The Deployable Structures Library

Cory Rupp   Laura Schweizer

ATA Engineering, Inc., USA, {cory.rupp, laura.schweizer}@ata-e.com

Abstract

Deployable structures are an enabling technology for many space- and ground-based structures and vehicles. Analysis of deployment mechanisms and structural dynamic responses in early design phases is key to ensuring deployment reliability and overall structural integrity. In this paper, a Modelica library is presented that provides a number of building blocks to enable and ease the development of models of deployable structures. Several examples using the library are presented that would be difficult or impossible to model using other technologies.

Keywords: Modelica, deployable structures, flexible structures, spacecraft, solar array

1 Introduction

Accurate modeling of multibody kinematics and dynamics is critical for design and analysis of deployable structures and mechanisms. This is particularly true for expensive, highly engineered space-based structures such as solar arrays, antennas, and booms where deployment failure results in mission failure. Not only is deployment the number one risk to these systems, they also must meet stringent mass and stiffness requirements that make structural dynamics responses an important consideration in preliminary design phases.

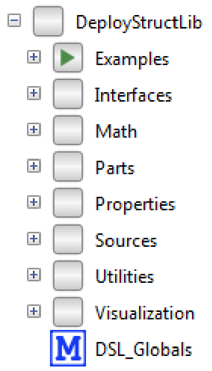
Existing software tools for performing multibody simulations are often domain-specific and limited in extensibility. The primary tool used in the aerospace industry is ADAMS, a proprietary software package provided by MSC. While ADAMS is primarily geared toward multibody analysis, it can be extended by integration with Simulink and some other domain-specific tools. Although possible, implementing multiphysics effects or specialized mechanisms is far from trivial and largely beyond the intended scope of the ADAMS toolset. On the other hand, Modelica, specifically the MultiBody library of the Modelica Standard Library (MSL), has far more freedom for the user to add new capabilities for analysis of deployable structures. Even so, the MSL is limited in the range of structures it can model.

This paper presents a new Modelica library, the Deployable Structures Library (DSL), which provides specialized structural and mechanism components that are useful for modeling deployable structures. This library is largely an extension of the MSL MultiBody library to expand its applicability. In addition to new modeling capabilities, built into the library is a modeling workflow that more closely follows the typical structural engineering design and analysis process. With the library being tailored to the engineering process, it is hoped that structural engineers will be able to more readily adopt the library and Modelica as a modeling tool in general.

2 The Deployable Structures Library

The Deployable Structures Library contains a mix of new modeling capabilities and reformulations or extensions of existing MSL blocks to provide a toolset for structural design and analysis engineers. The library enables the analysis of many space-based deployable structures and other difficult-to-analyze spacecraft components, as well as ground-based structures. Many of these systems include complex, one-off mechanisms to actuate their movement. As such, it is imperative to have a modeling technology such as Modelica that is extensible and can be used to model the mechanism's behavior at a fundamental level. The goal of the library is to make building models of these mechanisms easier through use of a set of common building blocks.

The DSL is organized into several packages, somewhat mimicking the organization of the MSL MultiBody library. At the top level are the Examples, Interfaces, Math, Parts, Properties, Utilities, and Visualization packages, as shown in Figure 1. The most relevant of these are described in the following sections.



DeployStructLib
- Examples
- Interfaces
- Math
- Parts
- Properties
- Sources
- Utilities
- Visualization
- DSL_Globals

Figure 1. Top-level view of the Deployable Structures Library.

2.1 DSL_Globals

At the top level of the library is the *DSL_Globals* block. This block is used as a Modelica inner model, holding global parameters that enable specialized structural analyses, including steady-state and quasi-static analysis, to be performed. Components in the Parts package reference this block with Modelica's outer construct to change their set of equations and model different behavior. For example, the quasiStaticFactor parameter will scale the mass values of all DSL components, typically using a construct such as

```
parameter SI.Mass mass =
  if DSglb.quasiStatic then
    DSglb.quasiStaticFactor*rho*xprop.A*L
  else
    rho*xprop.A*L "Beam mass";
```

What this does is reduce the inertia consistently across the entire simulation, thereby reducing the problem to a quasi-static problem. It is equivalent to pushing the mass matrix toward zero (i.e., $M \rightarrow 0$) in the standard dynamics equation

$$M\ddot{u} + C\dot{u} + Ku = f \qquad (1)$$

leading to (assuming also $C \rightarrow 0$) the static equation

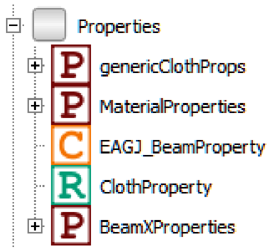
$$Ku = f \qquad (2)$$

With this change, time-varying static structural responses can be evaluated using a transient solver in a Modelica vendor-independent manner.

2.2 Properties

To help reduce the barrier to adoption and implementation of Modelica for engineers, much of the library has been designed with the structural engineering workflow in mind. As an example, typical structural engineering analysis software is organized with material and structural properties defined separately from the actual structure model. As such, a single property only needs to be defined once, thereby preventing model bloat and modeling error. This concept is implemented in the Properties package in a natural way through Modelica record parameters, which are used by structural elements in the library.

Within the Properties package are subpackages of records that can be used for material property, beam cross section, and cloth property definitions (Figure 2). When a model is created, these records can be implemented at the top level as parameters that are passed down through the model to individual Deployable Structures Library components. This workflow is similar to modeling procedures used by other structural engineering tools such as finite element software packages where elements are given a material property identifier that corresponds to a single property definition. With such a process, fewer mistakes are made because there are fewer opportunities for error. This implementation in Modelica has the added benefit that design studies with material, beam cross section, or cloth properties can be performed easily.

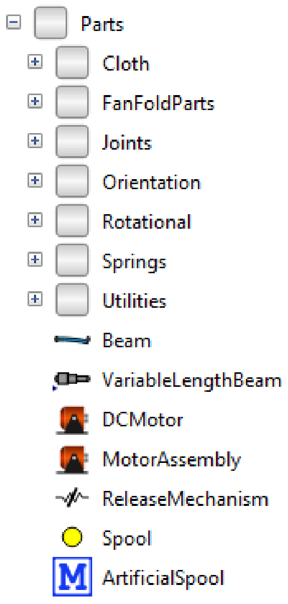


Properties
- genericClothProps
- MaterialProperties
- EAGJ_BeamProperty
- ClothProperty
- BeamXProperties

Figure 2. The Properties subpackage of the DSL.

2.3 Parts

The Parts package contains a plethora of new and extended components and mechanisms that can be used to model deployable structures (Figure 3). The most relevant and broadly applicable of these are discussed in the following sections.



Parts
- Cloth
- FanFoldParts
- Joints
- Orientation
- Rotational
- Springs
- Utilities
- Beam
- VariableLengthBeam
- DCMotor
- MotorAssembly
- ReleaseMechanism
- Spool
- ArtificialSpool

Figure 3. The Parts subpackage of the DSL.

2.3.1 Beams

It is virtually impossible to model deployable structures without some kind of beam model. While the MultiBody package in the MSL contains rigid components that can be used to model rigid beams (namely, the *BodyBox* block), changing the beam cross section requires explicit (and error-prone) parameterization, and there is no means to accurately model beam flexibility. Most deployable structures, especially space-based structures, contain lightweight and thin beam members, making it absolutely necessary for beam cross sections to be designed and beam flexibility included in models, as beam deformation dramatically affects overall dynamic motion and response.

To make it easier to model thin structural members, the Parts package within the DSL contains a *Beam* block that acts as a wrapper for both rigid and flexible beam models. The block requires appropriate material and cross-sectional properties as parameter inputs, which are defined through blocks in the DSL.Properties subpackage. The material properties are provided to the *Beam* block through a *MaterialProperties* record, which allows the user to define independent materials and pass them to individual beams. The beam cross-sectional properties can be passed to the *Beam* block either by setting the dimensions of the cross section in a *BeamXProperties* record and passing that record to the beam or by setting combined properties such as the bending rigidity EI and the torsional rigidity GJ in an *EAGJ_BeamProperty* block and passing these to the beam. In the case of the *BeamXProperties* records, several predefined cross-sectional shapes are provided in which cross-sectional moments of inertia are automatically computed when the model is built. Beam materials are currently assumed to be isotropic, and cross-sections are assumed to be constant along the length.

Whether a beam is modeled as flexible or rigid is determined by setting the value of the Boolean *rigid* parameter. This parameter selects between the *FlexBeam* and *RigidBeam* blocks described next. The beam model itself is essentially a dummy model layer that implements either a *FlexBeam* or *RigidBeam* depending on the value of the rigid Boolean parameter. The Modelica code for this implementation layer looks like this:

```
parameter Boolean rigid = false;
RigidBeam beamR(L=L, xprop=xprop, …) if
  rigid;
FlexBeam beamF(L=L, xprop=xprop, …) if not
  rigid;
equation
if rigid then
  connect(beamR.frame_a, frame_a);
  connect(beamR.frame_b, frame_b);
else
  connect(beamF.frame_a, frame_a);
  connect(beamF.frame_b, frame_b);
end if;
```

There is no requirement that all beams in a model be flexible or all be rigid since the flag is set at the level of the individual beam. However, efficiencies can be gained when creating and debugging a model by first using rigid beams and then later switching to a flexible model through the use of a single top-level parameter tied to all the beam *rigid* parameters.

2.3.2 Flexible Beam

The *FlexBeam* block provides a Bernoulli-Euler model of a beam. The formulation of this block follows that of Schiavo *et al* (2006), with some modifications to fit into the overall scheme of the DSL and without the internal element discretization (i.e., the block consists of a single finite element). In particular, the beam cross-sectional and material properties are passed in as parameter blocks from the DeployStructLib.Properties subpackage. While not as general as the DLR FlexibleBodies library (2006) or that described by Ferretti *et al* (2014), this Modelica-based flexible beam model is intended to provide basic functionality when flexibility is a concern in the analysis of deployable structures.

The key assumption of Bernoulli-Euler beams is that plane sections that are normal to the neutral axis of the beam remain plane and normal to the axis after bending. This implies that transverse shear effects are neglected in forming the stiffness matrix; thus, this block is primarily intended for modeling slender beams.

The mass matrix in the flexible beam can be computed either as a lumped mass matrix, where all mass is lumped into the beam degrees of freedom, or as a consistent mass matrix following typical finite element procedures through the Boolean parameter *useLumpedMassMatrix*. In structural dynamics analysis, the formulation of the mass matrix is often an important consideration for accuracy, and often the lumped mass matrix is used. The lumped mass matrix for the particular finite element formulation used here, however, is singular, as the inertia of the bending degrees of freedom lumps to zero. This leads to singular matrix errors during solving, for which the solution is to add an MSL *Body* block to the beam tip with a small mass value, which in effect adds rotational inertia terms to the beam bending degrees of freedom, thereby rectifying the singular matrix issue. Because this modeling caveat is a nonstandard, non-intuitive procedure, the *useLumpedMassMatrix* parameter is false by default so that general users can use the flexible beam block and need not understand the nuances of finite element mass matrices. Corresponding clarification on the use of the *useLumpedMassMatrix* parameter is provided in the *Beam* block documentation.

Structural damping in the beam is computed using Rayleigh damping; i.e., the damping is a weighted linear combination of the stiffness and mass matrices.

A second flexible beam block, *FlexBeamEAGJ*, is also provided. This block uses the same formulation as the *FlexBeam*, but it allows the user to specify beam properties such as the bending rigidity EI, rather than separately specifying the material and cross-sectional properties. This is useful for correlating models to test data, especially when the beam under consideration has a complicated cross section, is a compound beam, or is built from composite materials.

2.3.3 Rigid Beam

The *RigidBeam* block ignores the stiffness of the beam and assumes complete rigidity. It is based on the *BodyBox* block in the MultiBody library of the MSL but includes inertial calculations from material and cross-section definitions using the same DSL.Properties blocks as used in the *FlexBeam* model.

2.3.4 Variable Length Beam

The *VariableLengthBeam* block provides a flexible beam model that can change in length over time and simultaneously update its stiffness and inertia properties. At each time step, the block recalculates the inertia tensor of the beam, correctly accommodating large changes in beam length. The cross-section of the beam is assumed to be constant, so only the change in length is included when the stiffness and inertia are recalculated. The formulation is the same as that of the *FlexBeam* block but with the addition that the length of the beam is defined by an initial length parameter and a length rate of change input provided by a *VariableLengthSource* block (located in the Utilities subpackage of the library) or a standard MSL source block. The Modelica implementation of the variable length is implemented as

```
parameter Real L_start = 1.0 "Start value
  for L";
Real L(start=L_start, fixed=true) "Beam
  length";
Real dL "dL/dt";
Modelica.Blocks.Interfaces.RealInput
  dL_in;
equation
  dL = der(L);
  dL = dL_in;
```

from which downstream variables such as the beam mass, mass and stiffness matrices, and rigid body inertia matrices are no longer parameters but real variables that vary throughout the simulation.

It is important to note that additional terms attributed to time derivatives of inertia, mass and stiffness matrices, and other typically constant parameters are neglected. As such, this simple (and perhaps simplistic) implementation is restricted to analyses where the change in length of the beam is slow. This rate of change restriction can be generalized by ensuring that the rate of change of structural natural frequencies is much smaller than the natural frequencies themselves (i.e., the system parameters could be described as quasi-static). It can be argued that this assumption is valid for the vast majority of deployable structures, in particular space-based structures, as the deployment process often occurs over the course of several minutes to keep structural loads low. The *VariableLengthBeam* model, however, should be used with care and could potentially be extended to more general cases through implementation of missing terms via derivations such as those provided in Yang *et al* (2017).

The *VariableLengthSource* block used as input to the *VariableLengthBeam* block is primarily for convenience and outputs a user-defined change in length per unit time for a given beam deployment sequence. The rate of change is a parameter, so it is constant throughout the simulation. The user can also set a minimum length and a maximum length so that the beam length stops at a set deployed distance.

2.3.5 Weak Joints

The Joints subpackage contains two "weak" joint models: a *WeakSpherical* joint and a *WeakRevolute* joint. Most joints in the MSL use strong constraints, e.g., the positions of frame_a and frame_b must be exactly equal. Many deployable structures have kinematically complicated systems of joints that may ultimately connect in a kinematic loop. Handling of kinematic loops is a Modelica vendor-dependent operation, so use of a weak formulation overcomes these difficulties on the Modelica side by allowing the user to break the kinematic loop in a specific (and perhaps more desirable) location.

In the weak joint formulation, frame_a and frame_b are essentially connected by springs and dampers, the stiffness and damping of which are parameters that the user sets. The joints use a single value of stiffness and damping for all three translation and rotation directions.

2.3.6 Rotational

The Rotational package of the DSL contains rotational hard stops and rotational locks, both of which are common components in deployable systems.

There are two rotational stop models in the DSL: *RotationalStop* and *DampedRotationalStop*. These two blocks use linear 1D rotational springs to model the stiffness of a hard stop. When the hard stop is not engaged, frame_b of the block is allowed to rotate freely with respect to frame_a, and no torque is generated. When frame_b moves past the given stop angle in the direction opposite free-play, the rotational spring engages and acts to stop the motion of frame_b. The *DampedRotationalStop* block includes a rotational damper in parallel with the rotational spring. Both the damping coefficient and the spring stiffness are set by the user. The rotational stop models do not prevent frame_b bouncing against the stop. That is, if frame_b hits the stop, it can rebound, undamped and unrestricted, in the free-play direction, whether damping is included or not.

The Rotational package also includes two rotational lock models: *RotationalLock* and *DampedRotationalLock*. Both of these blocks monitor the angle of frame_b with respect to frame_a. Once that angle moves past the user-specified locking angle, a linear 1D spring engages to prevent further rotation in the free-play direction. The stiffness of the spring is a parameter set by the user. The *DampedRotationalLock* model includes damping in addition to the spring stiffness.

2.3.7 Springs

The Springs package contains several 3D linear springs that are useful for modeling deployable structures. We highlight several important blocks here.

The translational complement of the *RotationalStop* block is the *BarrierSpring* block. The *BarrierSpring* block provides a linear spring that acts only in compression. The length of the spring is computed as the distance between frame_a and frame_b in a user-specified direction that is normal to the plane of the hard-stop. Thus, as the spring length in the direction normal to the hard stop shrinks, the compressive force resisting that motion increases, effectively preventing frame_b from colliding with or passing frame_a. Since the spring does not act in tension, however, there is no resistance to frame_b bouncing against the hard stop in the extensional direction. The *BarrierSpring* block also includes damping; the use of damping in the model can be turned on by setting a nonzero damping coefficient. The spring stiffness, as with the rotational block, is also a user-specified parameter.

Many deployable structures are also tensioned structures, so the *TensionSpring* block is provided for modeling this type of behavior. This block reformulates the MSL *MultiBody.Forces.Spring* block with a *semiLinear* stiffness function to provide stiffness only when the spring is in tension. The *CompressionSpring* block is the compression-only complement to this block.

Deployable structures often contain some mechanism to release the structure from its undeployed configuration and begin the deployment process. The *ReleaseMechanism* block in the Springs package provides a method of modeling the release of the stiffness that holds the structure in place, and it consists of a 3D linear tension spring. When the spring is engaged, it acts to pull frame_a and frame_b together. A Boolean *released* variable can be set to *true* by the simulation at any time, at which point the mechanism will release, disengaging the spring. To reengage the spring, the *released* flag is set to *false*. Damping can be optionally specified by setting the damping coefficient to a nonzero value, which can be helpful for minimizing chatter in complex deployment mechanisms. Like the spring stiffness, the damping force is only active when the mechanism is not released.

2.3.8 Other Modeling Components

Several other modeling components are provided in the DSL that help in the modeling effort of deployable structures but are not significant enough for detailed description. Among these is a *Spool* block that enables modeling of spooled lanyards or similar gradual release mechanisms, a *GravityRamp* function for use with *MultiBody.World* that aids in quasi-static analysis by slowly ramping up the gravitational acceleration over a period of time, and a series of predefined *Orient* blocks that are simplified versions of the MSL *MultiBody.Parts.FixedRotation* block with preset angles and rotation directions.

2.4 Cloth

To achieve high specific power (the ratio of generating power to mass), space-based solar arrays often utilize solar blankets in which solar cells are affixed to a lightweight flexible fabric or cloth substrate. Because of their ability to easily fold into a small volume, the solar blankets also provide a high ratio of power to stowed volume, another important metric for spacecraft design. The downside to solar blankets is that they are very flexible and thus require complex structural mechanisms deploy and tension them. Naturally, their flexibility also introduces significant kinematic and dynamic behavior during the deployment process that can potentially damage the spacecraft, the deployment mechanism, or the solar blanket itself. One way this process can go wrong was illustrated when the solar blanket on the International Space Station (ISS) tore during deployment (Wright, 2007).

To model the behavior of solar blankets and other clothlike structures, the DSL provides the *Cloth* package and corresponding modeling block (Figure 4). The *Cloth* block is a high-level building block for creating a discretized mesh of specially formulated membrane finite elements on a defined surface geometry. The membrane elements have displacement degrees of freedom only, so bending moments are ignored in the formulation, which is a suitable assumption for most fabrics. Each membrane element node is connected to a new Modelica *Location* connector consisting of only a 3D position in space and corresponding cut forces for flow variables. The Location connector is defined as:

```
connector Location "Location of the
  component with one cut-force"
  SI.Position r_0[3] "Position vector from
    world frame to the connector frame
    origin, resolved in world frame";
  flow SI.Force f[3] "Cut-force resolved
    in world frame”
end Location;
```

The elements in the *Cloth* block are connected to each other by connecting their *Location* interfaces. A lumped mass approach is taken to model the fabric mass distribution whereby point masses with *Location* interfaces are connected to the *Cloth* element nodes.

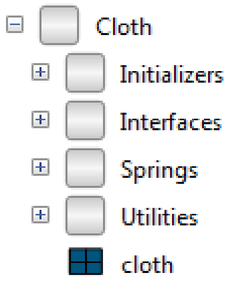


Figure 4. The Cloth package of the DSL.

The formulation of the cloth membrane finite element uses the relative displacement of nodes measured along the element edge as the primary elemental degrees of freedom $\boldsymbol{q}$. In the Modelica model, this is implemented as

```
equation
  d12 = location[1].r_0 - location[2].r_0;
  d23 = location[2].r_0 - location[3].r_0;
  d31 = location[3].r_0 - location[1].r_0;
  q = {Vectors.length(d23)-L[1],
       Vectors.length(d31)-L[2],
       Vectors.length(d12)-L[3]};
```

where d12 are relative displacement vectors between nodes i and j and where L_i are the undeformed edge lengths. Elemental strains along an edge, the so-called natural strains $\boldsymbol{\epsilon}$, can be calculated as

$$\boldsymbol{\epsilon} = \begin{bmatrix}\epsilon_1\\ \epsilon_2\\ \epsilon_3\end{bmatrix} = \begin{bmatrix}1/L_1 & 0 & 0\\ 0 & 1/L_2 & 0\\ 0 & 0 & 1/L_3\end{bmatrix}\begin{bmatrix}q_1\\ q_2\\ q_3\end{bmatrix} = \boldsymbol{B}_q\boldsymbol{q} \qquad (3)$$

which are merely a transformation of Cartesian strains $\boldsymbol{e}$ onto the element edges (Fellipa, 2017) assuming a constant strain triangle element, i.e.,

$$\boldsymbol{\epsilon} = \begin{bmatrix}\epsilon_1\\ \epsilon_2\\ \epsilon_3\end{bmatrix} = \begin{bmatrix}c_1^2 & s_1^2 & s_1c_1\\ c_2^2 & s_2^2 & s_2c_2\\ c_3^2 & s_3^2 & s_3c_3\end{bmatrix}\begin{bmatrix}e_{xx}\\ e_{yy}\\ 2e_{xy}\end{bmatrix} = \boldsymbol{T}_e^{-1}\boldsymbol{e} \qquad (4)$$

where s_i and c_i are direction sines and cosines of the element edges in the undeformed state. The Cartesian strains are then related to Cartesian stresses via the constitutive equation:

$$\sigma = \boldsymbol{Ce}. \qquad (5)$$

Variational analysis with strain energy (details not shown here for brevity) then leads to

$$\delta U = \frac{1}{2}(\delta\boldsymbol{e})^T\boldsymbol{Ce} = \frac{1}{2}(\delta\boldsymbol{\epsilon})^T\boldsymbol{C}_n\boldsymbol{\epsilon} \qquad (6)$$

where

$$\boldsymbol{C}_n = \tfrac{1}{2}\boldsymbol{T}_e^T\boldsymbol{C}\boldsymbol{T}_e. \qquad (7)$$

Substitution of (3) into (6) and adding in the contribution of work due to external forces leads to

$$\delta\Pi = \delta U - \delta W = \frac{1}{2}(\delta\boldsymbol{q})^T\boldsymbol{B}_q^T\boldsymbol{C}_n\boldsymbol{B}_q\boldsymbol{q} - (\delta\boldsymbol{q})^T\boldsymbol{f}_q \qquad (8)$$

from which, after integration over the element volume V, one eventually finds the simple matrix relationship

$$\boldsymbol{K}_q\boldsymbol{q} = \boldsymbol{f}_q \qquad (9)$$

where

$$\boldsymbol{K}_q = V\boldsymbol{B}_q^T\boldsymbol{C}_n\boldsymbol{B}_q \qquad (10)$$

is the element stiffness matrix and $\boldsymbol{f}_q$ are covariant nodal forces along the element edges that are summed at each node. In Modelica, these equations are written as

```
equation
  Kq * q = fq;
  -location[1].f =
    fq[2]*Vectors.normalize(d31) -
    fq[3]*Vectors.normalize(d12);
  -location[2].f =
    fq[3]*Vectors.normalize(d12) -
    fq[1]*Vectors.normalize(d23);
  -location[3].f =
    fq[1]*Vectors.normalize(d23) -
    fq[2]*Vectors.normalize(d31);
```

The primary advantage of this element formulation is that because the primary variables $\boldsymbol{q}$ are formed as relative displacements, the stiffness matrix $\boldsymbol{K}_q$ is constant under any translation or rotation (i.e., during the entire simulation). This makes the Modelica implementation particularly simple and efficient because the stiffness matrix is computed as a parameter while relative displacements $\boldsymbol{q}$ are readily calculated from the *Location* connectors of the element. The simplicity of the approach is particularly apparent in light of the complexity that would be necessary for implementing a total Lagrangian, corotational, or other geometrically nonlinear finite element formulation in Modelica.

Quadrilateral elements are formulated by overlaying two pairs of triangle elements such that the common edge of a pair crosses that of the other pair. In this case, each element has half the prescribed thickness. This is a not uncommon technique for creating quadrilateral membrane elements. Further details about the methodology used to develop the finite elements used in the *Cloth* block are discussed in a forthcoming paper (Rupp, 2018).

Discretization of the *Cloth* block is performed in Modelica by defining the locations of points on a quadrilateral patch and the number of divisions along two adjacent edges of the patch. A Coons patch (a simple bilinear interpolation approach) is then used to define the individual elements. Stiffness matrices of each element are then computed for the undeformed (stress-free) state of each element in the patch as part of the Modelica parameter evaluation procedure. A similar procedure is used to set the mass value of each lumped mass attached to the nodes, which replaces the traditional mass matrix for the element. The conversion of the mass matrix into discrete lumped masses is performed so that inertial calculations can be easily performed in the inertial (i.e., world) frame as opposed to the local frame used to perform elemental stiffness calculations. Initialization of the *Cloth* block is performed by defining parameterized initial locations of the four corners of the quadrilateral patch. In this way, the cloth can be pre-tensioned during model initialization. This mechanism also allows a folded initial blanket state to be defined, which is a common situation for many space-based solar array blankets.

The *Cloth* block is highly parameterized to allow for changes in material properties (defined via the *Properties.ClothProperty* record), thickness, undeformed shape, and folding configuration. Allowing for this design flexibility and setting up the corresponding initialization of the problem is the most complicated aspect of the *Cloth* block, as changing any one of them will affect the entire cloth formulation. To facilitate this process, several initializer functions have been created that set the cloth stiffness and mass matrices as well as interface positions in space based on various input configuration parameters. The initializer functions can handle several different cloth configurations such as z-folds, no folds, triangle and quadrilateral discretizations, and different boundary conditions.

To improve the performance and relieve a bit of processing effort from the Modelica compiler, the element mass and stiffness matrix generation, as well as the initial placement of nodes for the *Cloth* block, is performed via a library of external "C" functions. Modelica interfaces to these functions are found in the *Initializers* subpackage.

The *Cloth* package provides a powerful modeling capability for many structures that would otherwise be difficult to model. The block has been validated through comparison to test data gathered during deployment testing of a state-of-the-art solar array (Rupp *et al*, 2016) as well as via several numerical and analytical studies.

3 Examples

We now present three examples of deployable structures that make use of the Deployable Structures Library. Each example uses the *Cloth* modeling block—meaning that they cannot be modeled using multibody dynamics simulation codes (non-finite-element-analysis based) other than Modelica or using only the MSL. Further examples can be found in Rupp *et al* (2016).

3.1 Linear Deployment Solar Array

Space-based solar arrays are ubiquitous deployable structures used in the aerospace industry as the primary source of power for satellites, space stations, space-based telescopes, etc. A common style of solar array design is the linear deployment array in which a central boom or mast is used to tension a solar blanket between two cross-bars. An example is the solar array design used onboard the ISS. These arrays are known for their compact stowage volume and high specific power.

The central mast used in linear solar arrays is the primary deployment mechanism for the array and can take many forms. The ISS arrays use a foldable truss design (Knight *et al*, 2012), the compact telescoping array concept uses a telescoping boom consisting of concentric tubes that are simultaneously moved relative to each other via a motorized mechanism (Mikulas *et al*, 2015), and the recently developed MegaROSA design uses roll-out booms (Hoang *et al*, 2016). To characterize these different types of arrays without the need to consider specific deployment mechanisms, NASA developed the Government Reference Array (GRA) as a reference design for studying the scalability of these and other high-specific-power solar array designs (Pappa *et al*, 2013).

To study this type of design, a Modelica model of the GRA was created in which the central mast was modeled using the DSL *VariableLengthBeam* block. The cross-bars at the base and tip to which the solar blanket attaches were modeled with *FlexBeam* blocks. Each solar blanket was modeled with a *Cloth* block. The *Cloth* blocks were connected to the array structure at the trunk and tip beams only; there is no connection to the mast beam. Deployment was simulated by changing the length of the mast at a rate of 0.04 m/s. A visualization of the simulated deployment process is shown in Figure 5, and the in-plane cut force at the mast base is shown in Figure 6, where the sudden tensioning of the blanket introduces high-frequency loading. Due to the changing length of the central mast, it is not possible to perform this simulation in any other available software tool.

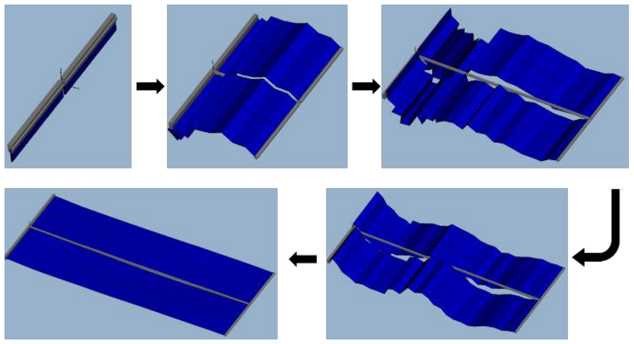


Figure 5. Deployment sequence of GRA linear solar array example.

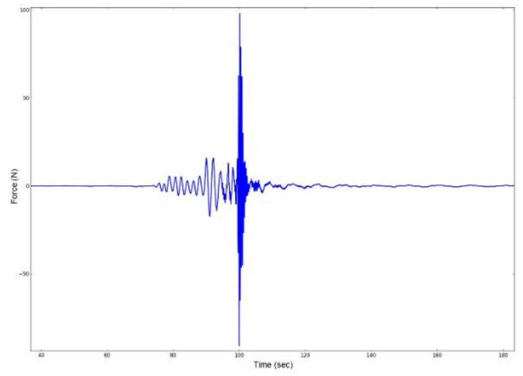


Figure 6. Transverse force at base of GRA mast during blanket tensioning event.

3.2 Origami Solar Array

One concept for large, deployable solar arrays that NASA has investigated is based on origami—the Japanese art of paper folding (Zirbel *et al*, 2013). The shape and folding patterns of these origami solar arrays are defined through three origami parameters, M, H, and R, which control the number and position of the folds as well as overall dimensions. Changing one parameter changes the design of the array, which may in turn affect stowed compactness, load distribution, and overall deployability. Thus, utilizing a model that considers these variables as design parameters is key for finding an optimal origami solar array design.

To enable rapid iteration through various design options, an origami solar array model was created using

Modelica and the *Cloth* block in the DSL. The model was parameterized based on the three origami parameters. The *Cloth* block is written such that creating and parameterizing various configurations such as this, and even completely different topologies, is simple to accomplish.

The origami array model contains only the *Cloth* block; there are no structural members, so all stiffness is provided by the cloth itself. In this case, we are exploiting the formulation of the *Cloth* in a way that each segment of the origami array consists of a single membrane element. As such, each segment acts as a semi-rigid panel without bending degrees of freedom and hinged at each edge. The model is constrained at the center of the array and is deployed by forces pulling radially outward on the tips of the array. Two configurations were studied by varying the origami parameter M, which controls the number of sides the array has when stowed. Nothing else in the model was changed. The deployment sequence for M = 6 is shown in Figure 7, and the deployment sequence for M = 3 is shown in Figure 8. Incidentally, the M = 3 design suffers from binding due to the geometric layout of the origami faces, which is readily apparent during the dynamic simulation.

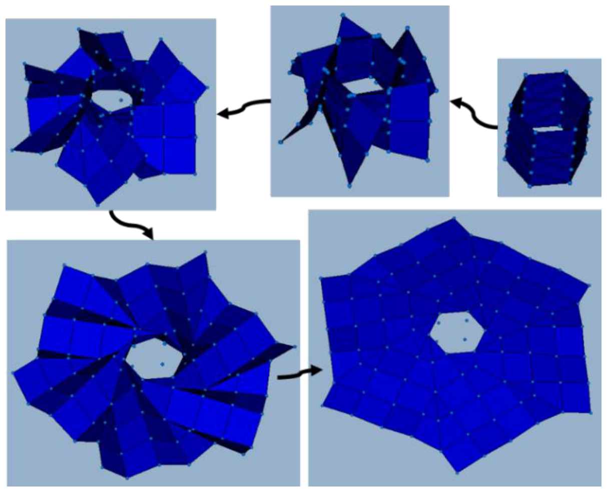


Figure 7. Origami solar array deployment sequence with origami parameters M = 6, H = 2, R = 2.

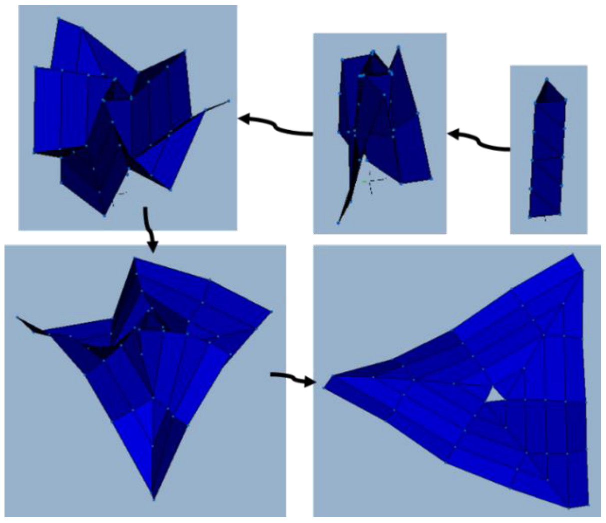


Figure 8. Origami solar array deployment sequence with origami parameters M = 3, H = 2, R = 2.

3.3 Solar Sail

Solar sails have long been proposed as a space propulsion technology, but only recently with the availability of inexpensive cubesat platforms have they been demonstrated in actual spaceflight. As such, solar sail design has yet to gain much flight heritage, and the means by which a sail is deployed can vary widely between design concepts. Solar sails are large, gossamer-thin reflective blankets that harness solar radiation pressure to exert small amounts of thrust on the spacecraft. Integrated over time, the thrust can translate into significant accelerations, possibly enabling interstellar travel (Landis, 1999).

One particular concern of solar sails is the blanket tensioning process. Because the sails are thin (on the order of tenths of millimeters), they are prone to tearing if excessive force is applied. However, if the sails are not taut, wrinkles in the reflective fabric will diffusely reflect solar radiation, resulting in a loss of momentum imparted by the radiation pressure.

A Modelica model using the *Cloth* block and the *VariableLengthBeam* block was created to examine forces exerted on the blanket during the tensioning process. The sail was modeled as a single, square *Cloth* block. The four deploying booms were modeled with *VariableLengthBeam* blocks fixed to each other at the center of the structure where the spacecraft bus would be located, and the sail was connected to the booms through extensional springs at each corner of the sail. This simulation started at the instant when the sail is fully deployed but not yet tensioned and ended when the booms reach their final predetermined length. The start and end states of the sail are depicted in Figure 9. The sail, which is initially square in its undeformed state, stretches at the corners and exhibits a catenary shape along its edges. This is the expected behavior for such a system and demonstrates how the DSL *Cloth* model can be used to perform structural analysis of solar sail structures.

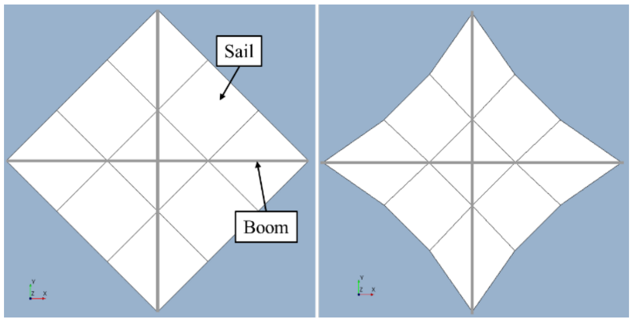


Figure 9. The solar sail model in its untensioned (left) and tensioned states (right).

4 Conclusions

This paper presents a Modelica library for the analysis of deployable structures. The library provides modeling blocks useful for creating models of deployment mechanisms and structures unique to deployable systems, particularly those used on spacecraft. In particular, a modeling capability for structural cloths or fabrics is presented that is not available in any other multibody dynamics software package.

The Deployable Structures Library is open-source and available via the github page https://github.com/ATAEngineering/DeployStructLib, which will likely be linked to via the Modelica Association website at https://modelica.org/libraries. While the library should work for any Modelica implementation per the Modelica standard, it was developed using OpenModelica and has not been tested using other software. External contributions and bug fixes or reports are encouraged.

Acknowledgements

This work was funded under Phase I and Phase II Small Business Innovation Research (SBIR) contracts sponsored by the National Aeronautics and Space Administration (NASA). The authors would like to thank technical monitors Geoffrey Rose and Richard Pappa at NASA Langley Research Center.

References

C. Fellipa. 2017. Introduction to Finite Element Methods course material, Chapter 15: Three-Node Plane Stress Triangles. https://www.colorado.edu/engineering/CAS/courses.d/IFEM.d/IFEM.Ch15.d/IFEM.Ch15.pdf

G. Ferretti, A. Leva, and B. Scaglioni. Object-oriented modelling of general flexible multibody systems. *Mathematical and Computer Modeling of Dynamical Systems*, 20(1): 1-22, 2014. doi: 10.1080/13873954.2013.807433.

A. Heckmann, M. Otter, S. Dietz, and J.D. López. The DLR FlexibleBodies library to model large motions of beams and of flexible bodies exported from finite element programs. In *Proceedings of 5th Modelica Conference*. Vienna, Austria, September 2006.

B. Hoang, W. White, B. Spence, S. Kiefer. Commercialization of Deployable Space Systems' roll-out solar array (ROSA) technology for Space Systems Loral (SSL) solar arrays. In *Proceedings of 2016 IEEE Aerospace Conference*. Big Sky, MT, 2016. doi: 10.1109/AERO.2016.7500723.

N. F. Knight Jr., K. B. Elliott, J. D. Templeton, K. Song, J. T. Rayburn. FAST Mast Structural Response to Axial Loading: Modeling and Verification. In *Proceedings of 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*. Honolulu, HI, April 2012. doi: 10.2514/6.2012-1952.

G.A. Landis. 1999. Advanced Solar- and Laser-pushed Lightsail Concepts. Final Report, NASA Institute for Advanced Concepts. http://www.niac.usra.edu/files/studies/final_report/4Landis.pdf

M. Mikulas, R. Pappa, J. Warren, and G. Rose. Telescoping Solar Array Concept for Achieving High Packaging. In *Proceedings of the 2nd AIAA Spacecraft Structures Conference*. Kissimmee, FL, January 2015.

R. Pappa, G. Rose, T. Mann, J. Warren, M. Mikulas, T. Kerslake, T. Kraft, J. Banik. Solar Array Structures for 300 kW-Class Spacecraft. *Space Power Workshop*, 2013. (https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20140000360.pdf)

C. J. Rupp. 2018. The Relative Finite Element Method. In Prep.

C. J. Rupp, L. Schweizer, and D. Murphy. Rapid Parametric Analysis and Design of Space-Based Solar Arrays. In *Proceedings of the 3rd AIAA Spacecraft Structures Conference*. San Diego, CA, January 2016.

F. Schiavo, L. Viganò, and G. Ferretti. Object-oriented modelling of flexible beams. *Multibody System Dynamics*, 15(3): 263–286, 2006. doi: 10.1007/s11044-006-9012-8.

J. Wright. 2007. https://www.nasa.gov/content/astronaut-scott-parazynski-works-near-solar-array

S. Yang, Z. Deng, J. Sun, Y. Zhao, and S. Jiang. 2017. A Variable-Length Beam Element Incorporating the Effect of Spinning. *Latin American Journal of Solids and Structures*, 14(8): 1506-1528. doi:10.1590/1679-78253894

S. A. Zirbel, R. J. Lang, R. W. Thomson, D. A. Sigel, P. E. Walkemeyer, B. P. Trease, S. P. Magleby, L. L. Howell. Accomodating Thickness in Origami-Based Deployable Arrays. *Journal of Mechanical Design*, 135, 2013. doi: 10.1115/1.4025372.