Measurement Model Refactoring

This section provides details related to the measurement model implementation in GMAT. The tracking data subsystem that provides observation data is discussed on the Tracking System Refactoring page (TO BE WRITTEN).

Measurement Model Scripting

In the current design, users interact with measurement models through two classes: MeasurementModel and TrackingSystem. Tracking systems are used to collect together measurement models and apply common properties to them. Since tracking systems require measurement models, we'll discuss the measurement model scripting first.

A user defines a measurement model using the scripting

```
Create MeasurementModel mmod;
mmod.ObservationData = {tdfObject};
mmod.Type = TDRSSTwoWayRange;
mmod.Participants = {ods, stat};
mmod.Bias = [ 0 ];
mmod.NoiseSigma = [ 1 ];
mmod.TimeConstant = 6000;
```

GMAT's measurement model resource provides the data needed to manage a measurement during simulation and estimation. It identifies the resources used in the measurement, the type of measurement used in the run, and the source of the tracking data associated with the measurement. In addition, it lets the user provide bias values for each components of the measurements, and provides initialization data for the measurement error covariances. Some notes:

- Measurement models are defined generically. The Type field identifies the measurement specifics
- Tracking data files are connected to measurements using the ObservationData field. ObservationData names a FileInterface resource used to read (for estimation) or write (for simulation) the tracking data
- The Type field identifies the specific measurement type used in the model. The type identifies an object instantiating a subclass of GMAT's CoreMeasurement class. That object is used to perform measurement computations.

- Measurements are made between objects in the simulation, identified in the Participants list.
- The Bias setting on the measurement is an a priori bias. The estimators can estimate biases during a run; if set to run in that mode, this is the initial bias value used. It is an additive value for the measurement
- During initialization, the noise sigma value is swuared to build the diagonal of the measurement error covariance matrix.
- The time constant is not used in the current code.

Tracking systems are scripted using the syntax

```
Create TDRSSTrackingSystem AQUA_TDRSS;
AQUA_TDRSS.Add = {mmod};
AQUA_TDRSS.TroposphereModel = 'None';
AQUA_TDRSS.IonosphereModel = 'None';
```

In the current implementation, atmospheric effects can only be applied to measurement models through tracking system containers. The identified models are applied top all measurement models identified in the list of measurement models.

- Tracking systems are defined by measurement type. For example, the TRDSSTrackingSystem container only contains TDRSS measurement models.
- The Add field lists the measurement model resources that the tracking system manipulates
- The TroposphereModel names the specific tropospheric model that is applied to the measurements. The only options at this writing are "None" and "HopfieldSaastamoinen"
- The IonosphereModel names the specific ionospheric model that is applied to the measurements. The only options at this writing are "None" and "IRI2007"

The atmospheric model names are passed into the measurement model, and from there to the core measurement objects in the measurement models.

Class Layout

The current class layout for the measurement models used in GMAT Navigation is shown in Figure 1. The light orange classes are the measurement types that users script inside of a measurement model. To avoid diagram complexity, the PhysicalMeasurement leaf classes were left off this diagram (for the most part). The bottom portion of the figure illustrates this feature: the TwoWayRange class has four derived classes: DSNTwoWayRange, USNTwoWayRange, TDRSSTwoWayRange, and SNTwoWayRange.

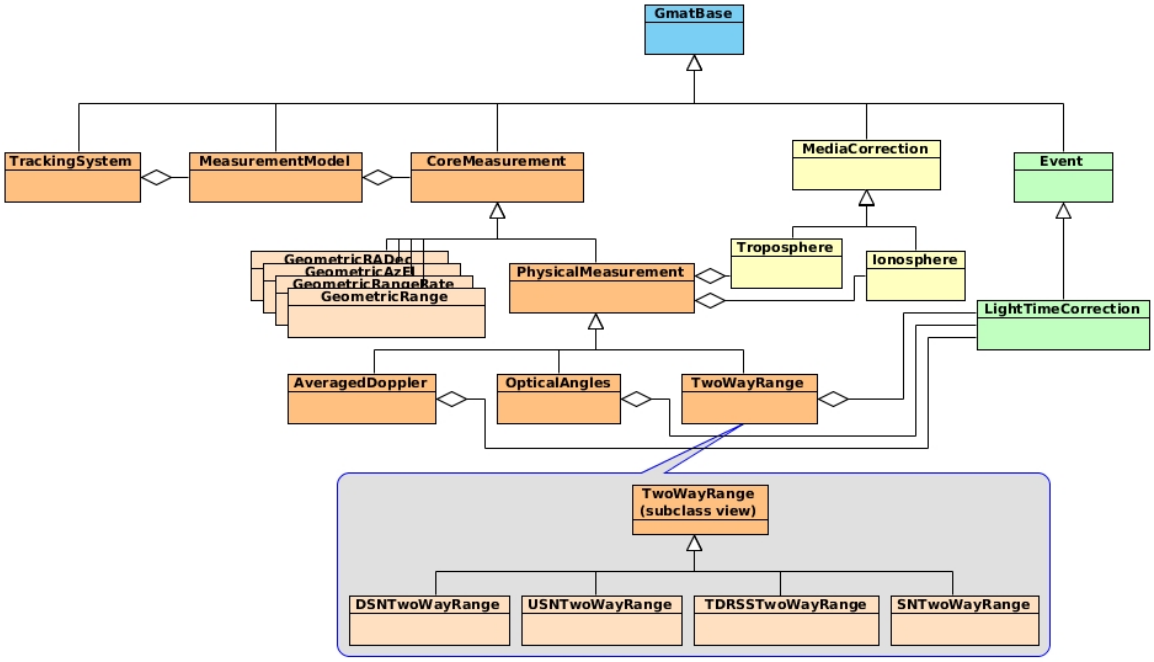

Figure 1: Measurement Modelling in GMAT, Current Approach

Refactoring Plan

GMAT's measurement models in their current form provide challenges for both users and software developers. From the user perspective, media corrections due to the atmosphere require that the user set up one or more measurement models, and then a tracking system that contains these measurement models. Each measurement model contains a core type of measurement. Measurement models inside of a tracking system must all be of the same type – "DSN", USN", TDRSS", etc – because the leaf classes of the tracking systems are typed that way. However, even a cursory examination of the class diagram in Figure 1 shows that this scripted typing does not match the code. DSN 2-way range and DSN Doppler measurements are contained in DSN tracking system containers, but the class structure for these measurement types does not match the user's view of the objects. This state of mismatch came about because of the implementation approach taken for the original code. In the original implementation, measurement classes were implemented before media corrections were designed for the system. Tracking systems were added next as containers that would group measurements together. Then media corrections were added, and specified as features of the tracking systems. The actual use of media corrections is applied in the measurement objects. The tracking system objects are used to script these corrections, and they are then passed to the measurements for use in calculations.

The results of this implementation is a basic disconnect between the programmer's view of the relationships between objects in the measurement model subsystem and the user's view of these same objects. The measurement model refactoring presented below addresses that feature of the current implementation.

For programmers, the goal of this refactoring is to have:

1. A set of signal building blocks that all measurement models use
2. A single measurement model for a given collection of signal paths used for a generic type – for example, a "two way range" measurement would be used for all range measurements that start at a ground station, go to a target spacecraft, and then return to the same ground station.
3. An interpreter layer that transforms that generic model into the form used for an associated observation.

Users would interact with this approach by creating resources for the tracking data side (using a modestly reworked design for the "DataFile" class structure), and for the measurement model in a generic form. We'll begin by looking at the implications of the proposed refactoring on the scripting, and then discuss the proposed rearrangement of the

code by presenting the architecture that makes this approach work.

User View of the Refactoring: Scripting

The updated scripting for the measurement modelling in GMAT is revised to meet the following criteria:

- Measurement models are named by type, rather than consisting of a single container with the type set internally
- The resource types are matched to the collection of measurement "legs" used to define the measurement: OneWayRange, TwoWayRange, TwoWayDoppler, etc. This list is started below but needs to be fleshed out.
- Corrections that apply to the legs of a measurement are set on the measurement model: relativity, media corrections, and so forth.
- The generic measurement model is designed to model the physics of signals being generated and passing between the participants of a measurement.
    - The connection between an observation and the measurement model is specified on the tracking data side of the code
    - The conversion between the physical modeling in the measurement model and the data that matches an observation is applied through an internal "adapter" class.

The last feature in this list is probably the biggest difference between the current code and the proposed refactoring. In this new approach, a TwoWayRange resource provides the structure needed to model a light time solved signal transmitted from the ground to a spacecraft, a delay at the spacecraft, and a return light time solved signal from the spacecraft to the ground. Delays at both ends of this signal can also be applied if needed.

The conversion of the data from that signal into a measurement is performed outside of the resource, in an internal piece of code that maps the signal based physics to an observation value. That connection is made in estimation by taking the tracking data format as specified in the observation data file (or data stream if the input observation is not file based) and calling the measurement model to provide the inputs needed to generate the calculated observation. The calculated observation is generated by a tracking data adapter intermediary class.

From the user's perspective, the scripting for the reworked measurement modelling subsystem rearranges the location of some elements of the models, and removes the tracking systems from the scripting. An example of the updated measurement model scripting for a two-way range model that includes light time solutions for each path in the measurement is

```
Create TwoWayRange twr;
twr.Participants = {Sat, GStation};
twr.Bias = [0.0031];
twr.Covariance.Bias = [1.0e7];
twr.Troposphere = HopfieldSaastamoinen;
twr.Ionosphere = IRI2007;
twr.UseRelativity = true;
```

Open Issues:

Are there other signal corrections that need to be included? (Example: ET - TAI should be here as well if we want to be able to toggle it)

Should we add scripted delays on this level, or continue to get those from the hardware? My inclination is to have it here following this approach:

- Default the delay to 0.0
- Script it like this: twr.Delay = 0.00013
- If set to zero, check for hardware attached to participants at initialization. If nonzero, then hardware delays are overridden.

You might notice that the "ObservationData" setting is not part of this new definition for a measurement model. The connection between an observation and the associated measurement model is handled separately from the definition of the measurement model itself. For estimation, the connection is based on settings in the observation data. For example, when an observation that is being processed is identified as a "DSN Range measurement," GMAT uses the raw data from the measurement module and adapts it to the data expected for the observation, converting the raw two-way range data into range units as expected for DSN ranging. If the observation is identified as "USN range," a different adapter is applied to generate the corresponding range in kilometers.

Proposed initial set of measurement models

Model Name	Description	Participants	Supported Types
GeometricMeasurement	Any instantaneous measurement between a spacecraft and a ground station	Spacecraft, Ground station	GeometricRange, GeometricRangeRate, GeometricAzEl, GeometricRADec
TwoWayRange	Ground station -> spacecraft -> ground station light time adjusted range measurements with all corrections available	Spacecraft, Ground station	DSNRange, USNRange
TwoWayDoppler	Ground station -> spacecraft -> ground station light time adjusted Doppler measurements with all corrections available	Spacecraft, Ground station	DSNDoppler
RelayedRange	Ground station -> relay spacecraft -> target spacecraft -> relay spacecraft -> ground station light time adjusted range measurements with all corrections available	Spacecraft, Spacecraft, Ground station	TDRSSTwoWayRange

RelayedDoppler	Ground station -> relay spacecraft -> target spacecraft -> relay spacecraft -> ground station light time adjusted Doppler measurements with all corrections available	Spacecraft, Spacecraft, Ground station	TDRSSTwoWayDoppler
AngularMeasurement	One way light time adjusted angular measurements	Spacecraft, Ground station	AzEl, RADec

Open Issues:
Can the adapter for the two-way measurements handle ramping? I think it can, but that piece needs to be factored into the detailed design. If it cannot be handled there, we might need separate RampedRange and RampedDoppler models.
TBD: The geometric measurement piece might need to be broken into separate resources.
It might be possible to combine the range and Doppler measurements into a single class in the code. They seem different enough to me to leave them separate here, though: ranging only needs one transit of the full path, while Doppler needs two.

Software Description of the Refactoring

The measurement model refactoring breaks apart the original design for the measurement classes and reworks them into a new set of components that collaborate to generate measurement data. The new design has a set of measurement model classes that replace the CoreMeasurement base class in the original design. Each measurement model consists of the specification of the measurement participants and one or more signal objects that model the paths between the participants. Observations, collected in tracking data files, provide measured data in a format that might not directly match the computations performed internally in the measurement models. A set of adapter classes is defined that provides the translation between the data computed in the measurement model and the data in the observation. These tracking data adapters recast the measurement model data into a format suitable for residual calculations. As an aside, users can control reporting about the data generated in each measurement calculation using a set of reporting controls that define the reporting granularity.

As an example of how the classes defined below work, consider the case of DSN Ranging. During estimation, a DSN Range observation, in range units, is retrieved from a tracking data file. The range observation data is passed to a TwoWayRange measurement model, providing that model with the specifics of the measurement epoch and signal characteristics (frequency, frequency band, ramping if applied, etc). The measurement model calls the signal objects to compute the light time solutions and measurement corrections, along with signal delays at the spacecraft and, if needed, the ground station. The resulting data consists of the participant states at the measurement nodes, information about the measurement feasibility, and properties of the received and transmitted signal. The tracking data adapter (a DSNRangeTDA object for this example) takes these raw data and computes the corresponding calculated measurement in range units. The estimator uses this value as the computed data in the O - C calculation.

Users of the system can generate a report of the estimation data used for the measurement. For each measurement, the user can report the measurement residual, the observation value (in range units), the computed

range (in range units), and the actual range (in km). Additionally, the user can report the tracking data converted into km, along with the range ambiguity for that observation. Finally, the participant states at the signal nodes can be reported as well.

Open Issues:
The mechanism for control over the data reporting described above is TBD.

The following sections describe the classes described above in more detail.

Measurement Composition

In the refactored measurement models presented here, GMAT treats a measurement as an object that ties together a set of participant objects to generate a computed value for a measurement. The participants communicate with each other through a set of signals passed from one participant to another. This signal based design in presented on the Anatomy of a Measurement page. Observation data connects to the signal based measurement through adapter classes that map the physics of the communications signals to the data available in the tracking data objects. The adapter for a given measurement is set by the tracking data reader, which selects the adapter appropriate to the observation.

The signals used in the measurements include light time solution code and the media corrections needed to generate a precise model of the measurement. Users script measurement objects using core descriptors that are independent of the tracking data type – "TwoWayRange", "TwoWayDoppler", etc. This refactored design for these components of GMAT's measurement modeling is presented on the remainder of this page. The first few sections provide a static overview of the classes used in the measurement subsystem of the estimation code. Once the class structure has been described, the interactions that accomplish estimation and simulation computations will be presented so that the reader can see how these classes work together in the estimation system.

Measurement Classes

Figure 2 shows the proposed refactoring of the measurement model classes.

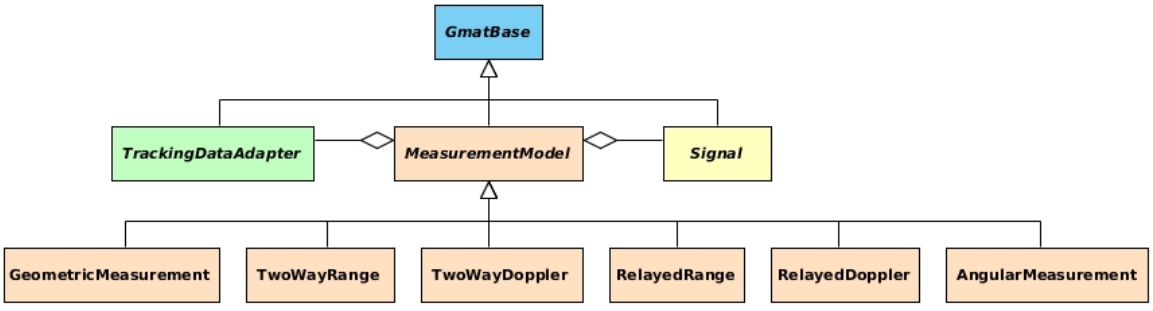

Figure 2: Proposed Measurement Model Classes

The class structure shown here is markedly simpler than that in the current code. Part of this simplification comes from two features of the proposed rework:

1. The CoreMeasurement and MeasurementModel classes have been merged, removing the MeasurementModel container class in the original design and replacing "CoreMeasurement" with the new "MeasurementModel" class.
2. The burden of translating the physics of signals moving through space into the data seen in an observation

has been shifted from the MeasurementModel class onto a set of adapter classes

The new measurement model structure assigns the following responsibilities to the MeasurementModel classes:

- The MeasurementModel classes are used to:
  - Generate the raw data needed for the "computed" side of the residuals calculations
  - Provide the raw data needed for measurement simulation
- Measurements are modeled as one or more signals passing between two or more measurement participants. (The participants endpoints may be referenced as "nodes" in the following text.)
- Nodes of the signal path are specified at measurement participants
- With the exception of "geometric" signals, all signals are modeled using a light time solution for the signal path.
- A measurement is "feasible" if all of the signals comprising the measurement are feasible.
  - Interruption in any signal in the measurement results in the the measurement being "infeasible."
  - Infeasible measurements do not return a calculated value.
- Delays, when needed, can be specified on the measurement model
  - Delays can be specified on a node-by-node basis. Scripting for this feature is TBD.
  - Nonzero delays override data in the participant hardware
  - When omitted or set to zero, the delay at each node can be obtained from hardware attached to the participant at that node
- Users control the corrections applied to the signals comprising the computed measurement through MeasurementModel resources:
  - Relativistic Corrections can be toggled on or off
  - Time corrections - specifically, the ET - TAI correction - can be toggled on or off
  - Tropospheric model corrections can be specified and toggled on or off
  - Ionospheric model corrections can be specified and toggled on or off
- Corrections are specified at the level of a computed measurement.

Open Issues:
Scripting for node-by-node delays is not yet specified. Is it something we need?
Do we ever need to have different correction models on the signals in a single measurement?

Signal Classes

Measurement models in the refactored design consist of a collection of signals passing between measurement participants. This design, shown in Figure 3, models signals between the ground and a spacecraft (both instantaneously as a "GeometricSignal", and, using a more precise "PhysicalSignal" that allows for light time solutions and signal path corrections), and between two spacecraft. Future enhancements will add signals involving celestial bodies.

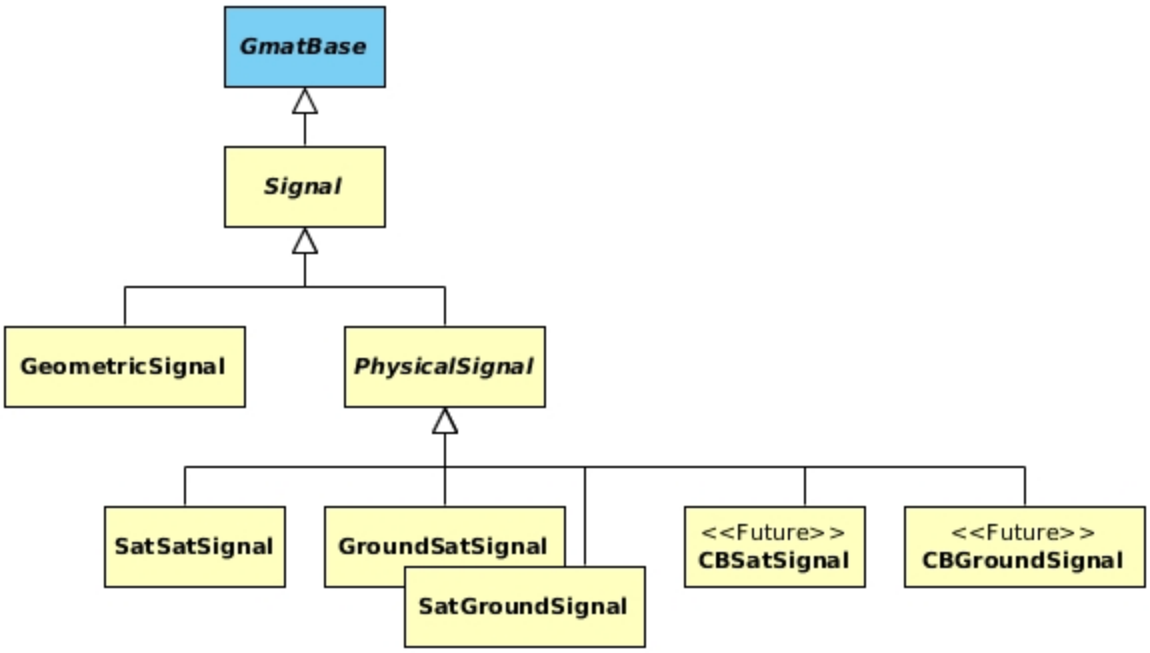

Figure 3: Signal Classes

The Signal Classes Have the following responsibilities and features:

- Modeling the signal path between two measurement participants
- A geometric signal class is included to support the existing instantaneous, uncorrected measurement models. These models are useful for testing other elements of the estimation system.
- The PhysicalSignal intermediate class adds structures needed to support light time solution and measurement corrections.
- All PhysicalSignal classes account for corrections to the measurement path due to:
  - Light travel time between the participants
  - Tropospheric corrections to the path
  - Ionospheric corrections to the path
  - Relativistic corrections to the path
  - Timing differences between the ET and TAI time systems
- All signal classes provide interfaces to account for signal processing delays at each participant
- All signal classes provide interfaces to test the signal path for obstructions that make the path infeasible

Open Issues:
Should the feasibility testing include other considerations? Example: the signal frequency Doppler shifting out of the measurable frequency band at the receiving node.
The figure has separate boxes for uplink (ground to space) and downlink (space to ground). Is there any reason that these should be separate, or can they be modeled as a single ground/spacecraft path?

Tracking Data Adapter Classes

The tracking data adapters provide the connections between the physical modeling performed in the measurement model classes and the observation data stored in the tracking data. Instances of these classes take the signal path

data from a measurement model and transform that data into the form that matches a related observation. The data adapter can supply measurement information on either side of this transformation.

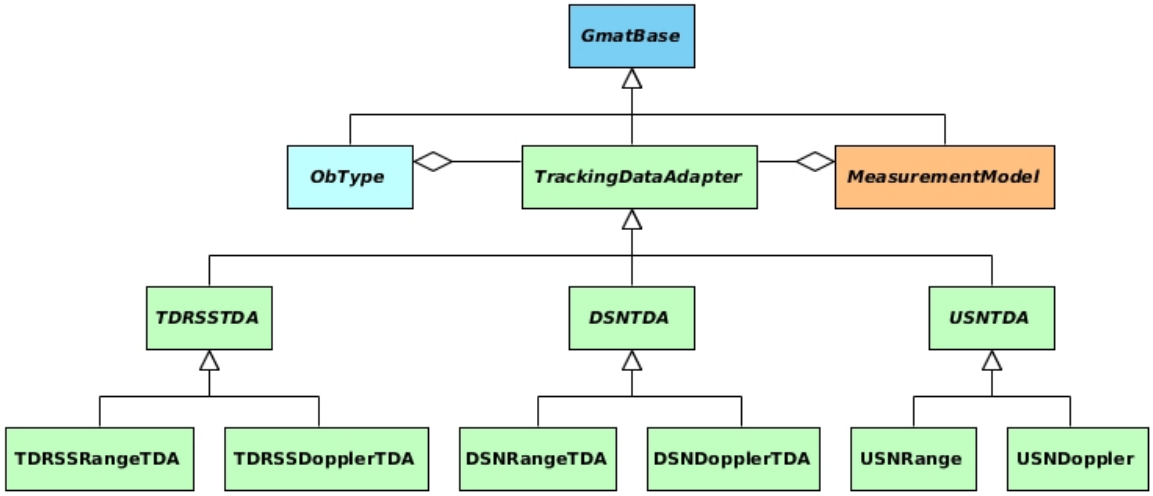

Figure 4: Tracking Data Adapters

The TrackingDataAdapter classes provide these functions:

- Provide the communications layer between measurements and observations
- Reformat measurement data into the form expected for a corresponding observation
- Reformat observation data into the form expected for a measurement, with associated ambiguity when appropriate

Open Issues:

(Here we get to the crux of the issue for the measurement refactoring.) The intent of this design rework is to modularize the measurement model code by breaking the physics of signals between participants apart from the observation data derived from those physical computations. Hence the goal here is have (1) A set of signal building blocks that all measurement models use, (2) a single measurement model for a given collection of signal paths used for a generic type, and (3) an interpreter layer that transforms that generic model into the form used for an associated observation. Does this refactoring work?

Tracking Data File Classes

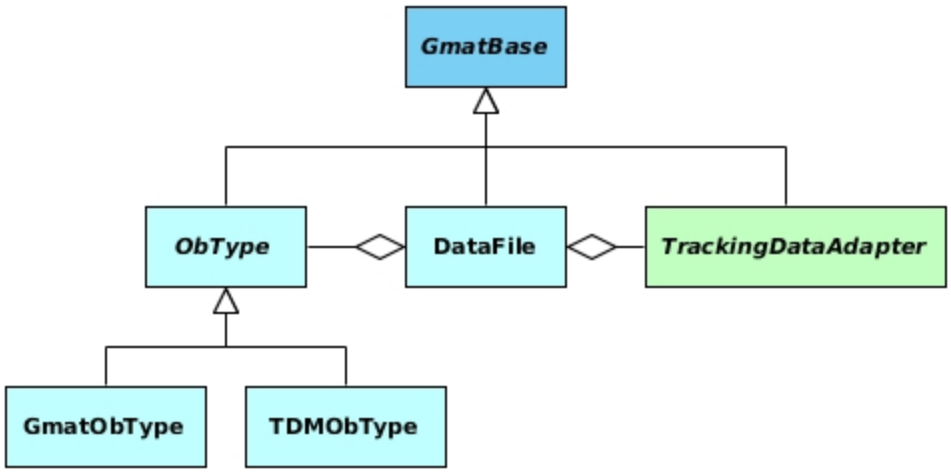

Figure 5: Tracking Data File Classes

Open Issues:

Class Interactions for Estimation and Simulation

The preceding sections provide the class framework for the measurement model subsystem in GMAT. Those classes work collectively to model measurements and relate those measurements to observations reported as a record in a source of tracking data. The data sources in the current GMAT work are data files of tracking data records, in GMAT's internal measurement data format or, by the summer of 2014, in CCSDS TDM format. The static class picture presented here does not show the scope of the interactions between the classes to perform the computations needed to perform estimation. This section fills in those details.

Measurement Model Initialization

To be supplied

Figure 6: Initialization of Measurement Models and Tracking Data

Open Issues:

The Measurement Process: Estimation

During Estimation, GMAT's estimator requests a tracking data record and propagates the simulation to the epoch of that record. It then retrieves the observation from the record and compares that value to a computed value obtained from the associated measurement model, resulting in the "O - C" for the simulation at the observation epoch. That process is shown in Figure 7 and described below.

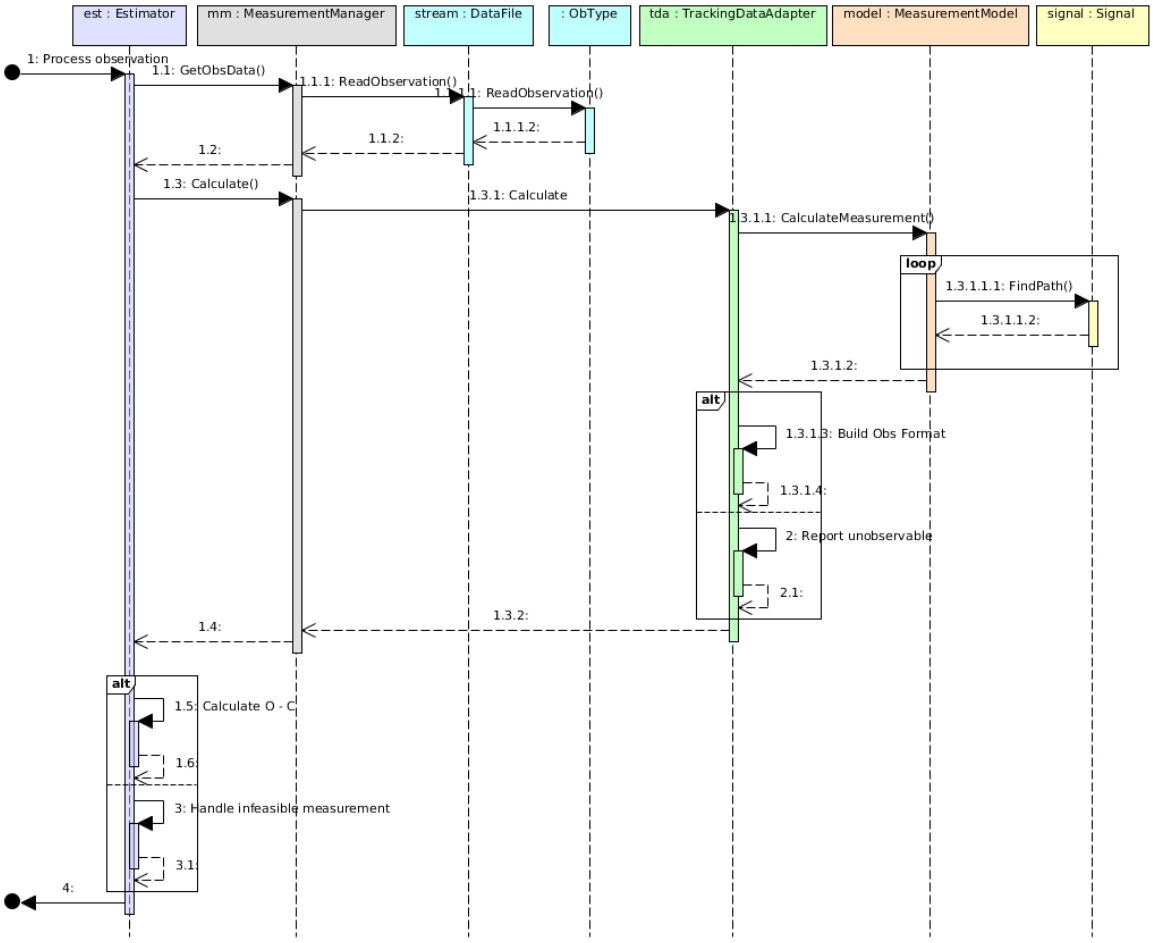

Figure 7: Computation of O - C (Click figure to enlarge)

Open Issues:

The Measurement Process: Simulation

Open Issues: