the fidelity of the subsystem models, so further improvement of the subsystem models will allow the tool to progress to a full design tool. Various features of the Spacecraft Budgeting Library, notably the command network model, are portable and may be reused for other applications.

References:

[1] W.J. Larson, and J.R.Wertz, editors. *Space mission analysis and design*. Microcosm Press/Kluwer Academic Publishers, 1999.

[2] *STK 8*, Analytical Graphics, Inc.

[3] T. Pulecchi, F. Casella, M. Lovera. A Modelica Library for Space Flight Dynamics. In *Proceedings of the 5th Modelica Conference, Vienna, Austria,* volume 1, page 107.

[4] *Spacecraft Control Toolbox,* Princeton Satellite Systems, Inc.

[5] Built by Dyna Benchergui and Andre-Claude Gendron

