Comparison with AstroPy

This page outlines and compares the astropy implementation against the IVOA Coords data model. The goal is to provide a mapping to illustrate the compatibility of the Astropy implementation model to the IVOA model. To date, I haven't seen a formal 'model' for the Astropy packages (but may find one in this effort), so the comparison is from a breakdown of the Classes and Relations defined in the implementation packages.

Reference:

- AstroPy docs: https://docs.astropy.org/en/stable/
- Coords model: pdf,html

AstroPy packages related to Coords (and the greater IVOA model suite):

- astropy.units - Units and Quantities
- astropy.coordinates - Astronomical Coordinate Systems
- astropy.time - Time and Dates
- Other packages related to extended model suite (Measurement, Cube, TimeSeries)
    - astropy.uncertainty - Uncertainties and Distributions
    - astropy.nddata - N-dimensional datasets
    - astropy.timeseries - Time Series

astropy.units:

- "handles defining, converting between, and performing arithmetic with physical quantities, such as meters, seconds, Hz, etc. It also handles logarithmic units such as magnitude and decibel."
- "does not know spherical geometry or sexagesimal (hours, min, sec): if you want to deal with celestial coordinates, see the astropy.coordinates package"
- Quantity: combines a value and a unit, where 'value' may be a scalar, sequence, or array
- includes 'dimensionless'
- handles equivalencies, such as that between wavelength and frequency.

astropy.coordinates:

- "provides classes for representing a variety of celestial/spatial coordinates and their velocity components, as well as tools for converting between common coordinate systems in a uniform way."
    - this is restricted to the spatial domain, and includes the concept of transforms (between standard space frames and representations)
- Class inheritance diagram
- **SkyCoord**: container class, provides convenient API to content, helper methods, etc.. Exposes attributes of underlying objects at this level, which somewhat hides the underlying hierarchy, but is more convenient to the user.
    - positions:
        - coordinate values: SkyCoord provides API to underlying values at this level.
            - API catered to Frame/Representation.
                - via 'named attributes' which depend on the Frame. eg: if frame='icrs' => c.ra, c.dec. if frame='galactic' => c.l, c.b
                - representing Frame/Representation-centric spatial coordinate types
            - values themselves exist at the Frame and Representation level
                - Frame values are frame-centric according to the default representation type: (ra,dec,dist), (l,b,dist), (az,alt, dist)
                - Representation values are named according to the coordinate space: (lon,lat,dist), (x,y,z), (rho, phi, z)
        - Quantity value may be array...
            - 'for efficiency when performing the same operation on multiple coordinates'
        - supports sexagesimal notation (in/out), the conversion is managed at the Coord level? (Quantities do not support it)
    - velocities:
        - associates differentials with the position: eg. (ra, dec,dist) + (pm_ra, pm_dec, radial_velocity)
        - may have multiple differentials associations (using Unit to identify the nature of the derivative (eg: 's' = time derivative))
            - so can have velocity and acceleration, and derivatives w.r.t. other properties.
    - Frame:
        - string key - eg 'icrs' used to instantiate specialized Frame class
            - Frame classes know how to convert to other, related Frames (see Transforms)
        - There is a hierarchy of Built-in Frames which can be mapped to the StandardReferenceFrame List.
            - Built-in Frames include the representation, and therefore, the coordinate value'
            - would need to check built-in frame specs for additional metadata associated with the particular frames (eg: FK4. equinox)
                - **NOTE**: equinox value is a Time type.

        - Can generate Custom frame
        - Transforms
            - Frame level internally supports transforms between frames (eg: 'icrs' 'galactic')
            - transform_to(frame) where 'frame' == string key ('fk5'), Frame class, or frame instance.
                - this form also allows conversion to AltAz; usage of that frame may require Earth rotation info which are "automatically downloaded from the International Earth Rotation and Reference Systems (IERS) service when required"
    - Representations
        - the 'coordinate space' (Spherical, Cartesian, Cylindrical) is described as a 'representation'
        - Transforms
            - Representation level handles conversions between spaces
                - managed through Cartesian via to_cartesian() and from_cartesian() methods implemented by all Representation types.
        - 'SphericalRepresentation' is default for built-in frames
        - coordinate values
            - available at this level for efficiency in performing operations on arrays of coordinates.
            - coordinate names here are Space-centric: Spherical(lon,lat,dist), Cartesian(x,y,z), Cylindrical(rho, phi,z)
                - may be different from names at SkyCoord level... especially for Spherical.
        - built-in representations:
            - SphericalRepresentation: "Representation of points in 3D spherical coordinates"
                - lon: longitude angle, range 0 to 360 deg, wrapped
                - lat: latitude angle, range -90 to +90 deg.
                - distance: radial distance
                - differentials: BaseDifferential[*]
            - UnitSphericalRepresentation: "Representation of points on a unit sphere"
                - same as spherical, but no distance
            - PhysicsSphericalRepresentation: "Representation of points in 3D spherical coordinates (using the physics convention of using `phi` and `theta` for azimuth and inclination from the pole)."
                - phi: angle, range 0 to 360 deg, wrapped
                - theta: angle, range 0 to 180 deg
                - r: radial distance
                - differentials: BaseDifferential[*]
            - CartesianRepresentation
                - x: 'x' component of the point
                - y: 'y' component of the point
                - z: 'z' component of the point
                - differentials: BaseDifferential[*]
            - CylindricalRepresentation: "Representation of points in 3D cylindrical coordinates."
                - rho: distance from 'z' axis to the point
                - phi: azimuth, angle range 0 to 360, wrapped
                - z: 'z' coordinate of the point
                - differentials: BaseDifferential[*]
    - Differentials:
        - The differentials are sibling types to the Representations, and provide differentials in the various 'spaces'
        - The nature of the differential is stored as the 'key' in the container object (not internal to the Differential itself)
        - SphericalCosLatDifferential: d_lon_coslat, d_lat, d_distance
        - UnitSphericalCosLatDifferential: d_lon_coslat, d_lat
        - SphericalDifferential: d_lon, d_lat, d_distance
        - UnitSphericalDifferential: d_lon, d_lat
        - CartesianDifferential: d_x, d_y, d_z
        - CylindricalDifferential: d_rho, d_phi, d_z
        - PhysicsSphericalDifferential: d_phi, d_theta, d_r
        - RadialDifferential: d_distance
    - EarthLocations
        - sub package with location of sites (observatories) on Earth, not sure what the frame is there..
            - EarthLocation.of_site('Apache Point Observatory') => <EarthLocation(-1463969.30185172, -5166673.34223433, 3434985.71204565) m>
        - name, address, location.
- Coordinates
    - three tiered system
        - representations: "a particular way of storing a three-dimensional data point (or points), such as Cartesian coordinates or spherical polar coordinates"
        - frames: "particular reference frames like FK5 or ICRS, which may store their data in different representations, but have well-defined transformations between each other"

        - frames know about representations; may change attribute name (eg: Galactic/Spherical == (l,b), ICRS/Spherical == (ra, dec).. renaming (lon,lat))
    - high-level class: "uses the frame classes, but provides a more accessible interface to these objects as well as various convenience methods and more string-parsing capabilities."
        - ie: a container object providing an interface to the underlying content

astropy.time:

- "provides functionality for manipulating times and dates. Specific emphasis is placed on supporting time scales (e.g. UTC, TAI, UT1, TDB) and time representations (e.g. JD, MJD, ISO 8601) that are used in astronomy"
- components
    - value(s) - scalar, list, or numpy array
    - format - 'how to interpret the values':
        - time.FORMATS dictionary; currently contains 18 entries. eg: 'fits', 'iso', 'jd', 'byear', 'jyear', 'ymdhms'
        - 'cxcsec' = Chandra X-ray Center seconds from 1998-01-01 00:00:00 TT
    - scale
        - time.SCALES dictionary; currently 8 (tai, tcb, tcg, tdb, tt, ut1, utc, local)
- each 'format' has a corresponding Class
- time from epoch formats (TimeOffset)
    - cxcsec, unix, gps provide offsets from a fixed reference date
        - each has an 'intrinsic' time scale, can't tell if that means 'fixed' or just 'default' (TT, UTC, TAI) respectively
- TimeDelta
    - result of difference between Time types
    - the scale MUST be of type where 1day == 86400s.
    - don't see anything here which stores the reference date.

Comparison/Compatibility:

- Coordinates:
    - Frame vs Coordinate system
        - Astropy 'BaseCoordinateFrame' is equivalent to the model 'SpaceSys' element, including both the 'Frame' and 'CoordSpace' concepts.
    - Association of Coordinate value to the corresponding Axis
        - The astropy.coordinates package seems to allow only spatial coordinates associated with standard coordinate spaces.
            - there is no concept of a CoordSpace as a collection of Axis.
            - Representations merge the model Coordinate 'values' with various Standard Coordinate Spaces to form space-centric versions of the Coordinate.
                - eg: Point + standard SphericalCoordSpace - SpaceFrame => SphericalRepresentation
                - eg: Point + standard CartesianCoordSpace - SpaceFrame => CartesianRepresentation
                - eg: Point + customized SphericalCoordSpace - SpaceFrame => PhysicsSphericalRepresentation
        - astropy.coordinates has no independent relation between a coordinate and its axis... the axis specification is built into the definition of the Representation type:
            - a 'SkyCoord' in Spherical Representation has
                - 'lon' is an angular value with range 0..360 deg, wrapped - by definition
                - 'lat' is an angular value with range -90:90 deg - by definition
            - a 'SkyCoord' in Cartesian Representation has perpendicular axes
                - 'x' a Quantity, no range restriction, not wrapped.
                - 'y' a Quantity, no range restriction, not wrapped.
                - 'z' a Quantity, no range restriction, not wrapped.
            - These are equivalent to instances of the Standard Coordinate Spaces defined in the model
        - Point => SkyCoord; where the constraint to use only standard coordinate spaces allows denormalization of content to more efficiently operate on them.
    - The coordinate value(s) are available at all three levels of the implementation
        - By merging the model Coordinate with standard CoordSpace concepts, the coordinate value becomes accessible by the Frame through SpaceSys. The various Frame types provide coordinate values via named attributes catered to the given Frame (c.ra, c.dec).
            - I'm interpreting these as essentially aliases to the underlying Representation coordinates assuming the representation is 'default'.
            - I'm assuming the code handles the translation if the SkyCoord.representation is not the 'default' and users attempt to access the values via these named attributes.
        - SkyCoord provides an API so that the coordinate values appear to reside at the SkyCoord level.
            - a 'SkyCoord' in "ICRS" frame and Spherical Representation has (ra, dec, distance)
            - a 'SkyCoord' in "ICRS" frame and Cartesian Representation has (x, y, z)
            - a 'SkyCoord' in "Galactic" frame and Spherical Representation has (l, b, distance)
    - Only standard coordinate spaces

        - The IVOA model allows users to customize the coordinate space specification.
            - limit axes to a finite domain (eg: chip coordinates)
            - to specify where 0 is in a 'pixel' (left, center, right, somewhere in between)
            - to define 'standard' spaces not yet included in the model (eg: PhysicsSpherical)
        - Dealing only with standard coordinate spaces, this package has no knowledge of Discrete, or Binned axis types.. only Continuous.
    - The transforms described here are going between various standard reference frames
        - These are not formally defined (ie: not modeled content, but implementations of 'commonly known transformations'.
        - so I'm not sure where Pixel coordinates come into play in Astropy. (image Pixel SkyCoord), and defining the PixelSpace
            - there is the AST package which knows the Transforms, but I'm curious how the pixel space and coordinates are defined.
    - IVOA model references an external list for supported reference frames and positions.
        - the current list covers most, but maybe not all of the Frame types in the astropy package
        - some astropy Frames show additional metadata not currently in SpaceFrame. These could be accommodated by extension
            - AltAz Space/Frame with associated IERS metadata.
            - LSR with Barycenter velocity
- Time:
    - similar to Coordinates, the value and associated axis/frame are more tightly bound than in the Coords model.
        - again, there is no customization of the axis, which is compatible with the default CoordSpace for TimeSys
    - astropy.time 'Frame' seems to only consist of the time scale.
        - is the use case which requires refPosition/refDirection rare enough to warrant extracting these to a subclass of TimeFrame?
    - astropy.time includes Epoch as a Time type where the model does not.
        - this was a comment by FB as well, during the review process.
    - astropy.time supports far more 'formats' than currently supported by the model.