Models and VOTables

~ Part I ~

Working with MIVOT

L. Michel

Let’s Take a Simple Example

What you have

VOTable 1

RA/DEC

VOTable 2

_RAJ2000/_DECJ2000

VOTable 3

RA_CORR/DEC_CORR

VOTable 4

RA_DET/DEC_DET

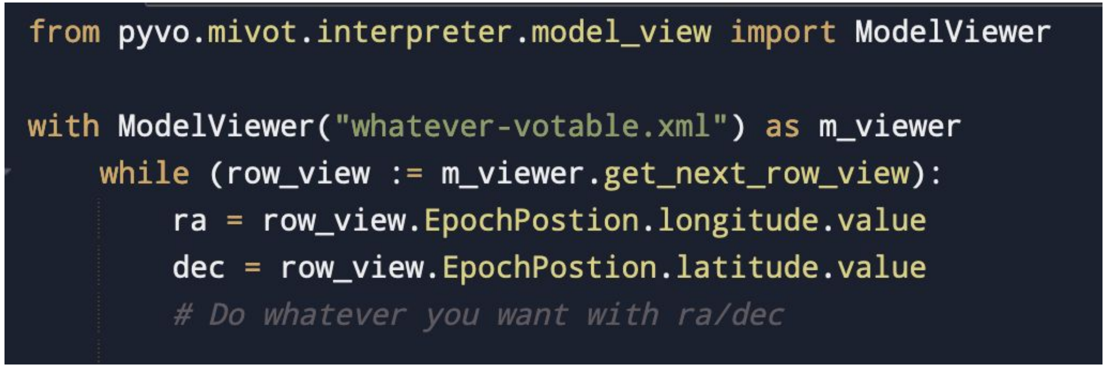
VOTable 5

RA_PNT/DEC_PNT

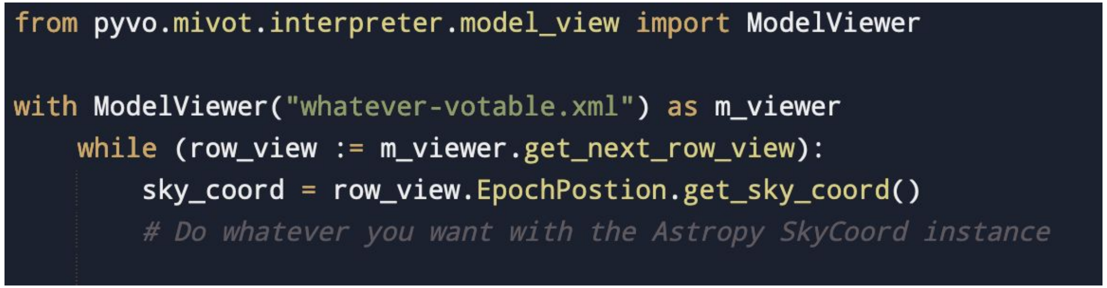
VOTable 6

GLON/GLAT

What we want to do

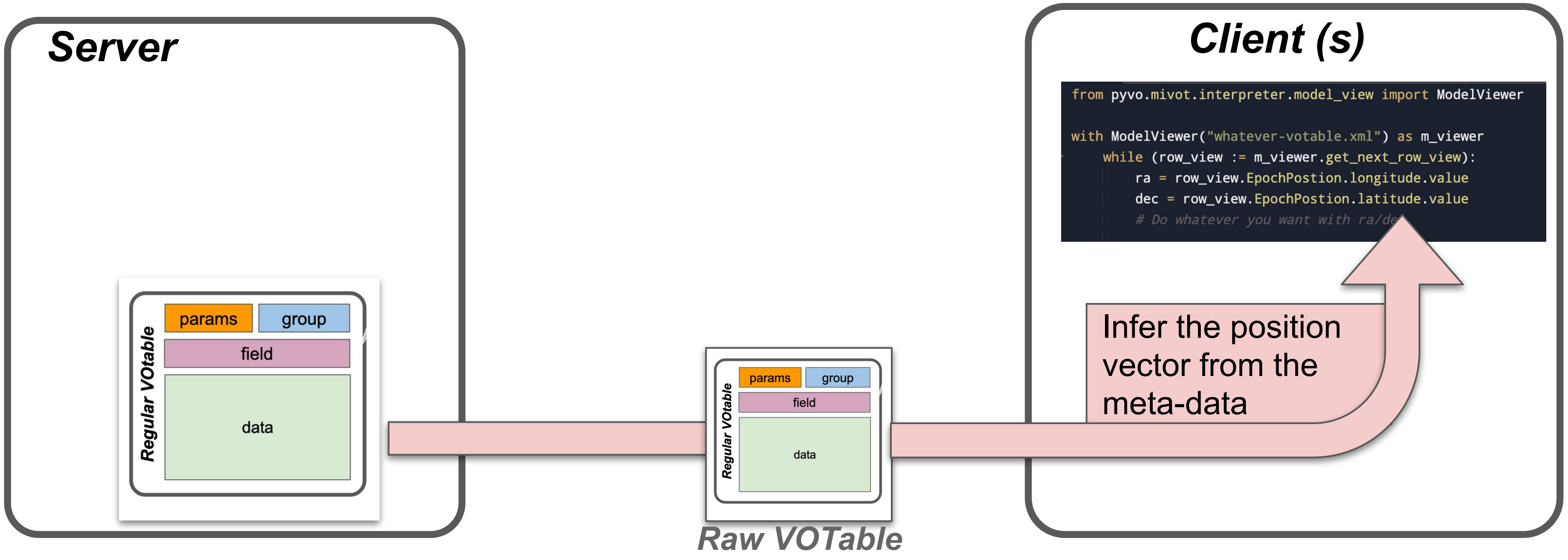

```python
from pyvo.mivot.interpreter.model_view import ModelViewer

with ModelViewer("whatever-votable.xml") as m_viewer
    while (row_view := m_viewer.get_next_row_view):
        ra = row_view.EpochPostion.longitude.value
        dec = row_view.EpochPostion.latitude.value
        # Do whatever you want with ra/dec
```

Or even better


```python
from pyvo.mivot.interpreter.model_view import ModelViewer

with ModelViewer("whatever-votable.xml") as m_viewer
    while (row_view := m_viewer.get_next_row_view):
        sky_coord = row_view.EpochPostion.get_sky_coord()
        # Do whatever you want with the Astropy SkyCoord instance
```

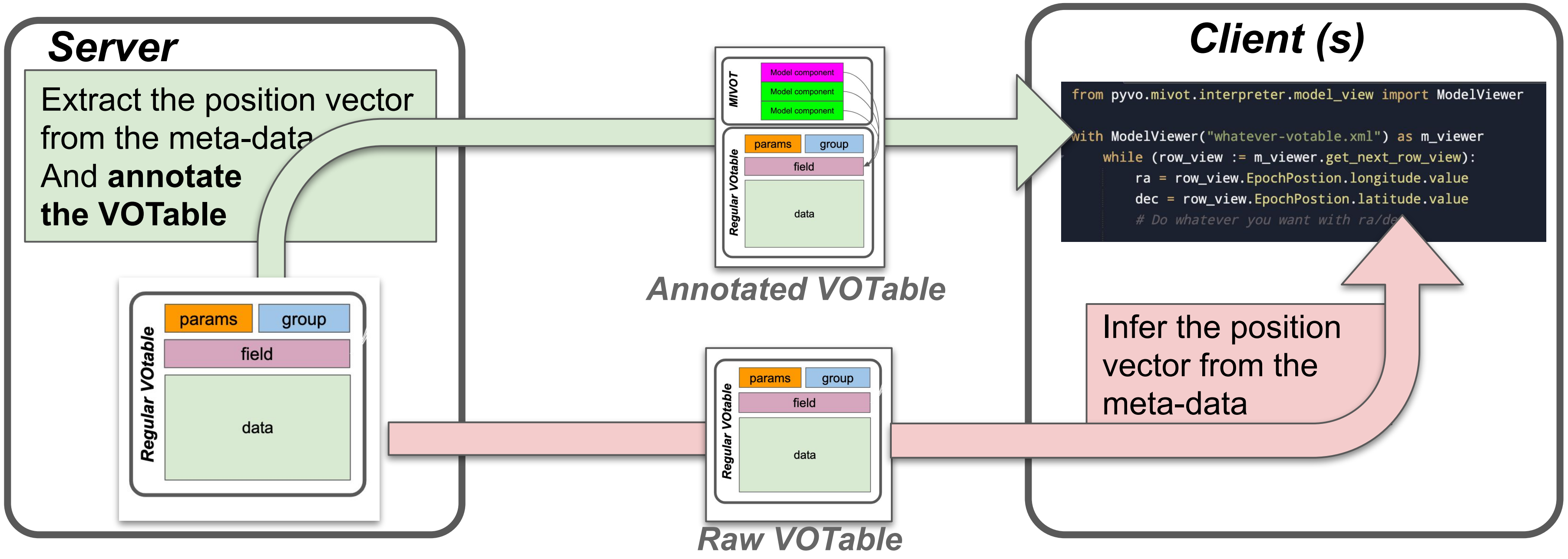
- **Goal**:
  - Get a straightforward access to the scientific quantities of my data sets.
  - Facilitate the implementation of new data patterns

Data Annotation on Server Side



```
from pyvo.mivot.interpreter.model_view import ModelViewer

with ModelViewer("whatever-votable.xml") as m_viewer
    while (row_view := m_viewer.get_next_row_view):
        ra = row_view.EpochPostion.longitude.value
        dec = row_view.EpochPostion.latitude.value
        # Do whatever you want with ra/de
```

Data Annotation on Server Side



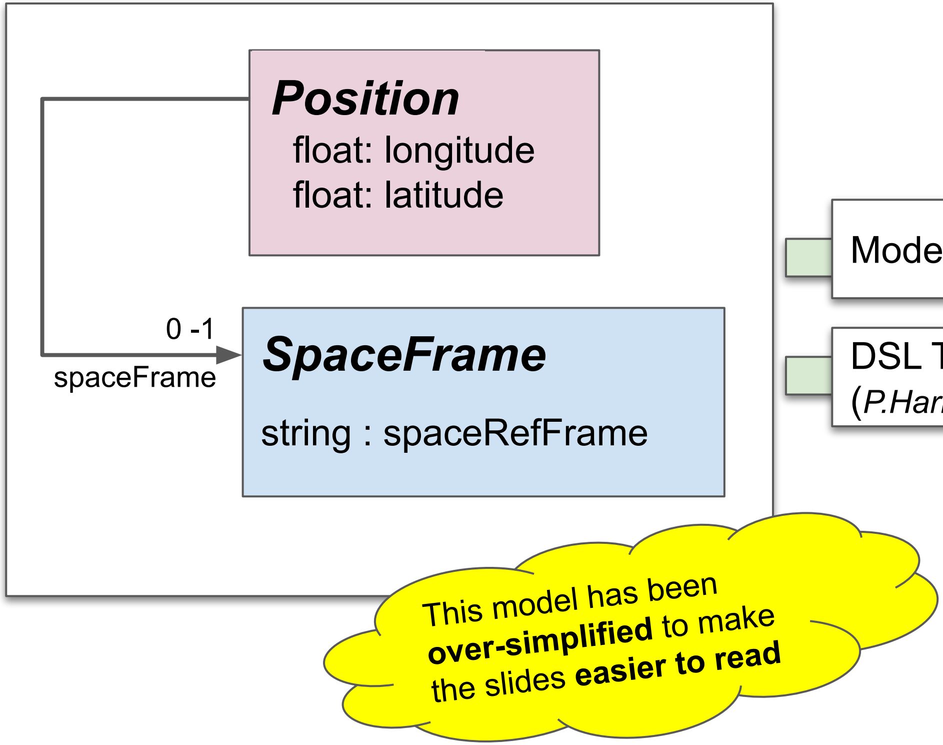
Extract the position vector from the meta-data
And **annotate the VOTable**

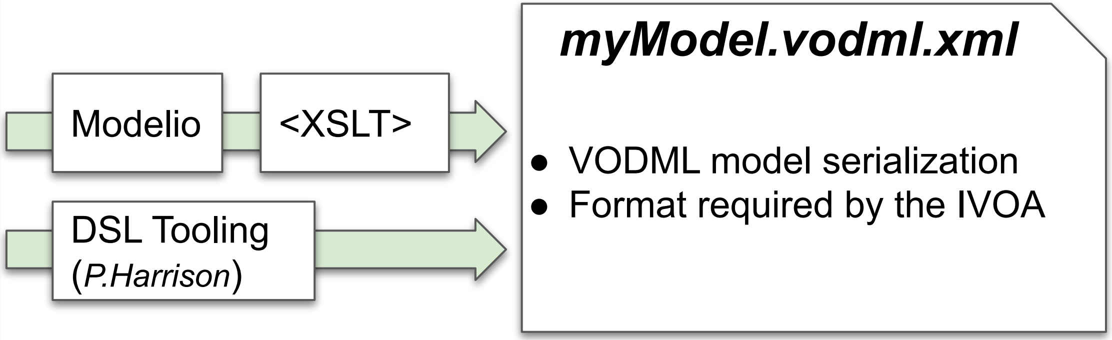
```
from pyvo.mivot.interpreter.model_view import ModelViewer

with ModelViewer("whatever-votable.xml") as m_viewer
    while (row_view := m_viewer.get_next_row_view):
        ra = row_view.EpochPostion.longitude.value
        dec = row_view.EpochPostion.latitude.value
        # Do whatever you want with ra/de
```

Infer the position vector from the meta-data

- **Mapping data on a reference model at server level makes more sense**
    - Avoid data misinterpretation
    - Make sure all clients will understand the same things
    - Done once for ever: save development time for clients
- **This is the purpose of MIVOT**

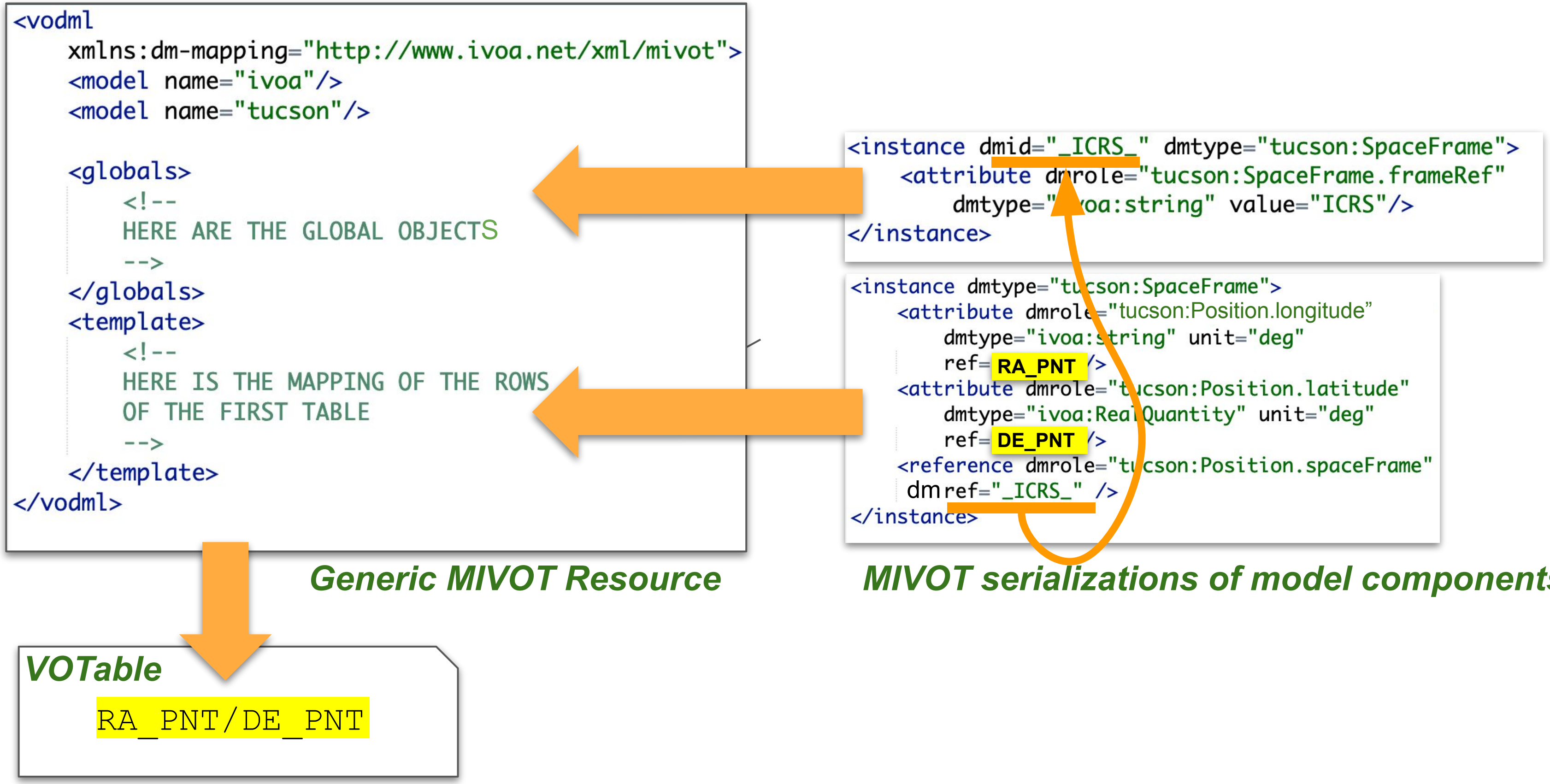
Step 1: Choose or Build a VODML Model





- VODML model serialization
- Format required by the IVOA

Step 2: Annotation Block above the Data Table



```
<vodml
    xmlns:dm-mapping="http://www.ivoa.net/xml/mivot">
    <model name="ivoa"/>
    <model name="tucson"/>

    <globals>
        <!--
        HERE ARE THE GLOBAL OBJECTS
        -->
    </globals>
    <template>
        <!--
        HERE IS THE MAPPING OF THE ROWS
        OF THE FIRST TABLE
        -->
    </template>
</vodml>
```

```
<instance dmid="_ICRS_" dmtype="tucson:SpaceFrame">
    <attribute dmrole="tucson:SpaceFrame.frameRef"
        dmtype="ivoa:string" value="ICRS"/>
</instance>
```

```
<instance dmtype="tucson:SpaceFrame">
   <attribute dmrole="tucson:Position.longitude”
      dmtype="ivoa:string" unit="deg"
      ref= RA_PNT />
   <attribute dmrole="tucson:Position.latitude"
      dmtype="ivoa:RealQuantity" unit="deg"
      ref= DE_PNT />
   <reference dmrole="tucson:Position.spaceFrame"
    dm ref="_ICRS_" />
</instance>
```

For What Purpose

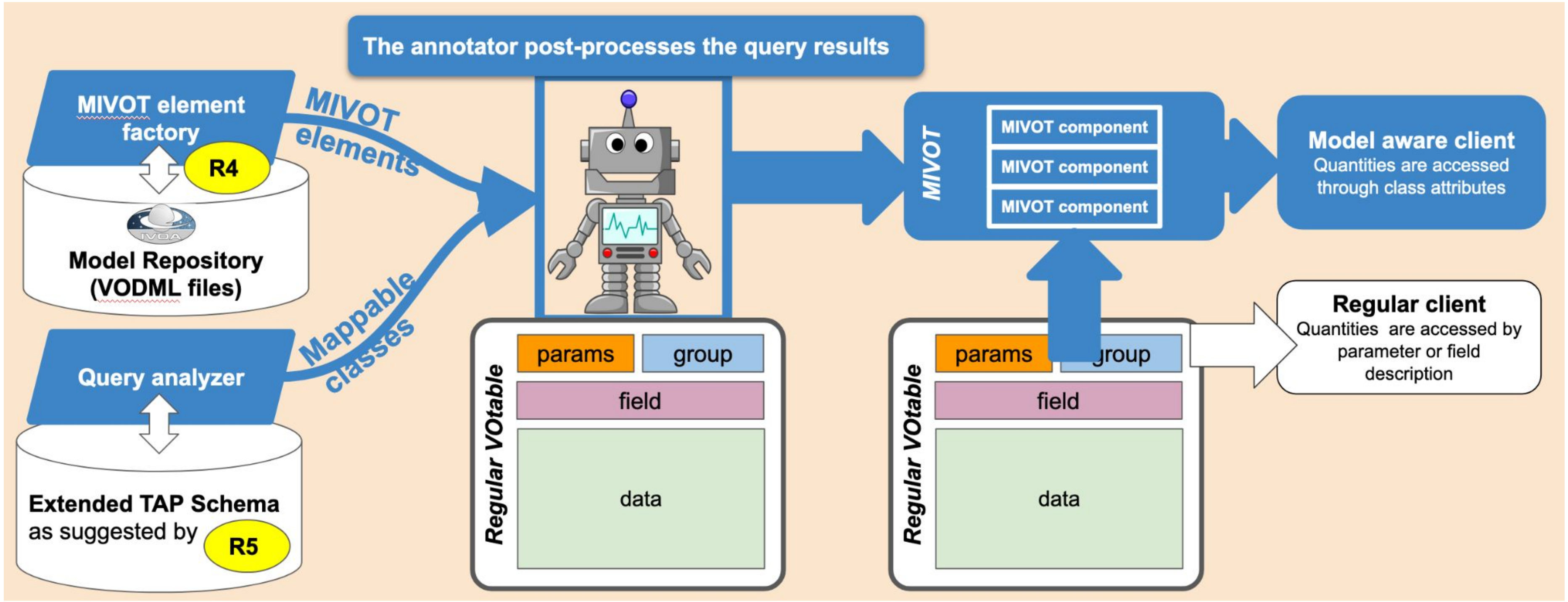
~~This model has been **over-simplified** to make the slides **easier to read**~~

The MIVOT workflow has not been designed to just handle positions.
(we know how to do it for decades)

It has been designed to improve the data description and to eventually add missing metadata in a standard way

- Epoch propagation (*apps1* session this afternoon)
- Fine description of the errors ([a]symmetrical, ellipse, covariance, correlation …)
- Dataset metadata (Model in progress MCD)
- Dataset provenance (IVOA REC)
- Photometric data (brightness, color, filters…)
- High energy data (outcome of the HEIG)
- Any other suggestion

Annotation Engine in a TAP Server (not standard)



R4

MIVOT Elements Generation

Command issued from the validator package

Selection of one concrete class

```
$ mivot-snippet-gen  “coords.SpaceFrame” “RefLocation=CustomSpaceLocation”
```

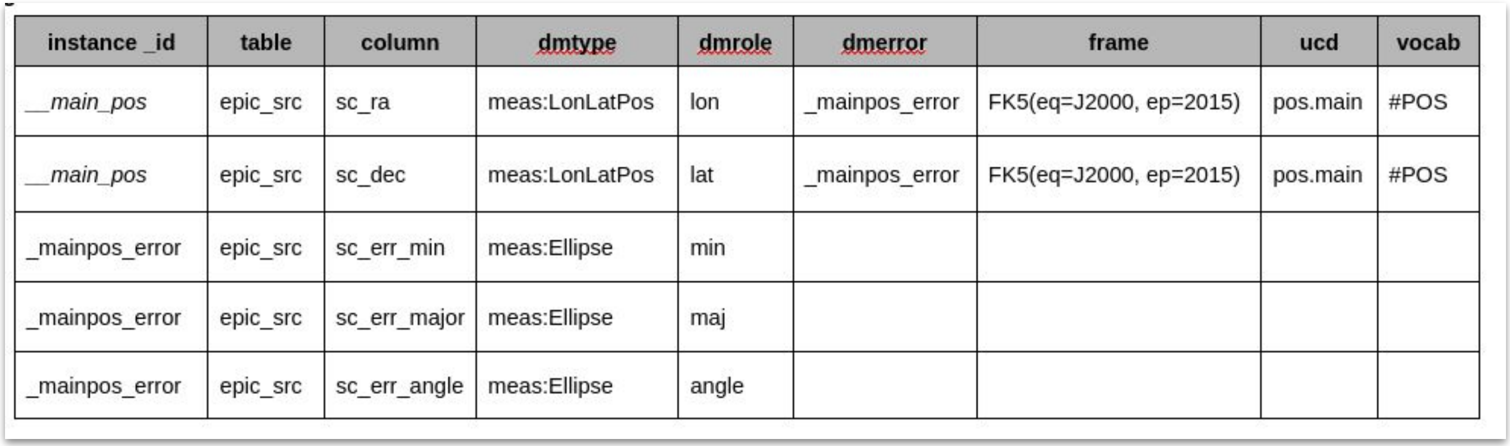
Class to be generated

R5 Providing the TAP Server with the Mapping Rules

One proposal to add one table per supported model to the TAP_SCHEMA
e.g., **`CUBE_mivot,MANGO_mivot`** binding table columns with model classes.

This approach allows data to be mapped on multiple models
It avoids to fill **`META_SCHEMA.columns`** with NULL cells

We could also consider as an alternative to add those columns
`META_SCHEMA.columns` instead of adding new tables .

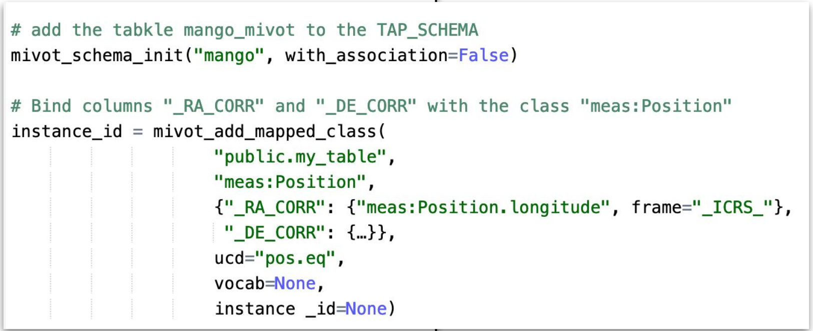


instance _id	table	column	dmtype	dmrole	dmerror	frame	ucd	vocab
__*main_pos*	epic_src	sc_ra	meas:LonLatPos	lon	_mainpos_error	FK5(eq=J2000, ep=2015)	pos.main	#POS
__*main_pos*	epic_src	sc_dec	meas:LonLatPos	lat	_mainpos_error	FK5(eq=J2000, ep=2015)	pos.main	#POS
_mainpos_error	epic_src	sc_err_min	meas:Ellipse	min				
_mainpos_error	epic_src	sc_err_major	meas:Ellipse	maj				
_mainpos_error	epic_src	sc_err_angle	meas:Ellipse	angle				

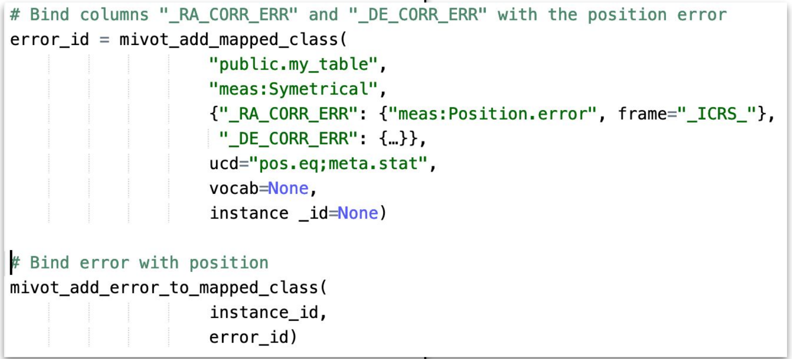
VOLLT Add-on

- **Operate Query Response Post-processing**
  - Easy to implement with the VOLLT API design
  - Direct access to the parser allows to easily determine which model classes can be mapped
- **Python administration tool**
  - Manage the TAP_SCHEMA extension
- **On GitHub**
  - Nothing but one Wiki page

https://github.com/lmichel/vollt-mivot-extension



```
# add the tabkle mango_mivot to the TAP_SCHEMA
mivot_schema_init("mango", with_association=False)

# Bind columns "_RA_CORR" and "_DE_CORR" with the class "meas:Position"
instance_id = mivot_add_mapped_class(
                    "public.my_table",
                    "meas:Position",
                    {"_RA_CORR": {"meas:Position.longitude", frame="_ICRS_"},
                     "_DE_CORR": {…}},
                    ucd="pos.eq",
                    vocab=None,
                    instance _id=None)
```



```
# Bind columns "_RA_CORR_ERR" and "_DE_CORR_ERR" with the position error
error_id = mivot_add_mapped_class(
                    "public.my_table",
                    "meas:Symetrical",
                    {"_RA_CORR_ERR": {"meas:Position.error", frame="_ICRS_"},
                     "_DE_CORR_ERR": {…}},
                    ucd="pos.eq;meta.stat",
                    vocab=None,
                    instance _id=None)

# Bind error with position
mivot_add_error_to_mapped_class(
                    instance_id,
                    error_id)
```

Mivot Validator

- **Public Python package**
  - Process annotated VOTable
- **2 validation levels**
  - XML Schema (XSD)
    - VOTABLE
    - MIVOT syntax
  - Model compliance
    - Annotation against declared models
- **Misc (required for above item)**
  - MIVOT snippet generator (see above)

https://github.com/ivoa/mivot-validator

MIVOT and the Registry

No need to declare the MIVOT capability to use annotations

- The presence of the MIVOT block does not impact the regular parsing

Some ideas to declare the annotation capability in the registry

- MIVOT adds a new feature to the VOTable format that allows a new way of consuming data. The standard has been designed so that this new functionality can be ignored by regular clients, as services are not required to provide annotated data anyway.
- This flexibility might require a standard way of declaring the availability of annotated data. The registration of this capability must be applicable to any service running any protocol that may provide data as VOTables.
- This can be done by defining a new IVOA Registry capability that would list the mapped models and that could be appended to the service capabilities.

A extra UWS parameter to ask query results to be annotated

- Example: get CUBE DM annotations :
  `query?format=application/x-votable+xml&` **`datamodel=cube`**`&query=SELEC`
- If the query result cannot be annotated, an empty MIVOT block is set with a FAILED status.

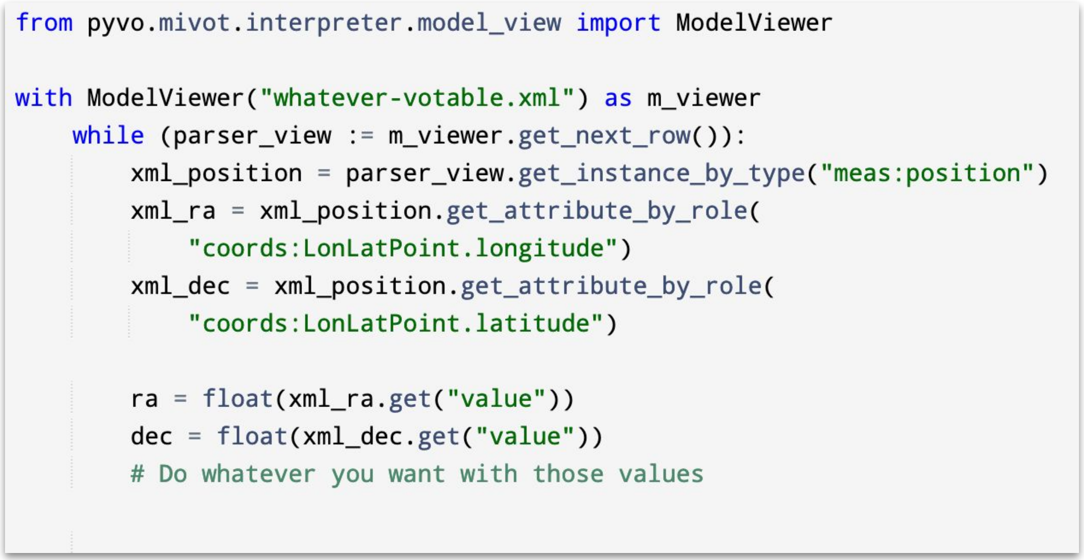
(our) CLIENT ARCHITECTURE

- **2 CLIENT APIs - ONE ARCHITECTURE**
  - **LEVEL 0:** Embedded in the VOTable parser
    - Extract the MIVOT block from the VOTable
    - Basic parsing operation
    - No further processing
  - **LEVEL 1-4:** Embedded in a VO aware package
    - Resolve reference
    - Provide a model view of each data row
    - Provide tools to make the best from data model views

Level number grows with the abstraction level.

1-2 Hide the Column Parsing

- **XML parser view:**
  - The API level 1 (and 2) provides access to XML elements as they are set after processing the current table row.
  - Various elements and attributes can be accessed using XPATH queries.
  - XPATH queries are based on both **`@dmtype`** and **`@dmrole`** MIVOT element attributes
- **The column parsing is hidden.**

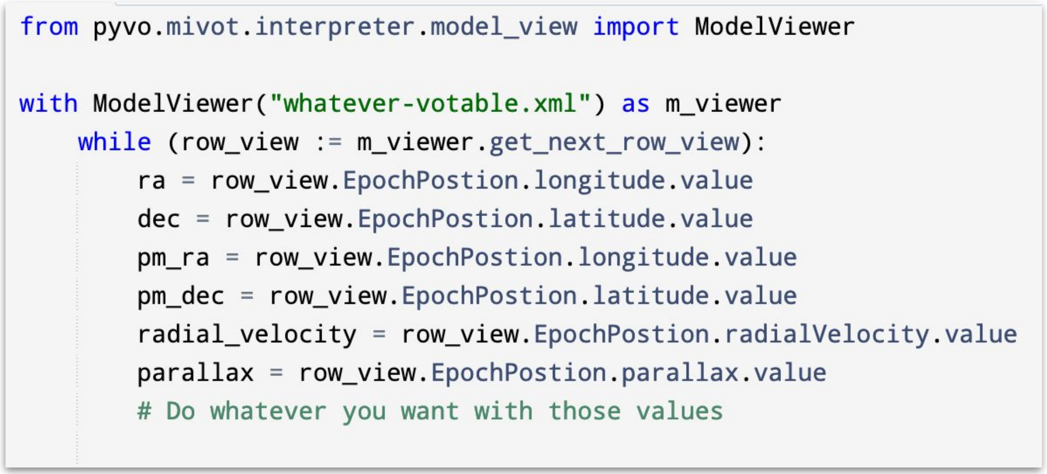


```
from pyvo.mivot.interpreter.model_view import ModelViewer

with ModelViewer("whatever-votable.xml") as m_viewer
    while (parser_view := m_viewer.get_next_row()):
        xml_position = parser_view.get_instance_by_type("meas:position")
        xml_ra = xml_position.get_attribute_by_role(
            "coords:LonLatPoint.longitude")
        xml_dec = xml_position.get_attribute_by_role(
            "coords:LonLatPoint.latitude")

        ra = float(xml_ra.get("value"))
        dec = float(xml_dec.get("value"))
        # Do whatever you want with those values
```

3 Hide both Column Parsing and MIVOT Identifiers

- **Dynamic Objects:**
  - The API level 3 builds dynamic objects matching the MIVOT instances.
  - These dynamic classes are model-related although their compliance is not checked.

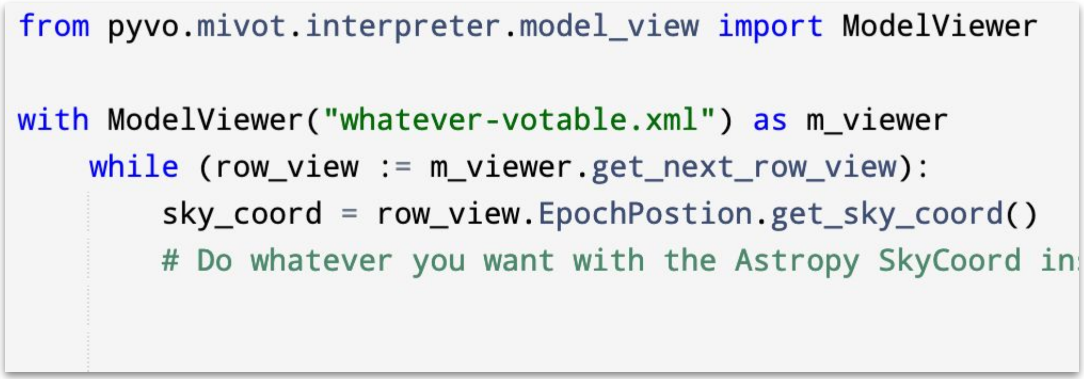
- **The column parsing is hidden.**
- **MIVOT identifiers are hidden**



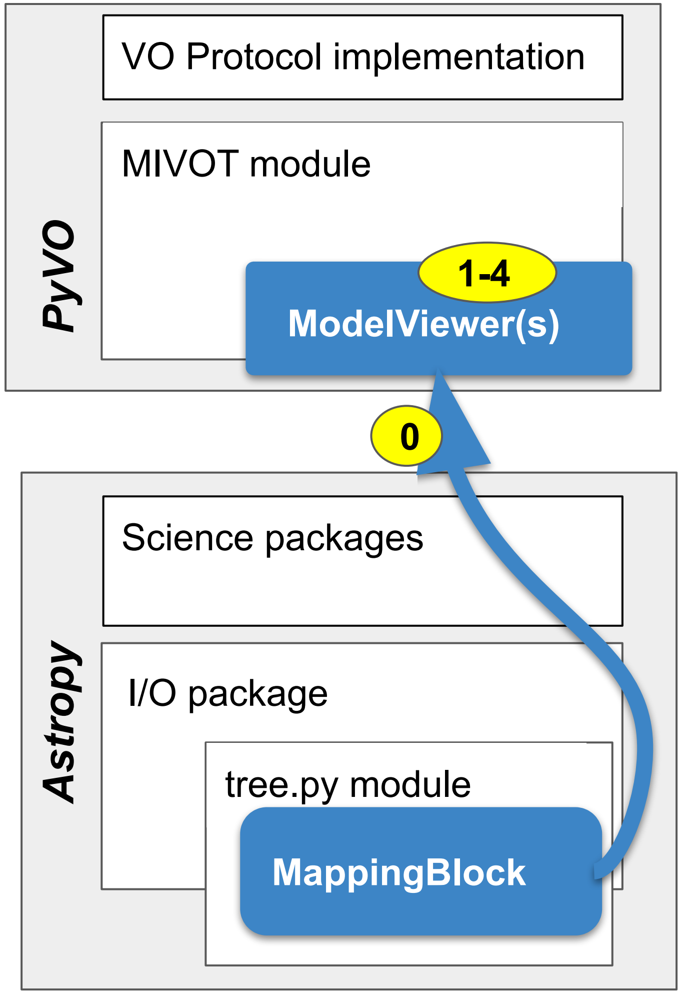
```
from pyvo.mivot.interpreter.model_view import ModelViewer

with ModelViewer("whatever-votable.xml") as m_viewer
    while (row_view := m_viewer.get_next_row_view):
        ra = row_view.EpochPostion.longitude.value
        dec = row_view.EpochPostion.latitude.value
        pm_ra = row_view.EpochPostion.longitude.value
        pm_dec = row_view.EpochPostion.latitude.value
        radial_velocity = row_view.EpochPostion.radialVelocity.value
        parallax = row_view.EpochPostion.parallax.value
        # Do whatever you want with those values
```

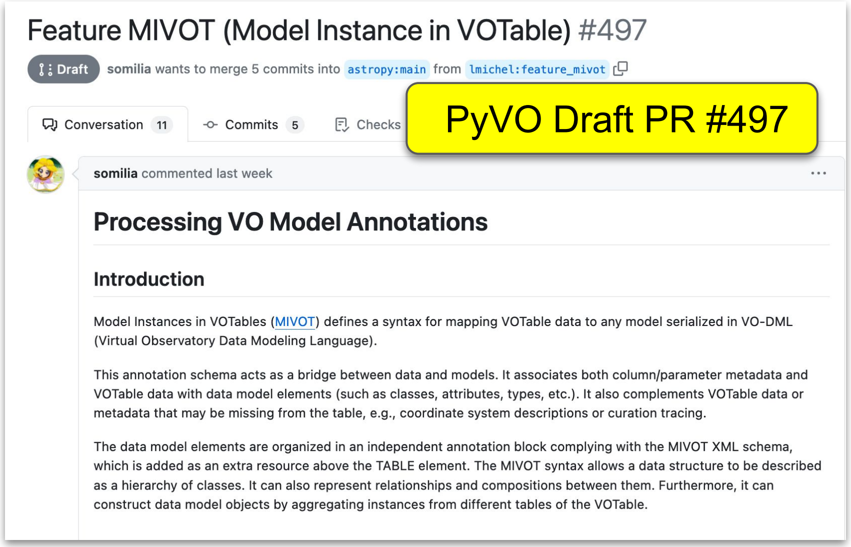
4 Hide Model Elements

- **Build library instances:**
  - The API level 4 builds library instances (e.g., AstroPy SkyCoord or Quantity) directly from the MIVOT block
- **The column parsing is hidden.**
- **MIVOT identifiers are hidden**
- **Models elements are hidden**



```
from pyvo.mivot.interpreter.model_view import ModelViewer

with ModelViewer("whatever-votable.xml") as m_viewer
    while (row_view := m_viewer.get_next_row_view):
        sky_coord = row_view.EpochPostion.get_sky_coord()
        # Do whatever you want with the Astropy SkyCoord in
```





Feature MIVOT (Model Instance in VOTable) #497

Draft somilia wants to merge 5 commits into `astropy:main` from `lmichel:feature_mivot`

Conversation 11 Commits 5 Checks

PyVO Draft PR #497

somilia commented last week

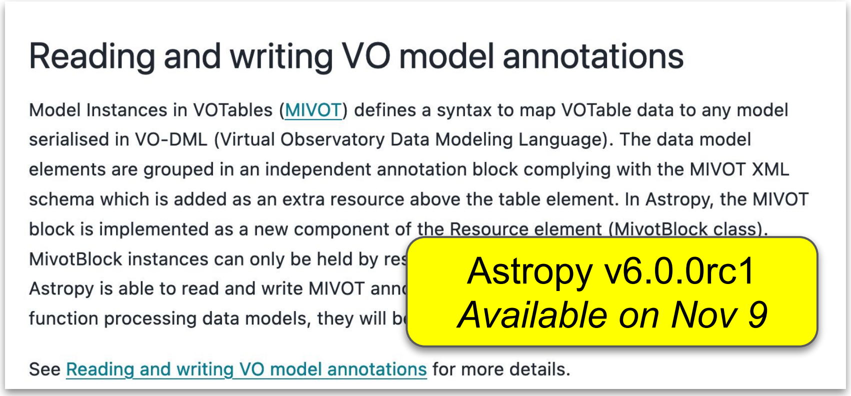
Processing VO Model Annotations

Introduction

Model Instances in VOTables (MIVOT) defines a syntax for mapping VOTable data to any model serialized in VO-DML (Virtual Observatory Data Modeling Language).

This annotation schema acts as a bridge between data and models. It associates both column/parameter metadata and VOTable data with data model elements (such as classes, attributes, types, etc.). It also complements VOTable data or metadata that may be missing from the table, e.g., coordinate system descriptions or curation tracing.

The data model elements are organized in an independent annotation block complying with the MIVOT XML schema, which is added as an extra resource above the TABLE element. The MIVOT syntax allows a data structure to be described as a hierarchy of classes. It can also represent relationships and compositions between them. Furthermore, it can construct data model objects by aggregating instances from different tables of the VOTable.



Reading and writing VO model annotations

Model Instances in VOTables (MIVOT) defines a syntax to map VOTable data to any model serialised in VO-DML (Virtual Observatory Data Modeling Language). The data model elements are grouped in an independent annotation block complying with the MIVOT XML schema which is added as an extra resource above the table element. In Astropy, the MIVOT block is implemented as a new component of the Resource element (MivotBlock class). MivotBlock instances can only be held by res
Astropy is able to read and write MIVOT ann
function processing data models, they will b

Astropy v6.0.0rc1
Available on Nov 9

See Reading and writing VO model annotations for more details.

F.X Pineau - J. Abid

Possible MIVOT support for CDS projects based on Rust like the X-match orAladin Lite V3

- **Level 0:** MIVOT is implemented in the CDS VOTable crate.
    - The VODML block is parsed and returned as as a structure of context dependent objects.
    - A VODML block can be visited and/or modified by implementing a **VodmlVisitor** trait.
    - One can also create a VODML block and add it to an existing VOTable.
    - https://github.com/cds-astro/cds-votable-rust/tree/main/src/mivot

- **Levels 1-4:** exploratory developments are ongoing in a MIVOT dedicated library using the VOTable crate.

- These Rust implementations will allow for a MIVOT library in **Javascript/WASM.**

Tooling Status

- **MIVOT is an IVOA recommendation since June 2023**
- **Validator: still needs to be published on Pypi**
- **Client Implementation**
  - Python
    - Astropy v6.0.0rc1: available on Nov 9
    - PyVO: Drfat PR #497
  - RUST
    - Parser released in the VOTable package
    - Client logic in dev
- **TAP Server**
  - Admistration tool in developement
  - VOLLT add-on to be released by the beginning of next year

Models and VOTables

~ Part II ~

Working with MANGO

L. Michel

Challenging Complexity and Diversity for an all Purpose Source Model

4XMM-DR9

Parameters

- Raw position with errors
- Corrected position with errors
- Detection flags
- Fluxes with errors in multiple Energy bands
- Hardness Ratio with errors in multiple Energy bands

Associated data

- Multiple Detections
- Xmatched sources
- Spectra
- Time series
- SEDs

How to make a distinction between raw and corrected position ?

How to interpret the flag values?

How to define the energy bands?

How to recognize an HR?

All of those are associated data, but how to tag their type

Paradox:

- No source model in the Virtual Observatory.
- Source data, which represent the basic building blocks of astronomers' work, are not modeled.

Reason:

- Observations of source objects are multi-faceted.
- The way features for source data are described and organized depends on the science cases.

Consequences:

- This diversity cannot be served by a single static data model describing a source item for all possible cases.
- Having a global source model would lead to a very complex solution not usable in practice.

The Proposal

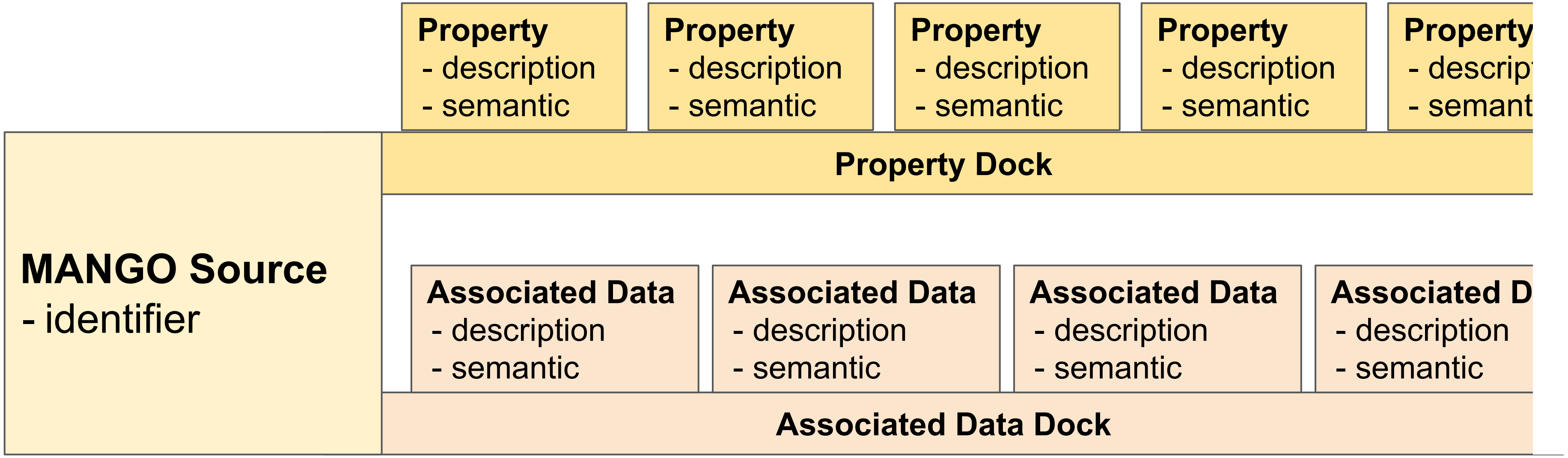
What MANGO is not:

- A model describing what what is source is

What MANGO is:

- An open model providing clients with a standard description of the quantities of interest.
- The list of those quantities of interest is open (as much as you want)
- Its content is set by the data provider
- The number of properties that can be modeled by MANGO is large
- They can either be imported from other models or be native MANGO classes.

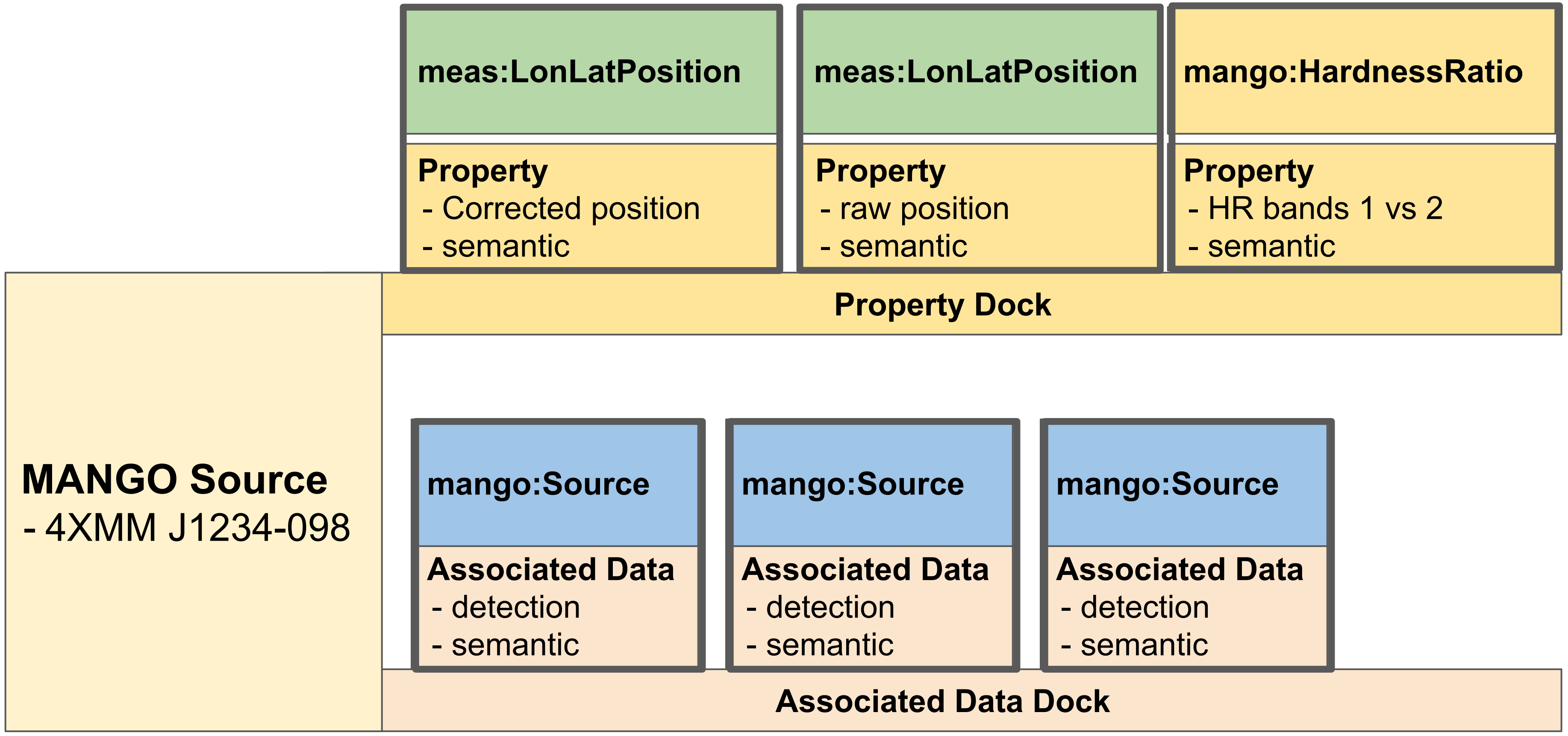
The Dock Metaphor



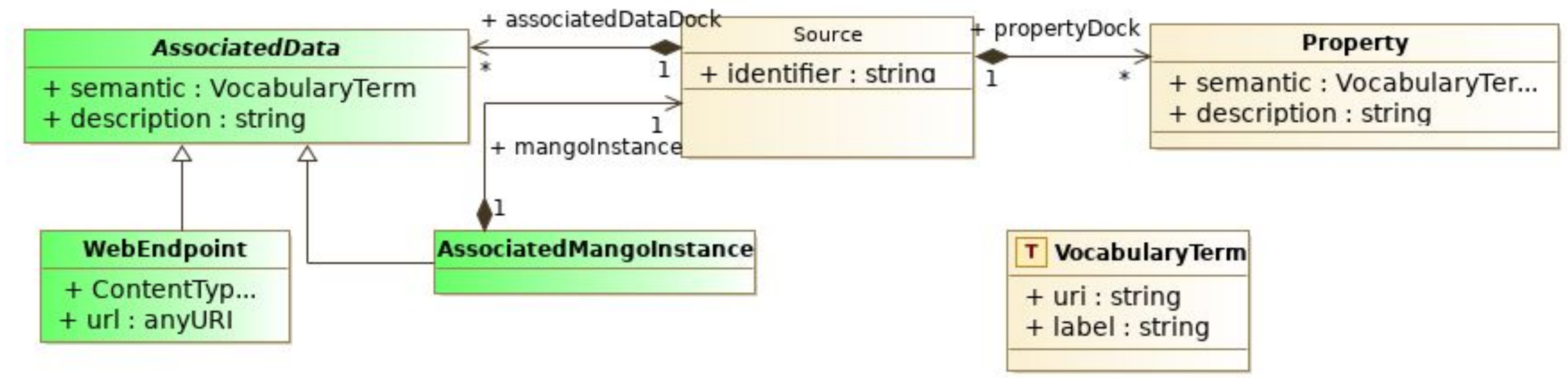
The docks are open ended collections of parameters

- One dock for the source parameters
- One dock for the associated data
- The dock slots are all similar: description + semantic
- The actual property description is hooked on those slots

The Dock Metaphor (example)



Shallow Dive into the UML

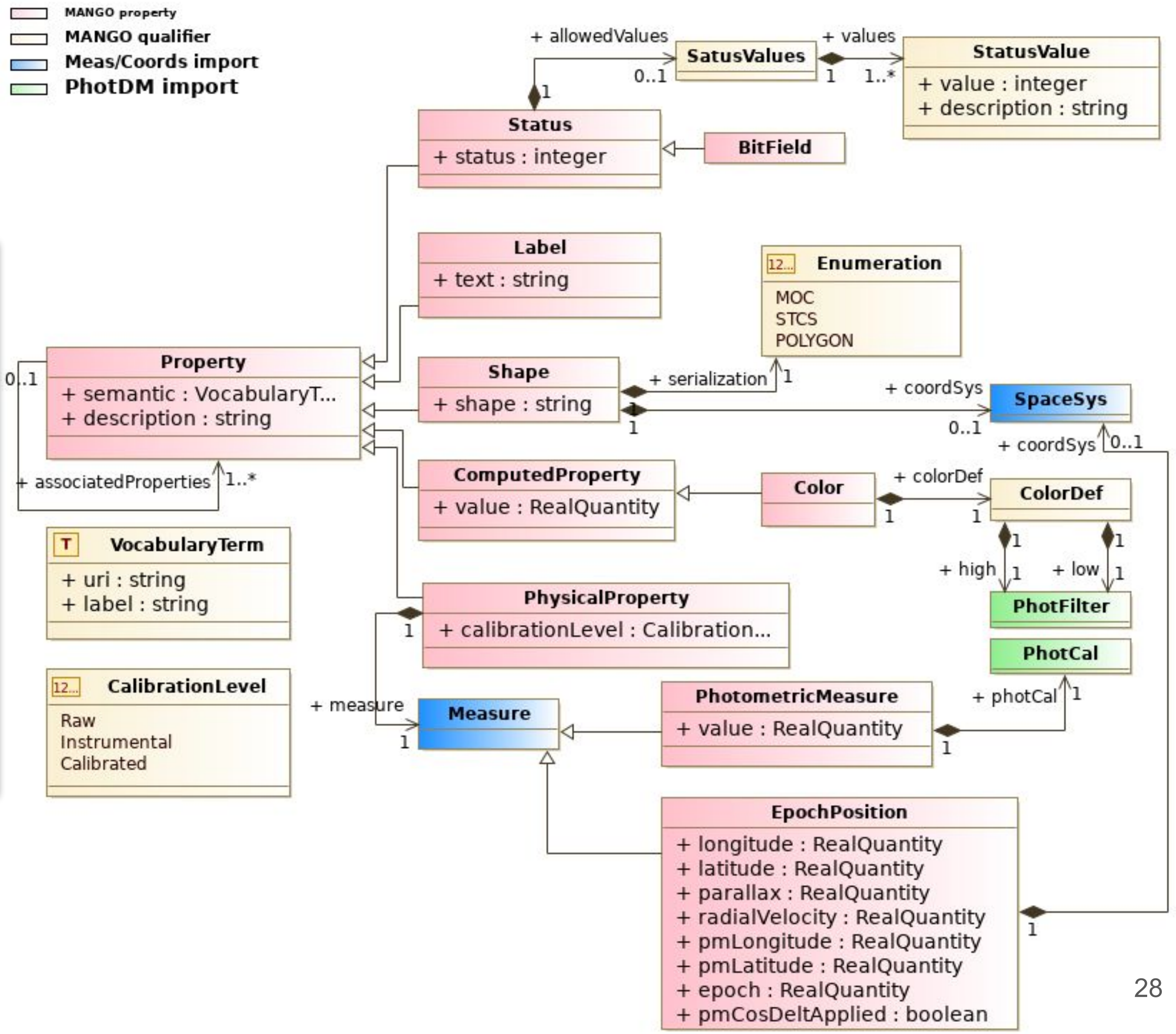


Property Connectors

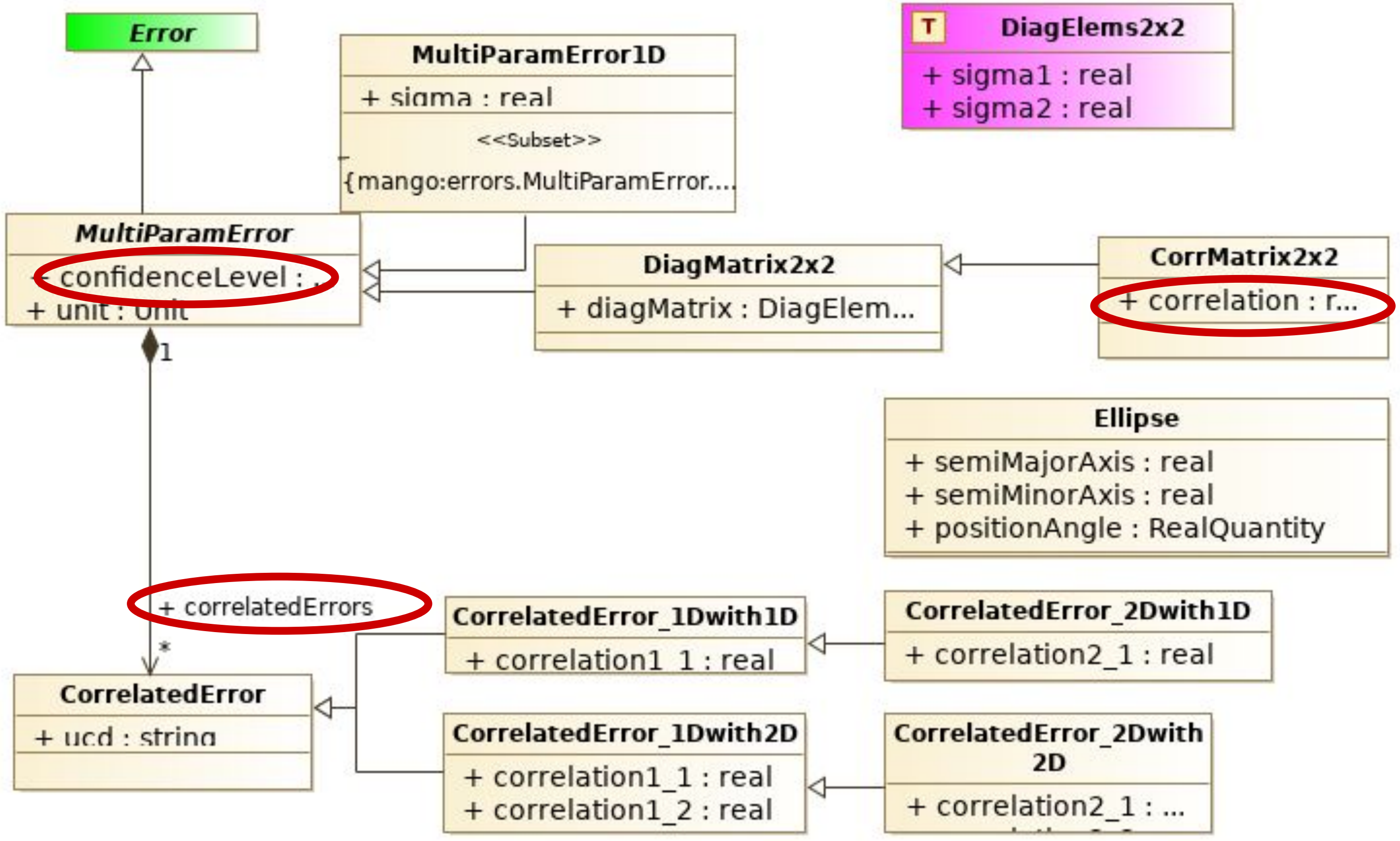
- Mango ***`Source`*** have one connector for each parameter value or associated data
- They have as **many connectors** as needed.
- Both parameters and associated data are hooked to the Mango ***`Source`*** by a wrapping class that allows to identify both natures and roles of the hooked data.
- ***`Properties`*** can be linked together to represent logical parameter sets.

GOING DEEPER

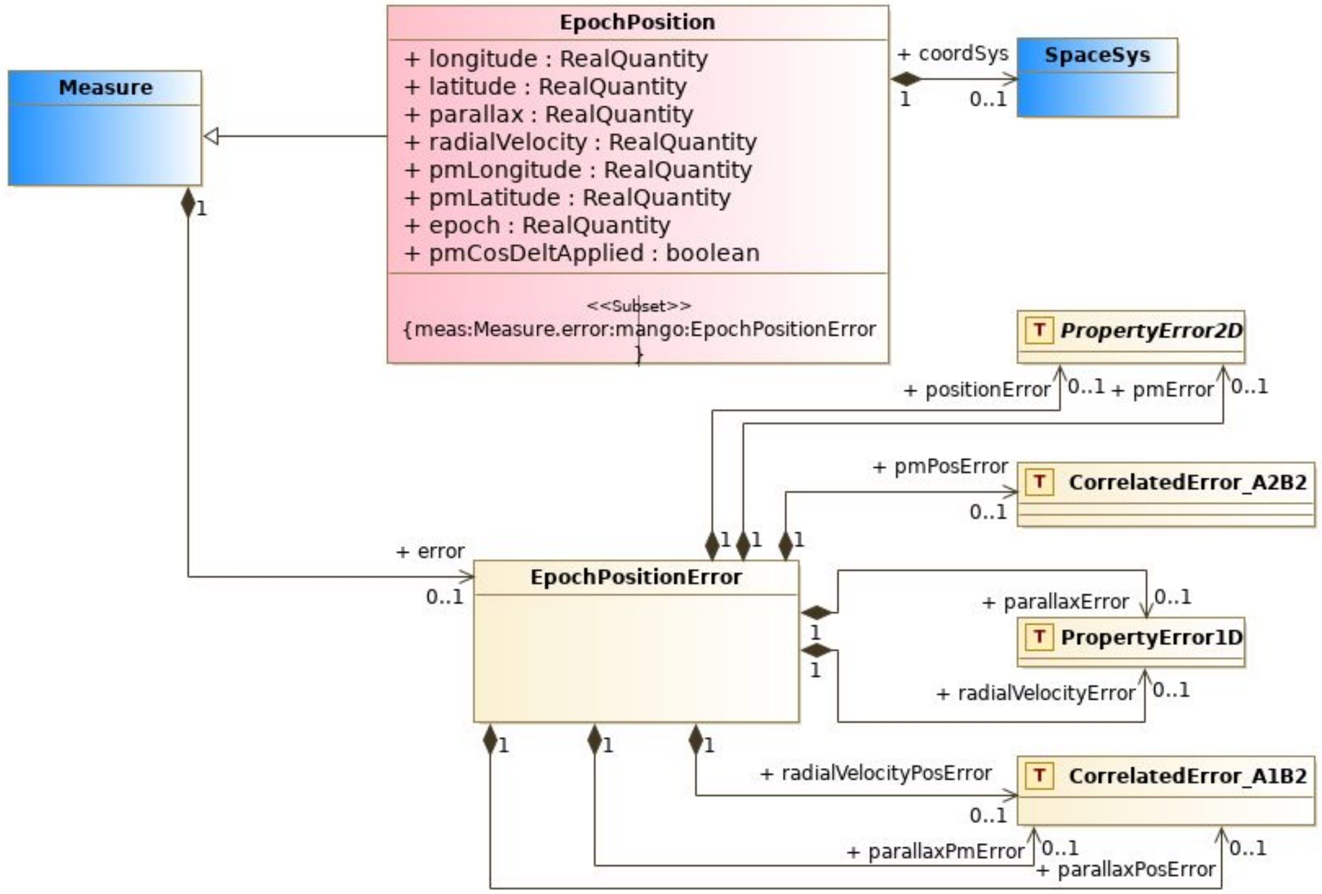
- Physical properties
  - ***Measure* sub-classes** are used whenever possible.
  - Others Parameters are built by **extending the *Measure* class**
- Other properties
  - Specific classes
  - Part of the MANGO model.



Support of Complex Errors



Epoch Propagation Support (see Apps talk today)



Mango Status and Call For Contributions

Model initially presented in 2019

- Draft document
- Dormant since that time
    - Pandemic
    - DM workshop
    - MIVOT RFC

Work have been resumed in september

- Same structure (docks)
- Less specialized classes
- Specialization of the property

Good slot now for adding new properties

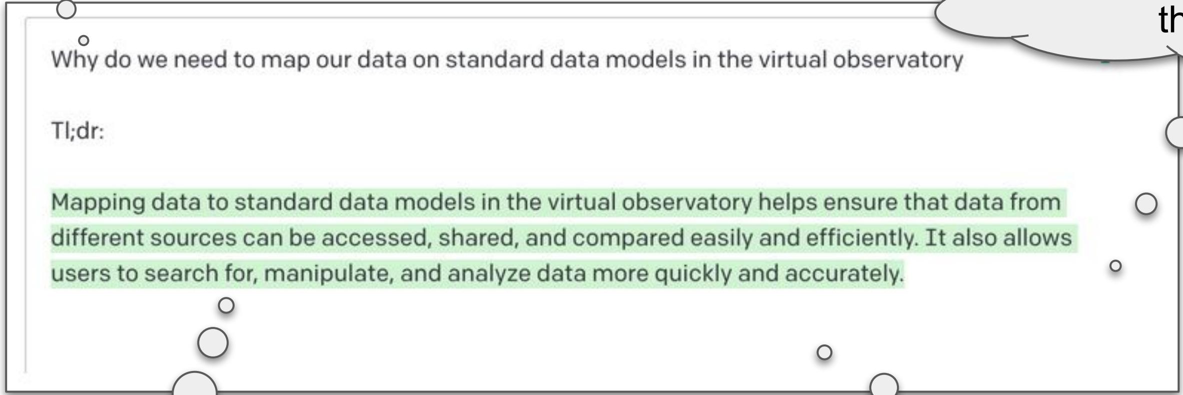
- Epoch Position
- HE Data
- ranking of similar properties (Pierre suggestion)

https://github.com/ivoa-std/MANGO

Working with Model Anotations
-
Laurent MICHEL on the behalf of all DMers

ChatGPT is a good guy

I admit there are a few input on the WEB on that topic



Why do we need to map our data on standard data models in the virtual observatory

Tl;dr:

Mapping data to standard data models in the virtual observatory helps ensure that data from different sources can be accessed, shared, and compared easily and efficiently. It also allows users to search for, manipulate, and analyze data more quickly and accurately.

I'm probably one of the main contributor of the ChatGPT training set on this topic

However, I enjoy hearing it from "someone" else