Firefly support for ObsTAP and SODA

Extension of TAP interface to ObsTAP

Starting with the existing Firefly TAP-query UI, extend the graphical query-builder interface as follows:

1. *Determine whether the selected TAP service is advertising an `ivoa.ObsCore` table via `TAP_SCHEMA` .*
2. *If so, display a modified UI for the schema/table selection row of the query screen:*
    a. *Provide a radio button choice for () "Image metadata search" vs. () "General table search". When the former is selected, display a message to the effect of "Searching table 'ivoa.ObsCore' for image metadata", hide the schema/table selection UI elements, and enable the behavior in items 3 and beyond below. When the latter is selected, retain the normal TAP-search functionality, but if the schema "ivoa" and table "ivoa.ObsCore" happen to be selected, enable the behavior below anyway.*
    b. *(Extra) If a table is selected, and the table schema appears consistent with ObsCore (contains the necessary columns for the behavior below), enable the behavior below.*
3. *Modify the rest of the UI to show that it is an "image metadata search" screen.*
4. Add a new collapsible subsection of the screen that provides for selection against the calibration level in the fixed list of levels defined in the ObsCore standard. Provide a checkbox or multiple-selection menu interface. Also allow for selection against the data product types defined in ObsCore, again as a checkbox or multiple-selection menu interface. Generate appropriate `WHERE` clauses for the ADQL as expressions involving `calib_level` and `dataproduct_type`. The resulting widget should also be able to generate the equivalent parameter expression for an SIAv2 query, using the `CALIB` and `DPTYPE` parameters. Provide for an application-level control that determines whether `dataproduct_type== "image"` is set as the default query state of this subsection.
    a. (Extra) Query the service to determine the available values of `calib_level` and `dataproduct_type` and display them in the UI, e.g., by greying out values that are not present.
5. Add a new collapsible subsection of the screen that provides for selection against the collection name as a string. If an "ObsTAP mode" control parameter of the widget is `True`, provide options for "exactly equal" and "matches substring" queries. Generate appropriate `WHERE` clauses for the ADQL as expressions involving `obs_collection`. The resulting widget should also be able to generate the equivalent parameter expression for an SIAv2 query, using the `COLLECTION` parameter, as long as the "matches substring" control is not asserted (which would be an error, as substring searches are not supported in the SIAv2 query parameters). Provide for an application to provide optional sample values to be displayed in the UI.
    a. (Extra) Query the service to determine the list of available values for `obs_collection`, and display a list of values as a multiple-selection menu or similar interface, as an alternative to explicit text query string entry. (Extra extra) Include in this display an indication of the number of observation datasets available in each collection. Write the code in a way that facilitates adding further information of this kind in the future.
6. Add a new collapsible subsection of the screen that provides for selection against the data product sub-type (which for Rubin/LSST is the Butler dataset type) as a string. Display of this subsection should be conditioned on whether the service actually has a `dataproduct_subtype` column. Provide options for "exactly equal" and "matches substring" queries. Generate appropriate `WHERE` clauses for the ADQL as expressions involving obs_collection. No SIAv2 support is required. Provide for an application to provide optional sample values to be displayed in the UI.
    a. (Extra) Query the service to determine the list of available values for `dataproduct_subtype`, and display a list of values as a multiple-selection menu or similar interface, as an alternative to explicit text query string entry. (Extra extra) Include in this display an indication of the number of observation datasets available in each collection. Write the code in a way that facilitates adding further information of this kind in the future.
7. Add a new collapsible subsection of the screen that provides for selection against the wavelength coverage of the observations. Allow input of wavelengths in nanometers and microns, as well as selection of a wavelength range from a menu of filters. Make the list of filters configurable by "application" code - that is, by code that's not in core Firefly. Generate appropriate `WHERE` clauses for the ADQL as expressions involving `em_min` and `em_max`, which requires conversion to meters. The resulting widget should also be able to generate the equivalent parameter expression for an SIAv2 query, using the `BAND` parameter.
    a. (Extra) Query the service to determine the minimum value of `em_min` and the maximum value of `em_max` available on the service, and display them in the UI, in nanometers if they are both less than 1500nm, otherwise in microns.
8. Extend the spatial search subsection of the screen to allow for queries against the boundaries of regions, similar to those currently supported in IRSA Viewer. and in the legacy PDAC portal. The resulting widget should be able to generate appropriate `WHERE` clauses for the ADQL as expressions involving `s_region` and the `CONTAINS()` and/or `INTERSECTS()` operators.
9. Extend the temporal search subsection of the screen to allow for queries against the exposure duration, expressed in seconds. Generate appropriate `WHERE` clauses for the ADQL as expressions involving `t_exptime`. The resulting widget should also be able to generate the equivalent parameter expression for an SIAv2 query, using the `TIME` and `EXPTIME` parameters.
    a. (Extra) Query the service to determine the minimum value of `t_min`, the maximum value of `t_max`, and the minimum and maximum values of `t_exptime` available on the service, and display them in the UI.
10. (Extra extra) Allow some or all of the "query the service to determine available values" features to interact with / be updated by selections made interactively. This could be accomplished by, say, doing a `SELECT DISTINCT` for tuples of calib_level, dataproduct_type, dataproduct_subtype, and obs_collection and then maintaining this table internal to the Javascript, updating the counts of available observations per value as appropriate to the currently configured query.
11. Include the `WHERE` clauses from all these subsections in the generated ADQL query.
12. Provide a version of this screen that entirely suppresses the service-selection and table-selection, allowing an application to provide a dedicated "image search" screen without distractions.

Display of the query result will rely on the existing Firefly capability for displaying an image browser based on ObsCore query results.

Note that there is no requirement in this particular task to actually implement SIAv2 queries - the point is just to make the query widgets reusable when we get around to implementing SIAv2.

Support for SODA queries

In the ObsTAP UI

Add an optional cutout-size parameter to the spatial-selection UI subsection. Ensure that the spatial-selection query is compatible with the cutout size, when provided, so that observations that do not cover the cutout are not returned. When a cutout is requested, ensure that `dataproduct_type==" image"` is added to the query.

In the existing Firefly ObsCore image-browser UI

(Requires more detailed collaboration with the rest of the Firefly team)

Analyze the returned ObsCore table for DataLink information indicating that a linked SODA service is available. If it is, invoke the SODA service (this will require the creation of a new search processor on the Java side of Firefly, most likely) and display the resulting image. If it is not, display a message to the user along the lines of "The image service being accessed does not currently provide a linked image cutout service; displaying the full image instead." In that case, display the cutout box as an overlay on each displayed image. Whether or not a cutout service is available may be a per-row property of the ObsCore table; design the UI behavior accordingly.

ALSO

Allow the application to provide a **default** cutout service. This should be able to be conditioned on acceptable values of the facility, instrument, calib_level, and dataproduct_subtype (if present) fields in the ObsCore table, and on a regular expression for the obs_publisher_did field. If present, this default is used whenever dataproduct_type is "image", DataLink information fails to yield a SODA service, and the value-matches are satisfied. The default cutout service definition should include a full DataLink service descriptor which will simply be used by Firefly instead of a service descriptor obtained from the query result table.