Let’s Play with Model Annotated Data

(call for) **Ideas for a Python API**

Laurent Michel - ObAS

Tools and Doc Status

- **Annotation syntax**
  - VO working draft
- **Annotation generation**
  - Mapping component builder (individual model classes)
  - On the fly annotation of TAP responses (ML et al.)
  - Jovial (OL MCD): Java library - DSL used to generate annotations
  - DM mapper (GL) interactive SPA
- **Annotation readout**
  - JAVA client in development (limited scope limited to FoV - see FB talk session #10)
  - Rama (OL MCD): Python pkg - parses annotation and instantiates model classes. Provides adapters to AstroPy types.
  - **Python demonstrator**
  - **AstroPyVO demonstrator**

To be presented in the hack-a-thon

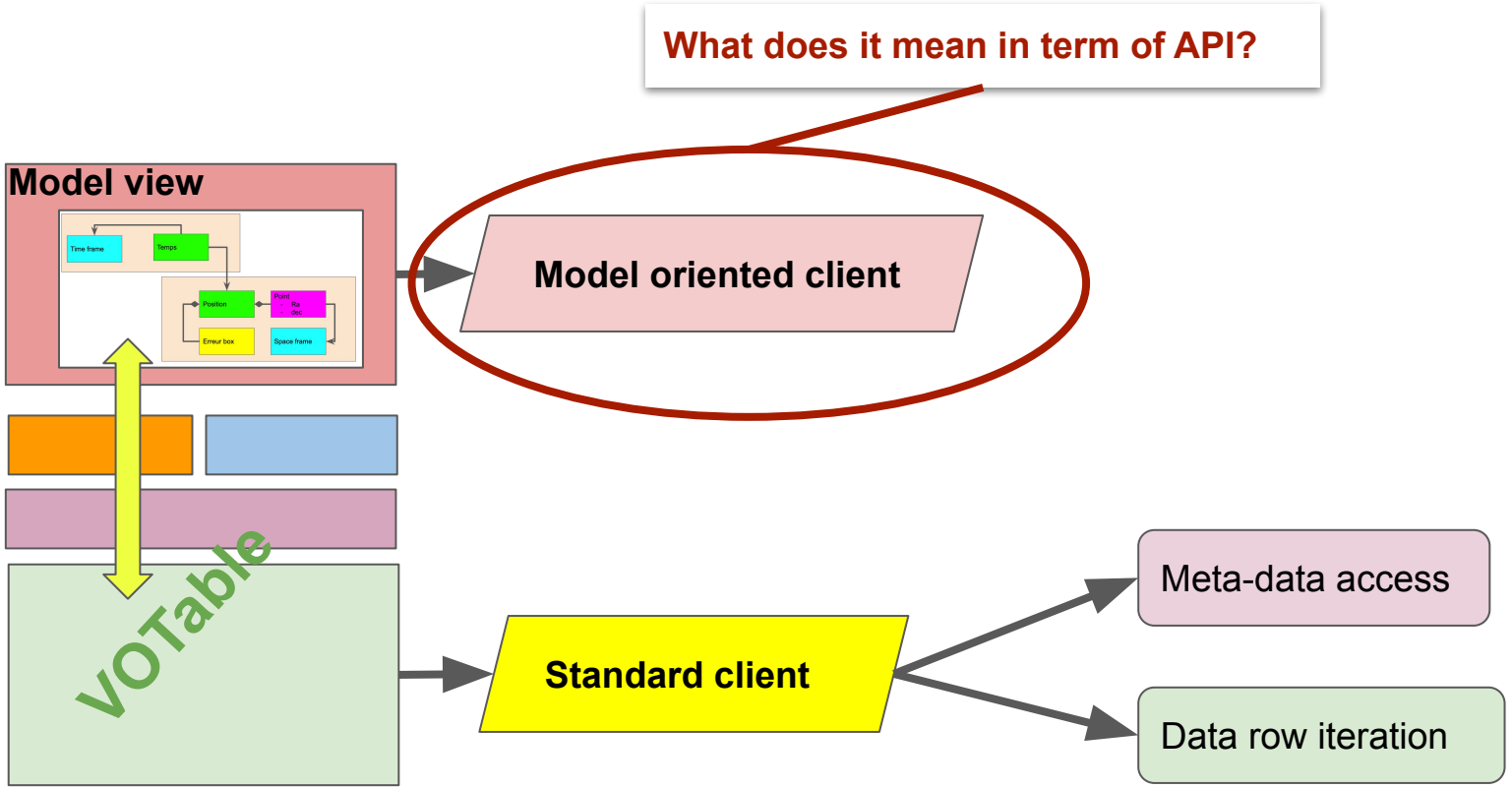
Data Model View

- **Providing Data with a Model View**
  - The VOTable can still be read in a regular way
  - Data rows can be interpreted as model instances
- **Data Model View: PRO**
  - Data can be interpreted in the same way regardless the way they are arranged.
  - The interpretation of the data is unambiguous and is documented in the model
- **Data Model View: CON**
  - Represent an additional workload for curators

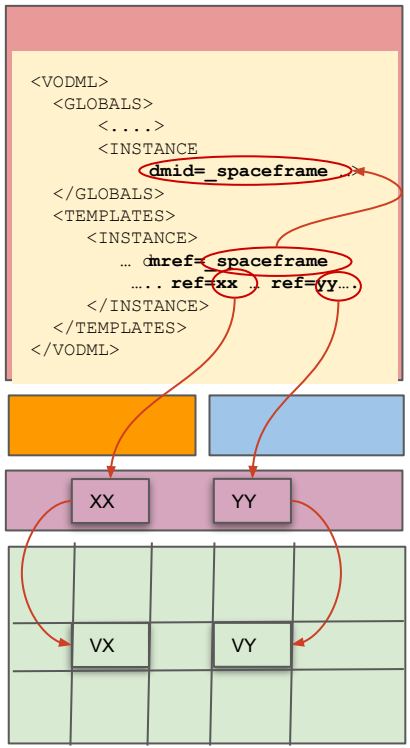
Model Oriented Client Expectations

- **Data retrieval**
  - **Retrieving missing data or meta-data**
    - Filter, provenance, FoV, dataset meta-data…
    - Missing units (this can happen)
    - Systematic errors
  - **Retrieving standard representations of usual quantities**
    - `Meas` model instance e.g. `Point, Time`….
    - Photometric calibration (`PhotDM`)
  - **Retrieving complete model instances**
    - Time Series
- **Minimising the parsing**
  - Hiding the bothering XML handling
  - Getting directly instances usable for the computation

Processing VOTable Data vs Modeled Data



Definition the appropriate API



Parser

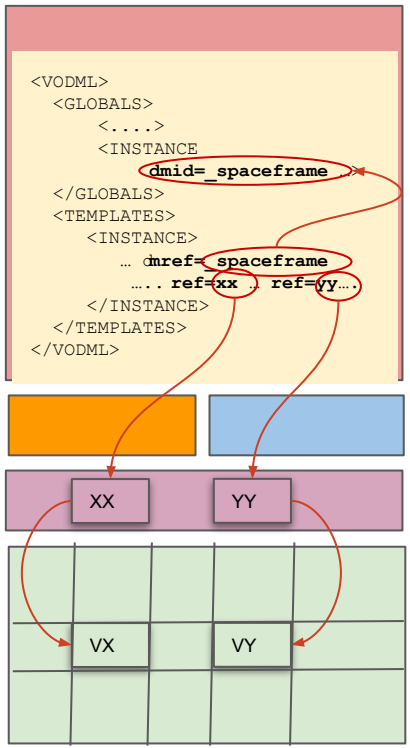
```
<INSTANCE>
    <... space frame …>
      <… val=VX …?>
      <… val=VY….>
</INSTANCE>
```

XML Serialization of the model view of the data rows

- References resolved
- Components gathered

USER

Definition the appropriate API



Parser

```
<INSTANCE>
   <... space frame …>
      <… val=VX …?>
      <… val=VY….>
</INSTANCE>
```

XML Serialization of the model view of the data rows

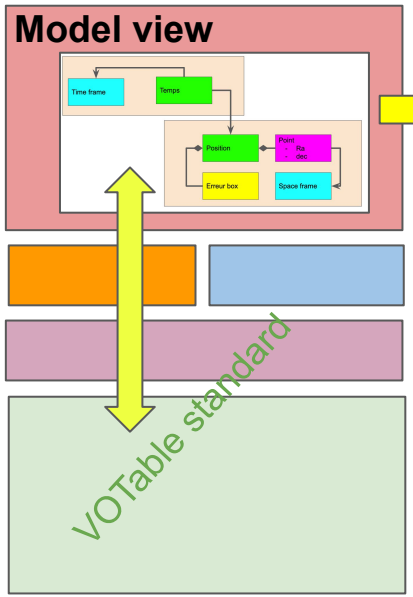
- References resolved
- Components gathered

formator

Convenient serialization of the model view on the data rows

USER

Which Serialization for the DM View



Python engine
Parser
+
formator

direct framework binding
- Instantiate framework classes

model-specific classes
- One Python class for each model class

Meas/Coord

PhotDM

Other?

generic classes
- One unique Python class for all classes of all models

Getters based on `XPath`

Getters based on `@dmrole`

Other?

API examples

Getters based on **`XPath`**

```
row = mviewer.get_next_row()
row_model_view = mviewer.get_model_view(resolve_ref=False)
hot_points = mviewer.get_model_component_by_type("sed:PhotPoint")
for ele in row_model_view.xpath('.//ATTRIBUTE[@dmrole="sed:Source.name"]'):
    print(f'name is {ele.get("value")}')
    break
```

API examples

Getters based on `@dmrole`

```
row = mviewer.get_next_row()
row_model_view = mviewer.get_model_view(resolve_ref=False)
phot_points = mviewer.get_model_component_by_type("sed:PhotPoint")
timescale = phot_points[0].get_attribute("coords:CoordinateCoordSys",
                                         "coords:PhysicalCoordSys.frame",
                                         "coords:TimeFrame.timescale").value
```

Getters based on **`XPath`**

```
row = mviewer.get_next_row()
row_model_view = mviewer.get_model_view(resolve_ref=False)
phot_points = mviewer.get_model_component_by_type("sed:PhotPoint")
for ele in row_model_view.xpath('.//ATTRIBUTE[@dmrole="sed:Source.name"]'):
    print(f'name is {ele.get("value")}')
    break
```

API examples

Meas/Coord

```
row = mviewer.get_next_row()
row_model_view = mviewer.get_model_view(resolve_ref=False)
phot_points = mviewer.get_model_component_by_type("sed:PhotPoint")
phot_cal = PhotCal(phot_points[0])
print(f"spec_loc = {phot_cal.photometryFilter.spectralLocation}")
```

Getters based on `@dmrole`

```
row = mviewer.get_next_row()
row_model_view = mviewer.get_model_view(resolve_ref=False)
phot_points = mviewer.get_model_component_by_type("sed:PhotPoint")
timescale = phot_points[0].get_attribute("coords:CoordinateCoordSys",
                                         "coords:PhysicalCoordSys.frame",
                                         "coords:TimeFrame.timescale").value
```

Getters based on `XPath`

```
row = mviewer.get_next_row()
row_model_view = mviewer.get_model_view(resolve_ref=False)
phot_points = mviewer.get_model_component_by_type("sed:PhotPoint")
for ele in row_model_view.xpath('.//ATTRIBUTE[@dmrole="sed:Source.name"]'):
    print(f'name is {ele.get("value")}')
    break
```

API examples

```
row = mviewer.get_next_row()
# get the astropy SkyCoord object for that row
position = mviewer.get_astropy_sky_coord()
```

Meas/Coord

```
row = mviewer.get_next_row()
row_model_view = mviewer.get_model_view(resolve_ref=False)
phot_points = mviewer.get_model_component_by_type("sed:PhotPoint")
phot_cal = PhotCal(phot_points[0])
print(f"spec_loc = {phot_cal.photometryFilter.spectralLocation}")
```

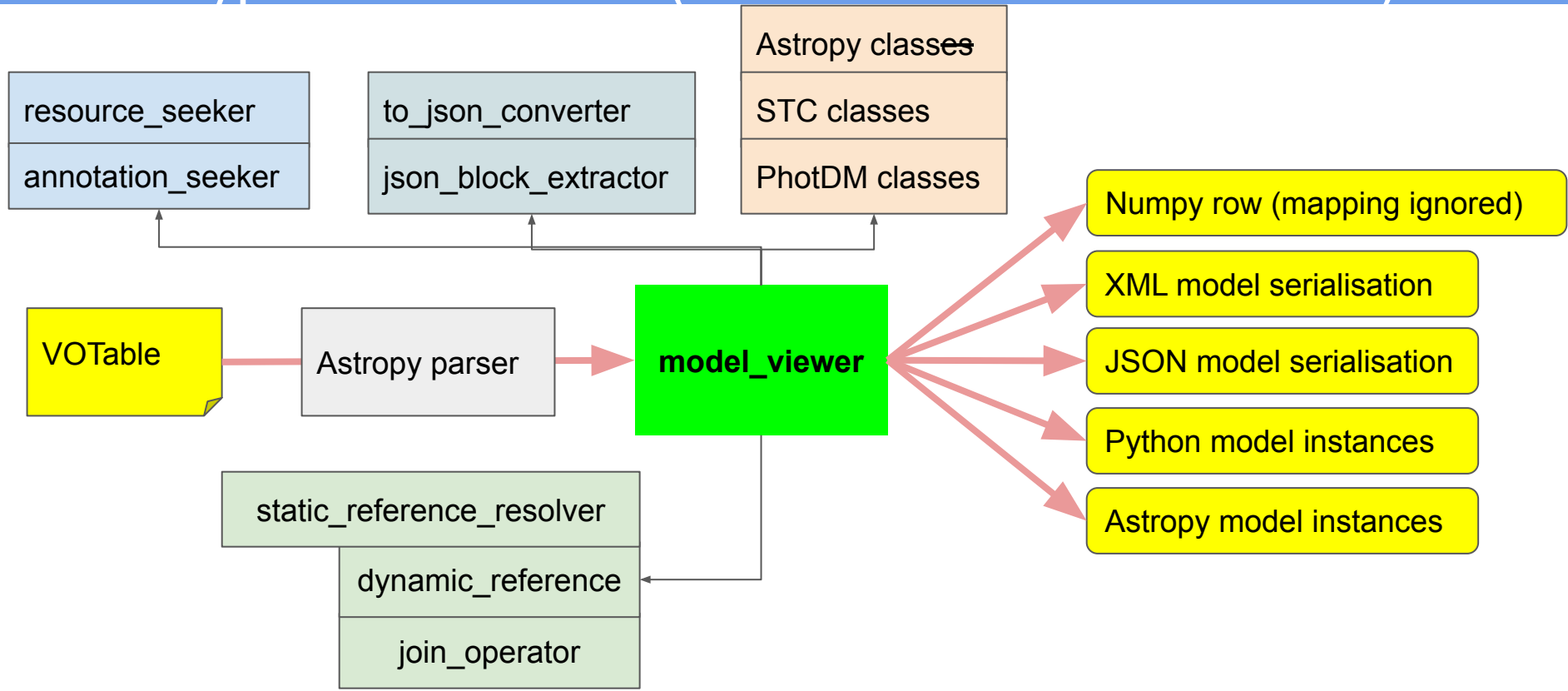
Getters based on `@dmrole`

```
row = mviewer.get_next_row()
row_model_view = mviewer.get_model_view(resolve_ref=False)
phot_points = mviewer.get_model_component_by_type("sed:PhotPoint")
timescale = phot_points[0].get_attribute("coords:CoordinateCoordSys",
                                         "coords:PhysicalCoordSys.frame",
                                         "coords:TimeFrame.timescale").value
```

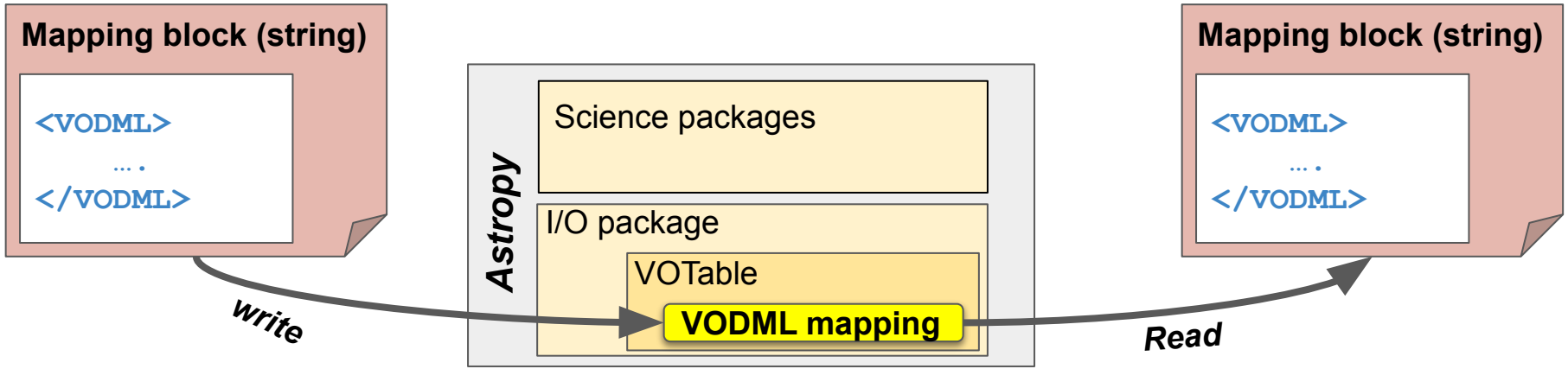
Getters based on `XPath`

```
row = mviewer.get_next_row()
row_model_view = mviewer.get_model_view(resolve_ref=False)
phot_points = mviewer.get_model_component_by_type("sed:PhotPoint")
for ele in row_model_view.xpath('.//ATTRIBUTE[@dmrole="sed:Source.name"]'):
    print(f'name is {ele.get("value")}')
    break
```

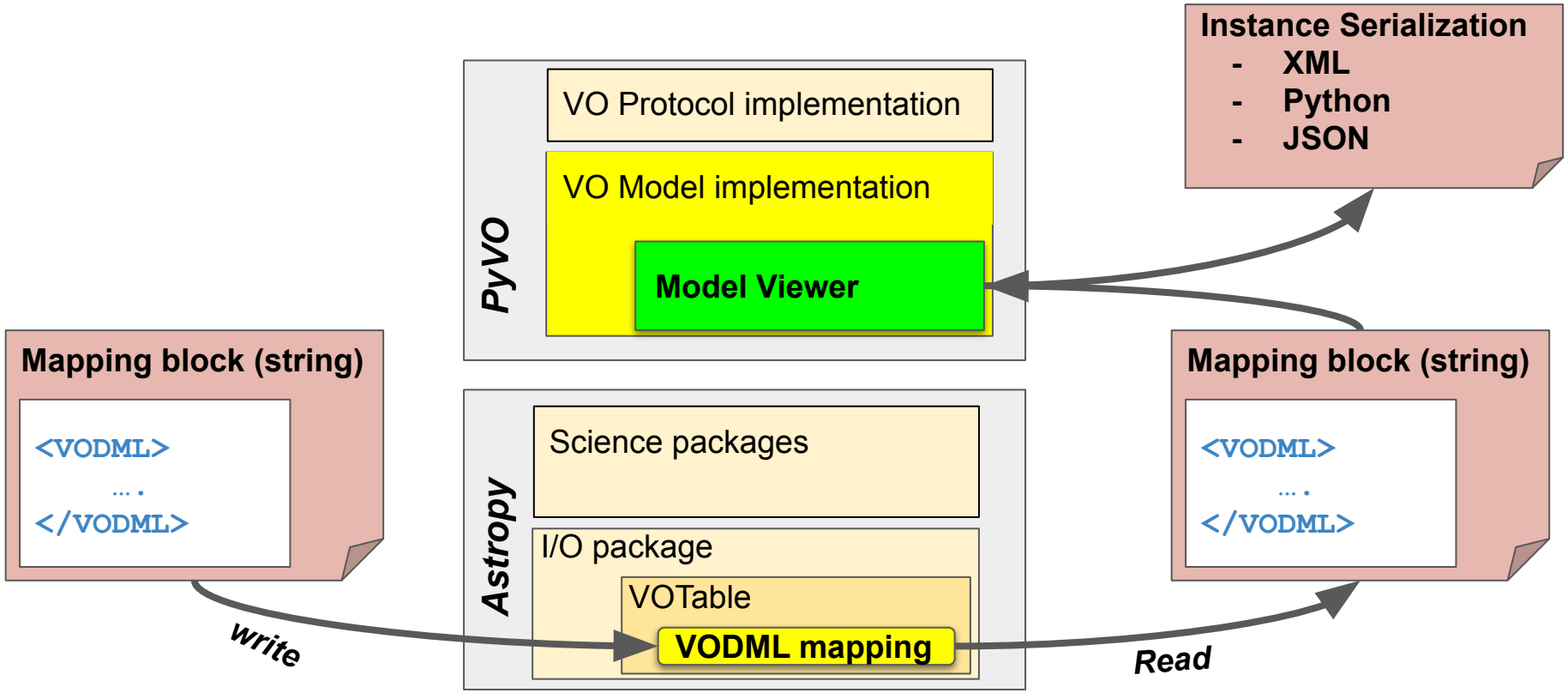
Prototype Architecture (modelinstanceinvot-code)



Astro[Py]Vo Implementation (lmichel@github forks)



Astro[Py]Vo Implementation (lmichel@github forks)



Notebooks

https://github.com/ivoa/modelinstanceinvot-code
https://mybinder.org/v2/gh/ivoa/modelinstanceinvot-code/merge-syntax

- **XML/JSON parsing**
  - `Measure.ipynb`
- **Meas/Coord + unit conversion + distance computation**
  - `gaia_3D.ipynb`
  - `moving.ipynb`
- **Astropy SkyCoord**
  - `gaia_3D_astropy.ipynb`
- **PhotDM implementation**
  - `photdm_impl.ipynb`

Making the Data Model View Affordable

- **The model view only renders the data hierarchy**
  - Model nuances are ignored
    - Inheritance
    - Association vs aggrégation vs composition
    - Object types vs data types

- **Data hierarchy faith to the model**
  - Model mapping in a VOTable = arrangement of
    - Attributes (i.e. JSON: `key:value`)
    - Tuples (i.e. JSON: `{...}`)
    - Collections (i.e. JSON: `[...]`)
  - Reference mechanisms:
    - Links between model components
    - Links between model components and VOTable data
  - Possible addition of missing metadata (e.g. filter definition, FoV components)