Designing the SRT control software: Notes to the Antenna package

Andrea Orlati[1]
Simona Righini[2]

1 - I.N.A.F. Istituto di Radioastronomia.
2 – Dip. Astronomia - Università degli Studi di Bologna.

Dicembre 2008

IRA 424/08

DOCUMENT OVERVIEW.

This document illustrates the design of the subsystem "Antenna" which, in the NURAGHE/ESCS control software, is the subsystem in charge of controlling the telescope main drives. The "Antenna" package is not only able to drive the azimuth and elevation axes, but it also includes all the required high-level functionalities to perform a radio observation.

The design is mainly based on three diagrams:

1. *The component diagram*; it reports (see illustration 3) the software components involved in the subsystem in particular the dependencies among them.
2. *The class diagram*; it presents (see illustration 4) all the components with full details and it also describes how they interact in order to meet all the required specifications.
3. *The deployment diagram*; it just shows how the system should be deployed for the final installation and gives also details about the interaction with the hardware devices.

Documentation on the implementation of the software or on the tools used during this process is out of the purposes of this paper. The interested reader is invited to look at the documentation of the software itself.

The following paragraphs will take into consideration the various components one at a time, in order to provide a description of each of them.

Before that, we need to fix some terminology about the coordinates.

FORMAL AGREEMENT ABOUT COORDINATE FRAMES.

Some notes about coordinate frames and coordinate conversion are due (see also illustration 1). The system will work in **equatorial celestial coordinates** (Right Ascension, Declination). All other coordinates will be converted to that frame before being used, included the ephemeris obtained from orbital parameters if, for example, we are dealing with Solar System bodies. The system can also treat the horizontal, and the galactic frames, ecliptic are not supported at the moment but could be added in a near future. At any time, in the system the following categories of coordinates are present:

1. ***Input Equatorial Coordinates***: *the coordinates of the source as given in input by the user or supplied by a catalogue. If coordinates are provided exploiting other frames the system will convert them into the equatorial one.*
2. ***Input Galactic Coordinates:*** *The coordinates of the source as given in input by the user in the galactic frame. In that case the input coordinates are transformed into the equatorial frame.*
3. ***J2000 Equatorial Coordinates:*** *The equatorial coordinates at J2000.0. They could be the same as input coordinates.*
4. ***Apparent Equatorial Coordinates:*** *These coordinates take into account effects like precession, nutation and annual aberration.*
5. ***Apparent Horizontal Coordinates:*** *Horizontal coordinates (Azimuth, Elevation) depend on the site of the observer. They are directly converted from the apparent equatorial ones. The apparent coordinates, both equatorial and horizontal, could also be comprehensive of the user-defined offsets.*
6. ***Raw Horizontal Coordinates:*** *The raw coordinates are the coordinates where the system believes the antenna can find the source. Atmospheric and instrumental effects (e.g. refraction and pointing model) are taken into account.*
7. ***Commanded Horizontal Coordinates:*** *These are the coordinates actually commanded to the antenna mount. They can differ from the raw ones because*

of "instant" measured effects like levelmeters readings or wind corrections from metrology system, etc...

8. ***Encoders Coordinates:*** *These coordinates are read directly from the mount encoders. They measure the current coordinates the antenna is pointing to.*
9. ***Observed Horizontal Coordinates:*** *These coordinates come from the encoder coordinates cleared from all the effects related to the mount. In case of an ideal mount, apparent and observed must be equal.*
10. ***Observed Equatorial Coordinates:*** *These are directly converted from the Observed Horizontal given the site and the time of the observation. They are referred to a standard epoch J2000.0.*
11. ***Observed Galactic Coordinates:*** *They are directly converted from the equatorial ones.*

Applying these conventions the ACU-measured pointing error will turn out to be the difference between the commanded coordinates and the encoders ones.

OBSERVATORY.

It implements all the operations related to the site. For example time computations. It also includes the Delta-UT1 information. Since the required tracking and pointing precision is 2 arcseconds, the difference between UT and UT1 must be taken into account. This value should be updated on a regular basis by an automatic script that collects the data from the web. At the moment in order to make the new DUT1 value available, the component must be restarted and so the whole antenna subsystem.

1. *getSiteInformation()* is an internally used operation that returns in one shot all the parameters regarding the site and the time of observation to all the other components which require to perform coordinate conversions.

The deployment of this component must take care of the different site coordinates.

MOUNT.

This takes care of the Antenna Control Unit (ACU) and, inside the subsystem, is the component that directly deals with the telescope hardware.

Since the Sardinia ACU will be significantly different from the one installed in Medicina the implementations have forcibly many differences. On the other hand, from the subsystem point of view they are functionally identical; in order to exploit this characteristic, a common interface has been added. This way the rest of the subsystem "sees" the mount component though that interface and does not care about the particular mount implementation it is dealing with. In this case, the mount turns out to be a sort of plug-in, installed in the system only during the deployment phase.

One of the common features enlisted in the mount interface that it is worth to highlight is that the ACU can work with various modes. Among them there are: *rate,* the telescope is driven commanding the speed, *preset,* the telescope is commanded to go to fixed positions and *programtrack,* the telescope is ordered to be in a given position at a certain time mark, if a sequence of points is loaded in advance the ACU can interpolate the correct tracking curve. The *changeMode()* can be called to select the desired mode. The *programtrack (ACU_PROGRAMTRACK)* mode was chosen to steer the antenna during normal operation because it allows to bypass all possible TCP/IP latencies and delays that could compromise the tracking precision. In fact, lots of tracking points are uploaded into the ACU in advance, so that a late command cannot compromise the overall behaviour of the telescope. On the other hand, this mode leads to some other difficulties that the component has to deal with. A source

change or a user offset, in fact, will force the component to clear the positions stack into the ACU and upload the new tracking curve in one shot. For that reason it is fundamental to keep in memory the commanded position not yet reached by the telescope and to fill the ACU stack to a reasonable level - that is a trade off between telescope precision and network performance. Too many positions to load at a time may saturate the available bandwidth.
Here a list of items that it is worth to discuss, follows:

1. *Stow*() and *Unstow*() take a long time, so they are designed to be asynchronous. The caller will be informed when the telescope is stowed or unstowed using a callback mechanism.
2. *DeltaTime* is an offset that will be added to the ACU current time. Since the ACU will be synchronized according the UT time, if *DeltaTime* is set to the current value of DUT1, the ACU will be capable to operate in UT1. The hardware of the Medicina VLBI antenna does not support this feature, so this field is just a place-holder.
3. *AzimuthOffset and ElevationOffset* can be considered hardware offsets, i.e. they are immediate corrections that come from measures like levelmeters or metrology. They have nothing to do with the user offsets that can be commanded by the observer.
4. *CommandedAzimuth and CommandedElevation* come out from the sum of the raw coordinates - issued by the operations *preset()* or *programTrack()* - and the offsets. In case the operating mode is the *programtrack*, the last commanded coordinate could also be several seconds in the future so the currently commanded point is the result of the linear interpolation between the two nearest (in time) points of the tracking curve.
5. *Azimuth and Elevation* are the encoders coordinates, the last known real position of the mount as reported by the ACU. Since the network latency is still an issue also when reading parameters from the ACU, this values could not be enough up to date for very precise measurements. For this reason a method *getEncoderCoordinates()* has been added so that the caller can ask for the position of the telescope at any given time. The algorithm is very easy; the ACU answers every position request with a triplet, encoder azimuth, encoder elevation and a time stamp; these data are stored by the software. When asked for the position of the telescope at a certain epoch, the software performs a simple linear interpolation between the two nearest (in time) points in its dataset. If the sampling rate of the positions is high enough the error can be considered negligible. This technique is used in all other parts of the subsytem when a network delay could compromise the results of the observation.
6. *mountStatus* is a monitor of the current status of the mount, that can be used to quickly recognize is the mount is having problems. This field can have three status:
    1. OK: everything is going well.
    2. Warning: a minor problem has been found, the observation can continue, but an investigation should be carried out.
    3. Failure: a critical error has been found, the observation had to be stopped.
7. *Section* indicates the section in which the azimuth axis is at present: Counter Clock Wise(ACU_CCW) or Clock Wise(ACU_CCW). The boundaries of the two sections and, in general, all the parameters that characterize the mount are not hard-coded but read from the Configuration Data Base (CDB) when the component initializes (see illustration 3).

8. *forceSection()* it can be used, before commanding a new position, to force the section of the azimuth rail that the antenna will travel to reach that point. For example if someone wants to make sure that there is enough travel range to track a source from rise to set this method can be called before loading the first point of the tracking curve. After the first point is loaded the preferred section is reset to the default value (ACU_NEUTRAL) that means that the antenna will go to the next point along the shortest trip from the last commanded point. Of course cable wrapper limits are taken into account, so it could be necessary to wrap the whole cable wrapper.
9. *getAntennaErrors()* is similar to *getEconderCoordinates()*, but it returns the pointing errors as reported by the ACU with the corresponding timestamp.
10. *Stop()* immediately ceases every pending command to the telescope and switches the brakes on. In order to unblock the telescope, a new mode must be commanded.

The coordinates reported by this component are all given in degrees as the ACU of the Medicina VLBI Antenna works in degrees (probably the ACU of SRT will, too). Degrees will appear only in this component. All the remaining parts of the system will work in radians since most of the astronomical routines,we are planning to adopt require radians.

POINTINGMODEL.

It gives the azimuth and elevation correction resulting from the pointing model. The pointing model parameters should be read directly from the proper tables stored in the CDB. It provides a method to set the working receiver and a method to read-back the offsets to be applied to the horizontal coordinates, *getAzElOffsets()*.

1. *setReceiverCode():* This method should be called by the Receiver package every time a new receiver is configured. It accepts the receiver code as a string; if the code is not present in the CDB tables an error is returned.

REFRACTION.

It gives the correction due to the refraction generated by the atmosphere. It needs site information and measurements from meteorological sensors: temperature (Celsius degrees), relative humidity (fraction from 0 to 1) and pressure (millibars).

EPHEMGENERATOR.

The subsytem must take care of all the possible observation strategies that directly involve the telescope main drives. Typically we go from the classical sidereal tracking, spanning to satellites or planets tracking, up to the On The Fly scans (OTF), etc... In order to fulfil all these requirements a different generator of coordinates is necessary. At the same time, the Boss (see below) needs to command the telescope without caring about which particular observation is going on at the moment. For that reason it has been defined the EphemGenerator interface that exposes all the features required for a coordinate calculator. Thus, changing the observation mode consists in changing the implementation of this interface, in other words it consists in loading a different component that inherits from the same interface.
All components inherited from this interface generate the tracking curve as a function of time. It also allows the user to provided specific offsets. The time must be UT1 based, so all the coordinates computation are performed using UT1. It needs Observatory for coordinates conversions and time computations. Any observing mode exploits one of these generators to move the antenna accordingly (pure sidereal

tracking, OTF, etc...). The coordinates/angles are expressed in radians. We can say that the coordinates computed here are all *apparent* as defined before.

1. *setHorizontalOffsets*() allows to set azimuth and elevation offsets as they are given by the user. The azimuth offset will be corrected for the cosine of the elevation before applying it.
2. *setEquatorialOffsets*() allows to set right ascension and declination offsets as they are given by the user. The right ascension term will be corrected for the cosine of the declination.
3. *setGalacticOffsets()* allows to set the longitude and latitude offsets as they are given by the user. The longitude is corrected for the cosine of the latitude. Depending on the specific implementation, this feature might be available or not.
4. *getHorizontalCoordinates*() is used internally by the Boss component to get in one shot the apparent horizontal coordinates (suitable to be commanded to the telescope) at any given time.
5. *checkTracking()* is used privately by Boss to check if the antenna is tracking or not. Boss provides the azimuth and elevation (observed) for a given time, the HPBW is also passed. The component checks is the coordinates given as arguments are close enough to the apparent computed by the generator itself. "Close enough" is determined comparing the tenths of HPBW to the mean squared root of the pointing errors.

SKYSOURCE.

It inherits from EphemGenerator. This component can generate a tracking curve so that any pure sidereal tracking can be performed. Several "applications" might need this kind of generator contemporaneously, it has been implemented as a dynamic component so that it can have an undefined number of instances in the system, each of them running as a "private" component of the application. It supports three coordinates frames: equatorial, galactic and horizontal. How these frames are handled is reported in figure 2.

1. *loadSourceFromCatalog*() permits to load the source parameters from a table stored in the ACS Configuration Data Base.
2. *setFixedPoint*() permits to set a fixed point in the horizontal frame, the telescope will not perform any tracking curve. For those familiar with the Field System, it is equivalent to a call to the FS snap command "source=azel,...",
3. *setSourceFromGalactic*(), *setSourceFromEqutorial*() allow to select a source giving directly galactic or equatorial coordinates.

SOLARSYSTEMBODY.

It inherits from EphemGenerator. This component, at present not implemented, is planned to generate a tracking curve for any Solar System body. The final goal is to perform the computation of the tracking curve for planets and minor Solar System bodies starting from their orbital parameters. A faster solution could be to interrogate on-line the JPL-Horizons Ephemeris Generator and automatically download the coordinates.

SATELLITE.

An additional special tool to compute tracking curves for artificial satellites or space debris orbiting around the Earth can be developed exploiting the TLE ("Two Line

Elements", orbital parameters) available on-line. Further investigations about this topic should be carried out.

MOON.

Very simple generator that allows to track the Moon. In the future it could be expanded with other features to calculate the lunar phase, the apparent dimension of the disk and the positions of specific sections – e.g. the limbs.

OTF.

We define "on-the fly scan" a set of one or more proper "subscans" mapping a given region of the sky, with the antenna continuously acquiring data while slewing along a defined path. The OTF component takes care of computing the UT-dependant sequence of pointings needed to execute a single subscan.
Each subscan is identified by a series of initialisation and setup parameters which the component must receive in input through the call of the setSubScan() method:

1. *initAz* (radians) is the azimuth the antenna is pointing to when the component is called
2. *initSector* indicates the CW-CCW section associated to the above azimuth
3. *initEl* (radians) is the elevation the antenna is pointing to when the component is called
4. *initTime* is the UT instant when the component is called
5. *description* is the way the boundaries of the subscan are described. It can be identified by a start and a stop position (SUBSCAN_STARTSTOP) or by a central point and the spans along the axes (SUBSCAN_CENTER)
6. *geometry* is the shape of the subscan with respect to the sky. For any of the supported coordinate frames it can be at constant latitude (SUBSCAN_CONSTLAT) or constant longitude (SUBSCAN_CONSTLON), otherwise it can be a great circle arc (SUBSCAN_GREATCIRCLE). A great circle subscan can only be described in the start/stop mode
7. *lon1, lat1* (radians) depending on the subscan description, these might be longitude and latitude of the subscan start point (if SUBSCAN_STARTSTOP) or of the subscan central point (if SUBSCAN_CENTER)
8. *lon2, lat2* (radians) depending on the subscan description (see below), these might be longitude and latitude of the subscan stop point (if SUBSCAN_STARTSTOP) or the total subscan spans along the frame axes (if SUBSCAN_CENTER)
9. *coordFrame* coordinate frame associated to the input values (lon1, lat1, lon2, lat2). Supported frames: J2000.0 equatorial, horizontal, galactic
10. *subScanFrame* coordinate frame associated to the execution of the subscan. They can differ from the coordFrame only in case of a pointing calibration scan: in this case, a sidereal source whose coordinates are given in a coordFrame - such as J2000.0 equatorial - is then scanned along one of the horizontal axes
11. *direction* in case the description is set as SUBSCAN_CENTER (for constant longitude or constant latitude subscans) it is necessary to specify the direction in which the subscan must be performed. This is accomplished by specifying if the varying coordinate must increase (SUBSCAN_INCREASE) or decrease (SUBSCAN_DECREASE) along the scan
12. *startUT* (100 ns since 1852-10-15 00:00:00) is the UT instant at which the antenna must reach the subscan start position, such beginning data acquisition

13. *subScanDuration* (seconds) is the time duration of the subscan

It must be noticed that methods to set user-defined offsets to the coordinates are implemented but, at present, empty.

Given the above parameters, OTF performs some preliminary checks and operations:

1. it adds an acceleration ramp before the actual subscan and a deceleration ramp after it. These ramps are defined by the constant acceleration value which is written in the Configuration Database
2. knowing the initial antenna position and computing the subscan ramp-start coordinates to be commanded, it checks if the antenna can reach the target within the designed UT instant
3. it simulates the subscan and controls that the azimuth and elevation speeds lie within the instrumental ranges along the whole subscan. For equatorial and galactic constant longitude and constant latitude subscans, the scanning speed is constant in those sky frames, but can lead to very high values for azimuth and elevation speeds. The same applies to great circle scans, where the "local speed" is computed as a constant angular speed along the arc.

OTF also retrieves some instrumental parameters from the CDB, such as the DUT1 value, the HPBW, the maximum antenna accelerations and speeds, etc...

The getHorizontalCoordinate() method is then called from AntennaBoss and activates the production of the horizontal coordinates to be commanded at the given UT. AntennaBoss asks for these coordinates in advance, obtaining a sequence of pointings which will constitute the subscan. When this interrogation takes place, OTF also gets the actual coordinates which the antenna is pointing to when the method is called – by means of the checkTracking() method, and confronts them to the previously predicted ones, checking if the antenna is tracking correctly (i.e. if the pointing error is lower than 0.1*HPBW).

ANTENNABOSS.

This component is the supervisor of subsystem, it is in charge of starting the observation and setting up the telescope mount and of checking the system status and health.

In order to use the antenna it is necessary to call the setup() method; in unstows the telescope, then it synchronizes the ACU with the host computer system time and finally it commands the ACU_PROGRAMTRACK as operational mode.

Since the observed and raw coordinates reported as attributes of this component are referred to the present time (coming from the last read of the encoders coordinates) and since the network latencies and the ACU_PROGRAMTRACK mode can cause the read to be late, the component allows to query for coordinate pairs at a given time; if the requested time is not available the boss will answer with the linear interpolation between the nearest time marks. The exposed methods are *getObservedEquatorial, getObservedHorizontal, getObservedGalactic, getRawCoordinates*.

The component also allows to set user offsets in all the supported coordinate frames. In each frame, the longitude is corrected for latitude cosine. In practice what this component does is to call the corresponding *setGalacticOffset()*, *setEquatorialOffset()* and *setHorizontalOffset()* of the current generator. Every time a new tracking is started the offsets are reset.

It exposes a lot of attributes, like:

1. *refractionOffset, pointingAzimuthOffset, pointingElevationOffset* are the computed offset to deal with pointing model and refraction so that observed coordinates can be derived.

2. *enabled* attribute reports about the availability of the component and indirectly of the subsystem, if not enabled the component does not expose all its functionality and the same is for the subsystem. For example: if disabled, the Boss won't forward the commands to the mount and so this can be considered a sort of "simulation" mode. See the operation *enable()* and *disable()* to alter this flag.
3. *correctionEnabled* attribute shows if the correction for refraction and pointing model has to be applied or not. In case the correction is on (the attribute is true in his case) before loading a position point into the mount the Boss will add the offsets computed by the specific component (Refraction and PointingModel). In case the correction is disabled the raw coordinates are coincident with the apparent ones. This field can be changed through the methods *correctionEnable()* and *correctionDisable()*
4. *status* indicates if the system led by this component is working properly (MNG_OK) or not (MNG_FAILURE). If the corresponding flag (*mountStatus*) in the Mount component is not MNG_OK, status will report the same value, otherwise it will show the inner status of the boss itself. The status of the boss may be altered by many events – e.g. an exception in the communication with other components. A status will be considered valid up to 3 seconds, then if not renewed it will be considered cleared. A status change to a lower priority level can not take place if the present status is not cleared. If the component is not enabled, the status reported is always MNG_WARNING.
5. *generatorType* indicates which ephemeris generator is in use at present. As mentioned before, generators can be loaded and unloaded according to the specific observation the telescope is asked to perform. For internal it is also possible to have a reference to the instantiation of the running generator – *getGeneratorCURL()*. For example a client that would like to show which apparent coordinate is commanded at the moment.

The "observed" coordinates, as defined in the first section of this document, are computed here starting from the "encoders" ones. The encoders coordinates are read directly from the mount. More specifically, when a request to perform a subscan is issued – *startTracking()* - the component performs the following steps.

1. First of all it determines which generator has to be used and, in case, it takes charge of loading it.
2. Then the component gets the tracking information from the ephemeris generator (apparent coordinates), the correction offsets are collected from the refraction and pointing model (raw coordinates). These coordinates are passed to the mount. This cycle is repeated until a tracking curve is uploaded for an adequate amount of time in advance. Figure 5 illustrates schematically this algorithm.
3. At the same time it collects from the mount the "encoders" coordinates and the offsets, so that a "trace back" in the computation of coordinates can be done in order to calculate the observed ones (both horizontal and equatorial).
4. raw and observed coordinates are accessed using a time stamp, TCP latency and delays force to interpolate linearly between two "real" samples in order to avoid to flood the network when readings of these values are required frequently.

Boss is also in charge of publishing to a common notification channel the information about the tracking and the status. The data are produced at least once per second

and every time the last published information changes. The tracking information is retrieved by calling *checkTracking()* in the current ephemeris generator component.
The component has also the *command()* operation, allowing to issue commands or requests to the component itself via a sequence of human-readable instructions. This operation has one string argument that will contain the text command. Only a limited set of instructions can be parsed and executed by the AntennaBoss via this operation. The grammar of these text commands has been defined but it's still open to updates. The chosen grammar is simple and it can reasonably be compared to a SNAP-like syntax (the VLBI Field System command language). The command is followed by a command separator symbol and then by the comma-separated list of arguments. Some wildcard characters are allowed. For homogeneity it is highly advisable that also the rest of the system adopts the same grammar. An example:

1. "*source=moon*"
2. "*source=3c123*"
3. "*source=4258+2010,441050.5,552250.4,2000*"
4. "*scan=GREATCIRCLE,CENTRE,EQUATORIAL,440000.0,220000.0,460000.0,230000.0*"

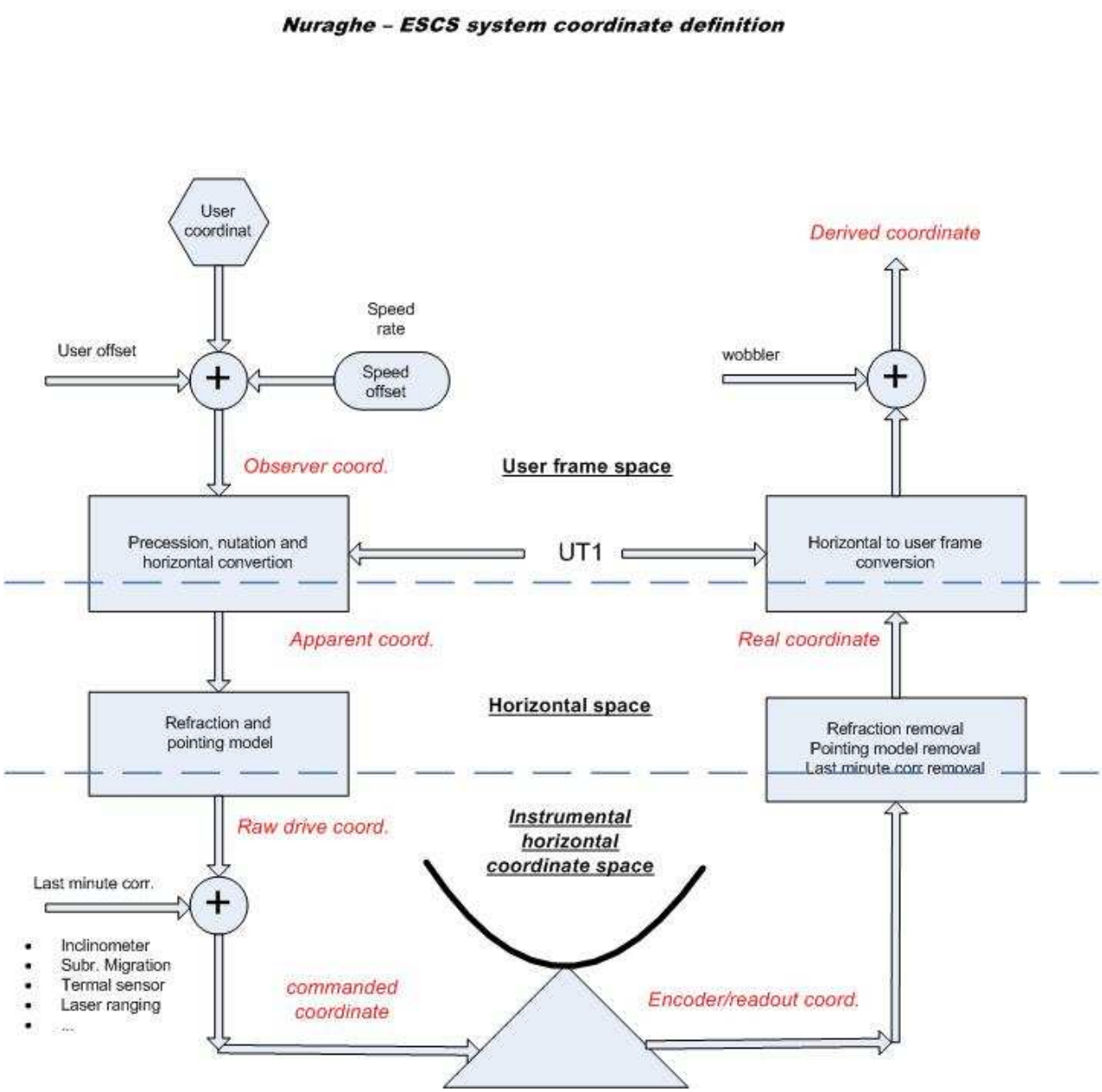

Illustration 1 - The coordinate definition and computation.

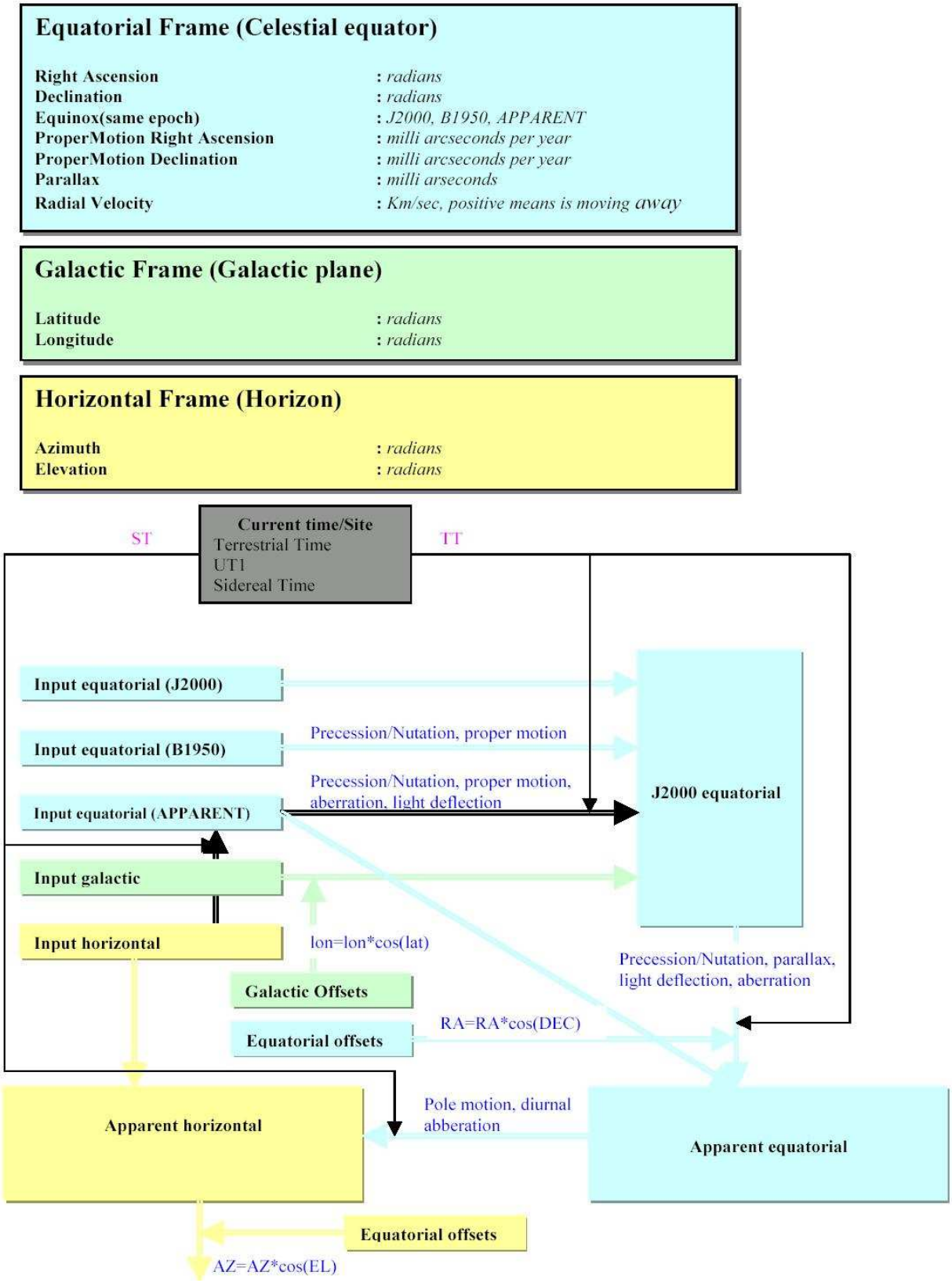

Illustration 2 - Coordinates conversion between supported frames

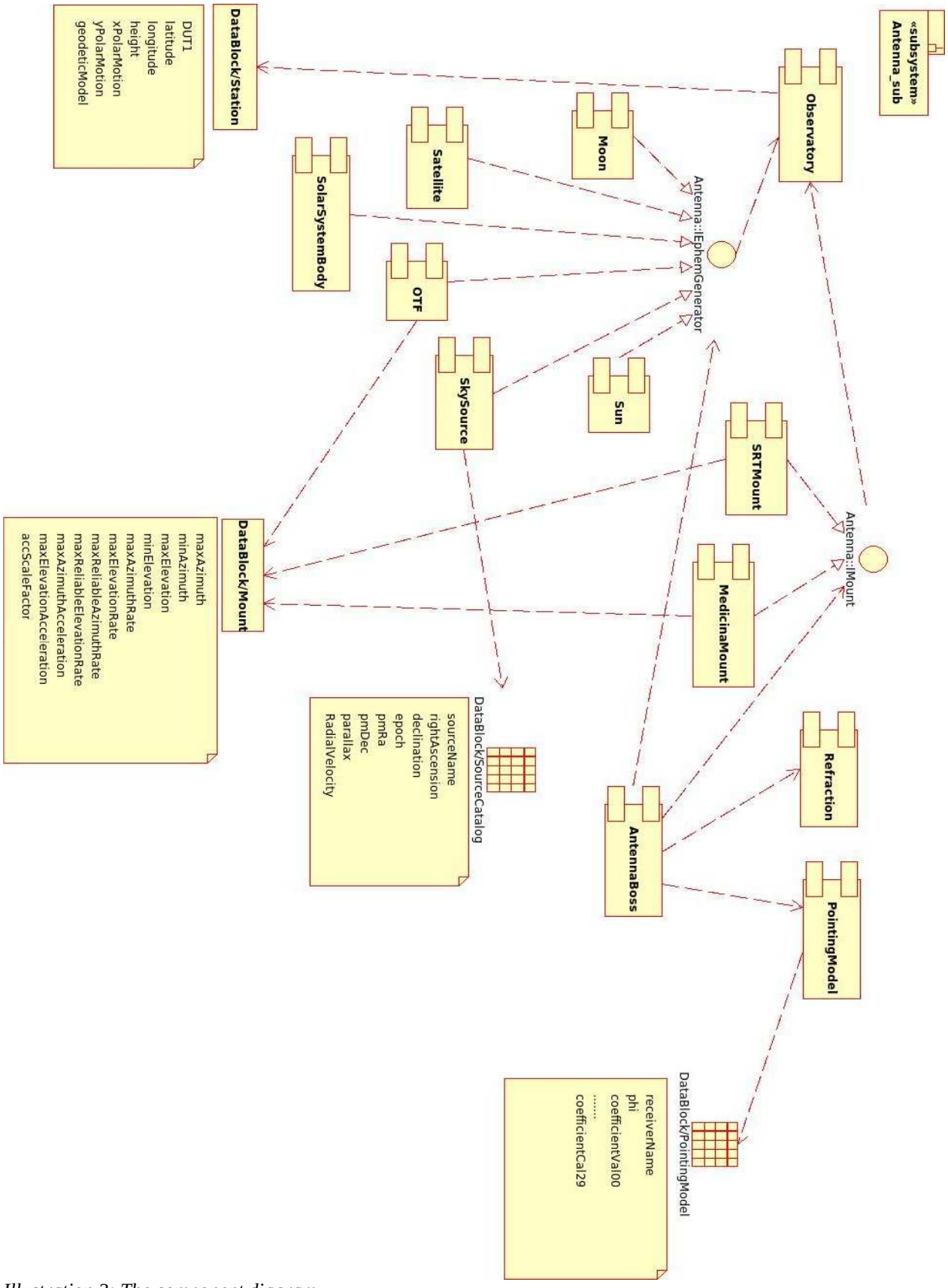

Illustration 3: The component diagram

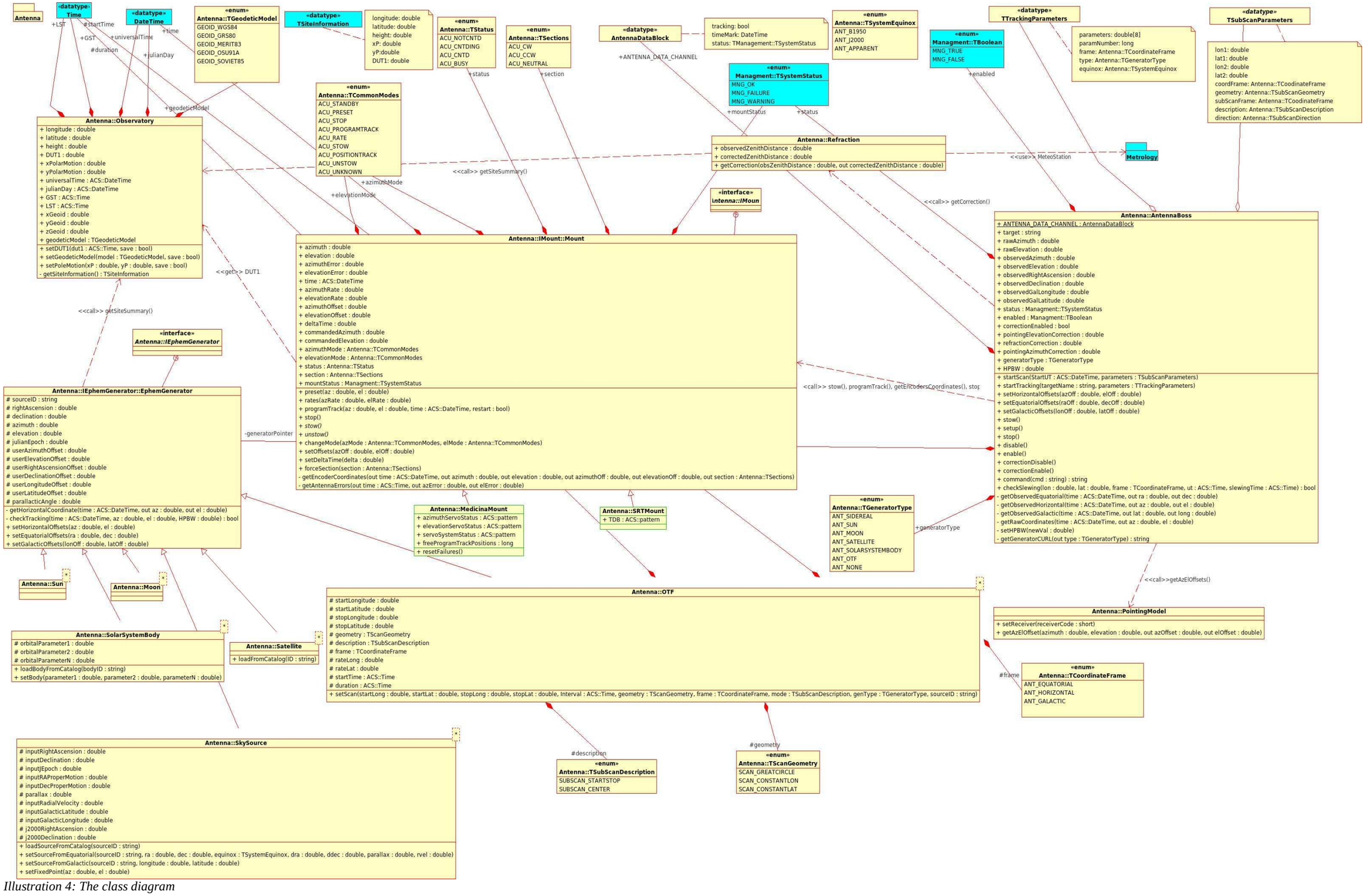

Illustration 4: The class diagram

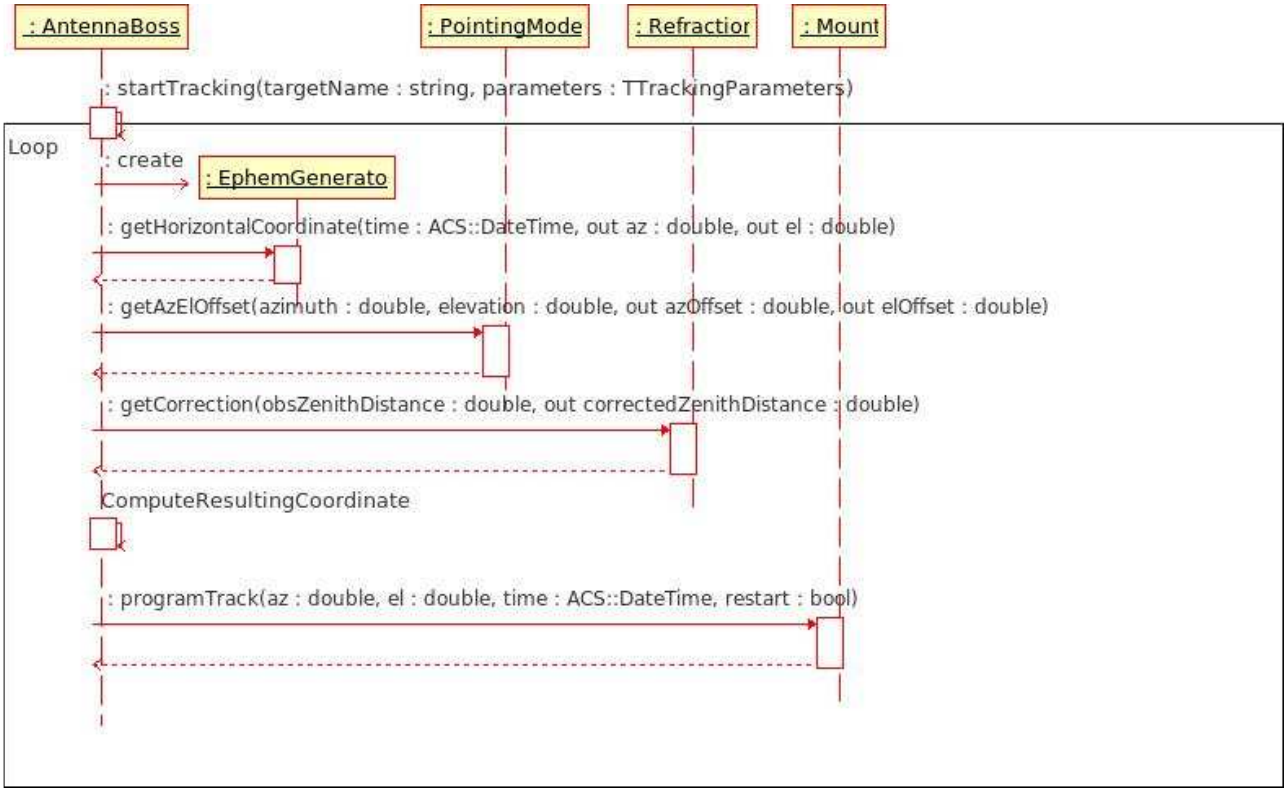

Illustration 5: Sequence diagram that illustrates the startTracking method of the Boss component