Astronomy and Computing 10 (2015) 88–98

Contents lists available at ScienceDirect

Astronomy and Computing

journal homepage: www.elsevier.com/locate/ascom

Full length article

Client interfaces to the Virtual Observatory Registry

M. Demleitner [a,*], P. Harrison [b], M. Taylor [c], J. Normand [d]

[a] Universität Heidelberg, Zentrum für Astronomie, Astronomisches Rechen-Institut, Mönchhofstraße 12-14, 69120 Heidelberg, Germany
[b] Jodrell Bank Centre for Astrophysics, Jodrell Bank Observatory, Macclesfield, SK11 9DL, UK
[c] H. H. Wills Physics Laboratory, Tyndall Avenue, University of Bristol, UK
[d] Observatoire de Paris VOPDC-IMCCE, 61 Av de l'Observatoire 75014 Paris, France

ARTICLE INFO

Article history:
Received 27 October 2014
Accepted 20 January 2015
Available online 30 January 2015

Keywords:
Virtual Observatory
Registry
Standards

ABSTRACT

The Virtual Observatory Registry is a distributed directory of information systems and other resources relevant to astronomy. To make it useful, facilities to query that directory must be provided to humans and machines alike. This article reviews the development and status of such facilities, also considering the lessons learnt from about a decade of experience with Registry interfaces. After a brief outline of the history of the standards development, it describes the use of Registry interfaces in some popular clients as well as dedicated UIs for interrogating the Registry. It continues with a thorough discussion of the design of the two most recent Registry interface standards, RegTAP on the one hand and a full-text-based interface on the other hand. The article finally lays out some of the less obvious conventions that emerged in the interaction between providers of registry records and Registry users as well as remaining challenges and current developments.

1. Introduction

In Demleitner et al. (2014a), henceforth Paper I, we described the design and maintenance of the Virtual Observatory (VO) Registry as a distributed information system. Conceptually, it is a collection of, by now, about 15 000 registry records. To give the Registry's users – astronomers, the library community, or even the general public – access to this collection, facilities have to be provided that allow focused queries against it. This includes common bibliographic constraints (by author, title or abstract term, year, etc.), but also constraints specific to a registry mainly concerned with data services (e.g., supported protocols or query parameters, metadata of published tables). In the design of such facilities, several challenges have to be addressed:

1. different users have very different expectations and requirements
2. the underlying data collection (i.e., the set of registry records) is changing over time
3. the underlying data structure is fairly complex, and evolves itself as new standards and techniques are introduced in the VO
4. as many uses require only a small subset of the types of metadata contained, partial resource descriptions should be retrievable
5. the total data set cannot efficiently be transferred to clients as a whole
6. registry records are frequently authored by persons not entirely familiar with the data model, resulting in inconsistent quality.

In consequence, no single *user* interface to the Registry can be sufficient. Instead, the VO community designed *client* interfaces, i.e., network endpoints with rigorously defined behavior and semantics, designed for use by programs that then present the actual user interfaces to Registry data.

We will begin this paper with a brief review of the various client interfaces that are or were used in the VO (Section 2). In Section 3, we proceed to describe the use some selected clients make of these facilities and the ways they apply and expose information obtained from the registry. A major part of the paper, Section 4, is devoted to a thorough discussion of the Registry Relational Model (RegTAP for short), one of the two registry interfaces currently being developed and deployed in response to the deficiencies of previous standards. In Section 5, the other new-generation interface is described.

While laying out some common use cases of Registry data in Section 6 we also point out common query patterns. Section 7 concludes with some speculation about probable future developments.

* Corresponding author.
E-mail address: msdemlei@ari.uni-heidelberg.de (M. Demleitner).

http://dx.doi.org/10.1016/j.ascom.2015.01.008
2213-1337/

In the following, we refer to common Registry standard texts by their abbreviated names as introduced in Paper I, and again the capitalized word "Registry" refers to the abstract concept, while concrete services are written in lower case (e.g., a "publishing registry"). Concepts from VOResource and its extensions are written in SMALL CAPS.

2. History

Although only explicitly written down in 2011, the use cases collected on the IVOA wiki (IVOA Registry WG, 2011) outline some of the challenges faced by the designers of the first client interfaces to the registry in the mid-2000s—finding tables containing columns with certain physics, locating services implementing certain protocols, and the like.

While on the maintenance side of the registry the ecosystem around OAI-PMH (Open Archives Initiative, 2002) provided guidance for many technology choices, in developing the client interfaces much more new ground had to be broken. For instance, the OPACs (Online Public Access Catalogs; see Kani-Zabihi et al. (2008) for a treatment from about the time of RI1 design) established in the library community, while comparable for the purpose of locating information resources, could not efficiently address the use cases, and no broadly accepted standard for client, rather than user, interfaces to OPACs, lent itself to adoption by the VO community.

Given that the interface to be designed was expected to be expressive enough for requests of the type "find all TAP services exposing a table having some word in the description and a column with a given UCD,[1]" it was determined fairly early on that an interface based on simple, atomic parameters would not be sufficient, and Registry information crucial to certain discovery tasks would not be queryable through it. Client interfaces making explicit too much of the underlying data model would also unduly restrict future developments of that data model. Thus, at least one interface to the Registry would have to support a full query language. Since the Registry data model was defined in XML Schema, an obvious choice for the query language was XQuery (Robie et al., 2014), a language that essentially extends SQL concepts to querying XML trees.

However, factors against the adoption of XQuery included:

- the heavy use VOResource makes of XML namespaces, which tended to make queries hard to write by hand;
- the much larger installed base of relational databases compared to XQuery-capable engines (compounded by the fact that translating XQuery to a given relational schema is hard);
- the desire to open up the full registry data model to queries written by end users, i.e., astronomers. As it was expected that many of these would familiarize themselves with the VO's SQL dialect ADQL (Astronomical Data Query Language; Ortiz et al. (2008)), requiring yet another query language for Registry access appeared undesirable.

With these considerations, it was decided to base the primary Registry interface on conventional relational technology.

While the complex queries XQuery and ADQL allow were needed for identified use cases, it was also acknowledged that "Google-like" searches – more or less loose matching of words in documents modeled as bags of words – was the dominant mode of searching for resources outside of the VO in the targeted user base. At least if common "comfort" features like stemming or phrase searches are desired, this type of search is hard or impossible to simulate through plain ADQL given its very basic set of text search capabilities. Therefore, a keyword search operation with significant freedom for implementors was also defined.

The result of these considerations was Section 2 of RI1 (Benson et al., 2009). It defines two required search operations *Search* (with constraints in ADQL) and *KeywordSearch* (with operator-defined matching of keywords against an operator-extensible minimal set of fields) as well as an optional *XQuerySearch* operation. All search operations return either identifier lists or sequences of full resource records in OAI-PMH style. In addition, two OAI-PMH-like operations were defined, *GetResource* to obtain a resource record from an identifier, and *GetIdentity* to discover metadata about the registry service itself.

Several implementations of the standards are available; services are provided by STScI, ESA, and AstroGrid.

As the RI1 design significantly predates the final standardizations of both ADQL (Ortiz et al., 2008) and the transport protocol for queries and results – that was eventually defined in the TAP standard (Dowler et al., 2010) –, RI1 further defined an ad-hoc transport based on the RPC mechanism SOAP, and it adopted ADQL at a time when experiments were underway with passing ADQL statements to client interfaces in parsed (XML) form. In consequence, modern TAP clients cannot use registry endpoints, and writing queries in the aging XML serialization of ADQL became at least difficult as software components translating SQL expressions into the XML forms went unmaintained.

Further critique came from implementor feedback (e.g., Taylor, 2010) and was collected together with the use cases (IVOA Registry WG, 2011). For instance, in practice the use of a restricted set of XPath to specify constraints instead of defining an actual relational schema lead to severe interoperability problems between different registries, which were further exacerbated by not specifying rules for case folding. The apparent flexibility towards registry extensions provided by the XPath-based column references also did not pay off as originally expected since registries still needed to do internal mapping as registry extensions were developed. In contrast to the (optional) XQuery interfaces, the (mandatory) ADQL interfaces frequently lagged behind standards deployment.

In this situation, the most advanced Registry clients relied on the optional XQuery interface or even used entirely proprietary interfaces.

As TAP services entered the registry in the early 2010s, RI1's response format also became a liability. Registry records contain table metadata, and with TAP services exposing many tables, resource records of several megabytes are not exceptional. This made relatively common queries like "Retrieve basic metadata on all TAP services" expensive in terms of transfer time and processing required.

Therefore, starting in 2011, it was decided to design a new Registry interface, dubbed "RESTful" to contrast it from the RI1 SOAP-based protocol. With TAP and ADQL now available, a replacement of the RI1 *Search* operation was mainly a matter of designing a schema and a mapping to this schema from VOResource. This can be seen as creating a second serialization of an abstract data model implicit in VOResource's XML schema files.

The combination of a defined schema and a TAP service had a model in ObsCore (Louys et al., 2011). The resulting new standard ("RegTAP"), discussed in Section 4, is in the last phases of IVOA peer review as this article is written.

A replacement for the *KeywordSearch* operation is also being developed. Here, the wide availability of feature-rich fulltext engines such as Apache Lucene offers the possibility of enriching the bag-of-words model and allows some advanced operators as well. We will revisit this development in Section 5.

[1] Unified Content Descriptors or UCDs in the VO denote physical concepts like "angular distance" or "radio flux" in a simple formal language (Derriere et al., 2004).

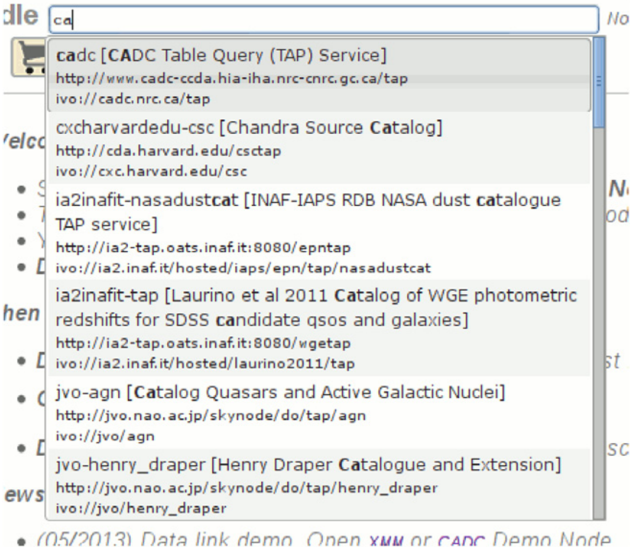

Fig. 1. TAPHandle uses registry information to provide input completion for TAP service access URLs, where the completion items are complemented by additional metadata.

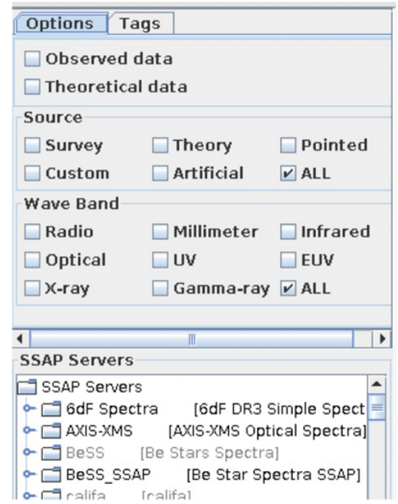

Fig. 2. SPLAT's rendering of the metadata of spectral services in the VO: Various metadata obtained from the registry records are made selectable through checkboxes.

3. Registry use in clients

Many VO clients integrate Registry access, frequently without advertising the actual source of the data. Depending on the scope of the application, different parts of Registry metadata are used, and different presentations of this information appear appropriate. In the following, we look at Registry usage in a number of, we believe, representative applications, concluding with an in-depth look at TOPCAT's use of the registry.

TAPHandle[2] is a TAP client operated through web browsers (Michel et al., 2014). It uses the Registry to discover all registered TAP services. With this information, it can provide input completion in the selector for the TAP service queried (Fig. 1), thus facilitating simple discovery tasks ("I want to query the CADC TAP server"). As it is a TAP client, it is natural for TAPHandle to use RegTAP as its Registry interface. Indeed, its use case is one of the standard tasks identified in the collection of requirements for a revised Registry interface (IVOA Registry WG, 2011). It uses a hard-wired RegTAP endpoint, performing essentially a single query per session within its server component. Thus, TAPHandle users are isolated from technical details of registry access and are also not exposed to visible registry queries.

Similarly, the spectral analysis tool VOSpec (Osuna et al., 2005) queries the registry for all services implementing specific standards (spectral and line access, in this case), but since in contrast to TAPHandle it has no server-side component, it does so directly from the user's client, using one of two built-in registry endpoints implementing the RI1 *Search* operation. Data extraction from the registry records retrieved is performed with an XSLT stylesheet. The discovered resources are presented to the user in a tree view for individual selection or de-selection. This UI is employed both in the selection of the spectral services and in the selection of servers providing information on the location of spectral lines.

Another VO-enabled spectral tool, SPLAT (Castro-Neves and Draper, 2014), takes this approach somewhat further by exposing DATASOURCE (from SimpleDALRegExt, allowing, for instance, the separation of theoretical and observational services) and WAVEBAND from VODataService via checkboxes in its UI (Fig. 2). The UI shown is built from a simple query for all services implementing SSAP. SPLAT furthermore allows users to add, as it were, private registry records (e.g., for unpublished services) that are then integrated into this interface.

A drawback of hiding actual registry queries in this way is that metadata quality of the resource records directly influences the user experience for the application itself. For instance, when a resource record author neglected to give correct waveband metadata, users knowing a certain resource serves optical spectra were frequently confused when the service was deselected after restricting queries to optical data.

The VO client Aladin (Bonnarel et al., 2000) supporting the major VO protocols could build upon a registry-like system called GLU that predates the definition of the VO Registry (Fernique et al., 2003). GLU, with automatic mirror selection and an Aladin-customized metadata format, to this day distributes Registry information to Aladin. Registry records enter GLU not through a client interface but rather by harvesting an OAI-PMH endpoint. The operators of the GLU system at CDS perform additional curation, e.g., by removing invalid records or records for known-defective services. By removing all resource metadata not immediately relevant to the client, Aladin can keep, in effect, a local cache of the entire GLU content, which is impractical for the actual Registry content, as that would currently entail managing and updating several hundreds of megabytes. Responsiveness is further enhanced by persisting this data between executions of Aladin.

There are also clients specifically built around the Registry. One of the most advanced to date is VOExplorer (Tedds et al., 2008), developed by the UK's VO project AstroGrid in the late 2000s as part of its VODesktop suite. In its user interface it guides users in the construction of constraints, mixing menu-based selection with free-text queries as appropriate. VOExplorer communicates with registries via the XQuery client interface. By thus retrieving only parts of the full VOResource record it significantly reduces network traffic compared to a RI1 *Search* client. When a full VOResource record is required, it is cached for use in future query results.

Though the major discovery protocols defined while VODesktop was still being developed are supported, the application's focus is clearly the integration of Registry data into a workflow, and it offloads visualization to specialized clients that can work in full integration with VOExplorer thanks to the SAMP inter-application communication protocol (Taylor et al., 2012). Regrettably, VODesktop's development ceased in 2009, with the demise of the AstroGrid project.

[2] Online at http://saada.unistra.fr/taphandle.

To replace the comprehensive graphical Registry UI provided by VODesktop, WIRR[3] was developed. It is essentially a browser-based query builder for RegTAP, where, much like in VODesktop, the user can successively add constraints on the search results. Notable constraint types include queries for resources containing columns with specific UCDs, "inverted queries" to obtain registry information from a service's access URL – which is useful for finding contact information when services fail –, or query with regular expressions on IVORNs. Even more than VODesktop, WIRR relies on external applications to use the resources found, employing SAMP messages for transmitting resource lists. TOPCAT is one application that already supports these.

To support Registry use from within custom user programs, libraries have been written that encapsulate details of registry access. Given the widespread adoption of Astropy (Astropy Collaboration et al., 2013), we note here the registry functions within the Astropy affiliated package PyVO (Graham et al., 2014). For Registry access, it contains a single function `regsearch` that supports constraints by keywords (essentially, a full-text search within resource record text fields), service types (e.g., image or spectral service) and wavebands. The function also has a parameter to pass in custom SQL fragments executed within the VAO's registry. Due to the limitations of RI1 standard client protocols, a custom, VOTable-based interface is employed at the moment, with a change to RegTAP in the back-end planned.

Finally, there are uses of Registry client interfaces not directly connected to actual VO clients. As an example we mention VO Fresh,[4] an RSS feed of metadata for services newly published or updated in the Registry; new resources are also announced through microblogging services. VO Fresh initially obtained registry information from a full registry's OAI-PMH endpoint but moved to obtaining registry information through RegTAP as that became available.

3.1. Case study: TOPCAT

TOPCAT (Taylor, 2005) is a tool for analysis of astronomical tables. Part of its function is to provide a user-friendly GUI for acquiring tabular data from Virtual Observatory services, most importantly TAP (Dowler et al., 2010) and Cone Search (Williams et al., 2008), but also SIA (Tody and Plante, 2009) and SSA (Tody et al., 2012). To achieve this, it needs the Registry to locate services with the relevant capabilities and to allow the user to assess their suitability for the science job at hand.

From a user point of view, TOPCAT's registry interaction consists of selecting a particular type of data service, optionally supplying some keywords to match against one or more of a handful of fixed resource metadata fields, and dispatching a search which results in presentation of basic metadata for each matching service. The user then peruses this list and selects one of the returned services for subsequent use in the application.

TOPCAT makes only a single type of registry query to support this functionality, the user interface which is illustrated in Fig. 3: locate all registry resources which offer a fixed standard capability (e.g. TAP) and which satisfy zero or more additional user search constraints (e.g. "Title contains the term UKIDSS"), and for each one return a small fixed amount of metadata (ID, Title, Publisher, Access URL and a few others). There is other information stored in the Registry records that TOPCAT may require, such as `vs:CatalogService` records describing table and column metadata. However, for newer VO protocols such information is also

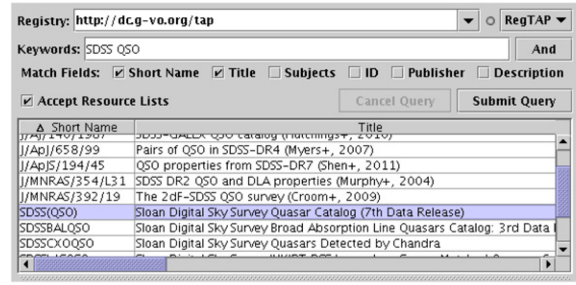

Fig. 3. TOPCAT's Registry interface: the user specifies a query using a selector for registry service endpoint and interface protocol (RegTAP or RI1), a keyword text field, and a set of checkboxes for what resource fields to match against. An additional constraint is the service type, which depends on the context these widgets are shown in. Below the input widgets is a listing of matched resources from which one may be selected. "Accept Resource List" allows filling the resource selector from SAMP messages.

available from the registered data services themselves, and TOPCAT prefers to acquire it from the latter source, since it may be more reliable and is also available for unregistered services.

Implementing these queries in RI1 presented some difficulties. The *KeywordSearch* operation is unsuitable since keyword searches cannot be combined with restrictions on service type. The *XQuerySearch* operation offers suitable functionality, but being an optional part of the standard it is only available from a subset of registry services (in fact, only the AstroGrid implementation), and so would have restricted the choice of registries with which the tool could interact. The only remaining option is the *Search* operation. Syntax and semantics of the fields to match in the required ADQL queries were somewhat under-documented, and there is a problem with the way case sensitivity is defined, but the most serious issue is that *Search* always returns the whole, perhaps large, record for each matched resource, the bulk of which is not needed. Patchy service implementation quality also contributed to make RI1-based registry interaction generally slow and unreliable.

With the introduction of RegTAP, registry interaction is much improved. The user interface is almost unchanged, but queries are more precise, thanks to more careful mapping of the RM data model into its relational counterpart, and much faster, since it is possible to restrict the query response to items of interest only. This latter point can lead to a reduction of two orders of magnitude in the required data transfer. To give an admittedly drastic – but in practice not uncommon – example, the response size for a query for all TAP services registered in May 2014 went down to about 150 kB from previously roughly 25 MB.

Note that although for both RI1 and RegTAP the client uses an essentially SQL-like language to select resources, what the user sees is a keyword-based or "Google-like" interface. Mapping from the latter to the former can result in verbose query text, but this text is not difficult for client code to generate. Therefore, for this purpose there has been no requirement for an essentially keyword-like client interface to the registry.

There is scope for richer interaction with the registry from TOPCAT, for instance queries on fine-grained metadata (column UCDs) or more detailed display of descriptive or curation metadata from selected records. These options may be explored in the future.

4. The registry relational model

The Registry Relational Model – briefly called RegTAP for mainly technical reasons – is the successor to the *Search* method in RI1. It essentially defines a relational schema and rules to map VOResource records into this schema. Using TAP as an access protocol and ADQL as the query language, this is enough to completely define a client interface to the registry.

[3] Online at http://dc.g-vo.org/WIRR.
[4] Online at http://dc.g-vo.org/regrss.

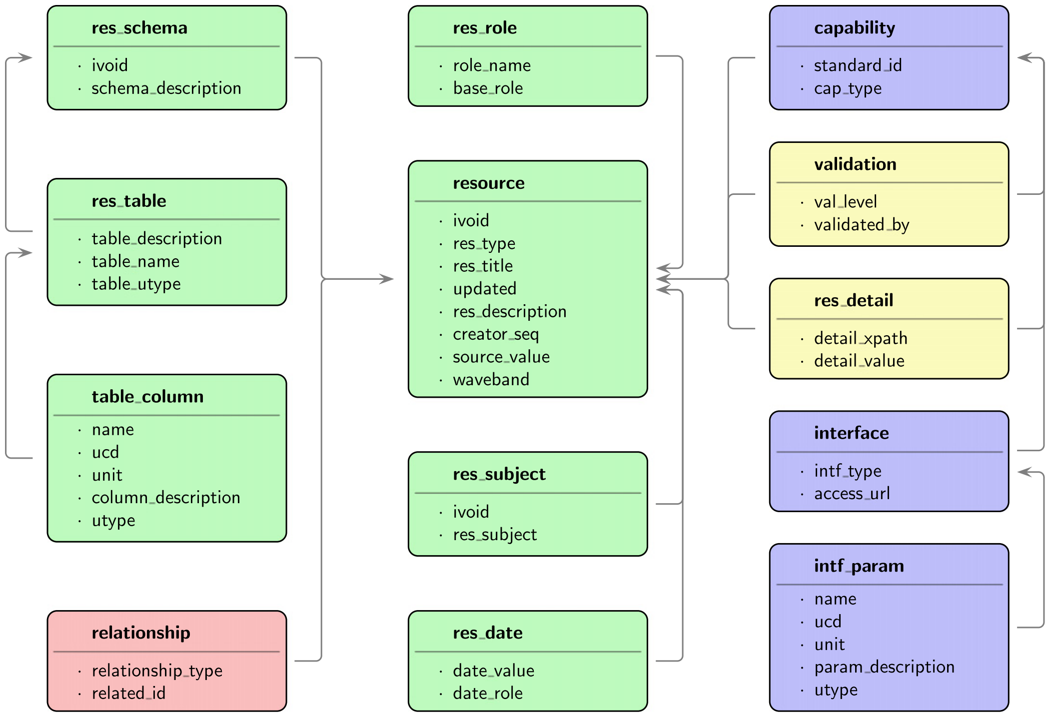

Fig. 4. A sketch of the database schema of the relational registry, adapted from Demleitner (2014). The arrows indicate foreign key relationships, the "attributes" enumerate the fields most likely interesting to clients or scientists writing queries. When joining through `relationship` (red) in the discovery of data collection access services (cf. Section 7.1), the green tables would be "resource-bound", the blue ones "capability-bound", whereas the yellow ones might be either resource or capability-bound depending on query semantics. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

A sketch of this relational schema is given in Fig. 4. Although the authors first experimented with alternative structures that would have been derived from VOResource algorithmically (Harrison, 2011), it turned out design considerations did not lend themselves to formalization, as discussed in the next subsection.

4.1. Design goals and their consequences

In the following discussion of RegTAP's design, the model is derived from several partly conflicting design goals, which are written *slanted* in the following. Additionally, RegTAP names are marked up in `slanted typewriter`, while we continue to write VOResource concepts in SMALL CAPS.

While RegTAP attempts to *represent all concepts* of VOResource that could plausibly be of use in locating resources, it is not a full relational mapping of VOResource. An overarching design goal was to *keep the model compact*. In version 1, the model defines 13 tables, a number that would have been significantly higher for a full mapping without proportionally adding discovery capabilities. From VOResource and its current extensions, the model primarily left out:

- From TAPRegExt (Demleitner et al., 2012) the descriptions of user defined functions; these say what extra functions are available in ADQL queries, how to call them, and what they do. Representing this would have required an extra table, which appeared hard to justify given that no major discovery scenarios were found for this metadata (it is there for TAP client use, and the TAP clients get the information directly from the service's capabilities endpoint).
- Also from TAPRegExt the declaration of how clients can upload tables into TAP services, with a similar rationale.
- From StandardsRegExt (Harrison et al., 2012) the enumerations of the input parameters defined by the standard itself. They do not appear valuable for discovery given the moderate number of existing standards. Representing them would, however, break the simple foreign key relationship between interface and capability if they were kept in the interface table, and an additional relatively complex table otherwise.
- Also from StandardRegExt the detailed information on the versions of documents issued. This would have required an extra table, and again, given the moderate number of standards, no credible discovery scenario is apparent.
- VOResource's ability to have multiple access URLs for a single interface; this feature has essentially not been used in practice, and keeping it would have introduced another join in all queries for access URLs and hence the vast majority of current Registry queries. It is planned to drop the feature in future VOResource versions; resources that actually need multiple access URLs for a single interface would then have to represent each endpoint as an interface of its own.

To leverage existing VOResource expertise, RegTAP tries to *follow VOResource names*. However, again compromises had to be made to meet some other design goals. First, as the subject domain of VOResource partly coincides with the data definition language of SQL, many of the terms in VOResource are reserved words in database engines. As RegTAP also tries to *avoid requiring delimited identifiers*[5] for usability reasons (e.g., difficult to understand parse errors resulting from forgotten quotes), conflicting names were amended with tags indicating entities' roles. In this way, VOResource's TABLE becomes `res_table`, and COLUMN `table_column`.

Another important design goal was to *hide foreign key relationships*. This is again a usability concern—having to write explicit join conditions would necessitate a more intimate familiarity with the

[5] Delimited identifiers in SQL, syntactically marked by enclosing the identifier name with double quotes, allow using arbitrary strings as column names and also suspend SQL's case folding.

data model than can be expected from a possibly casual user. Instead, query writers should need only to identify columns of interest and then use NATURAL JOIN to build their query's FROM clause.

This implies more name mangling, as in VOResource many elements can be children of different parents, for instance TYPE, NAME, DESCRIPTION. Again, disambiguation is effected using tags prepended with an underscore, indicating the source table, abbreviated when names would attain excessive length. Thus, DESCRIPTION in RESOURCE becomes *res_description*, whereas in CAPABILITY it becomes *cap_description*. Only the two column-like tables (*table_column* and *intf_param*) are an exception. This implies that *intf_param* and *table_column* are the only tables that cannot be naturally joined.

The key used for joining is obvious for all tables directly referencing *resource*, as Registry semantics ensure *ivoid* – the record's IVORN – is a suitable primary key for that table. However, RegTAP also has foreign keys into the tables *capability*, *interface*, and *res_schema*, for which VOResource does not provide suitable primary keys, as the respective relationships are represented by lexical inclusion in XML. RegTAP instead introduces surrogate keys, the nature of which is implementation-defined. Hence queries should never explicitly use them, and since the tables are naturally joinable, they have no reason to do so. In general, as the declaration of primary and foreign keys has no impact on service behavior, RegTAP makes no requirements in this area but restricts itself to recommendations.

A further design goal requiring changes to VOResource names is that *quoting must not hurt*. It is not uncommon that SQL authors and query generators employ delimited identifiers when they do not need to. In these cases, mixed-case column names easily lead to execution errors that again may not be easy to understand. Therefore, all identifiers in the standard are completely lowercase. Internal capitalization to indicate compound words is not uncommon in VOResource, however. In RegTAP compound words are concatenated with underscores, such that, for instance, RELATIONSHIPTYPE becomes *relationship_type*.

If only for reasons of ease of implementation across different back-end database engines, it was important for us to *not grossly violate the relational model*. However, an analogue of the object-relational impedance mismatch impacts RegTAP as well: for VOResource, being an XML application, hierarchy and sequences are natural and easy. In a relational model, these translate into foreign keys and extra tables and thus complicate the schema. In order to avoid an inflation of tables, RegTAP supports what in effect are arrays of simple strings.

These are only used where values are taken from controlled vocabularies, specifically for LEVEL and TYPE from CONTENT, WAVEBAND and RIGHTS from RESOURCE, FLAG from COLUMN, and QUERYTYPE from INTERFACE. Here, multiple values from VOResource are concatenated with hash characters (#). To allow reliable querying in these columns, RegTAP services must implement an ADQL user defined function called ivo_hashlist_has. We specifically did not use that pattern for SUBJECT, as its vocabulary, while governed by a recommendation to use the IVOA Thesaurus, is deliberately open – indeed, it is well conceivable that, for instance, hashtags might at some point be used here – and it stands to reason that complex queries over *res_subject* will be performed when clients make use of, say, ontologies that may themselves be represented in database tables.

In a model so heavily dealing with natural language, another violation of strict relationality is almost unavoidable: treating text as, at least, bags of words. RegTAP therefore requires conforming services to offer a user-defined function

```
ivo_hasword(txt VARCHAR(*), pat VARCHAR(*))
  -> INTEGER
```

that returns true at least when pat is present in txt. Operators are urged to match pat to txt in an information-retrieval (IR) sense (i.e., "Google-style" as document vectors). This is the main violation of RegTAP's design goal that *different registries yield identical results* for identical queries. This violation is regrettable, as experience shows that users are at least confused if their familiar result lists change after a change in the registry endpoint used by their client. However, given that IR facilities in back-end databases are inconsistent with each other and an independent implementation of them is nontrivial, the design goal that the standard *does not exclude a major database back-end* overrode the consistency concern.

A final salient design goal is that *Registry extensions are possible without schema updates*. Registry extensions change the XML schema, and hence RegTAP would have to represent arbitrary XML trees within a fixed relational schema if this design goal were to be fully achieved. The result would have been very hard to query indeed. RegTAP's designers therefore identified a subset of extensions that is relatively straightforward in queries, powerful enough to satisfy foreseeable use cases, and reasonably compact: atomic values in 1:n relationships over either resource or capability. The result is RegTAP's *res_detail* table.

This table on the one hand references resources or capabilities by their *ivoid* and, as appropriate, the surrogate key on *capability* (which is NULL for items pertaining to the entire resource). On the other hand it contains keys (*detail_xpath*) and values (*detail_value*). The keys in this table are essentially XPath expressions within the resource record, much like the references in RI1 query constraints. The values are always strings, even when the VOResource elements represented have other types.

Thus, a data collection's ACCESSURL child is accessible through the key /accessURL, the maximum size of files returned from an image service (defined in SimpleDALRegExt) is retrieved as its decimal serialization under the key /capability/maxFileSize, and the authorities managed by a registry are in, if necessary multiple, rows with the key /managedAuthority.

When mapping existing Registry extensions, it was found this was sufficient to express the concepts contained with the exceptions outlined above. A full list of the keys from the registry extensions published before RegTAP is given in Demleitner et al. (2014b), and future registry extensions should specify which additional keys they define.

4.2. Addressing particular issues

In going from VOResource to a relational schema, properties of either the relational or the XML model or restrictions of the query language forced us to introduce additional rules for several entities. We mention some major special cases in this subsection.

Case issues. A particular challenge in the mapping rules from VOResource to RegTAP was case-insensitive values. For instance, IVORN (Plante et al., 2007), UCDs, and utypes in current VO usage (Graham et al., 2013) all have to be compared ignoring case. Even if ADQL had an operator for case-insensitive string comparison, having to consider case issues in comparisons would invite bugs in queries that are hard to detect—when a query author forgets that a column must be compared ignoring case, the queries might still return some records and thus appear to work. RegTAP therefore mandates that all such values must be lowercased during ingestion. In this way, queries not taking into account case insensitivity will at least reliably produce an empty result list. RegTAP further case-normalizes other columns filled from controlled vocabularies to be as consistent as possible. Only columns intended for presentation (essentially the descriptions and titles, role name, and subject) and those where case normalization might lead to ambiguities (mainly *detail_xpath* and *detail_value*) are exempt from normalization. Where case normalized comparisons are desired for such mixed-case columns, RegTAP offers a UDF *ivo_nocase_match* in addition to *ivo_hasword* (that ignores case as well).

Order. While in most parts of VOResource, the order implied in XML trees is irrelevant and thus no particular attention is necessary in the translation to the sets of the relational model, CREATOR is an exception. Typically used to convey authorship information, order there matters to many data providers. Rather than add sequencing capabilities to *res_role*, RegTAP adds a column *creator_seq* to *resource* that contains a pre-formatted author list. This has the additional benefit that clients do not need to worry about reconciling the (correct) practice of having one author per CREATOR element with the (widespread) practice of including multiple names in one element in order to produce a flat author list at least for display purposes; any necessary special handling happens at the registry.

QNames. Several VOResource values are really XML qualified names (QNames). This concerns some fairly fundamental VOResource concepts, in particular the types of resources, interfaces, and capabilities. For instance, a query might be interested in locating all resources that are VODataService DATACOLLECTIONS. These are identified by having an XML schema type of {http://www.ivoa.net/xml/VODataService/v1.1 } `DataCollection`, using the conventional notation that prepends the namespace part of a QName in curly brackets. As this notation is cumbersome for input, serialized XML maps the namespace URIs to namespace prefixes. VOResource and extensions strongly recommend the use of canonical prefixes that would, for instance, bind the prefix "vs" to the VODataService namespace. Hence, the above name becomes a much more manageable `vs:DataCollection`. Unfortunately, the canonical prefixes are not mandatory in VOResource, which means that registries might use entirely different prefixes, and indeed, in registry practice, several do.

As long as the RegTAP ingestor knows which attributes contain QNames – and that is defined in VOResource XML schema files –, it can, however, unify prefixes by turning the namespace prefixes of the instance document into namespace URIs and then translating them back into the canonical prefixes. To ensure consistent results over registries, RegTAP requires this prefix normalization. Essentially, the recommendation to use its canonical prefix contained in all VOResource standards becomes a hard requirement for RegTAP.

5. Full-text based registry interface

In parallel with the relatively complex RegTAP interface, a successor to RI1's *KeywordSearch* is also being developed. This full-text based registry addresses the difficulty of extracting information from the previous registry interface. As field values describing resources are mostly text, a full-text search engine such as the Apache Lucene library – as used in popular server components like ElasticSearch or Apache Solr – is suitable to index and search the contents of the registry. It was therefore decided to develop a RESTful API using ElasticSearch as a client interface to the Registry. The requirements were to fulfill the registry use cases defined by the IVOA Registry Working Group (IVOA Registry WG, 2011) and to support the web clients developed at VO-Paris Data Centre, among them several Registry curation tools. The full specification of the RESTful interface is currently maintained at http://api.vo.obspm.fr/registry/. That page also provides several examples.

5.1. Query interface

The `/search` method is derived from the *Search* and *KeywordSearch* operations of the RI1 searching interface. It allows querying common Registry items individually or all together. For requests specifying multiple constraints logical AND is used by default, but a logical OR is available by adding a parameter "orValues" in the query string.

The initial set of fields to filter a request has been extended to fulfill the requirements of all clients using it. One of the VO client use cases is to find services by capability type (Spectrum, Image, TAP, etc.), and specific words or expressions from its DESCRIPTION, CONTENT.SUBJECT, TITLE, or SHORTNAME attributes. Other useful selection criteria come from CURATION'S PUBLISHER, CREATOR.NAME, and CONTRIBUTOR. The COVERAGE is also used to filter services in spectral, time, and soon in spatial domain. The spatial coverage actually is an optional field in the resource description and not well described in the service declaration. However this information will soon be available in a standard way, using the HEALPix Multi-Order Coverage maps (cf. 7.3).

As the developers of the full-text search based interfaces are also data curators in the VO Registry, some specific information was kept and is available for selected resources:

- the IVORN of the resource record
- the dates of the publication and last update of the resource record
- the registry where the resource has been declared and which is responsible for this record
- information from VALIDATION.

5.2. Query examples

A prototype service implementing the full-text query is being maintained at VO Paris.[6] The following example queries can be executed there by passing them as URL query strings; for reasons of readability, the query strings are shown here not URL-encoded.

- Search all resources containing the keyword "infrared":
```
keywords=infrared
```
- Ditto, but only return services implementing the Simple Image Access protocol:
```
keywords=infrared
  +"ivo://ivoa.net/std/SIA"
```
- Search for all resources published by the Centre de données de Strasbourg (CDS) implementing the Simple Cone Search protocol, with a CONTENTLEVEL of Research, and return the 100 resources starting from match number 200:
```
keywords=publisher:cds
  +standardid:"ivo://ivoa.net/std/ConeSearch"
  +contentlevel:Research
&max=100
&from=200
```
- Return the full resource record for some IVORN:
```
identifier=ivo://vopdc.obspm/luth/exoplanet
```

5.3. Service response

As the aim of the full-text query interface is to provide the simplest system for VO application developers, and most of the new clients are JavaScript based, the API returns query results formatted in JSON responses. Those responses give back the most useful fields as a subset of the whole service declaration plus a link to the original VOResource XML file in the registry. The subset of information returned is relatively compact and has proven sufficient for the clients already using the API.

[6] Access URL http://voparis-registry.obspm.fr/vo/ivoa/1/voresources/search.

6. Common registry queries

VOResource is a complex data model that sometimes offers multiple ways of expressing apparently very similar concepts. Driven by both registry record authoring practices and the queries employed by popular clients, some usage patterns have evolved that should be followed for successful Registry use. Other patterns are recommended for ease of use. We describe the patterns in RegTAP terms, but most would be equally applicable to endpoints speaking XQuery, and partly even to keyword-based services.

What is special to RegTAP is the query construction technique. The way the schema is designed, one looks for the fields to be constrained and for the fields to be retrieved in a schema description such as the RegTAP specification, an implementing service's TAP_SCHEMA, or its VOSI table metadata. The query can then be written by collecting all source tables, concatenating them by NATURAL JOIN and treating the result as a single table.

6.1. Locating standard services

A very common type of query is finding SERVICES implementing a certain standard. In VOResource terms, it is actually not the service but one of its capabilities that complies with a standard. For instance, a service could at the same time implement a cone search for telescope pointings, and two image services each conforming to a specific version of SIAP. Each facility is then represented as a different CAPABILITY.

The *capability* table offers two ways to identify the kind of interface—one could constrain *cap_type* or *standard_id*. The correct constraint is on *standard_id*, as it would be perfectly legal to register an SSA service, say, with a *cap_type* of vr:capability (i.e., the minimal capability description only consisting of a standard identifier and the interfaces). While the record would miss essential metadata, clients should have no trouble operating a service registered in this way, and hence the Registry query should find it. All known clients' queries by service type follow the pattern of matching against the standard identifier.

As the standard identifier is an IVORN, it needs to be lowercased in queries for RegTAP. So, to locate all services (say, by their IVORN and titles) having SSA interfaces, the query would look like this:

```
SELECT ivoid, res_title
FROM rr.resource
  NATURAL JOIN rr.capability
WHERE
  standard_id='ivo://ivoa.net/std/ssa'
```

Other relevant standard identifiers are given in the respective specifications or in one of the examples in RegTAP.

6.2. Locating standard interfaces

Locating the capability is not enough to operate a service. In addition, the endpoint – which in VO practice is identified by an access URL – needs to be located. A single capability can have multiple interfaces, and while this practice is not recommended, there are resource records that have capabilities declaring adherence to a standard with interfaces for web browsers in addition to the standard interface.

VOResource's INTERFACE element has a ROLE attribute to distinguish the standard interfaces from custom ones. For the former, ROLE would contain a special string formed according to certain rules. In practice, many resource record authors have neglected to set ROLE, and therefore actual clients started to ignore it. Current VO practice therefore is to regard the (hopefully unique) interface of type vs:ParamHTTP as the interface exposing the standard.

Hence, the pattern to locate interfaces complying to standards right now is, in RegTAP (this time looking for TAP interfaces):

```
SELECT ivoid, access_url
FROM rr.capability
  NATURAL JOIN rr.interface
WHERE standard_id='ivo://ivoa.net/std/tap'
  AND intf_type='vs:paramhttp'
```

We expect this pattern to be stable, mainly because the development of StandardsRegExt now very strongly suggests that services supporting multiple versions of a single standard will have to treat each such interface in a single capability. Hence, distinguishing different versions by the ROLE attribute (or the *intf_role* column in RegTAP) appears dispensable, and assuming the vs:ParamHTTP interface within a standard capability must be the standard service endpoint is straightforward and robust.

6.3. Query by physics

A type of discovery query not yet widely supported in Registry UIs is the query by physics. The fact that resource records can and in many cases do contain table metadata giving UCDs helps locating resources exposing a certain type of data. As UCDs follow a grammar that ADQL does not understand, it is frequently advisable to use wildcards in such queries. For instance, columns containing infrared magnitudes could be found like this:

```
SELECT name, ucd, column_description
FROM rr.table_column
WHERE ucd LIKE 'phot.mag;em.ir%'
```

To illustrate again RegTAP's principle of natural joins, let us show how to add a constraint on the embedding table here:

```
SELECT name, ucd, column_description,
  table_description
FROM rr.table_column
  NATURAL JOIN rr.res_table
WHERE 1=ivo_hasword(
    table_description, 'quasar')
  AND ucd LIKE 'phot.mag;em.ir%'
```

—then constraining on tables accessible via TAP is simply a matter of combining select list and the FROM and WHERE clauses from Section 6.2 with this query.

6.4. A sketch of TOPCAT's query

By way of example, we present a query submitted by TOPCAT to acquire service metadata for presentation to the user, incorporating some user-supplied constraints. The following ADQL would locate TAP services concerning galaxies:

```
SELECT ivoid, short_name, res_title,
  reference_url, base_role, role_name,
  email, intf_index, access_url,
  standard_id, cap_type, cap_description,
  std_version, res_subjects
FROM rr.resource AS res
  NATURAL JOIN rr.interface
  NATURAL JOIN rr.capability
  NATURAL LEFT OUTER JOIN rr.res_role
  NATURAL LEFT OUTER JOIN (
    SELECT
      ivoid,
      ivo_string_agg(res_subject, ',␣')
        AS res_subjects
     FROM rr.res_subject GROUP BY ivoid
  ) AS sbj
WHERE
```

```
standard_id='ivo://ivoa.net/std/tap'
AND intf_type='vs:paramhttp'
AND (
  1=ivo_hasword(res_title, 'galaxy')
  OR 1=ivo_hasword(res_subjects, 'galaxy')))
```

Note how in this query outer joins are used to make sure rows are returned even for records that, for instance, do not give roles. In the case of *res_subject*, VOResource guarantees that at least one subject must always be present, so doing an outer join here should not be necessary. On the other hand, in particular in queries executed on behalf of a UI, it is good practice to assume minor violations of VOResource will be present in the Registry.

The sub-query for *res_subjects* also shows an example for how to reduce the number of rows transferred by server-side aggregation. Another application for this pattern could be, using suitable strings as separators, retrieving pairs of capability identifiers and their access URLs.

7. Open issues

Even after the introduction of RegTAP, Registry development is not completed. In addition to Registry extensions as new service and resource types are defined within the VO, several fields of work are currently actively being explored. We discuss them here as they delineate what the Registry should be doing but does not do so far, as well as to document approaches tried in the VO to problems that may similarly arise in other communities.

7.1. Data collection and relationships

Some TAP services today expose dozens or hundreds of tables[7]. In ObsCore services[8] (Louys et al., 2011), data from many individual data collections are queryable through a single endpoint. In the same way, some SIAP services make data from several individual observatories accessible.

In all these cases, the contributing data collections should all be present with their full metadata in the Registry. Using GAVO's Lens Image Archive[9] as an example, a title query for one of the contributing data collections, MiNDSTEp, say, should yield the full metadata for the data collection[10], and clients should, from there, be able to infer the access URL of the service exposing the data.

The DataCollection type of VODataService provides a type for such cases, and through relationship – in this case, with a relationshipType of `servedBy` – the associate data service can be successfully located.

However, client support for querying through relationship has been lacking, even in the most advanced registry clients. WIRR at least shows the presence of related resources explicitly, but an additional query is required to retrieve them. Also, a query for ''Image services exposing data from MiNDSTEp'' would fail unless the registry record of the embedded service were carefully crafted. In RegTAP, writing ADQL for queries that would simultaneously find ''direct'' (i.e., services exposing exactly one data collection) and ''indirect'' (i.e., data collection metadata managed separately from service metadata) services is at least highly nontrivial (Demleitner, 2013). To understand why joining tables through relationship requires great care, consider again Fig. 4. The colors there distinguish between ''capability-bound'' metadata that in such queries would have to be queried from the service (e.g., access URL, capability ids, accepted parameters) and ''resource-bound'' metadata that needs to come from the data collection itself (e.g., description, title, or UCDs from a published table). The two tables that can reference both *resource* and *capability*, shown in yellow in the figure, additionally complicate query construction. In any case, natural joins of tables from different groups will not produce meaningful results, thus requiring query authors to add explicit join conditions.

These difficulties have spurred activity to consider changing VOResource such that clients do not need to follow relationships to locate access URLs. A reasonable solution explored was adding (some of) the capabilities of the data service to the resource records of the data collections themselves. That, however, leads to an inflation of such capabilities that will make the very common queries for all resources implementing a certain standard as laid out in Section 6.1 much harder to handle for clients. For instance, to prepare for an all-VO query for images, clients would have to filter out duplicate access URLs in order to avoid querying a service exposing n resources n times (instead of once).

The least burdensome solution is still to be found. Discussions within the Registry working group are currently investigating the use of ''auxiliary'' standard identifiers for the capabilities on the data collections, which would, by lexical convention, facilitate the discovery of either unique services (by using the standard identifiers already in use) or all endpoints exposing data constrained by further metadata (by using an appropriate and index-friendly regular expression).

7.2. Education and internationalization

In the context of work done within the IVOA working group on Education (Molinaro et al., 2014), the issue of multilinguality arose. While in professional astronomy, all-English metadata seems sufficient and, indeed, preferable, the situation is not as clear when certain resources – in this case, educational material – should be made discoverable for educators or even the general public. For instance, if a worked-out use-case on open clusters is available in Italian, should it not be discoverable by querying for ''Ammasso Aperto''? This would entail allowing the relevant text fields (title, description, possibly subject) to be present multiple times in resource records, each element containing text in a different language, and it would probably also entail allowing language constraints in client interfaces to avoid losing precision due to homographs in different languages. Alternatively, different registries might be set up for different languages.

So far, the Education WG only plans to allow discovering which language specific resources are available in rather than supporting queries in non-English languages. If, however, takeup of Registry technologies outside of the research community were to increase, the issue would have to be revisited, presumably from both the client and the data model side.

7.3. Coverage in space and time

VODataService allows the specification of resource coverage, i.e., the spatial area covered on the sky as well as the ranges in time and spectrum, in resource records. Apart from the controlled vocabulary in waveband, this is done through embedding STC-X (Rots, 2005) within registry records. No standard way of querying this information exists to date. An attempt to include coverage information through four tables giving sets of coordinate intervals per resource as proposed in Demleitner (2012) did not gain much traction, partly because of the flexibility and complexity of the underlying STC data model, partly because it was felt that for spatial coverage, coordinate ranges in the equatorial system were

7 Examples for such services include `ivo://org.gavo.dc/tap`, `ivo://nasa.heasarc/services/xamin`, as well as the TAP interface to VizieR.

8 The CADC TAP service `ivo://cadc.nrc.ca/tap` belongs in this category.

9 `ivo://org.gavo.dc/lensunion/q/im`

10 In this case, `ivo://org.gavo.dc/danish/red/data`.

too inflexible to be generally useful even for discovery purposes. An obvious example illustrating the shortcomings would be a survey along the galactic equator: either many ICRS ranges would have to be given, or the coverage would be dramatically overrepresented.

In the meantime, multi-order coverage maps (Boch et al., 2013, MOCs) were developed as a standard way of representing spatial coverages. Work is ongoing on how these could be integrated into VOResource on the data model side and exposed to the clients; if these were to be included in an extension to RegTAP, an ideally indexable way of representing MOCs in databases would be required, and no technically feasible solution has been proposed so far.

8. Conclusions

The VO Registry is an essential source of metadata about the services and data that can be used within the VO, and no non-trivial interaction with the VO can take place without using its discovery capabilities. Many VO clients embed Registry information and protocols in various forms.

By necessity, standardization of the Registry protocols occurred relatively early in the history of the VO. While standards on the server side have held out very well, the early standards on the client side have, in the meantime, proved insufficient for today's advanced Registry use.

This resulted in the creation of second-generation client interfaces. In this article, we have discussed principles and design goals of the two currently developed interfaces. The keyword search interface provides a simple language to constrain results that accommodate users' habits in taking up patterns from general search engines, with a response format designed for easy integration into browser-based applications. RegTAP, on the other hand, is a relatively faithful mapping of essentially the entire data model to a relational database schema, targeted towards "thick" clients and expert users writing ADQL queries by hand.

While RegTAP goes much further than RI1 in defining the mapping between the XML schema that defines the Registry data model and the relational model that is in practice used to represent the data set in queryable form, there are still some small areas where the relational schema and its XML counterpart do not precisely match each other's expressiveness, precluding, for instance, roundtrip ingestion and recreation of registry records through a RegTAP tableset. We propose as a lesson to be learned from this that future data modeling efforts should be done in an implementation-neutral language with well-defined and well-understood mappings to the common implementation languages. Within the VO, an effort is underway to enable this (Lemson et al., 2014).

This is not to say that the Registry model needs fundamental work or a technology switch any time soon. The new interfaces to the Registry expose its functionality fairly completely and interoperably between their implementations, and building on proven technologies like TAP and JSON, they also lower the cost of integrating VO registry information into client programs.

Some open issues remain in the registry's client interface; the most urgent ones are probably the formulation of constraints on spatial coverage and the handling of capabilities associated with data collections.

Acknowledgments

We thank Laurent Michel, Pierre Fernique, and Pedro Osuna for providing information on the Registry interfaces of TAPHandle, Aladin, and VOSpec.

This work was in part supported by the German Astrophysical Virtual Observatory GAVO, BMBF grant 05A11VH3.

References

Robitaille, T.P., Tollerud, E.J., Greenfield, P., Droettboom, M., Bray, E., Aldcroft, T., Davis, M., Ginsburg, A., Price-Whelan, A.M., Kerzendorf, W.E., Conley, A., Crighton, N., Barbary, K., Muna, D., Ferguson, H., Grollier, F., Parikh, M.M., Nair, P.H., Unther, H.M., Deil, C., Woillez, J., Conseil, S., Kramer, R., Turner, J.E.H., Singer, L., Fox, R., Weaver, B.A., Zabalza, V., Edwards, Z.I., Azalee Bostroem, K., Burke, D.J., Casey, A.R., Crawford, S.M., Dencheva, N., Ely, J., Jenness, T., Labrie, K., Lim, P.L., Pierfederici, F., Pontzen, A., Ptak, A., Refsdal, B., Servillat, M., Streicher, O., Astropy Collaboration, 2013. Astropy: A community Python package for astronomy. Astron. Astrophys. 558, A33.

Benson, K., Plante, R., Auden, E., Graham, M., Greene, G., Hill, M., Linde, T., Morris, D., O'Mullane, W., Rixon, G., Stébé, A., Andrews, K., 2009. IVOA registry interfaces version 1.0. IVOA Recommendation. URL: http://www.ivoa.net/Documents/RegistryInterface/.

Boch, T., Donaldson, T., Durand, D., Fernique, P., O'Mullane, W., Reinecke, M., Taylor, M., 2013. MOC—HEALPix multi-order coverage map. IVOA Working Draft. URL: http://www.ivoa.net/documents/MOC/.

Bonnarel, F., Fernique, P., Bienaymé, O., Egret, D., Genova, F., Louys, M., Ochsenbein, F., Wenger, M., Bartlett, J.G., 2000. The ALADIN interactive sky atlas. A reference tool for identification of astronomical sources. Astron. Astrophys. Suppl. 143, 33–40.

Castro-Neves, M., Draper, P.W., 2014. SPLAT-VO: Spectral Analysis Tool for the Virtual Observatory. Astrophysics Source Code Library, February.

Demleitner, M., 2012. Towards registry interfaces 2, talk given at the Fall 2012 IVOA Interop, São Paulo. URL: http://docs.g-vo.org/talks/2012-sampa-ri2.pdf.

Demleitner, M., 2013. News on RegTAP, talk given at the Fall 2013 IVOA Interop, Waikoloa, HI. URL: http://wiki.ivoa.net/internal/IVOA/InterOpSep2013Registry/regtap.pdf.

Demleitner, M., 2014. RegTAP—a new API to the VO Registry, poster presented at ADASS XIV.

Demleitner, M., Dowler, P., Plante, R., Rixon, G., Taylor, M., 2012. TAPRegExt: a VOResource schema extension for describing TAP services, version 1.0. IVOA Recommendation, August. URL: http://www.ivoa.net/Documents/TAPRegExt.

Demleitner, M., Greene, G., Le Sidaner, P., Plante, R.L., 2014a. The Virtual Observatory Registry. ArXiv e-prints, July.

Demleitner, M., Harrison, P., Molinaro, M., Greene, G., Dower, T., Perdikeas, M., 2014b. IVOA registry relational schema. IVOA Proposed Recommendation. URL: http://www.ivoa.net/documents/RegTAP/.

Derriere, S., Gray, N., Mann, R., Preite Martinez, A., McDowell, J., McGlynn, T., Ochsenbein, F., Osuna, P., Rixon, G., Williams, R., 2004. UCD (Unified Content Descriptor)—moving to UCD1+. IVOA Recommendation. URL: http://www.ivoa.net/Documents/latest/UCD.html.

Dowler, P., Rixon, G., Tody, D., 2010. Table access protocol version 1.0. IVOA Recommendation, March. URL: http://www.ivoa.net/Documents/TAP.

Fernique, P., Schaaff, A., Bonnarel, F., Boch, T., 2003. A bit of GLUe for the VO: Aladin experience. In: Payne, H.E., Jedrzejewski, R.I., Hook, R.N. (Eds.), Astronomical Data Analysis Software and Systems XII. In: Astronomical Society of the Pacific Conference Series, vol. 295. p. 43.

Graham, M., Demleitner, M., Dowler, P., Fernique, P., Laurino, O., Lemson, G., Louys, M., Salgado, J., 2013. UTypes: current usages and practices in the IVOA. IVOA Note, February. URL: http://www.ivoa.net/documents/Notes/UTypesUsage.

Graham, M., Plante, R., Tody, D., Fitzpatrick, M., 2014. PyVO: Python access to the Virtual Observatory, Astrophysics Source Code Library, February.

Harrison, P., 2011. Relational registry DM. IVOA Wiki page. URL: http://wiki.ivoa.net/twiki/bin/view/IVOA/RelationalRegistryDM.

Harrison, P., Burke, D., Plante, R., Rixon, G., Morris, D., 2012. StandardsRegExt: a VOResource schema extension for describing IVOA standards, version 1.0. IVOA Recommendation, May. URL: http://www.ivoa.net/Documents/StandardsRegExt/20120508/REC-StandardsRegExt-1.0-20120508.html.

IVOA Registry WG 2011. Requirements and use cases for new registry search interface. IVOA Wiki page. URL: http://wiki.ivoa.net/twiki/bin/view/IVOA/RestfulRegistryInterfaceReq.

Kani-Zabihi, E., Ghinea, G., Chen, S.Y., 2008. User perceptions of online public library catalogues. Int. J. Inf. Manage. 28, 492–502.

Lemson, G., Bourgès, L., Laurino, O., 2014. VO-DML: a consistent modeling language for IVOA data models, Data Model WG Internal Working Draft 2014-09-20, 9. URL: https://volute.googlecode.com/svn/trunk/projects/dm/vo-dml/doc/VO-DML-WD-v1.0.pdf.

Louys, M., Bonnarel, F., Schade, D., Dowler, P., Micol, A., Durand, D., Tody, D., Michel, L., Salgado, J., Chilingarian, I., Rino, B., de Dios Santander, J., Skoda, P., 2011. Observation data model core components and its implementation in the Table Access Protocol, version 1.0. IVOA Recommendation. URL: http://www.ivoa.net/Documents/ObsCore.

Michel, L., Louys, M., Bonnarel, F., 2014. Browsing TAP services with TAPhandle and datalink. In: Manset, N., Forshay, P. (Eds.), Astronomical Society of the Pacific Conference Series. In: Astronomical Society of the Pacific Conference Series, vol. 485. p. 15.

Molinaro, M., Demleitner, M., Ramella, M., Iafrate, G., 2014. Educational resources in the virtual observatory, IVOA Education IG internal note, 2014-02-25, 2. URL: http://volute.googlecode.com/svn/trunk/projects/edu/edumatters/edumatters-fmt.html.

Open Archives Initiative, 2002. The open archives initiative protocol for metadata harvesting, version 2.0. URL: http://www.openarchives.org/OAI/openarchivesprotocol.html.

Ortiz, I., Lusted, J., Dowler, P., Szalay, A., Shirasaki, Y., Nieto-Santisteba, M.A., Ohishi, M., O'Mullane, W., Osuna, P., 2008. The VOQL-TEG, the VOQL Working Group, IVOA astronomical data query language. IVOA Recommendation. URL: http://www.ivoa.net/Documents/latest/ADQL.html.

Osuna, P., Barbarisi, I., Salgado, J., Arviset, C., 2005. VOSpec: a tool for handling virtual observatory compliant spectra. In: Shopbell, P., Britton, M., Ebert, R. (Eds.), Astronomical Data Analysis Software and Systems XIV. In: Astronomical Society of the Pacific Conference Series, vol. 347. p. 198.

Plante, R., Linde, T., Williams, R., Noddle, K., 2007. IVOA identifiers, version 1.03. IVOA Recommendation, March. URL: http://www.ivoa.net/Documents/REC/Identifiers.

Robie, J., Chamberlin, D., Dyck, M., Snelson, J., 2014. XQuery 3.0: An XML query language. W3C Recommendation. URL: http://www.w3.org/TR/xquery-30/.

Rots, A., 2005. STC-X: Space–time coordinate (STC) metadata XML implementation. IVOA Note, March. URL: http://www.ivoa.net/documents/latest/STC-X.html.

Taylor, M.B., 2005. TOPCAT & STIL: starlink table/VOtable processing software. In: Shopbell, P., Britton, M., Ebert, R. (Eds.), Astronomical Data Analysis Software and Systems XIV. In: Astronomical Society of the Pacific Conference Series, vol. 347. p. 29.

Taylor, M., 2010. Registry interfaces 1.0: A service consumer's experience. Talk given at the IVOA Interop May 2010, Victoria. URL: http://wiki.ivoa.net/internal/IVOA/InterOpMay2010Reg/reg.pdf.

Taylor, M., Boch, T., Fitzpatrick, M., Allan, A., Fay, J., Paioro, L., Taylor, J., Tody, D., 2012. Simple application messaging protocol. IVOA Recommendation. URL: http://www.ivoa.net/documents/SAMP/.

Tedds, J.A., Winstanley, N., Lawrence, A., Walton, N., Auden, E., Dalla, S., 2008. VOExplorer: visualising data discovery in the virtual observatory. In: Argyle, R.W., Bunclark, P.S., Lewis, J.R. (Eds.), Astronomical Data Analysis Software and Systems XVII. In: Astronomical Society of the Pacific Conference Series, vol. 394. p. 159.

Tody, D., Dolensky, M., McDowell, J., Bonnarel, F., Budavari, T., Busko, I., Micol, A., Osuna, P., Salgado, J., Skoda, P., Thompson, R., Valdes, F., 2012. Simple spectral access protocol version 1.1. IVOA Recommendation. URL: http://www.ivoa.net/documents/SSA/20120210/REC-SSA-1.1-20120210.htm.

Tody, D., Plante, R., 2009. Simple image access specification. IVOA Recommendation. URL: http://www.ivoa.net/Documents/latest/SIA.html.

Williams, R., Hanisch, R., Szalay, A., Plante, R., 2008. Simple cone search. IVOA Recommendation. URL: http://www.ivoa.net/Documents/latest/ConeSearch.html.