Basic Modeling Geometry in SysML v2

INCOSE SysML v2 Readiness Exchange

26 August 2026

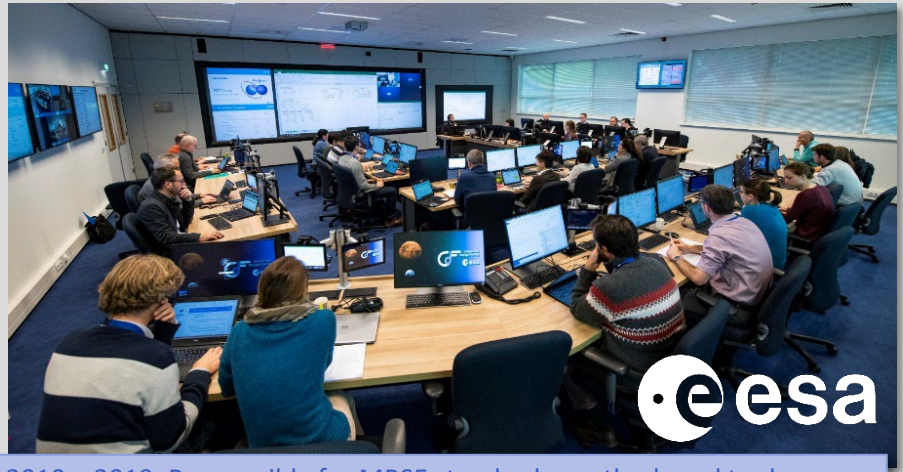
Hans Peter de Koning (hanspeter.dekoning@dekonsult.com)

My Professional Background

- 1984: MSc Applied Physics – Delft University of Technology
- 40+ years' experience as thermal | software | systems engineer
- 1984 – 1999: Industry (predominantly space)
- 1999 – 2019: European Space Agency – retired end 2019
- 2020 – now: Own consultancy company DEKonsult

«DigitalEngineering»
DEKonsult

- Digital engineering R&D projects (ESA, other customers)
- Core member KerML and SysML v2 development team in OMG
- Active member INCOSE MBSE Initiative and INCOSE/NAFEMS Systems Modeling & Simulation WG (Standards Lead)



- 2010 – 2019: Responsible for MBSE standards, methods and tools for the ESA Concurrent Design Facility for space mission development as well as MBSE support to ESA space projects
- Author of / contributor to many standards (ECSS, ISO, OMG)
- Lecturer at ESA Academy

Basic Geometry in KerML and SysML v2

Why Geometry in SysML? What's the use?

- Early lifecycle problem and design specifications need rough geometry
  - inform engineering team and stakeholders – easy basic 3D configuration view(s) for everyone
  - early detection of possible volume / interference / arrangement problems
- Later lifecycle needs up-to-date, quick visualization of 3D configuration(s)
  - system-of-interest in verification or operation environment
  - exchange with analysis / simulation / test models and tools
- Precise and rigorous specification of spatial aspects of interfaces
  - mounting surfaces, enveloping volumes, fields of view, stay-out areas or volumes, human factors, accessibility, ...
  - spatial aspects of operational scenarios / use cases, preparation of analysis and verification cases
- Basic geometry (shapes and placement) shared between systems engineering and other engineering disciplines
- Assessments of geometry-related quantities
  - center of gravity, moments of inertia, ...

Note:
- However, spatial modeling limited to basic, simple geometry, useful for system view.
- Detailed 3D spatial modeling remains the domain of mechanical engineering and CAD.

Geometry Requirement from SysML v2 RFP

- PRP 1.18: Basic Geometry
  Proposals for SysML v2 may include a capability to represent basic two- and three-dimensional geometry of a structural element, including a base coordinate frame as well as relative orientation and placement of shapes through nested coordinate frame transformations, where the basic shape definitions are provided in a model library.

Supporting Information:
- These capabilities are intended to provide basic geometry and coordinate frame representations to support specification of physical envelopes.
- The intent is that each block or equivalent will have its own reference coordinate system, and transformations can be applied between coordinate systems of different blocks.
- The shape of a block is defined as a property (e.g., 3-dimensional rectangular shape with length, height, and depth) whose values can be defined in its reference coordinate system.
- Consider references to standard formats (e.g., ISO 10303 (STEP), IGES)

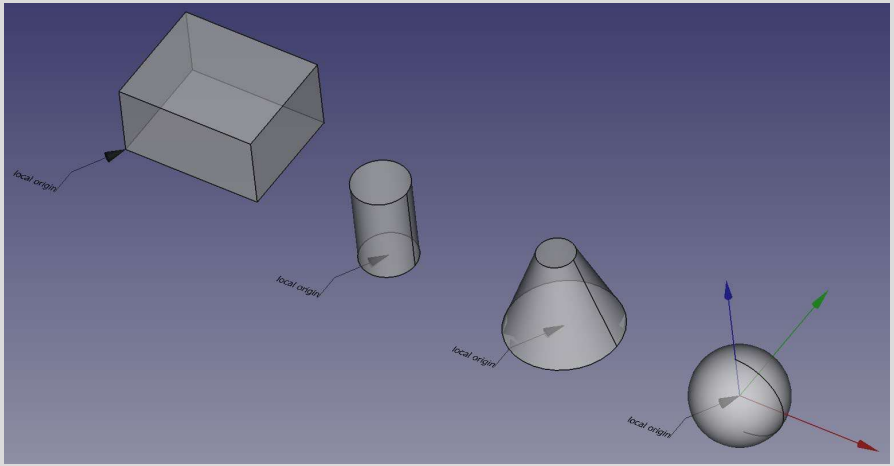
Topology & Geometry in KerML and SysML (from r2022-02 onwards)

Standard Libraries

The spatial aspect of the KerML/SysML 4D Occurrences

- Objects.kerml
  - Body, Surface, Curve, Point
- SpatialFrames.kerml
  - SpatialFrame
  - PositionOf, DisplacementOf
- SpatialItems.syml
  - SpatialItem
  - shape, envelopingShapes
- ShapeItems.sysml
  - Shell, Path,
  - primitive shells / B-reps

Some primitive shells / B-reps: Cuboid, Cylinder, Cone, Sphere

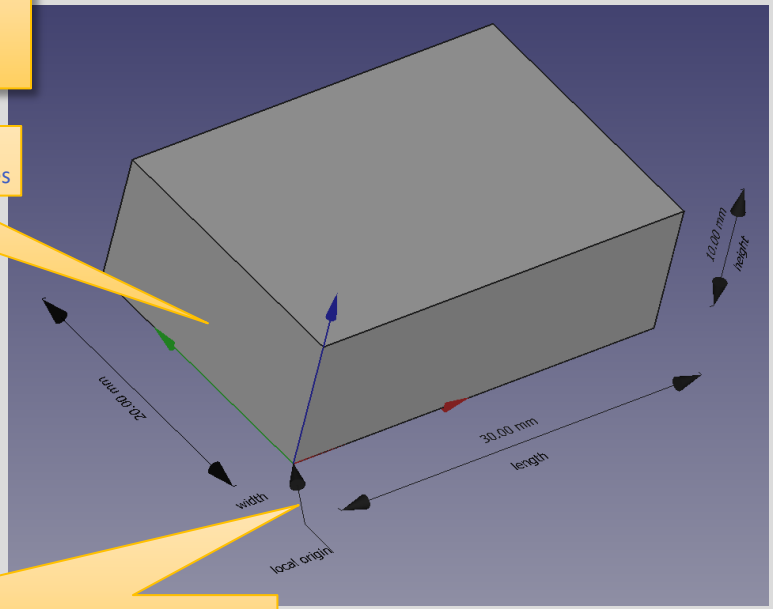


Note: Local coordinate frame definitions for all primitive shapes still need to be included in KerML and SysML specs

Example: Cuboid (alias Box)

```
attribute envelopingBox : Cuboid {
    :>> length = 30[mm];
    :>> width = 20[mm];
    :>> height = 10[mm];
}
```

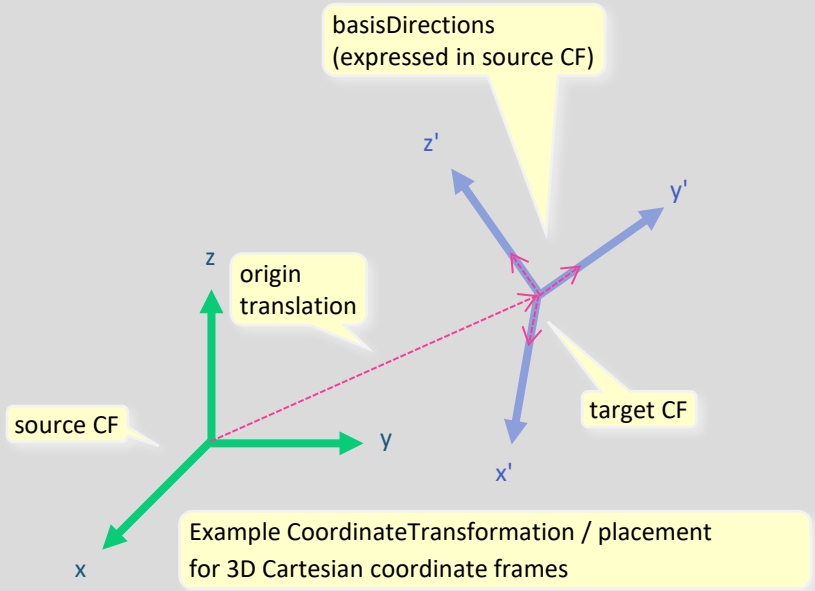
Most 3D tools use Red-Green-Blue coding
to denote (positive) X-Y-Z cartesian frame axes



Every Shape has its own *shape coordinate frame*,
which allows placement within the local frame of the owning (composing) Item/Part

Nested Coordinate Frames

- Need to position and orient each local frame for primitive or compound shape within an Item or Part
- Done via CoordinateTransformation
  - E.g. ISO 10303-42 (STEP) like “Placement”
    - translation of origin
    - target frame basisDirections expressed in source frame
  - Also supported:
    - rotation about axis & translation
    - sequence of rotation / translation
    - (affine) transformation matrix

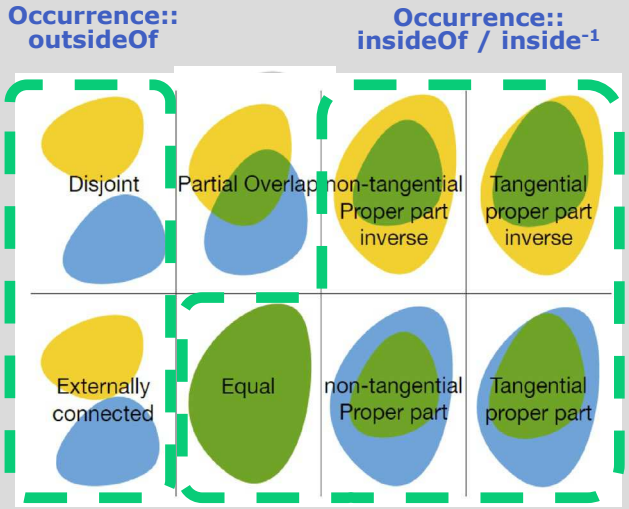


Example CoordinateTransformation / placement for 3D Cartesian coordinate frames

Transformation and Composite Structure

- SpatialItem has attribute coordinateFrame
- An Item or Part that is a specialization of SpatialItem is by default placed (i.e., located and oriented) w.r.t. to the coordinate frame of its owning Item or Part
- The root Item or Part of a composite structure establishes the top-level coordinate frame of the composite structure as a whole: the "system-of-interest"
- The origin of that coordinate frame acts as the reference point and its basis vectors as primary directions to position and orient the "system-of-interest" in 3D space

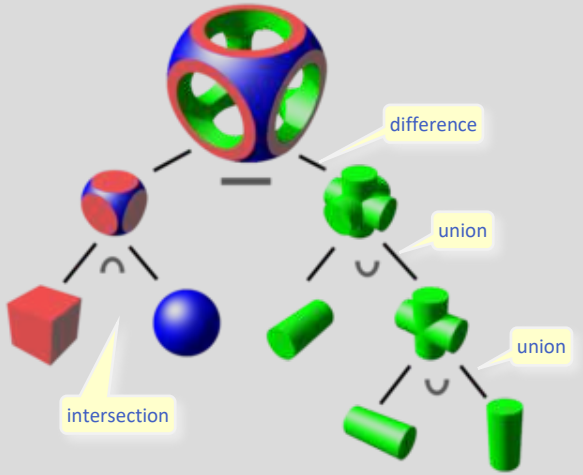
Spatial Modeling Logic and Method



Region Connection Calculus (RCC8)

- Analogous to Allen's logical relations for time
- See https://en.wikipedia.org/wiki/Region_connection_calculus

RCC8 diagram from:
John G. Stell, Spatial Connections An Informal Introduction, 2015
Note: Diagram is 2D for simplicity, but RCC extends to 3D.



Constructive Solid Geometry (CSG)

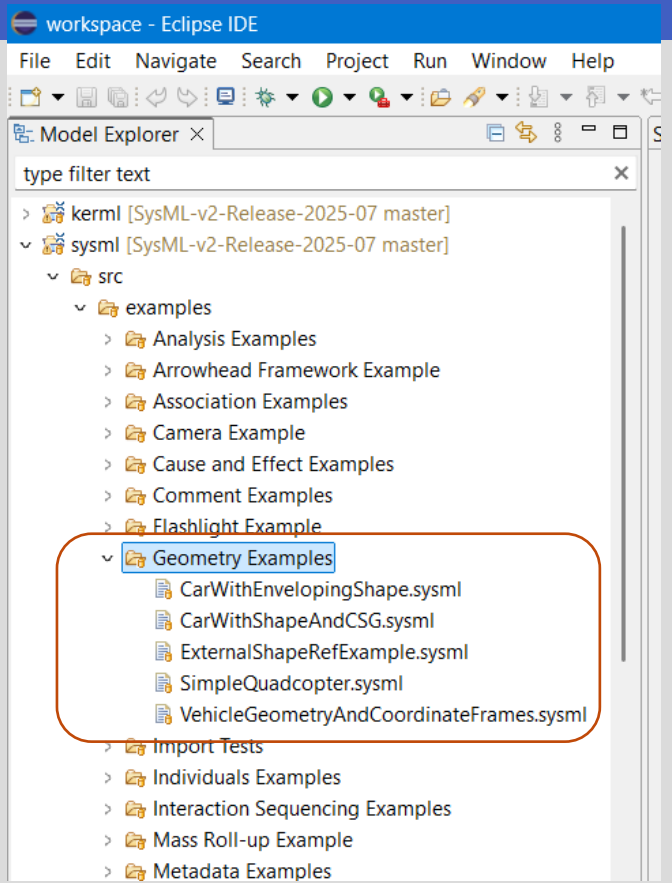
- Union, intersection, and subtraction of shapes
- Not assembly, shapes overlap
- Enables automatic derivation of resulting geometry and topology.

See https://en.wikipedia.org/wiki/Constructive_solid_geometry

Examples

5 Geometry Examples in Releases since 2022

- See SystemsModeling GitHub repos
  - SysML-v2-Release
  - SysML-v2-Pilot-Implementation
- folder sysml/src/examples/Geometry Examples
- CarWithEnvelopingShape.sysml
- CarWithShapeAndCSG.sysml
- ExternalShapeRefExample.sysml
- SimpleQuadcopter.sysml
- VehicleGeometryAndCoordinateFrames.sysml



Walkthrough of SimpleQuadCopter

Shows examples of:

- Use of primitive 3D shapes like box, cylinder, cone
- Enveloping geometry of a simple quadcopter concept
- CSG (Constructive Solid Geometry) union, difference and intersection
- Parameterized construction of shapes
- Coordinate frame transformations, using translation and rotation sequences
- Modeling a sensor’s field of view

SimpleQuadcopter

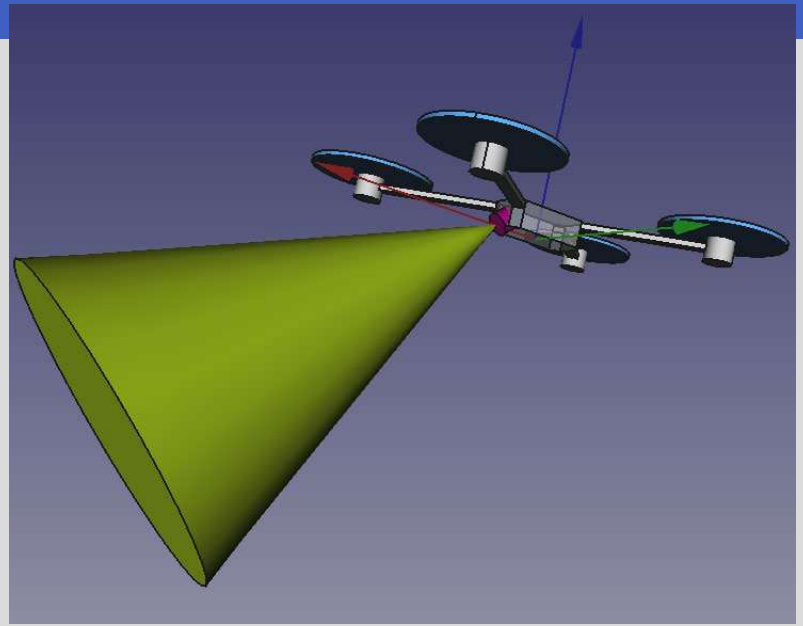
SimpleQuadcopter.sysml

```
package SimpleQuadcopter {
    private import ISQ::*;
    private import SI::*;
    private import SpatialItems::*;
    private import ShapeItems::*;
    private import RealFunctions::sqrt;
    private import TrigFunctions::pi;
    private import TrigFunctions::tan;
    private import MeasurementReferences::CoordinateFrame;
    private import MeasurementReferences::TranslationRotationSequence;
    private import MeasurementReferences::Translation;
    private import MeasurementReferences::Rotation;

    part motorShape : SpatialItem {
        item :>> shape : Cylinder {
            :>> radius = 18 [mm];
            :>> height = 30 [mm];
        }
    }

    part def Strut :> SpatialItem {
        // By default will get same coordinateFrame.mRefs as owning SpatialItem, i.e.:
        // attribute :>> coordinateFrame { :>> mRefs = (mm, mm, mm); }

        /* rawStrut is a construction shape: a rectangular beam */
        part rawStrut :> subSpatialParts {
            item :>> shape : Box {
                :>> length = 160 [mm];
                :>> width = 15 [mm];
                :>> height = 8 [mm];
            }
            attribute :>> coordinateFrame {
                :>> transformation : TranslationRotationSequence {
```

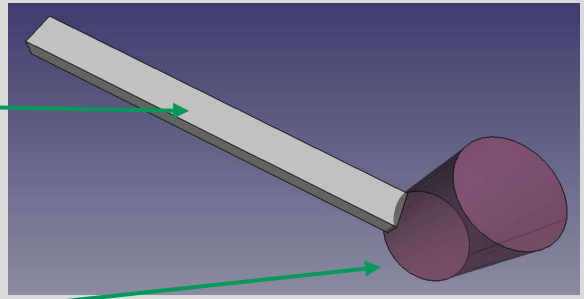


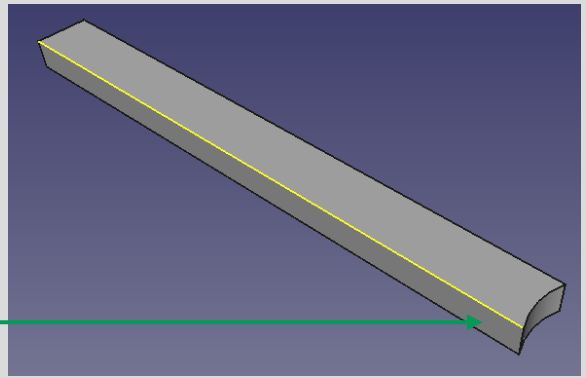
Note:
Graphical screenshots from FreeCAD 3D modeler

SimpleQuadcopter – Strut

SimpleQuadcopter.sysml

```
21    part def Strut :> SpatialItem {
22        // By default will get same coordinateFrame.mRefs as owning SpatialItem, i.e.:
23        // attribute :>> coordinateFrame { :>> mRefs = (mm, mm, mm); }
24
25        /* rawStrut is a construction shape: a rectangular beam */
26        part rawStrut :> subSpatialParts {
27            item :>> shape : Box {
28                :>> length = 160 [mm];
29                :>> width = 15 [mm];
30                :>> height = 8 [mm];
31            }
32            attribute :>> coordinateFrame {
33                :>> transformation : TranslationRotationSequence {
34                    :>> elements = (new Translation( (0, shape.width/2, 0)[source]));
35                }
36            }
37        }
38
39        /* motorCutout is a construction shape: a cylinder of the same shape as the  */
40        part motorCutout :> subSpatialParts {
41            item :>> shape = motorShape.shape;
42            attribute :>> coordinateFrame {
43                :>> transformation : TranslationRotationSequence {
44                    :>> elements = (new Translation( (175, 0, -1)[source]));
45                }
46            }
47        }
48
49        /* Strut shape is CSG difference of rawStrut minus motorCutout */
50        attribute :> differencesOf[1] {
51            item :>> elements = (rawStrut, motorCutout);
52        }
53    }
```

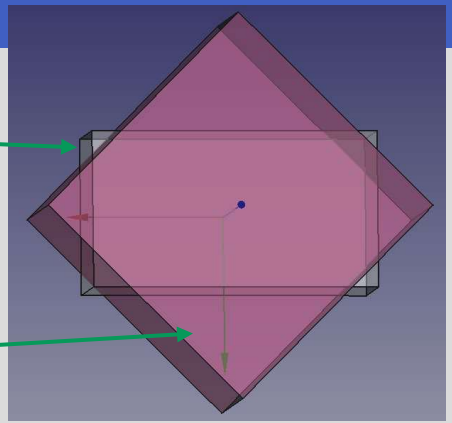


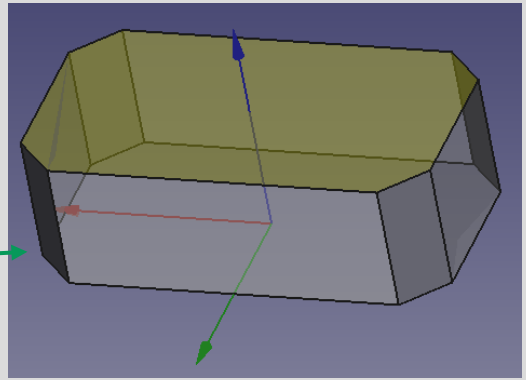


SimpleQuadcopter – MainBody with cut corners

SimpleQuadcopter.sysml

```
124       part mainBody :> subSpatialParts {
125
126           /* rawBody is a construction shape: the enveloping rectangular box */
127           part rawBody :> subSpatialParts {
128               item :>> shape : Box {
129                   :>> length = 160 [mm];
130                   :>> width = 15 [mm];
131                   :>> height = 8 [mm];
132               }
133               attribute :>> coordinateFrame {
134                   :>> transformation : TranslationRotationSequence {
135                       :>> elements = (new Translation( (0, shape.width/2, 0)[source]));
136                   }
137               }
138           }
139
140           /* cuttingBox is a construction shape: the enveloping rectangular box */
141           part cuttingCornersBox :> subSpatialParts {
142               item :>> shape : Box {
143                   :>> length = 105 [mm];
144                   :>> width = 105 [mm];
145                   :>> height = 60 [mm];
146               }
147               attribute :>> coordinateFrame {
148                   :>> transformation : TranslationRotationSequence {
149                       :>> elements = (new Translation( (0, -shape.length/sqrt(2), -10)[source]),
150                                       new Rotation((0, 0, 1)[source], 45['°']));
151                   }
152               }
153           }
154
155           /* Main body shape is the CSG intersection of rawBody and cuttingCornersBox */
156           attribute :> intersectionsOf[1] {
157               item :>> elements = (rawBody, cuttingCornersBox);
158           }
```

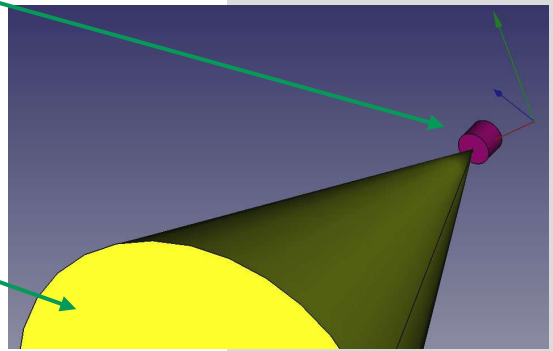




SimpleQuadcopter – Camera

SimpleQuadcopter.sysml

```
84    part def Camera :> SpatialItem {
85        // By default will get same coordinateFrame.mRefs as owning CompoundSpatialItem, i.e.:
86        // attribute :>> coordinateFrame { :>> mRefs = (mm, mm, mm); }
87
88        part cameraHousing :> subSpatialParts {
89            item :>> shape : Cylinder {
90                :>> radius = 15 [mm];
91                :>> height = 24 [mm];
92            }
93        }
94
95        /* The field of view is modeled as an item, since it is not a part of the quadcopter but rather a stay-out volume
96         * that can for example be used to formulate a constraint.
97         */
98        item fieldOfView :> subSpatialParts {
99            doc /* Conical field of view with half-top angle 20 degree */
100           item :>> shape : Cone {
101               :>> radius = height * tan(20 * pi/180) [mm];
102               :>> height = 500 [mm];
103           }
104           attribute :>> coordinateFrame {
105               :>> transformation : TranslationRotationSequence {
106                   :>> elements = (new Rotation( (0, 1, 0)[source], 180['°']));
107               }
108           }
109       }
110
111       // By default the shape of a Camera is the union of its owned composite items and parts that are SpatialItems.
112   }
```

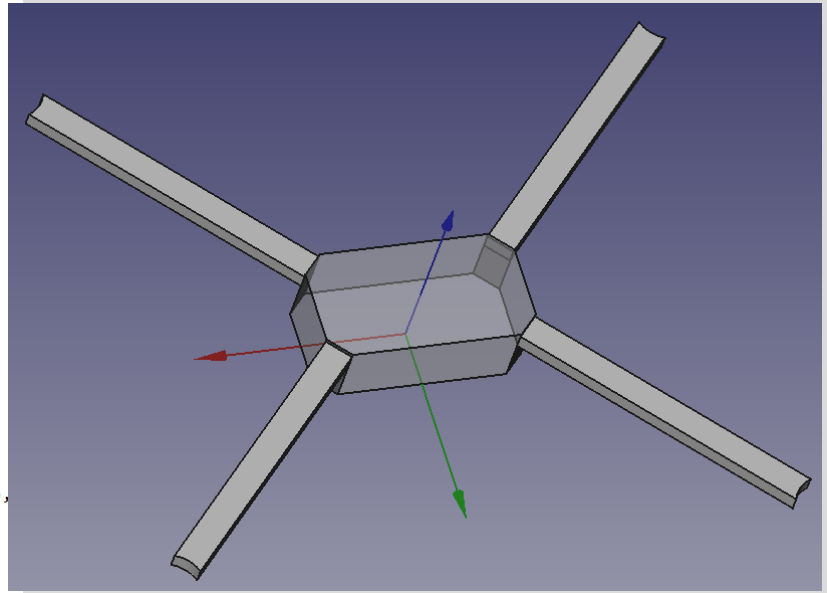


SimpleQuadcopter – 4 struts (usages)

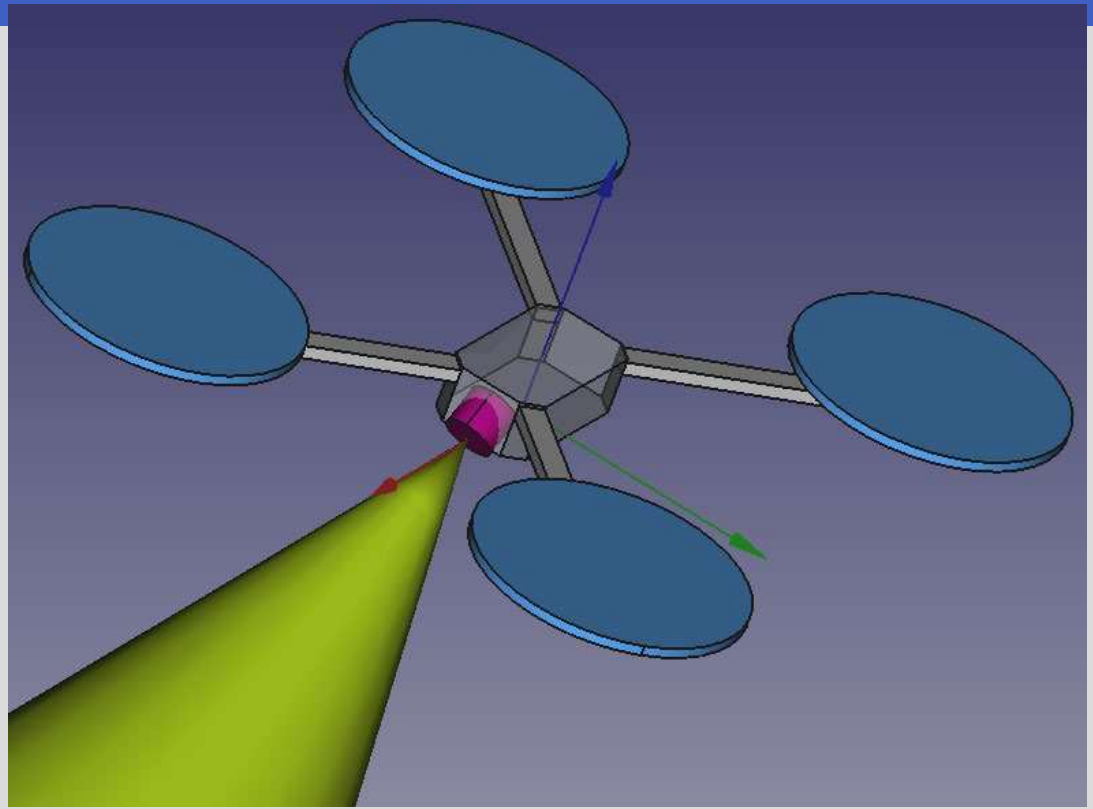
SimpleQuadcopter.sysml

```
        // Helper construction parameters
        private attribute xStrut : LengthValue = 49.60[mm];
        private attribute yStrut : LengthValue = 24.65[mm];
        private attribute zStrut : LengthValue = 25[mm];
        private attribute zPMAssy : LengthValue = 12[mm];

        part strut1 : Strut :> subSpatialParts {
            attribute :>> coordinateFrame {
                :>> transformation : TranslationRotationSequence {
                    :>> elements = (new Translation( (xStrut.num, yStrut.num, zStrut.num)[source]),
                                    new Rotation((0, 0, 1)[source], 45['°']));
                }
            }
        }
        part strut2 : Strut :> subSpatialParts {
            attribute :>> coordinateFrame {
                :>> transformation : TranslationRotationSequence {
                    :>> elements = (new Translation( (-xStrut.num, yStrut.num, zStrut.num)[source]),
                                    new Rotation((0, 0, 1)[source], 135['°']));
                }
            }
        }
        part strut3 : Strut :> subSpatialParts {
            attribute :>> coordinateFrame {
                :>> transformation : TranslationRotationSequence {
                    :>> elements = (new Translation( (-xStrut.num, -yStrut.num, zStrut.num)[source]),
                                    new Rotation((0, 0, 1)[source], 225['°']));
                }
            }
        }
        part strut4 : Strut :> subSpatialParts {
            attribute :>> coordinateFrame {
                :>> transformation : TranslationRotationSequence {
                    :>> elements = (new Translation( (xStrut.num, -yStrut.num, zStrut.num)[source]),
                                    new Rotation((0, 0, 1)[source], 315['°']));
                }
            }
        }
```

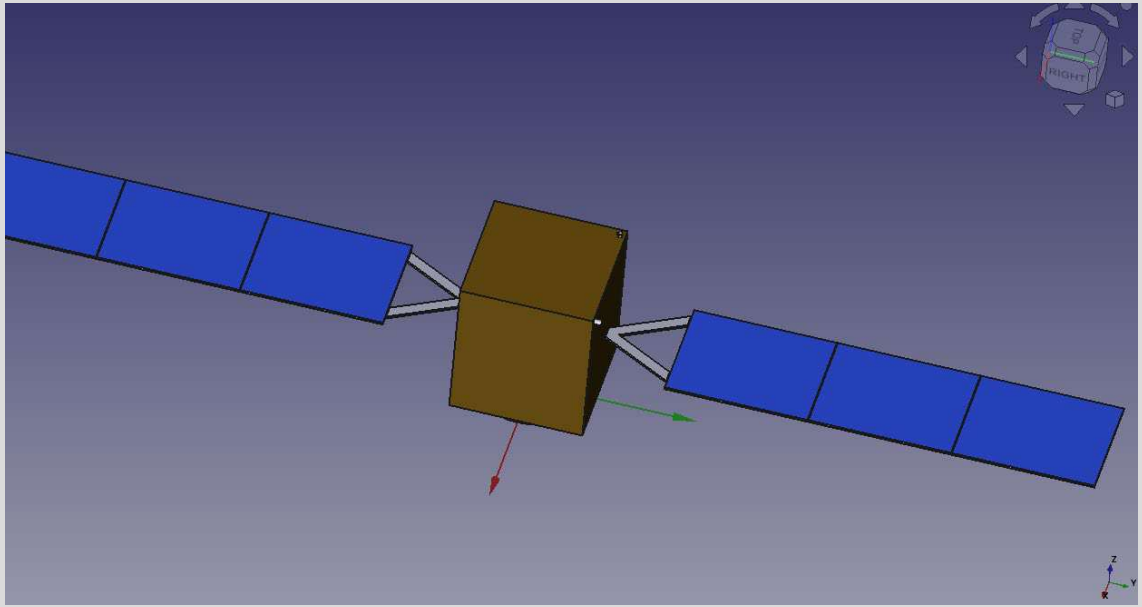


SimpleQuadcopter – Full Assembly



Explore Further Use Cases – DemoSpacecraft

- Example simple DemoSpacecraft
  - Explore proof of concept combining geometry and control
  - Deployment of solar array(s)
  - Explore sensor field of view and thruster plume impingement
- Geometric model is really basic for use at SE level
- No details – only enveloping shapes

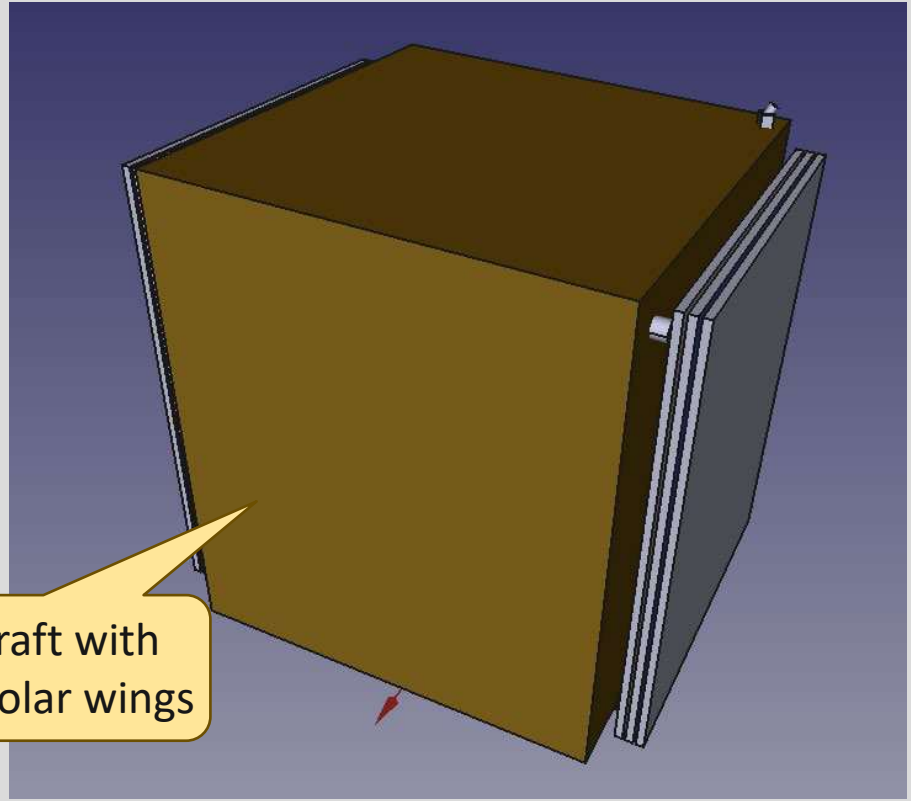


Model a Rigid Body Kinematics Chain

Solar wing consists of:

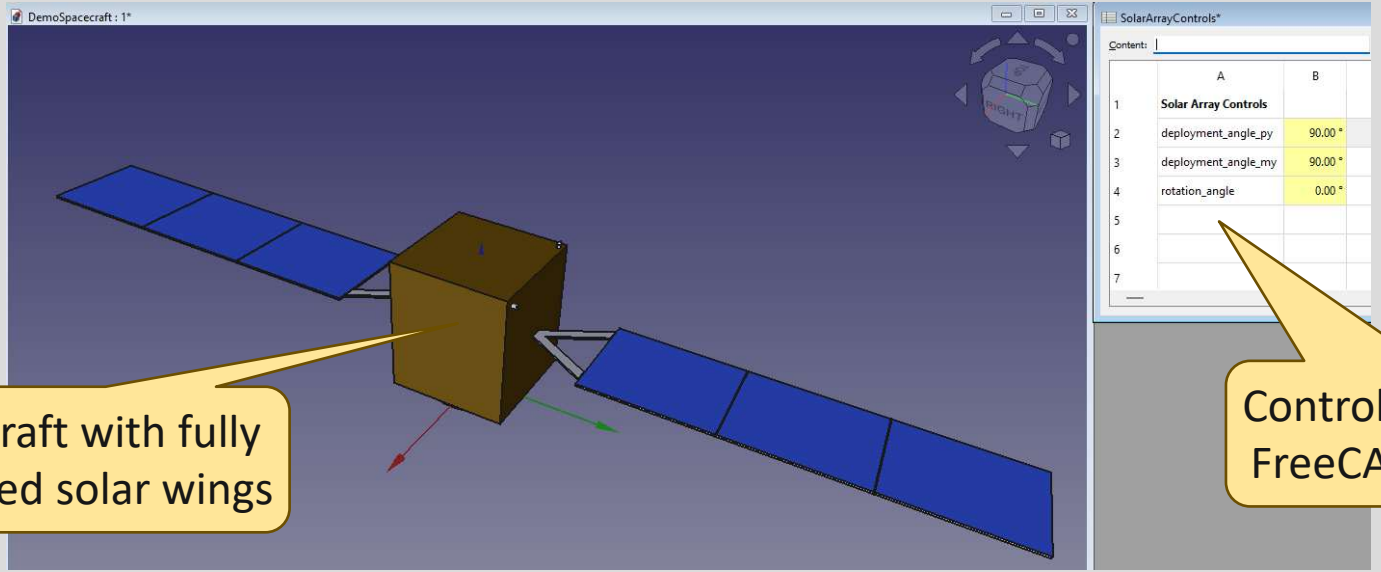
- yoke - attached to spacecraft body via 2DOF hinge
- solar panel 1, 2, 3 – connected via 1DOF hinge

Hinges are not modeled explicitly as parts, but rather as parameterized transformation of coordinate frames



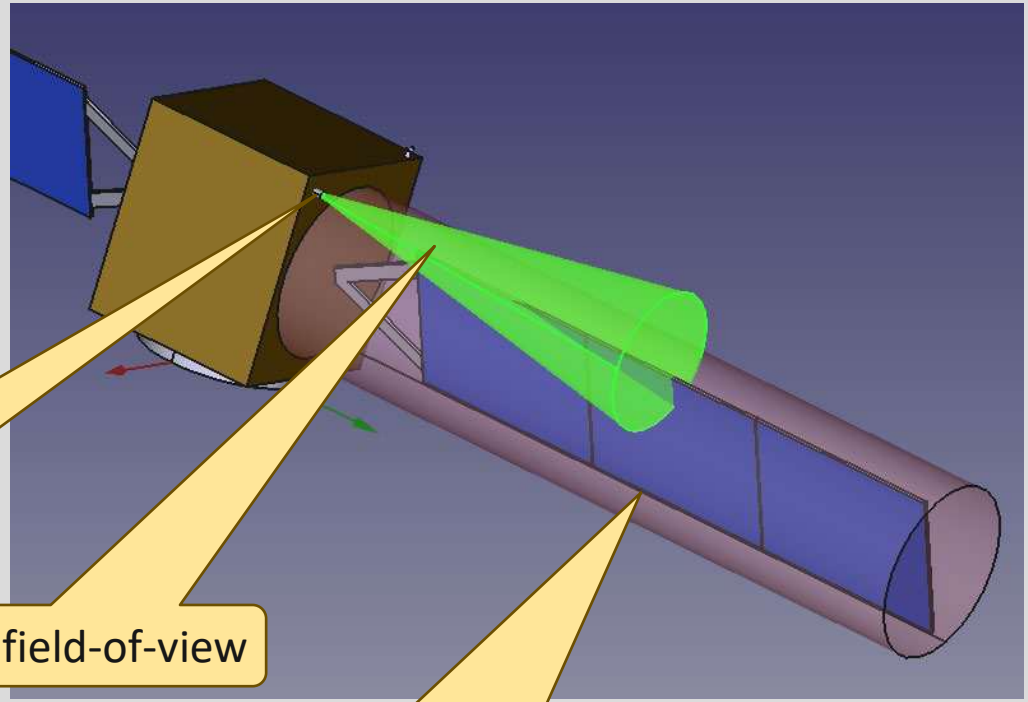
Spacecraft with Solar Wings Deployed

- FreeCAD spreadsheet reflects the SysML control parameters
- deployment_angle_py(_my) ranges from 0 ° (stowed) to 90 ° (fully deployed)
- rotation_angle range is 0 ° to 360 ° about solar wing axis for sun tracking



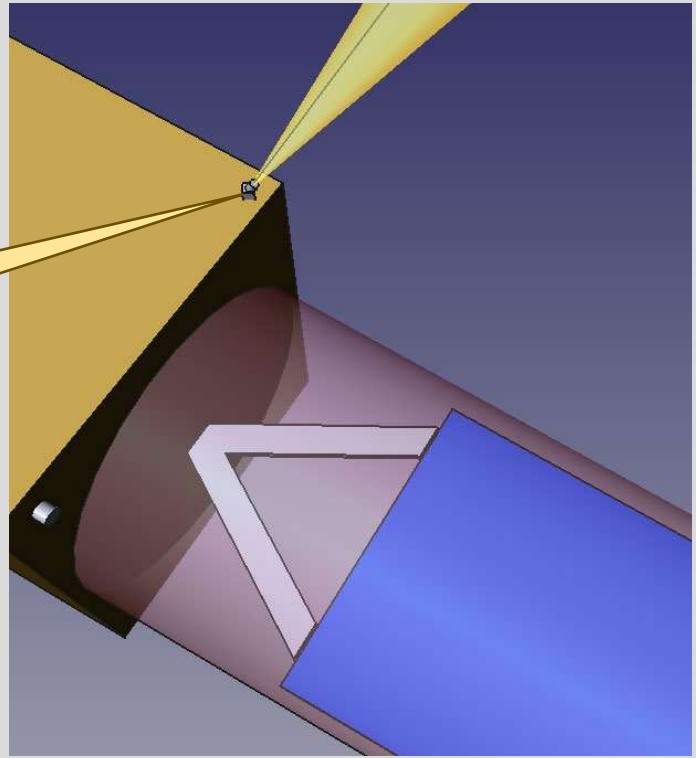
Check Field of View of Star Tracker

- Star Tracker field-of-view is partially obscured by the stay-out volume of the operating +Y solar wing



Check Plume Cone of Thruster

- Plume cone of Bi-propellant thruster does not impede the solar wing
- No risk of solar cell contamination with propellant particles



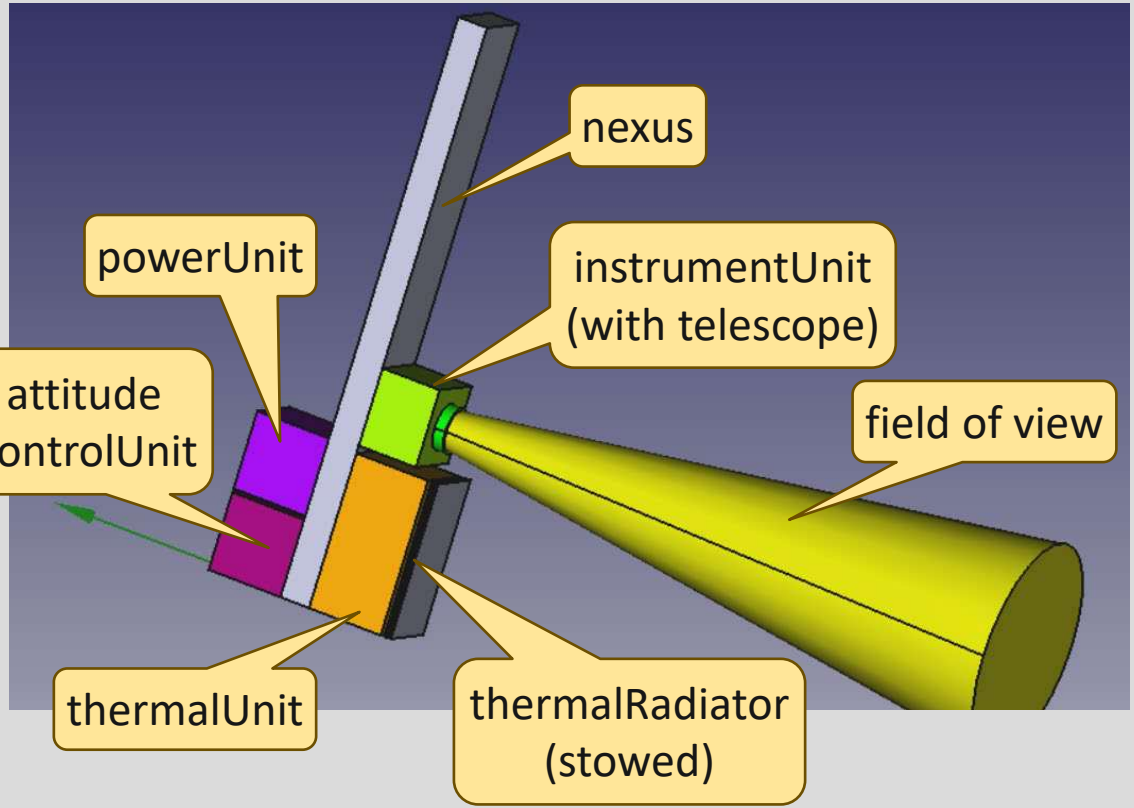
Example textual notation of DemoSpacecraft

```
89          part solar_wing_py :> subSpatialParts {
90              attribute solar_wing_py_CF :>> coordinateFrame {
91                  :>> transformation : TranslationRotationSequence {
92                      :>> elements = (new Translation( (0, 705, 920)[source]),
93                          new Rotation( (0, 1, 0)[source], SolarArrayControls.rotation_angle ));
94                  }
95              }
96              part yoke :> subSpatialItems {
97                  attribute yoke_py_CF :>> coordinateFrame {
98                      :>> transformation : TranslationRotationSequence {
99                          :>> elements = (new Translation( (0, 705, 920)[source]),
100                             new Rotation( (0, 1, 0)[source], -180 ['°'] + SolarArrayControls.deployment_angle_py ));
101                     }
102                 }
103                 item yoke_shape :>> shape : Box {
104                     /* details suppressed */
105                 }
106                 attribute :>> coordinateFrame {
107                     :>> transformation : TranslationRotationSequence {
108                         :>> elements = (new Translation( (0, -25, 5)[source]));
109                     }
110                 }
111                 part solar_panel1 :> subSpatialParts {
112                     item solar_panel1_shape :>> shape : Box {
113                         :>> length = 1200 [mm];
114                         :>> width = 25 [mm];
115                         :>> height = 1500 [mm];
116                     }
117                     attribute solar_panel1_CF :>> coordinateFrame {
118                         :>> transformation : TranslationRotationSequence {
119                             :>> elements = (new Translation( (0, -30, 760)[source]),
120                                 new Rotation( (1, 0, 0)[source], -180 ['°'] - 2 * SolarArrayControls.deployment_angle_py ));
121                         }
122                     }
123                     part solar_panel2 :> subSpatialParts {
124                         item solar_panel2_shape :>> shape : Box {
125                             :>> length = 1200 [mm];
126                             :>> width = 25 [mm];
```

Reference to control parameter

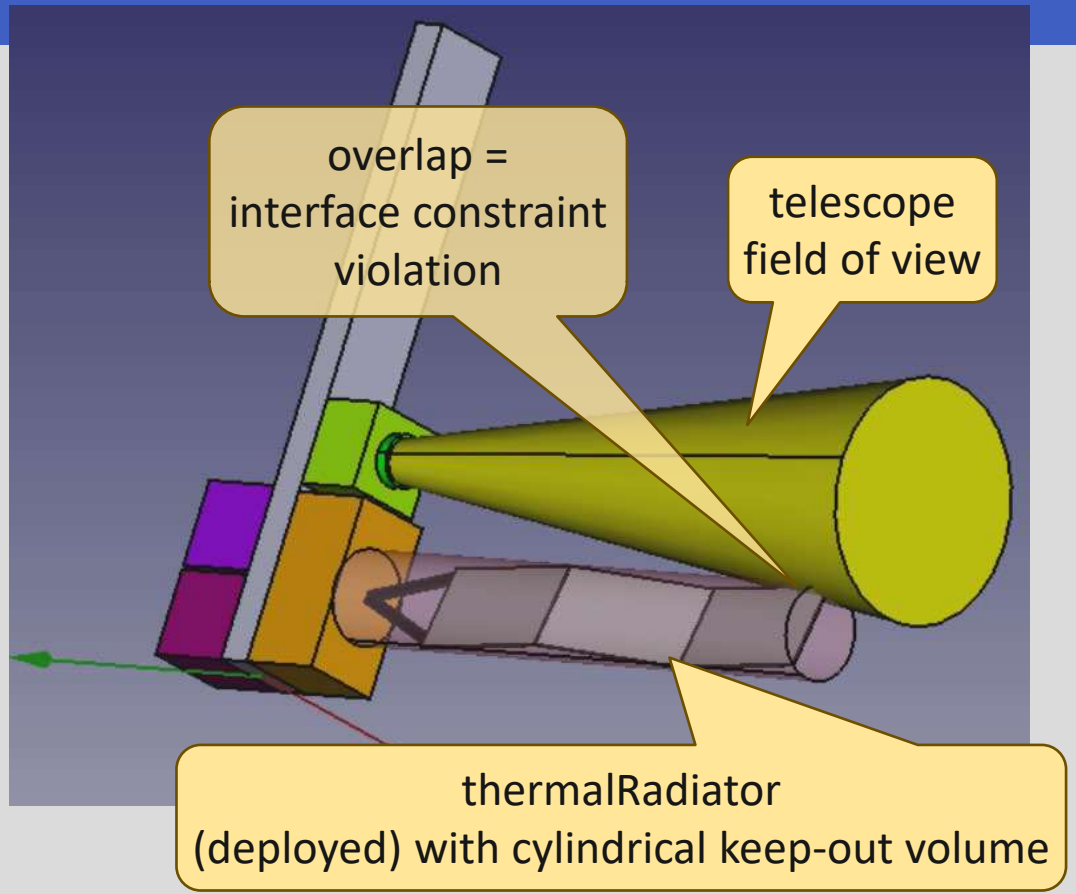
Explore Further Use Cases – NexusCraft

- New example simple “NexusCraft”
  - Explore generation of geometry from SysML2 model
  - Show interface checks
  - Prototype interaction via API
- Geometric model only - basic enveloping shapes
- Interaction FreeCAD via standard API with Starforge by Planetary Utilities
  - Data store is Open-MBEE Flexo quad-store



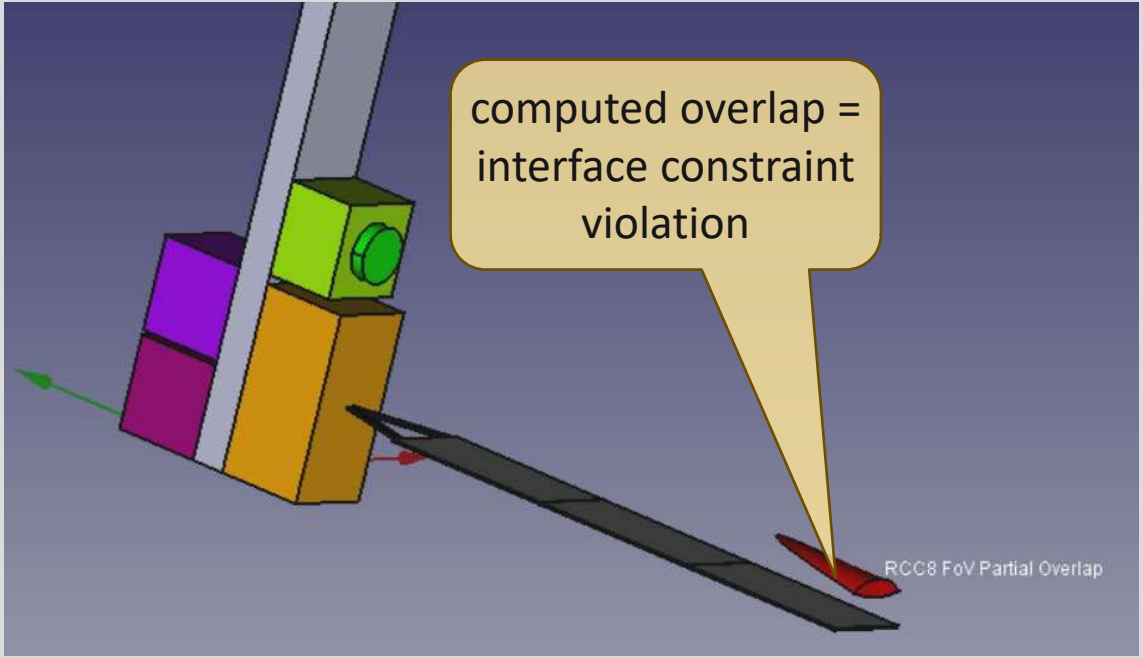
Starforge Nexus
Violation of Keep-out Volume

- Interface check example on simple “NexusCraft”
- Field-of-View of the telescope is overlapping with the keep-out volume linked to the rotating deployed thermalRadiator, i.e. it is partially obscured (similar to previous solar array example)
- Such checks on interface definitions / constraints can be automated and consistently performed as part of continuous model-based verification



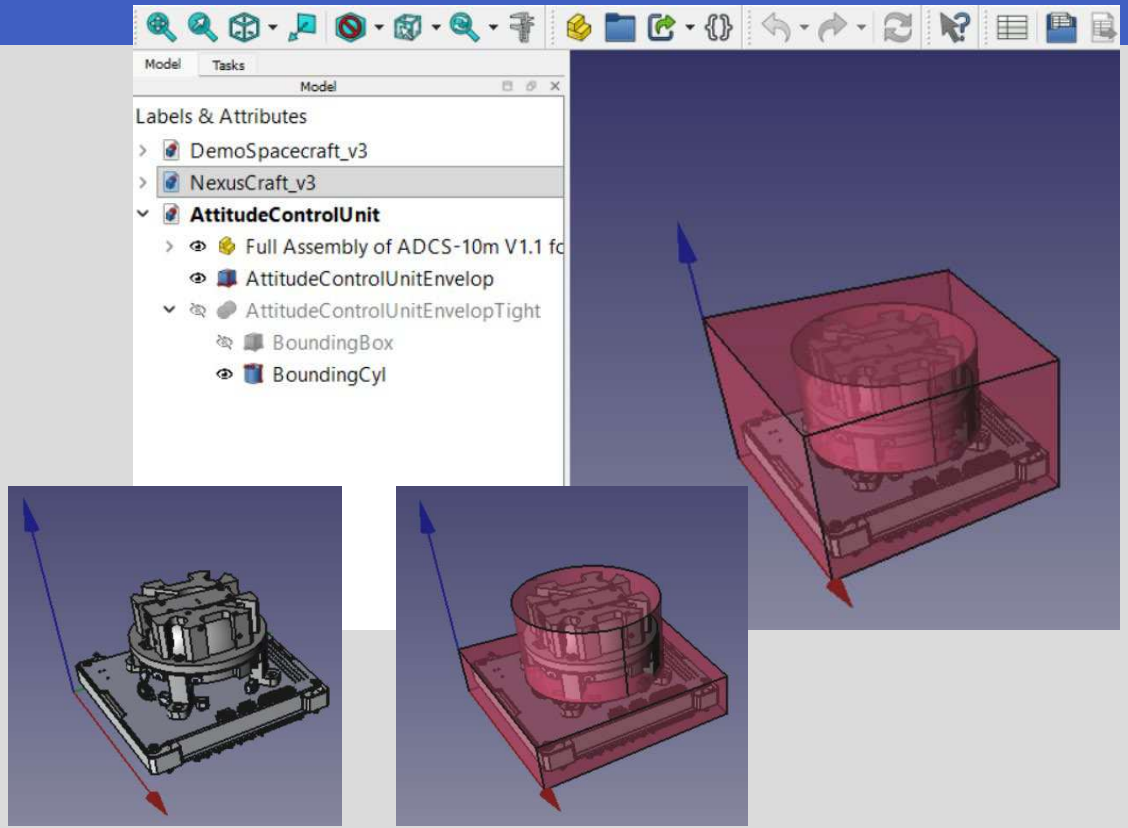
Starforge Nexus – Computation Result of Keep-out Volume Violation

- Interface check example on simple “NexusCraft”
- Result of solid volume intersection computation by the FreeCAD geometry engine (in red)
- Can be automated in CI/CD workflow as systematic checking



Enveloping Shapes over Detailed CAD Shapes

- Simplified Geometric Enveloping / Bounding Box Shapes to communicate 3D Physical Architecture to multidisciplinary team
- Needs standard external reference construct (into CAD database, STEP file)



Live Examples in FreeCAD

Current and Future Work

Preparation of SysML v2 Geometry Tutorial supported by open source example implementation

- Prepare geometry modeling tutorial (in same spirit as current textual and graphical notation tutorials)
- Using open-source FreeCAD tool (as representative of industrial 3D CAD tools)
- Full prototype with SysML pilot implementation and FreeCAD
- Bi-directional link via standard API implemented in Python
- Share and sync SysML composite part structure with CAD part assembly tree
- Add shapes in CAD model and reflect shapes (ShapeItem) back to SysML model
- Make iterative refinements between SysML and CAD with bi-directional updates
- Model moving structures (e.g. rigid body kinematics, robotics) by parameterizing coordinate frame transformations – Basic kinematics work but need improved for usability
  - Mix of externally referenced detailed CAD models and SysML enveloping shapes
- Elaborate RCC8 logic examples and show basic application of 3D reasoning