TAT-C Orbit and Coverage Module Architecture

Design Spec

This is the main focal point for development of a new feature (resource, command, GUI feature, etc.). This will become the feature spec when development is finished.

Status	Prototype Complete *(One of: Started, Prototype Complete, Spec Complete, Integration Complete, QA Complete, Documentation Complete)*
Lead	SPH
GMAT POC	SPH
GMAT Developer	WCS/MES
GMAT GUI Tester	TR
JIRA ID	(Label: eventlocation)

Contents

Orbit and Coverage Context

Design Drivers

- Components communicating with O-C inteface may be in different languages
- Data volume. There will be large amounts of data produced by OC component

Design Principles

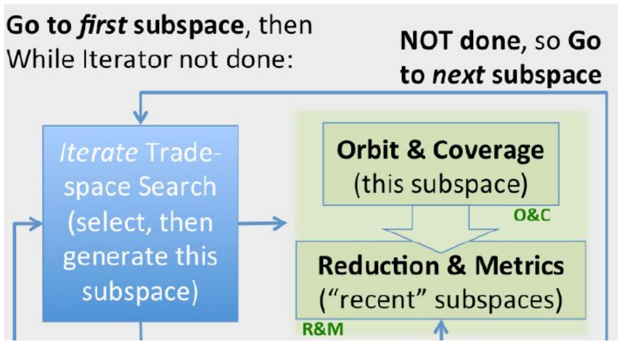
- Hide implementation details behind generic interfaces

Open Questions

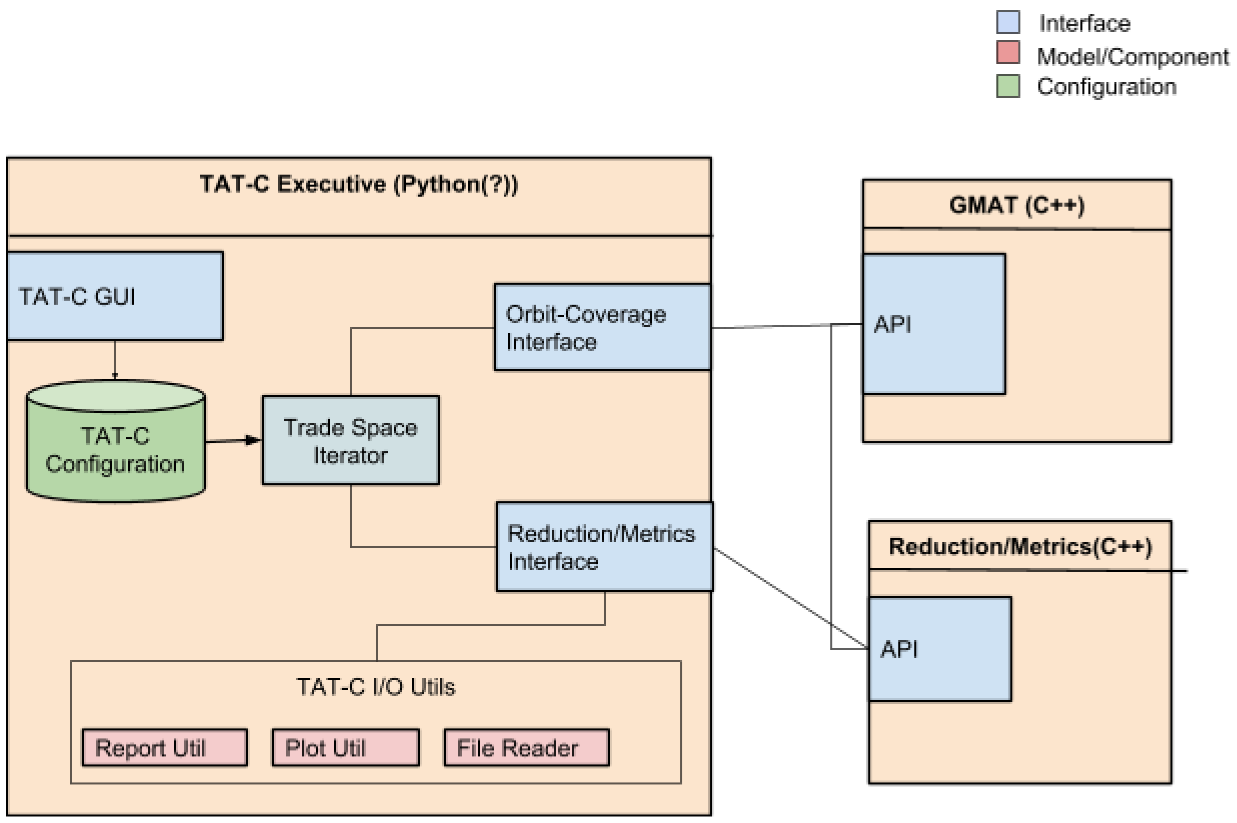
- What language will the system core be written in?
- What language will each component be written in? This drives interfaces if components are not in the same language/environment.
- Which components will communicate with each other directly (and which ones will not)?
- Will Reduction and metrics communicate directly with OC? If so, does it drive the computation, or only request pre-computed data?

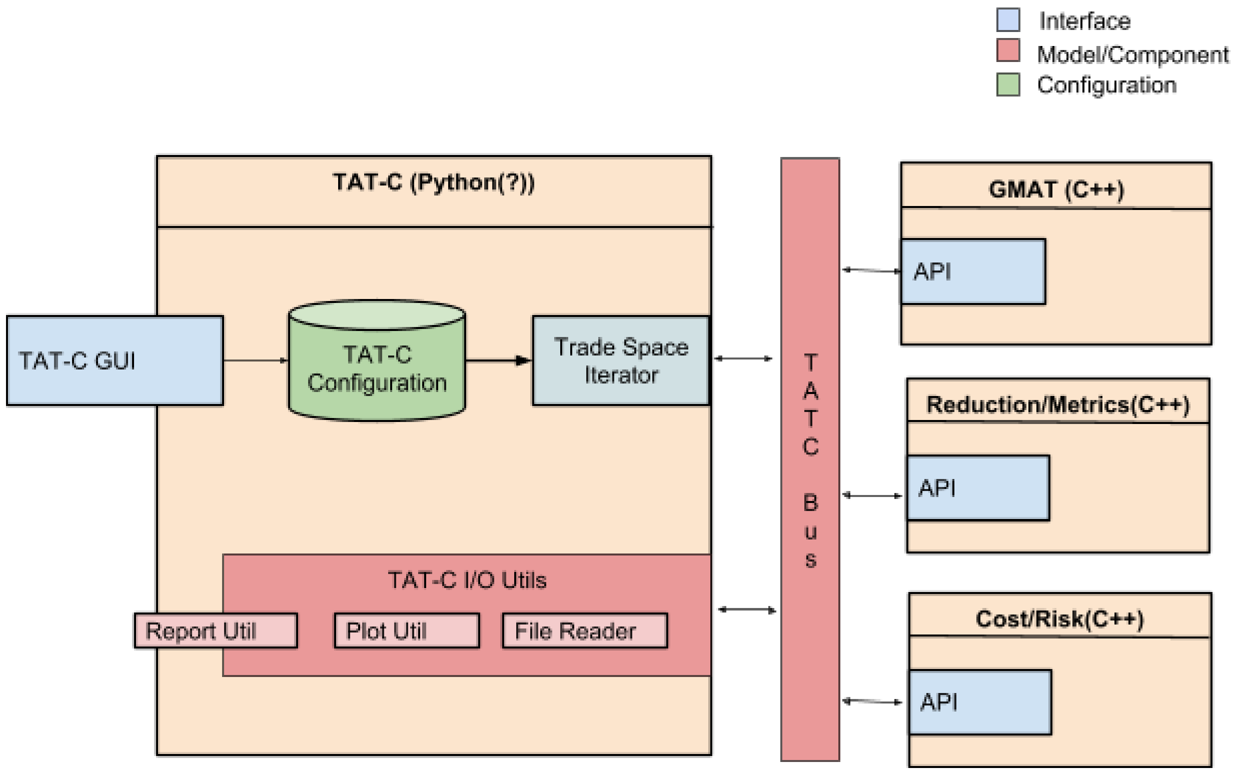
System Diagram for Components Relevant to Orbit Coverage

According to the TAT--C Subsystems and Control Flow diagram, OC communicates with two other system components, the trades space iterator, and Reduction and metrics. The relevant portion of that figure is shown below.


Go to *first* subspace, then
While Iterator not done:
NOT done, so Go to *next* subspace
Iterate Trade-space Search (select, then generate this subspace)
Orbit & Coverage (this subspace)
O&C
Reduction & Metrics ("recent" subspaces)
R&M

The system diagram below shows all key components that directly interface with orbit coverage, or that indirectly provide data to orbit coverage or process data from orbit coverage. This diagram is followed by more descriptions of the components (it does not stand on its own). The current TAT-C executive is assumed to be in Python, while the OC and Reduction components are assumed to be in C++.


Interface
Model/Component
Configuration
TAT-C Executive (Python(?))
TAT-C GUI
TAT-C Configuration
Trade Space Iterator
Orbit-Coverage Interface
Reduction/Metrics Interface
TAT-C I/O Utils
Report Util
Plot Util
File Reader
GMAT (C++)
API
Reduction/Metrics(C++)
API


Interface
Model/Component
Configuration
TAT-C (Python(?))
TAT-C GUI
TAT-C Configuration
Trade Space Iterator
TAT-C I/O Utils
Report Util
Plot Util
File Reader
TATC Bus
GMAT (C++)
API
Reduction/Metrics(C++)
API
Cost/Risk(C++)
API

Overview of Classes, Responsibilities, and Collaborations

Relevant TAT-C Executive/Core Classes

These are classes that drive the interface with the Orbit Coverage tool.

Class Name	**Class Responsibility**
Trade Space Iterator	Drives computation/analysis of a single point in the architecture trade space.
Orbit-Coverage Interface	A generic interface between TAT-C and the orbit coverage tool. The base class is abstract with NO specifics regarding the actual tool performing orbit coverage calculations. Derived classes implement the specific interfaces to the OC tool.
GMAT API	The GMAT API as exposed in the GMAT software. This is the API independent of TAT-C.
Reduction and Metrics	The TAT-C component that receives and process coverage data and ancillary data from the Orbit-Coverage.

Input to Orbit Coverage from TAT-C

These classes are populated with data from the TAT-C user interface, and passed to the Orbit-Coverage interface so the Orbit-Coverage tool can configure accordingly. NOTE: CONSTELLATION, SPACECRAFT, PAYLOAD, AND GROUND SEGMENT ARE THOROUGHLY DISCUSSED IN CHAPTER 7. O-C NEEDS A FEW MINOR ADDITIONS TO THAT DATA.

Class Name	**Responsibility**
Constellation	Class that contains data describing a single constellation in the architecture trade space. It has a vector of spacecraft, and contains other data like the simulation time etc.
Spacecraft	A model of the spacecraft and attached payload.
Payload	A payload model.
Coverage Config	Configuration for point grid definition and user defined custom points.

Ground Segment	Ground station configuration.
Orbit Model	Contains the orbit model

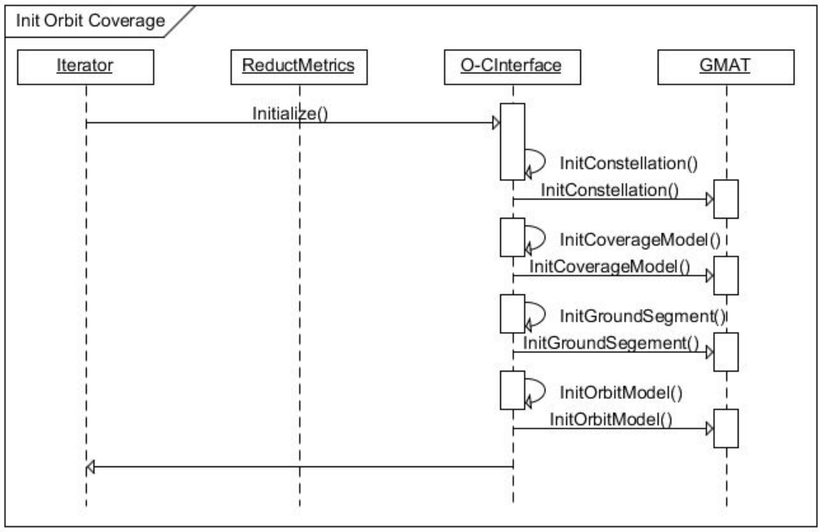
Output from Orbit Coverage to TAT-C and or Reduction and Metrics

Lower level classes containing output from orbit/coverage. NOTE: MOST OF THIS IS DEFINED IN CURRENT SECTION 4.

Class Name	**Responsibility**
OrbitCoverageData	A container for date related to orbit coverage (which points are seen when)
OrbitStateData	A container for orbit state data.
OrbitAncilaryData	A container for other orbit data the is required for cost/risk and performance.
OrbitTimeData	A container for time information.

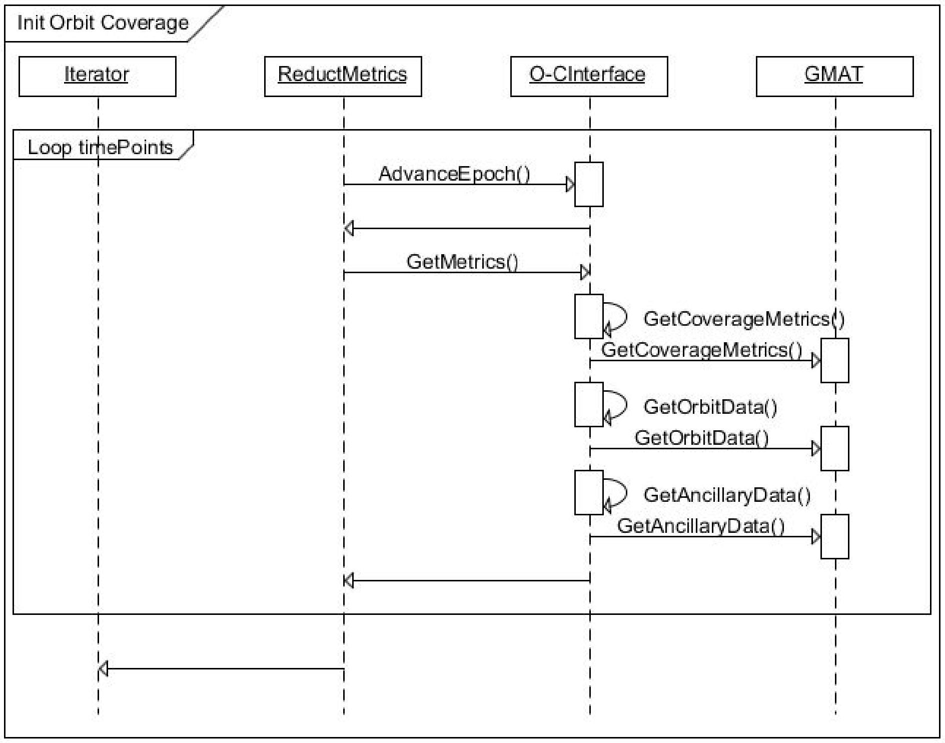
System Initialization

The figure below is a preliminary sequence diagram that shows the initialization sequence for the Orbit-Coverage subsystem.



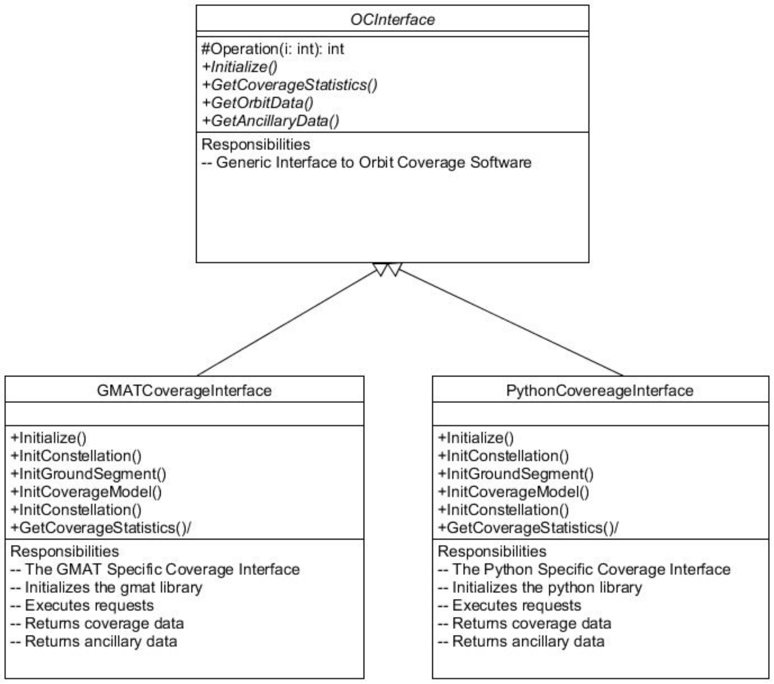
System Execution

The figure below is a preliminary sequence diagram that shows the execution sequence for the Orbit-Coverage subsystem. This diagram assumes that sending data back to Python, then into Reduction and Metrics will cause performance problems and they must communicate directly. This needs to be clarified soon.



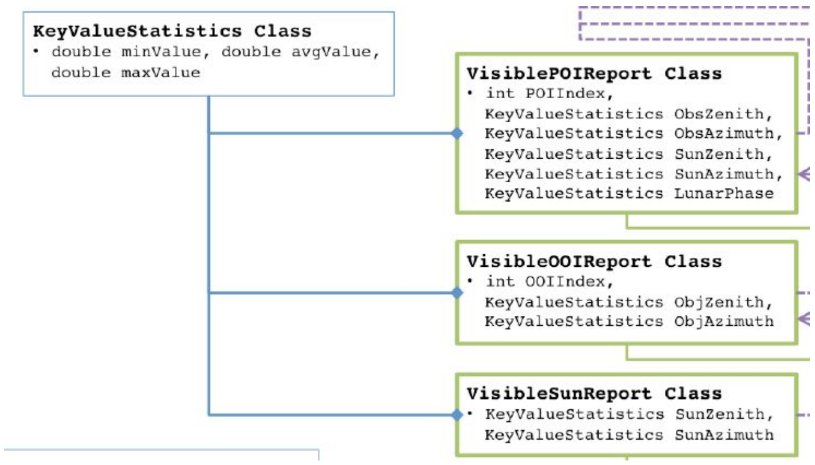
Key Class Diagrams

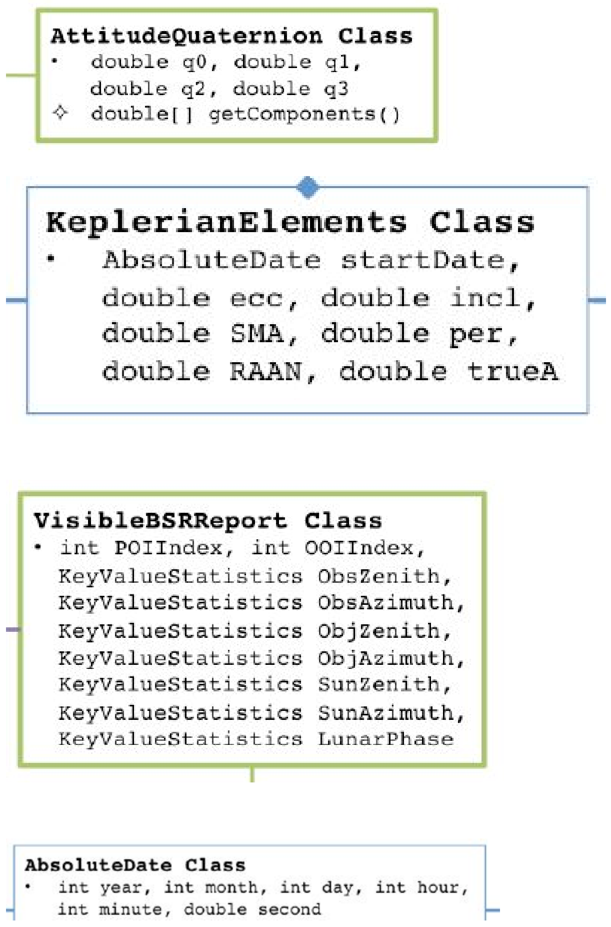
Orbit-Coverage Interface Diagram
The OCInterface class is an abstract class and defines the generic interfaces to the system computing orbit coverage. The derived classes implement the specific interfaces.



Classes Returning Data from Orbit Coverage

When Orbit Data returns data to TAT-C or Reduction and Metrics, via calls to GetCoverageStatistics, and GetOrbitData, etc, the data is returned in the form of objects instantiated from the classes below.


KeyValueStatistics Class
• double minValue, double avgValue, double maxValue
VisiblePOIReport Class
• int POIIndex, KeyValueStatistics ObsZenith, KeyValueStatistics ObsAzimuth, KeyValueStatistics SunZenith, KeyValueStatistics SunAzimuth, KeyValueStatistics LunarPhase
VisibleOOIReport Class
• int OOIIndex, KeyValueStatistics ObjZenith, KeyValueStatistics ObjAzimuth
VisibleSunReport Class
• KeyValueStatistics SunZenith, KeyValueStatistics SunAzimuth



```
AttitudeQuaternion Class
•   double q0, double q1,
    double q2, double q3
✧   double[] getComponents()
```

```
KeplerianElements Class
•   AbsoluteDate startDate,
    double ecc, double incl,
    double SMA, double per,
    double RAAN, double trueA
```

```
VisibleBSRReport Class
• int POIIndex, int OOIIndex,
  KeyValueStatistics ObsZenith,
  KeyValueStatistics ObsAzimuth,
  KeyValueStatistics ObjZenith,
  KeyValueStatistics ObjAzimuth,
  KeyValueStatistics SunZenith,
  KeyValueStatistics SunAzimuth,
  KeyValueStatistics LunarPhase
```

```
AbsoluteDate Class
•   int year, int month, int day, int hour,
    int minute, double second
```

Refactoring

Drivers

- Improved performance. Avoid propagating at small time steps when event location requires small times steps. Simulate orbits at a rate appropriate for orbits and interpolate to perform event location
- Eventual attitude models
- Prepare for GMAT API for high fidelity propagation

Trades

Which Interpolators   (not an either or)

- SPICE?
- GMAT Interpolators?
- Do we need to precisely locate event boundaries

Interface Design

- Propagate for duration then locate events
- Locate during propagation stepping

Eventual sensor updates

- Orientation in body system
-

High level design notes

- Spacecraft will have an interpolator that stores and interpolates orbit data to requested times.

Pseudocode for event location

```
While t < tmax
      Step and Buffer orbit data
      If CovChecker.CanComputeCoverage()
            ComputeCoverage
           End
End
```

The CoverageChecker::CheckPointCoverage() function changes

- Geometry and coord conversions move to spacecraft
- This function would loop over points buffered during propagation. Sends time and target vector in TDB coords to spacecraft. Spacecraft returns true or false depending for inView.

Class Responsibilities

Executive Package/Components Overview

Class	Responsibility
Spacecraft	Container for state at the current epoch, the attitude and sensor models, and holds the accumulated ephemeris data via an owned object. The spacecraft is responsible for providing a flag if a requested epoch is contained in the buffered ephem data, and determining if an object is in view of a given sensor if the date is contained in buffered ephem data.
Propagator	Models the orbit dynamics, propagates the orbit state to a new epoch and sets that new state on the spacecraft.
Coverage Checker	Stores the users grid points via an owned object, and stores the discrete times where grid points were in view of the spacecraft. Manages the location of event times in the CheckPointCoverage() method, and performs preliminary feasibility tests to avoid calling sensor model if there is no way the sensor could see a requested grid point. Processes the discrete events to compute coverage intervals. **Note: CheckPointCoverage currently computes attitude and rotates the view point vector into the sensor frame. This will be moved to s/c.**

O-C Driver Pseudocode

```
Create Date date
Create OrbitState orbitState
Create Spacecraft mySat(date, orbitState, attitude)
Create Sensor
mySat.AddSensor(theSensor)
Create Propagator(mySat)
Create PointGroup thePointGroup
Create CovChecker  covCheck(mySat,thePointGroup,eventStepSize)
prop->AccumluateEphemData() // I think we decided to remove this line
While propTime < propEndTime
```

```
     covCheck. CheckPointCoverage();
      date->Advance(propStepSize);
     prop->Propagate(*date);

      Ancillary calculation
End
// processes all in view events to compute interval events (rise/set)
coverageEvents = covChecker->ProcessCoverageData();
```

Pseudo-code CheckPointCoverage Function

```
IntegerArray CoverageChecker::CheckPointCoverage()
If propEndTime > covEndTime
   % Note, we need to address an interpolation sliding window so that
interpolation does not always occur at the end of the data span, when it
could interpolate in the middle.
   timeCov = timeCovStart
   While sc.canInterp && (timeCov < timePropEnd)
      Convert s/c state to central body fixed
      CheckGridFeasibility() // Check which points could possibly be inview
      For pointIdx = 1 to num Coverage points
         If feasibilityTest.at(pointIdx)  // if current point could
feasibly be seen by sat
            Rotate current point (pointIdx) to inertial
                                   call this refactored function
             inView =
sc->CheckTargetVisibility(inertialTargetVec,sensorNum,time)
                  EndIf
                  If inView
                        Store coverage geometry data
                  EndIf
       EndFor
timeCov += eventLocationTimeStep
EndWhile
EndIf
```

Development Steps

1. Modify Propagator::Propagate() and call new method Spacecraft::SetOrbitState(): Currently Propagate() gets a pointer to the OrbitState and sets state directly. Instead, we should call SetState() on s/c, and that method should update OrbitState object AND buffer the data.
2. Implement the Interpolate() and CanInterpolate() functions on spacecraft. Need to write algorithm for CanInterpolate().
3. Refactor Attitude
    a. Done: Write the new attitude class with a single leaf class that computes LVLH attitude.
    b. Write new overloaded CheckTargetVisibility() function that takes inertial target vector, uses attitude to convert to body, then calls sensor CheckTargetVisibility with vector.
    c. Done: Attitude is effectively hard coded in CoverageChecker::CheckPointCoverage(). This should be moved to the spacecraft and the spacecraft should convert the target vector to the sensor coords. This is in lines 269 to 280 in CoverageChecker starting at comments // Convert satToTargetVec to the nadir attitude system. Note, if a point is in view based on passing CheckGridFeasibility() test, then we would rotate the earth fixed point to inertial, and provide it to the sensor via spacecraft to check if it is in view. This function needs to be changed:
    d. inView = sc->CheckTargetVisibility(inertialTargetVec,sensorNum);
    e. Modify sc.CheckTargetVisibility to convert from inertial target vector to body fixed target vector.
    f. [DONE] Rename "bodyFixedState" in CheckPointCoverage to CentralBodyFixedState
    g. Test we get the same answers

Next Steps

Open Questions

1. How do we force interpolation when propagation is complete and we need to interpolate in non-optimal way.
2. What does the event location loop look like in CheckPointCoverage() given the difference in step sizes between propagation and event location.

- Use case 1: Event location time step is much smaller than propagation times step
- Design 1: … propagate for sim duration the locate events in post processing
- Use case 2: Event location time step and prop time step are the same
- Design 2: Propagate to time, locate events at that time.

```
// Function when not provide input
CoverageChecker::AccumulateCoverageData()

Time = sc->GetTime
scState = sc->GetState()
dateData.push_back(sc->GetJulianDate());
timeIdx++;
this->CheckPointCoverage(scState,time)

// Function when provide a time for event location
CoverageChecker::AccumulateCoverageData(time)
scState = sc->InterpolateState(time)
dateData.push_back(sc->GetJulianDate());
timeIdx++;
this->CheckPointCoverage(scState,time)

// New Function that performs event location and stores the data
// This is a refactorization of CheckPointCoverage()
CoverageChecker::CheckPointCoverage(scState,time)

// convert scState to Earth fixed at time
CheckGridFeasibility(bodyFixedPos, rangeVector, isFeasible);

// Unresolved issue..  How to store/set sc state so we don’t interpolate
twice.  May need an interpolated state on spacecraft, or maybe should reset.

 for ( Integer pointIdx = 0; pointIdx < numPts; pointIdx++)
      //  Modify so this only checks a single point.
      CheckGridFeasibility(bodyFixedPos, rangeVector, pointIdx, isFeasible);
      if (isFeasible )

          // convert grid point at pointIdx to inertial
          // Compute the inertial target vector

          if (sc->HasSensors())
                inView = sc->CheckTargetVisibility(inertialVector
,sensorNum,time);
          Else
              Leave this alone?
          End
          Store event data
end
```