IEAD: A Novel One-Line Interface to Query Astronomical Science Archives

MARCO LOMBARDI
University of Milan, Department of Physics, via Celoria 16, I-20133 Milan, Italy

Received 2011 December 21; accepted 2012 February 8; published 2012 March 5

ABSTRACT. In this article I present IEAD, a new interface for astronomical science databases. It is based on a powerful, yet simple, syntax designed to completely abstract the user from the structure of the underlying database. The programming language chosen for its implementation, JavaScript, makes it possible to interact directly with the user and to provide real-time information on the parsing process, error messages, and name resolution of targets; additionally, the same parsing engine is used for context-sensitive autocompletion. Ultimately, this product should significantly simplify the use of astronomical archives, inspire more advanced uses of them, and allow the user to focus on what scientific research to perform, instead of on how to instruct the computer to do it.

Online material: color figure

1. INTRODUCTION

Nowadays, a well-maintained and easily accessible data archive is critical to the success of a mid-to-large telescope facility. This is best appreciated if one looks at the large amount of pure archival articles, i.e., articles written using data from observations that were not proposed by any of the authors. For example, as noted already by Walsh & Hook (2006), approximately half of the publications based on Hubble Space Telescope data are purely archival. Furthermore, archival research is, of course, largely dominant for surveys and for dedicated telescopes, such as the Sloan Digital Sky Survey (York et al. 2000) or the Two Micron All Sky Survey (Skrutskie et al. 2006).

Astronomical archives are very complex. On one hand, the internal structure of the database is often complicated by the need to collect intrinsically different kinds of data, taken for different purposes by different instruments. On the other hand, the archive interface must serve technical customers, the astronomers, who sometimes need to perform very specific queries. As a result, typical archive interfaces (almost always accessible through a dedicated World Wide Web page) are often plagued by a large list of different fields and buttons to be able to accommodate queries from the most demanding (and technically inclined) user.

Unfortunately, this also means that many archive interfaces are also clumsy and unfriendly for the large majority of users. For example, a recent survey among the ESO Science Archive users (Delmotte et al. 2006) has shown that the most requested improvements for the archive interface are the possibility to perform more complex queries (23%), an easier-to-use interface (20%), and a less dense main-query page (17%). Clearly, these three suggestions cannot be followed at the same time using a classical interface.

As of recently, an increasing number of astronomical archive interfaces now accept queries written in SQL or extensions of this language (such as ADQL). Unfortunately, while presenting a clean query page, these solutions require the user to know the internal structure of the database used at a relatively deep level, and they ultimately make the database inaccessible to the less technically inclined users. Additionally, since different astronomical databases often have completely different structures, for each archive used, one is forced to learn very specific details that are of no use in different contexts.

However, perhaps the most serious limitation of the currently available interfaces to astronomical databases is the fact that they force the user to express the query in a form that (in all cases) strictly reflects the structure of the database. Ideally, instead, the user should focus on the science, and the interface should be flexible enough to allow a direct formulation of the user’s wishes.

In this article I present the Interface for the Exploration of Astronomical Databases (IEAD), a new query interface for astronomical science archives that solves the limitations discussed previously. The concepts presented here have been developed for the Hubble Space Telescope (HST) Science Archive, but could be applied equally well without significant modifications to any astronomical data archive (and the software has been developed with this aim in mind). IEAD is currently available (integrated within a more standard query interface) at the Space Telescope European Coordinating Facility (ST-ECF) HST.
2. IEAD: THE CONCEPT

With the advent of powerful and “smart” search engines, we all are used to the idea that simple interfaces should be provided to perform recurrent tasks. However, behind the apparent simplicity of these search engines, there are a lot of complex tasks that are performed behind the scenes; for example, some Web search engines allow queries to be formulated using natural languages, or with Boolean operators, or using special keywords to restrict the outputs. In general, it appears that the current focus in the development of search engines is to adapt them to the user’s needs, rather than to force the user to adapt to their designs and limitations.

A few search engines are designed to be able to perform both simple and (very) complex queries. Examples can be found in many Google products (the standard Web search engine, but also the search interface for messages in Gmail or for really simple syndication (RSS) in the Google RSS reader) and on many e-commerce sites (such as Amazon). In the astronomical field, a similar but much simpler product can be found in the one-line NASA Astrophysics Data System (ADS) interface. In a different context, the get script command of Aladin can associate automatically a set of keywords (that can be entered in an arbitrary order) with the server query vocabulary. It is along these lines that I designed the IEAD search engine for the Hubble Space Telescope Archive.

Ideally, the “perfect” interface should be very intuitive to use, and little or no explanations should be needed; still, it should be powerful enough to allow complex queries. The interface should use a simple syntax or should accept queries formulated in a natural language. Finally, the user should be able to profit from the archive without knowing the structure of the database in detail.

Natural-language user interfaces are generally difficult to implement and represent a very active field of research in computer science (e.g., Androutsopoulos et al. 1995; Popescu et al. 2003). In particular, a critical task in a natural-language interface is the entity identification: i.e., the classification of the various terms present in a query. Fortunately, astronomical data archives are very favorable in this respect, because many query terms can be uniquely associated with a particular data field: a fact that makes it possible to have unambiguous and still very simple queries. For example, quantities such as instrument names, camera names, filters, and optical element types (such as filter, grism, or prism) can assume only a fixed set of single-word values. Slightly more complex values, such as principal investigator (PI) names or data type names (such as imaging or 2D spectroscopy) still assume values from a limited set of single or multiword values. Real values such as astronomical coordinates, search radii, exposure times, exposure dates, or observing wavelengths can usually be easily disentangled from one another by their format (say, hh:mm:ss for right ascension vs. ±dd:mm:ss for declination) or from their units (30" is probably a search radius, while 2 h is likely an exposure time).

Finally, everything else, i.e., everything that are not recognized as one of the fields mentioned previously, has to be a target name, and this assumption can then be verified using the SIMBAD (Wenger et al. 2000) or NED (Helou et al. 1991) name resolvers.

In this simple approach, the string (queries are case-insensitive) acs m42 f775w is immediately understood to be a query for all instrument=acs data taken with the filter= f775w around the (SIMBAD-resolved) coordinates of m42. This simple example immediately highlights one of the main advantages of the IEAD system over the other query interfaces commonly found for astronomical databases: a level of abstraction is removed, and the user is free to express the query in a much more natural way. This unique feature makes the use of the database more direct and ultimately makes database research much easier by bringing the query language closer to the astronomer. Note that although the use of automatic entity identification is not entirely new in the astronomical context (see the aforementioned NASA ADS query interface and the Aladin get script command), IEAD pushes this concept greatly forward: now entire database queries, possibly composed of more than 20 different kinds of entities, are automatically parsed. Compared with standard archive interfaces, as an additional bonus, the user does not need to look for the right quantities and enter different values in the correct fields, but can instead mix together all values and still obtain sensible results.

As a second example, the string stis oii imaging planetary nebula is interpreted as a query for all instrument=stis data type=imaging taken with the filter =oii for description="planetary nebula". Finally, the example nic3 2d spectroscopy hdf-n thompson >30min can be used to select all camera=nic3 data type="2d spectroscopy" observations made by pi=thompson around the target=HDF-N, with exptime>30min.
3. USER AIDS

Since the IEAD query interface is based on JavaScript, it runs entirely on the user computer; additionally, the parser is custom-written and is therefore fast enough to parse complex queries in real time. This makes it possible to complement the interface with two user aids: an automatic display of the parsing result and a smart autocompletion on the query.

The automatic display of the parsing result shows not only how every word entered is interpreted, but also the way that the various constraints are linked together. For example, for the first example discussed previously, the interface shows (\texttt{inst=acs and (target=m42[ok,HII] and box=600arcsec) and filter=f775w}). This text provides a wealth of information. First, it is obvious that \texttt{acs} is taken to be an instrument, \texttt{f775w} is a filter, and \texttt{m42} is a target name. Additionally, the target has been correctly resolved and is a H\textsc{ii} region ([\texttt{ok}, \texttt{HII}]); note that the resolver (SIMBAD) and the meaning of the H\textsc{ii} code (H\textsc{ii} ionized region) are visible if the user leaves the cursor over the H\textsc{ii} text. All terms are combined using the predefined Boolean operator, and the coordinate search is performed around the target name with the default searching box (10').

In case of a parsing error, the system is also able to immediately show a descriptive error message and to indicate where the error took place. Similarly, the system also informs the user in real time when a target cannot be resolved with a no-stopping error (in the sense that the user can still type text in the field), since the problem might rely on the target name to be incomplete; however, the query will not be executed if the target cannot be resolved.

Additionally, when the user starts filling in the query field, the system shows all possible completions. The autocompletion is automatically shown as soon as the last query text has an incomplete word at the end (i.e., no completion is shown if the query text is empty or if its last character is a space). The autocompletion uses the complete parsing engine and is therefore fully context-sensitive; at any given moment the system knows the exact possible completion words and can show them to the user.

Finally, it is worth mentioning that both the ST-ECF and the CADC \textit{HST} Archive pages embed the IEAD system inside a traditional query form, composed of different entries for each field. In these pages each entry from the traditional query form can be dragged and dropped into the IEAD one-line interface. This should help users to get acquainted with the new interface and to enter their constraints for difficult-to-remember uncommon keywords.

4. GENERAL SYNTAX

The general syntax accepted by IEAD is represented in Figure 1 and will be explained in detail in this section. It is based on qualified terms: i.e., on a combination of \texttt{keyword-operator-value}, such as \texttt{instrument=ACS}. Both the keyword and the operator can be omitted. If the keyword is omitted, it is inferred from the value (in the preceding example, \texttt{ACS} is obviously an instrument); if the operator is omitted, the equal sign (=) is assumed, unless the value is preceded by a minus sign, in which case the assumed operator is nonequal (!=). Therefore, all these queries are equivalent to the one written previously (\texttt{instrument ACS =ACS ACS}), while \texttt{! =ACS} is interpreted as \texttt{instrument !=ACS}.

4.1. Automatic Keyword Identification

As discussed previously, the entity identification is a key feature of the IEAD system and a critical step toward a natural-language query. This task is performed by attaching a keyword and operator to each word (or group of words), when these have not been explicitly assigned.

The automatic identification of the keyword uses a simple but effective scheme. The value is compared with a set of known values or formats, and the first matching keyword is used. Keywords are sorted in a way that the most used ones or the most restrictive ones (i.e., those that give a match only in rather specific cases) are at the beginning, so that in the automatic identification is successful the large majority of cases (see Table 1 for a list of accepted keywords). Note, in particular, that the target name is the last tried keyword: target names can have many different formats; additionally, it is very time-consuming to verify if a phrase can be used as a target name. As mentioned in § 3, the user can verify the automatic keyword identification in real time; if necessary, in the rare cases where the identification is inappropriate or where the keyword cannot be automatically recognized (see the last column of Table 1), the user can also specify the desired keyword.

4.2. Formats for Values

A second key feature of the IEAD system is the treatment of values and their identification. This part of the parsing process influences the automatic keyword identification and has therefore been designed with particular care. The currently possible kinds of values include the following:

\textbf{Word}.—A single word can be used to specify an instrument, a filter, or a data set ID, for example. Typically, these values can be assigned an automatic keyword for quantities that can only assume a value among a limited list of values (such as the instruments) or that have a specific format (such as the data set ID).

\textbf{Phrase}.—A few values are composed of several words (see, for example, the \texttt{dataset} keyword in the examples of § 2). These can be placed one after the other or can be enclosed within single or double quotes to avoid ambiguities.

\textbf{Number}.—Integer or floating-point values. Floating-point numbers can use the scientific notation \texttt{d.ddde \pm dd}: i.e., with the letter \texttt{e} as a separator between the mantissa and the exponent.
Fig. 1.—Simplified syntax diagram for IEAD. Nonterminal nodes are indicated using rectangles, and terminal nodes use circles and ovals. See the electronic edition of the PASP for a color version of this figure.
nates are always taken to be equatorial; in particular, coordi-
nates without sign are interpreted as right ascension, and

cordinates with sign are interpreted as declination.  

Date.—Dates must be always entered in the format yyyy-

mm-dd.  

Range.—A range can be entered using the format min...max,

where either min or max can be omitted (but not both). Ranges

are accepted for all cases where a numerical value is valid, in-

cluding angles, dates, numbers, and numbers with unit (in this
case, a unit entered only for one of the two extremes is also

applied to the other extreme). A range can only be used with

Table 1

List of Currently Accepted Keywords and Aliases

| Keyword       | Aliases                                      | Type     | Units | Automatic |
|---------------|----------------------------------------------|----------|-------|-----------|
| inst          | instr, instrument                            | Word     | ...   | Yes       |
| camera        | detector                                     | Word     | ...   | Yes       |
| filter        | opt_elem, optical_element                    | Word     | ...   | Yes       |
| opt_elem_type | optical_element_type                         | Word     | ...   | Yes       |
| data_type     |                                              | Phrase   | ...   | Yes       |
| date          | obsdate, observation_date                    | Date     | ...   | Yes       |
| released      | release_date, observation_date               | Date     | ...   | No        |
| ra            |                                              | Anglec   | ...   | Yes       |
| dec           | decl, declination                            | Anglec   | ...   | Yes       |
| glon          | galactic_longitude, gal_lon                 | Anglec   | ...   | Yes       |
| glat          | galactic_latitude, gal_lat                  | Anglec   | ...   | Yes       |
| elon          | ecliptic_longitude, ecl_lon                 | Anglec   | ...   | Yes       |
| elat          | ecliptic_latitude, ecl_lat                  | Anglec   | ...   | Yes       |
| box           | radius, r, within_search_box                | Unit     | Anglec| Yesd      |
| hst_target    | hst_target_name, hst_name                   | Phrase   | ...   | No        |
| description   | descr, target_description, targetdescription, targetdescription | Phrase   | ...   | Yesd      |
| exptime       | exposure, exposure_time                      | Unit     | Time  | Yes       |
| prop          | proposal, proposalid, proposal_id, prop_id  | Integer  | ...   | Yes       |
| pi            | pi_name, piname, principal_investigator     | Pi       | ...   | Yes       |
| dataset       | dataset_name, data_set_name                 | Word     | ...   | Yes       |
| title         | proposal_title, prop_title                  | Phrase   | ...   | No        |
| resolution    | spatial_resolution                          | Unit     | Arcsec| Yesd      |
| scale         | pixel_scale, pixel                          | Unit     | Arcsec| No        |
| slew          | moving, moving_object, moving_target        | Flag     | ...   | No        |
| wavelength    | wave, lambda                                 | Wavelength| Wavelength| Yes      |
| bandwidth     |                                              | Unit     | Wavelength| No       |
| spec_res      | spectral_resolution                         | Unit     | Wavelength| No       |
| res_power     | resolving_power, respower                   | Float    | ...   | Yes       |
| time_start    | start                                       | Date     | ...   | No        |
| time_end      | end                                         | Date     | ...   | No        |
| members       | no_members                                  | Integer  | ...   | No        |
| mode          | photon_mode                                 | Phrase   | ...   | No        |
| extension     | science_extension                           | Phrase   | ...   | Yes       |

a Coordinate pairs.
b Coordinate pairs.
c Coordinate pairs.
d The box term is valid only near a coordinate, and in that case takes priority over resolution. Therefore, resolution is automatic only when there is no nearby coordinate.
e The description is automatically recognized for the most common description phrases, as deduced from a periodic analysis of the proposal database; typical examples include “star-forming region” or “cluster of galaxies.”
f In addition to wavelength units, the following astronomical bands are also recognized: ultraviolet, optical, infrared, u, b, g, r, i, z, j, h, and k.
the equality (=) or inequality (≠) operator and is then converted internally into expressions such as keyword ≥ min and keyword & lt; = max for the equality and keyword < min or keyword > max for the inequality operators.

4.3. Special Cases

A few terms are interpreted in a special way to accommodate particular cases. Many of these are especially important because the interface performs a query expansion for them; i.e., it extends the meaning of the human-entered values to adapt them to the database.

**Target name.**—All terms that cannot be automatically associated with any keyword are taken to be target names. These are resolved in real time through a call to the Sesame name resolver, which queries the SIMBAD, NED, and VizieR (Ochsenbein et al. 2000) databases in sequence, by default. The result of the Sesame check is immediately shown to the user as soon as it is available: typically, within a couple of seconds. The results are cached, so the same target is never queried again to Sesame within the same session.

**Coordinate pair.**—Two close coordinate terms (right ascension and declination, or Galactic latitude and longitude, or Ecliptic latitude and longitude) are interpreted as a coordinate pair. This is important when parsing positional constraints, since these are almost always taken with an implicit or explicit “fuzziness” (see “Search radius,” next).

**Search radius.**—An isolated angle quantity (i.e., a number followed by an angular unit, such as 2°) is taken to be an indication for an angular resolution of the observation. However, when the same quantity appears close to a coordinate term (such as 12:32:45, which is parsed as a right ascension coordinate), to a coordinate pair, or to a target name, it is taken to be a search radius for around the coordinate, the coordinate pair, or the target. When not specified, all coordinates with an (implicit or explicit) equality or inequality operator are taken to have a search radius of 10’. Note also that a search radius works differently, depending on if it is applied to a point in the celestial sphere (a coordinate pair) or to a single coordinate (in this case, the search “radius” does not identify a disk, but rather a stripe in the sky).

**PI names.**—Since different people have different habits for writing names, the system processes PI names so that the first name can be entered before or after the last name (more complicated cases in which a middle name is present or in which the last name is composed of more words are also contemplated). Internally, the entered PI name is mapped into the format used by the database. Additionally, for PIs, the dot is equivalent to the * wildcard (see § 4.4).

**Spectral range.**—This is a special kind of number with unit, where the value entered is interpreted by requiring that the specific wavelength is within the filter sensitivity of the data set (or the specified wavelength range has a nonvanishing intersection with the filter sensitivity). Additionally, the spectral range can also be entered using standard Johnson-Cousins filter names, which are approximately translated into the corresponding pivot wavelengths. This feature, together with the capability of the system to recognize simple phrases for data types (such as imaging or 2d spectroscopy), makes it possible to use the interface without a specific knowledge of the HST instrument capabilities.

### Table 2

| Unit type   | Default       | Units                          | Factor |
|-------------|---------------|-------------------------------|--------|
| Angle       | arcsec        | arcsec, arcsecond, arcseconds | 1/3600 |
| Angle       | arcmin        | arcmin, arcminute, arcminutes | 1/60   |
| Angle       | deg, degree   | degrees                       | 1      |
| Time        | s             | s, sec, second, seconds       | 1      |
| Time        | m             | min, minute, minutes          | 60     |
| Time        | h             | hour, hours                   | 3600   |
| Wavelength  | nm            | nm, nanometer, nanometers     | 1      |
| Wavelength  | a, ang, angstrom, angstroms | 1/10 |
| Wavelength  | um, micron, microns | 1000 |

**4.4. Wildcards and Anchors**

IEAD also accepts two wildcards for text (nonnumerical) values: the asterisk (*), matching any text, and the question mark (?), matching a single character. These are the same wildcards used in globbing by UNIX shells and should therefore be familiar to many users.

Additionally, the system also accepts the caret (^) to match the beginning of a value and the dollar sign ($) to match the end of a value. Therefore, to force a keyword to have an exact given value, one should use both anchors, as in title=^star$. Note that the simple title=star would also match observations with titles such as “A peculiar star in a nebula.” Both wildcards and anchors can be escaped using the backslash, as in \\ or \^.

**4.5. Multiple Constraints**

Multiple constraints can be simply written one after the other. Again, the specific framework of our query language,
astronomical databases, makes it simple to define rules that correspond to those of a natural language:

1. All terms with (implicit or explicit) equality operator that share the same (implicit or explicit) keyword are combined with an or Boolean operator;
2. Other terms are combined with a and Boolean operator.

Note that in different contexts, this particular problem, i.e., the so-called conjunction and disjunction ambiguity, is difficult to solve, because it requires a knowledge of the relationships among the various entities, while in the astronomical context, the solution is obvious: all entities (for example, instruments, cameras, and filters) are mutually exclusive, in the sense that an observation can only use one of them at a time.

These rules ensure that the simple query m42 acs wfc3 is interpreted as a search for observations around M42 carried out either with the ACS or with the WFC3 instruments, while m42 - acs - wfc3 is interpreted as a search for observations around M42 carried out with neither the ACS nor the WFC3 instruments.

4.6. Full Boolean Queries

The combination of the automatic identification and the simple syntax for combined constraints nicely solves the large majority of queries. However, in specific cases one might desire or need to perform more specific queries involving a combination of parameters.

In these situations it is possible to include the Boolean operators and, or, and not in the query and to use them as one would normally do in any programming language. For example, the string (acs and grism) or (stis and prism) might represent a sensible query. It is possible to mix multiple constraints without Boolean operators with queries involving Boolean operators; for example, the query acs grism or stis prism would be interpreted exactly as the preceding example (note that the implicit Boolean operators inserted between multiple constraints have a higher priority than explicit Boolean operators).

5. IMPLEMENTATION

As mentioned earlier, IEAD is entirely written in JavaScript, and so the code runs directly on the user’s browser and can perform truly interactive actions, such as the automatic display of the parsing result and the context-sensitive autocompletion. Indeed, the choice of the programming language for the final implementation has been mainly driven by the possibility to display interactive messages to the user (the original prototype of the interface was written in Python).

The code is object-oriented and is built over the concept of a term: i.e., a combination of keyword-operator-value. It defines a different kind of object for each different type of term: number (integer or float), angle, date, word, phrase, unit, flag, wave-length, and pi (see Table 1). All term classes are organized hierarchically with a common ancestor.

Each object has a constructor, which defines the keyword (and associated aliases) of the term, the associated field in the database, and type-dependent options (for example, for the angle in the allowed range, the obligatory nature of the sign, and a flag to indicate if the field refers to right ascension or not). Additionally, all term classes define a number of common members to deal with several common actions, such as the verification of the input, the parsing, the error handling, and the autocompletion. Finally, a member function takes care of the translation of the term into SQL code or into a human-readable string.

The program also defines metaterms: i.e., classes to modify the behavior of other terms. For example, there is a metaterm that modifies other terms to make the keyword and/or the operator of a term compulsory; another metaterm makes other terms optional (in the sense that if not present, then a default value is used; see the use of the search box, explained previously). However, the most important metaterm is the one to generate ranges that use a pair of dots as the separator.

Finally, all terms are combined into a parser for an expression that can involve Boolean operators and parentheses: again, this process is realized within a special class with a structure similar to those of a term.

In summary, the code uses the following simple scheme:

1. The query is initially handled by a simple lexer that splits the string into tokens: i.e., words that have an individual meaning in the language used by IEAD.
2. The tokens are passed to the expression parser, which analyses them in order.
3. The parser tries to parse the various terms by trying, in sequence, all term types that define an expression. The first matching term is used for the rest of the parsing. The last possible term tested is the target one, which accepts all tokens by default (unless a different keyword is specified). When the target token is used, a query to the Sesame database is also started in parallel.
4. The preceding point is repeated until all tokens are consumed.
5. The code uses the parser to translate the query into a human-readable string that is shown to the user.
6. When the query is finally executed, the parser is called again to generate an SQL code. This code is used in a special field in an HTML form and is then passed to the server, which uses it to directly interrogate the database with very little manipulation.

5.1. Internal Database

The IEAD code uses a simple internal JSON database to save the values for the various terms that have a fixed set of permitted values, such as the instrument names or the filter names.
Periodically, this database is updated to reflect the status of the full HST Archive; this process is particularly important for values such as the PI names, which are likely to change often (basically, each time observations from a new PI are carried out).

This simple process is handled by a straightforward Python script. The only interesting point to note here is that the script performs the same analysis for the description term. Since (almost) each proposal has a different description, often containing several words, it would be unpractical and probably not very useful to use these values as they are for the description term. Rather, the Python script analyzes the various descriptions and extracts from them the most common phrases. These are automatically recognized and used for autocompletion in the interface. Of course, one is still free to query for a particular description using a fully qualified term, as in description=phrase.

6. FUTURE PROSPECTS

The research presented here is the first step toward more advanced, efficient, and intuitive query interfaces for astronomical databases. However, this should not be considered the definitive, optimal solution. Instead, given the role that astronomical databases will play in the future astronomical research, it is critical to develop even more advanced interfaces to fulfill the needs of the users and to stimulate different uses of the astronomical databases. The interface discussed in this article could be improved in various ways, and it is useful to briefly consider future research directions here.

The interface could provide instant previews of the entered query, without forcing the user to press the search button each time. This would allow one to explore the archive in real time and, as such, would represent a major step forward. Unfortunately, at the present time, the structure and the query performance of most archives do not allow such a search to be performed in real time, and major enhancements in the database software and servers would be needed for this task.

Improvements could also be made in the autocompletion scheme. So far, the autocompletion system presents a truncated (or, optionally, the full) list of possible completions for a query, but it does not try to perform a “smart” job. First, the list produced is presented in alphabetical order, which is not the most useful approach (it would be much more sensible to present the list of completions by sorting them by popularity: i.e., frequency of use). Second, the autocompletion is only grammatically context-sensitive, not semantically; the proposed completions are likely to contain possibilities that would be considered to be obviously wrong by an experienced user (for example, the interface would include the WFPC2 instrument as a possible completion of “WF,” even if the user already selected the grism G280, which is not available for this instrument). This improvement, however, appears to be rather complicated to develop.

As mentioned already, the new interface implements many of the tasks needed in a natural-language query, such as entity identification, query expansion, and conjunction-disjunction disambiguation. The ad hoc implementation is computationally efficient, but also has various limitations: the set of keywords must be provided to the system together with a way to discover from the database the set of allowed values, and the query language is kept at a very basic level. There are various possible solutions for these issues. The need for a specific configuration of the system (keywords and predefined values) could be removed through the use of virtual observatory (VO) registries: this would make it possible to port the interface to all VO-compliant archives, possibly even without the explicit collaboration from the maintainers. Regarding possible extensions of the query language, it would be very interesting to investigate modern techniques used in natural-language research, involving statistical inference and machine learning. However, one should also be aware of the possible risks of a full natural-language query; on one hand, the lack of well-defined grammar and, therefore, of the set of possible queries might keep the users away from queries that are considered too complex; on the other hand, many users would expect the system to be “intelligent” and might request queries that are outside of its capabilities.

7. CONCLUSIONS

In this article I presented IEAD, a new one-line query interface for astronomical science databases. The major advantages of this interface over standard ones are as follows:

1. Queries are performed on a single line. This makes query pages very clean, avoiding the clumsiness that is often present in astronomical archive interfaces, and lets the user concentrate on the query (instead of on the search of the right field of each constraint).

2. The interface uses a simple syntax, designed to minimize the quantity of text that the user has to enter and to be close to a natural language.

3. Queries involving complex combination of parameters, Boolean operators, and parentheses are possible.

4. The interface provides immediate feedback on the parsing of the entered string and on the resolution of astronomical object names. Additionally, it has context-sensitive autocompletion.

5. The code is easily integrable in any SQL-compliant astronomical database and is extensible and usable for different telescopes or observatories.

I am grateful to F. Stoehr and J. Haase for helpful and constructive interactions, and to R. Fosbury and J. R. Walsh for encouraging and supporting this project.

---

7 See http://www.ivoa.net/Documents/RegistryInterface/.
REFERENCES

Androutsopoulos, I., Ritchie, G. D., & Peter, T. 1995, Nat. Lang. Eng., 1, 29

Delmotte, N., Dolensky, M., Micol, A., Padovani, P., Rino, B., Rosati, P., Wicenec, A., Retzlaff, J., et al. 2006, Messenger, 125, 41

Helou, G., Madore, B. F., Schmitz, M., Bicay, M. D., Wu, X., & Bennett, J. 1991, in Databases and On-Line Data in Astronomy, ed. M. A. Albrecht, & D. Egret (Heidelberg: Heidelberg Univ.), 89

Ochsenbein, F., Bauer, P., & Marcout, J. 2000, A&AS, 143, 23

Popescu, A.-M., Etzioni, O., & Kautz, H. 2003, in Proc. Int. Conf. Intelligent User Interfaces, (New York: ACM), 149

Skrutskie, M. F., Cutri, R. M., Stiening, R., Weinberg, M. D., Schneider, S., Carpenter, J. M., Beichman, C., Capps, R., et al. 2006, AJ, 131, 1163

Walsh, J., & Hook, R. 2006, Space Telescope European Coordinating Facility Newsletter, 40, 6

Wenger, M., Ochsenbein, F., Egret, D., Dubois, P., Bonnarel, F., Borde, S., Genova, F., Jasniewicz, G., et al. 2000, A&AS, 143, 9

York, D. G., Adelman, J., Anderson, J. E., Jr., Anderson, S. F., Annis, J., Bahcall, N. A., Bakken, J. A., Barkhouser, R., et al. 2000, AJ, 120, 1579