Defined Ontologies Exploration

Authors: Sebatien Derriere, Andrea Preite-Martinez, Alexandre Richard

Introduction

The ontology work in DS5 covers a wide spectrum of ontologies, associated technologies and their applications. Among these, an in-depth exploration of formal defined ontologies is performed.

These defined ontologies, while being more restrictive and difficult to build since they require formal definitions of the concepts, allow the use of automated inference tools ranging from consistency checkers to advanced semantic reasoning engines. This is especially interesting when considering databases since a semantic layer with such tools would allow automated consistency checks of the entries or advanced querying.

To experiment on these possibilities and the feasibility of a defined ontology-based system, a test case was chosen: an ontology of astronomical object types. Indeed the field is well-known, of manageable size, and related potential use-cases existed. Furthermore, standardizations of astronomical object types existed and could be used as a starting point. The SIMBAD[1] database list of object types[2] was a good candidate since it was of good size and the goal was to create and test an knowledge engine to couple with databases.

Summary of the activities

The following technologies have been explored:

- Formal ontologies representation using the Web Ontology Language (OWL) in its versions OWL-DL, OWL 1.1 and OWL2-RL
- Description Logics, from ALCN to SHOIQ(D) and the performance of their OWL implementation in automated reasoner Pellet, RACER and FaCT++
- Jena RDF framework and Protégé-OWL API to implement prototype applications.
- Protégé ontology editor.
- Graph generation with Graphviz and the DOT language.

In terms of publications the work on this ontology has lead to two IVOA Technical Notes:

- Ontology of Astronomical Object Types[3] which provides in-depth information on the ontology itself, updated with each major revision of the ontology.
- Ontology of Astronomical Object Types Use Cases[4] which covers some use cases for the ontology, including prototype implementation.

Additional information can also be found on the VOTECH wiki at:
http://wiki.eurovotech.org/twiki/bin/view/VOTech/OntologyOfObjectTypes

1 http://simbad.u-strasbg.fr/
2 http://simbad.u-strasbg.fr/guide/chF.htx
3 http://www.ivoa.net/Documents/latest/AstrObjectOntology.html
4 http://www.ivoa.net/Documents/latest/AstrObjectOntologyUseCases.html

Also, aside from DSRPs and IVOA interops, the work on the ontology of object types was the subject of presentations during the following workshops :

- Practical Semantic Astronomy Workshop 2008 in Caltech, Pasadena[5]
- Practical Semantic Astronomy Workshop 2009 in Glasgow[6]

An Ontology of Astronomical Object Types

Ontology Construction

Building a defined ontology requires formalizing conditions and definitions on the ontology's concepts. Description Logics[7] is an adequate and mature means of representing such ontologies and the Web Ontology Language (OWL)[8] is based on description logics and is probably the most widespread language for implementing ontologies. As for the OWL flavor to use, OWL-DL and its evolution OWL1.1[9] were a natural choice since only them allowed enough expressiveness to build exploitable definitions while still being decidable. Moreover, both are well-supported by existing automated reasoners. The ontology is also compliant with the latest OWL2-RL[10] flavor.

The ontology's construction was done by hand, using the Protégé-OWL[11] editor, and relied on formalizing in description logics the knowledge on object types from both documentary sources and experts of this field. However, for both performance and maintenance reasons, the goal is to include all the knowledge to be used by applications but no more.

The guidelines for ontology construction were:

- Only add conditions on concepts that are always true. This is necessary to ensure correct inferences from the reasoner.
- As a consequence, conditions expressing possibilities have to be expressed backwards (e.g. It cannot be guaranteed that a given stellar object has an proper motion in the databases though it *can* have one, but it can be guaranteed that a proper motion is always associated with stellar or sub-stellar objects)
- The main hierarchy being based on subsumption (a more general/more specific relationship), relationships between compound objects and their components are to be represented with properties *hasComponent/hasPortion* created towards this end.
- Reasoning complexity has to be kept low. This has lead to avoid using qualified cardinality restrictions when possible, and avoid putting restrictions on enumerations or intervals. Testing on intervals or even enumerations can be externalized though, so it is possible to keep the complexity lower in the ontology without sacrificing such restrictions.
- The consistency of the structure and the performance level of the reasoning is to be checked as often as needed using the reasoner
- To help linking real-world objects such as entries in object databases to the abstract concepts of the ontology, real-world data from databases such as measurements and labels is added or linked to the concepts using annotation properties.

The resulting ontology covers the whole field of object types, most of them being at least partly defined. Also, externalizing the restrictions on intervals and optimizing some restrictions enabled keeping the reasoning times quite low though the complexity of ALCIN(D), the description logic used to describe the ontology, is exponential[12].

5 http://www.cacr.caltech.edu/semast/
6 http://www.dcs.gla.ac.uk/workshops/semast09/
7 http://wiki.eurovotech.org/twiki/bin/view/VOTech/DescriptionLogics
8 http://www.w3.org/TR/owl-guide/
9 http://owl1_1.cs.manchester.ac.uk/
10 http://www.w3.org/TR/owl2-profiles/#OWL_2_RL
11 http://protege.stanford.edu/overview/protege-owl.html
12 http://www.cs.man.ac.uk/~ezolin/dl/

Implementation

Alongside with the ontology building, means of developing applications were set-up. This required mainly a reasoner and an API able to handle OWL-based ontologies manipulation and reasoner calls.

Reasoner

Choosing a reasoner required a thorough study[13]. It temporarily lead to the choice of RACER[14]. But eventually Pellet[15] turned out to be better since it reached higher levels of performance while having a much better support for non-commercial applications.

An API for OWL manipulation

The choice of an OWL API is basically a problem of compromise. On the one hand, until recently most developments were made using the Jena[16] RDF/RDFS Framework, but a great shortcoming is that it lacks specific primitives for OWL-based applications. OWL being an evolution of RDFS, it is possible to manipulate it with Jena, but at the cost of heavy additional development.
On the other hand, most OWL API are young and still in alpha stages. Eventually the Protégé-OWL API[17] was judged the best compromise since it provides all the wished functionalities and is well supported, being derived from Jena and used as basis for the Protégé-OWL editor which is itself upgraded on a regular basis.

Also, OWL elements are usually manipulated via their URI but for convenience dictated that they could be manipulated by their names (i.e. their URI without any namespace). Hence the implementation includes a class named *OntoManager* which deals with name-based manipulation. A positive side-effect is that since this class basically acts as an interface between the main part of the program and the calls to OWL API functions, porting the program to another API mostly means porting this single class instead of all the program.

Use-cases Implementation Choices

Both the Protégé-OWL API being written in Java, and the wish to be able to have the test applications running as web services lead to use Apache Tomcat as the web server and develop everything in Java / Servlet[18] / JSP[19], beginning with an extension of the API. This extension is designed for handling defined ontologies in conjunction with a reasoner and is not specific to the ontology of object types.

Additionally, a bridge to the Graphviz[20] representation software was implemented to allow graph representations of data and especially parts of the ontology subsumption structure. It includes methods that automatically build from basic data a script in DOT language to be

13 http://wiki.eurovotech.org/twiki/bin/view/VOTech/InferenceEngineTests
14 http://www.racer-systems.com/
15 http://pellet.owldl.com/
16 http://jena.sourceforge.net/
17 http://protege.stanford.edu/plugins/owl/api/
18 http://java.sun.com/products/servlet/
19 http://java.sun.com/products/jsp/
20 http://www.graphviz.org/

interpreted by Graphviz as well as methods to retrieve the image data within the Java program calling the bridge so that no knowledge of Graphviz itself is required to use it.

Application prototypes

In-depth information on the different use-cases and their prototype implementation can be found in the corresponding IVOA technical note :

http://www.ivoa.net/Documents/latest/AstrObjectOntologyUseCases.html

Registry Request Builder

The first application exploiting the ontology of astronomical object types was a request builder for querying astronomical registries. The idea of such a tool came from the limitations of existing registry querying methods. Indeed, when putting conditions on object types within a registry query, one must use existing keywords of the registry. But the following problems arise when considering astronomical object types:

- Some object types do not have a keyword associated.
- More specific keywords are not taken into account in a broader query.
- All the keywords have to be selected manually by the user if he wants the best query possible.

The ontology's main relationship – the subsumption – is the one needed to retrieve more specific or more general keywords. Starting with the concept queried on, going down the subsumption leads to more specific concepts and going up the subsumption leads to broader concepts. Hence, if the concepts are tagged with registry keywords, harvesting more specific or more general keywords. At the time only the VizieR registry keywords were added as annotations to the concepts. Indeed, though the builder is not dependent on any specific registry it requires object types keywords to achieve some results, and VizieR was richer than most registries with regard to such keywords.

Starting from the concept queried on, the search for keywords is done in two times:

- first find any keywords associated to the queried concept and any associated to more specific concepts
- Then, if no keyword has been found at this point another search is performed, this time to get the most specific subsumer having an associated keyword in order to be able to propose a query as close as possible to the original concept, albeit broader.

Registry Resource Finder

The idea with this application was to Capitalize on the first attempts to provide better registry queries with the Registry Request Builder by extending it to other registries than VizieR and allowing the user to type a free text as the input instead of having to choose the concept corresponding to the object type he wishes to query a registry on.

The main problem to solve was to interpret the input in order to know which object type the user wished to query on. To do this the resource finder relies of the following :

- The concepts of the ontology are tagged with keywords from various services.
- On top of those keywords, the concepts of the ontology are tagged with an annotation named *MISCgeneralKeywords* which is a collection of words in natural

language describing part or all of the object type represented by the concept.
- If the input is actually identical to an existing keyword annotating a concept then the annotated concept is the one to base the query on.
- Else, break the input into words and try and match them to the content of the *MISCgeneralKeywords* annotation. The closest match indicates the best concept to base the query on.

So mostly it is a preprocessing before the Registry Request Builder. Indeed the input from the user is matched to annotations in the ontology. The best match indicates the concept representing the object type to query on. Then get the keywords to use to query the registry as it is done in the registry request builder and build the query.

As far as querying registries go, the application is able to query either VizieR or the Astrogrid registry. In fact, the application could even query other services as long as the keywords for that service are present in the ontology. To show that possiblity the prototype application was made able to query the SAO/NASA Astrophysics Data System [21]

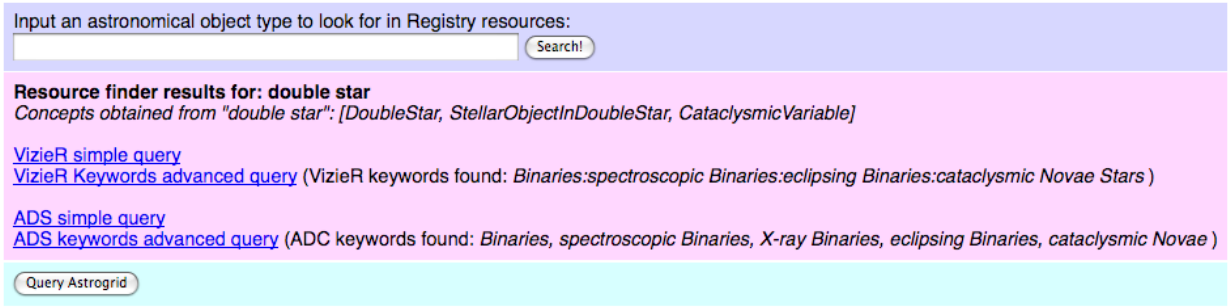


Figure 1: Screenshot of an example based on a query for the string “double star”.

Ontology Explorer

Another application using this ontology is a prototype of ontology explorer which was designed both to allow browsing the contents of the object types ontology (or any other one) and to test the performance of reasoning engines when it came to identifying an unknown concept from conditions put on it.

The interface give the details of the current concept. Conditions can be put on the concept using drop-down boxes which only allow building conditions from material the ontology and reasoner can relate to. Each a concept is altered, be it an existing concept or a new one just added, the reasoner checks if the concept is still consistent.

Asserted knowledge on the current concept and knowledge inferred by the reasoner are shown separately. Additionally, a dynamic graph showing the neighborhood of the concept (the direct ancestors and children) is shown to help visualize the hierarchy within concepts. The graph also shows which concepts are defined and which are not by using different colors and the current concept can be changed by clicking on the graph, which allows an easier navigation for users willing to browse the ontology without furthers needs like testing the performance of a reasoner.

21 http://adsabs.harvard.edu/index.html

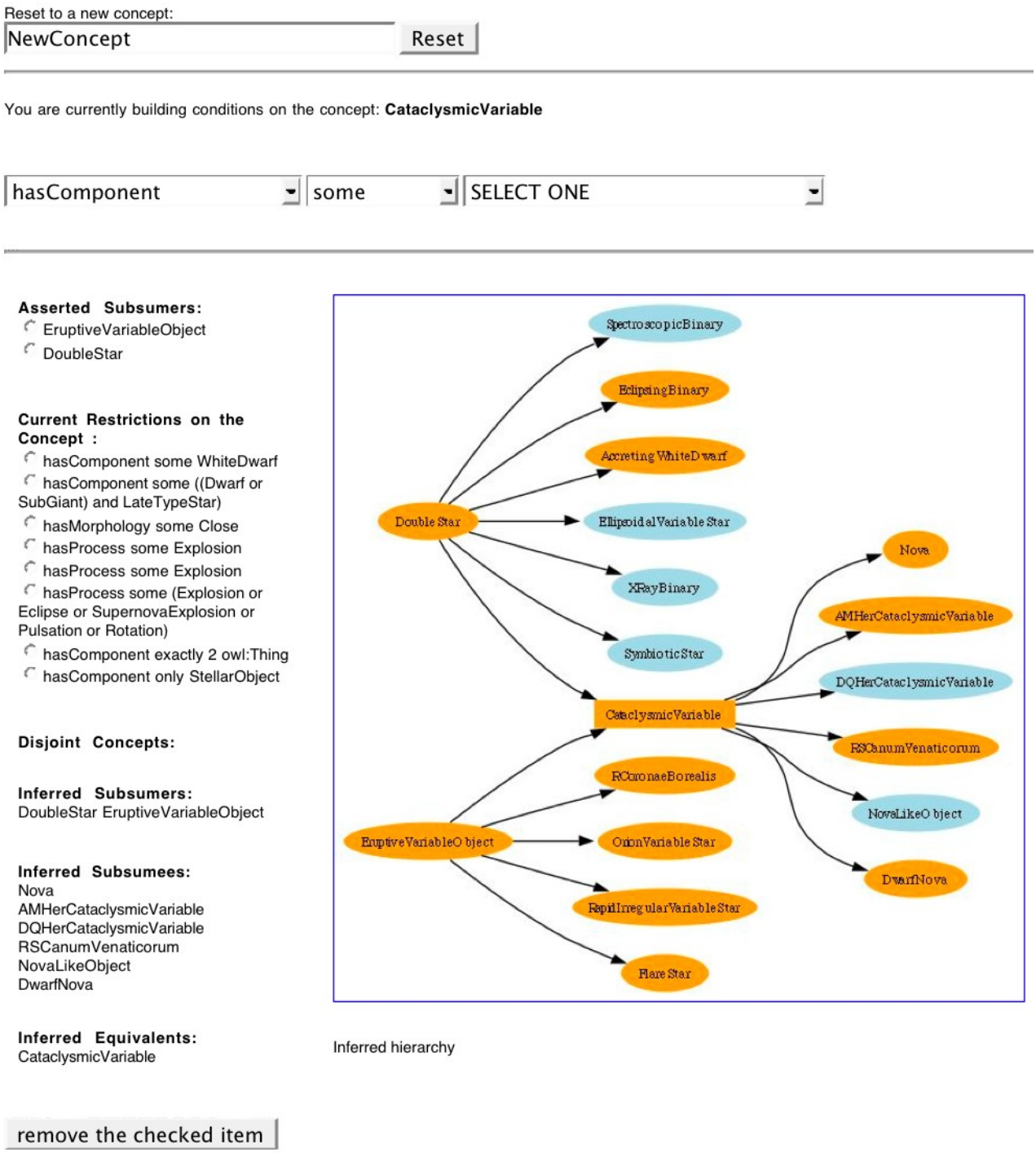


Figure 2: Screenshot of the concept explorer used to browse the ontology and showing the details and neighborhood of the concept CataclysmicVariable.

Extensive tests have lead to the conclusion that getting good inference results was highly dependent on the definitions of the concepts and the input data, which has led to building more adequate definitions for astronomical object types.

SIMBAD Consistency Checker

Cross-identification's Consistency Checker

A consistency checker for entries of the SIMBAD database was also developed. Indeed, there are currently about 4.5 million objects in SIMBAD, each of which is tagged with *otypes* which are the SIMBAD object types keywords. But most of the time, only the main otype has been set by an expert, the other otypes are inherited: a SIMBAD object inherits the dominant otype of each catalog where it is referenced. Consequently if a catalog covers a field where very different object types are considered, this can lead to inconsistencies.

The reasoner is able to check the consistency of any new element with regard to the ontology. Therefore if a concept with the same characteristics as the SIMBAD item to check is created, its consistency can be checked with regard to the ontology, which is consistent itself.

To create such a new concept conveniently, the concepts of the ontology have being annotated with their corresponding otypes. This way, the otypes from the SIMBAD entry provide a list of concepts that the new concept to check is to inherit from. After the check, if the concept is inconsistent, then the program indicates the inconsistent otypes.

Components And Measurements Checker

Once the cross-identification has been completed, additional consistency checks have been implemented both to detect more potential inconsistencies but also give more explanations about found inconsistencies. This work has taken two directions.

The first part of this extension has been checking if inconsistent otypes are not the result of the merging of compound objects and their components (e.g. A double star and its main component). If the inconsistent otypes refer to concepts one of which can be a component of the other, then it is likely that there is no real inconsistency but rather a merging of the two.

The second is checking if measurements from the SIMBAD entry are consistent with the object type of the entry itself. Currently, the measurements taken into account are the redshift and radial velocities. If such measurements are found for a given entry then they are checked with regard with the ontology. To be consistent with the ontology, it is impossible to have a radial velocity for extra-galactic objects or having a redshift for a star.

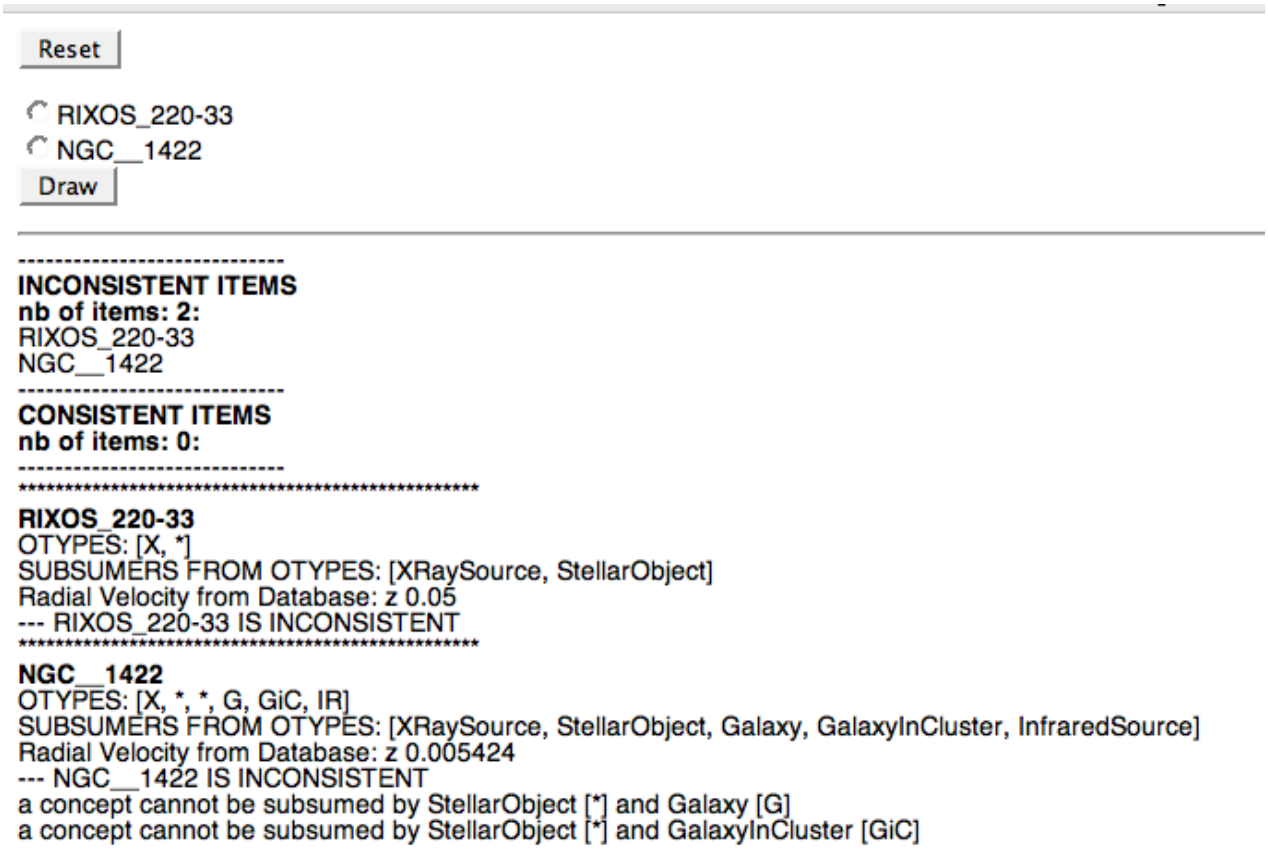


Figure 3: Screenshot of the SIMBAD Consistency checker on two items detected as inconsistent, the first for having a redshift value while its otypes state it should not have not and the second for being tagged both as a stellar object and a galaxy.

NED Consistency checker

Following the work on a consistency checker for the SIMBAD database, a version of that consistency checker was adapted to the NASA/IPAC Extragalactic Database (NED)[22].

It shows very little differences with the SIMBAD version. In fact, the most important differences are that, unlike the SIMBAD consistency checker :

- The application queries the NED database.
- Only one object can be checked at a time.
- Less measurements from the database are used for consistency checking, mostly because different measurements are present in the NED and SIMBAD databases.

Annotation hierarchy viewer

The annotation hierarchy viewer is meant to show hierarchies of keywords with regard to the ontology.

Items of the ontology -usually concepts- can be annotated, annotations having no part in reasoning. Therefore annotations are a good way of providing additional information on concepts. In particular, it is a convenient way of indicating which keywords in astronomy vocabularies or data sources correspond to a given concept.

It may be interesting to know how these keywords are organized within the ontology. Since they are present as annotations, they follow the same hierarchy as the concepts they annotate which the viewer shows as a graph.

22 http://nedwww.ipac.caltech.edu/

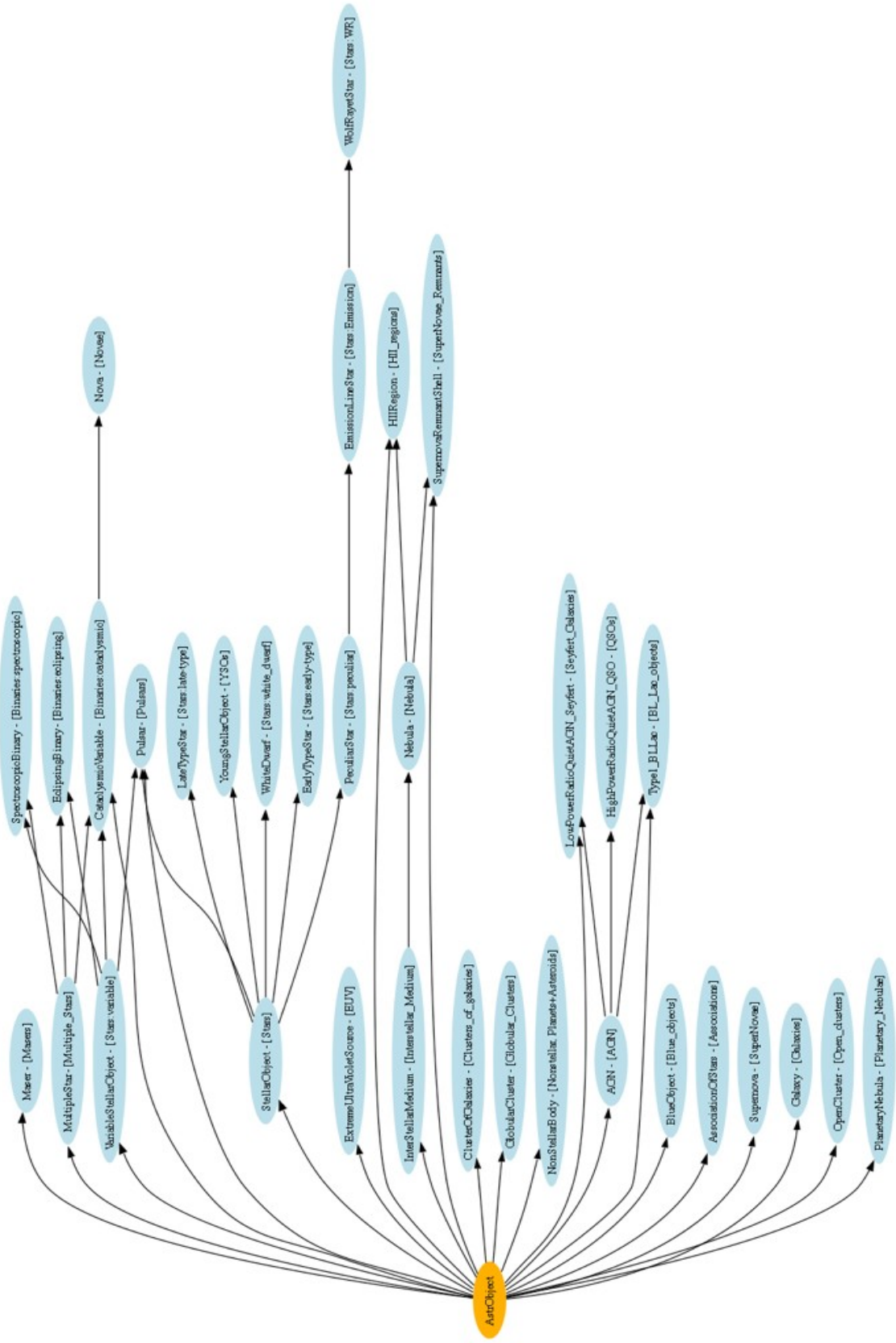


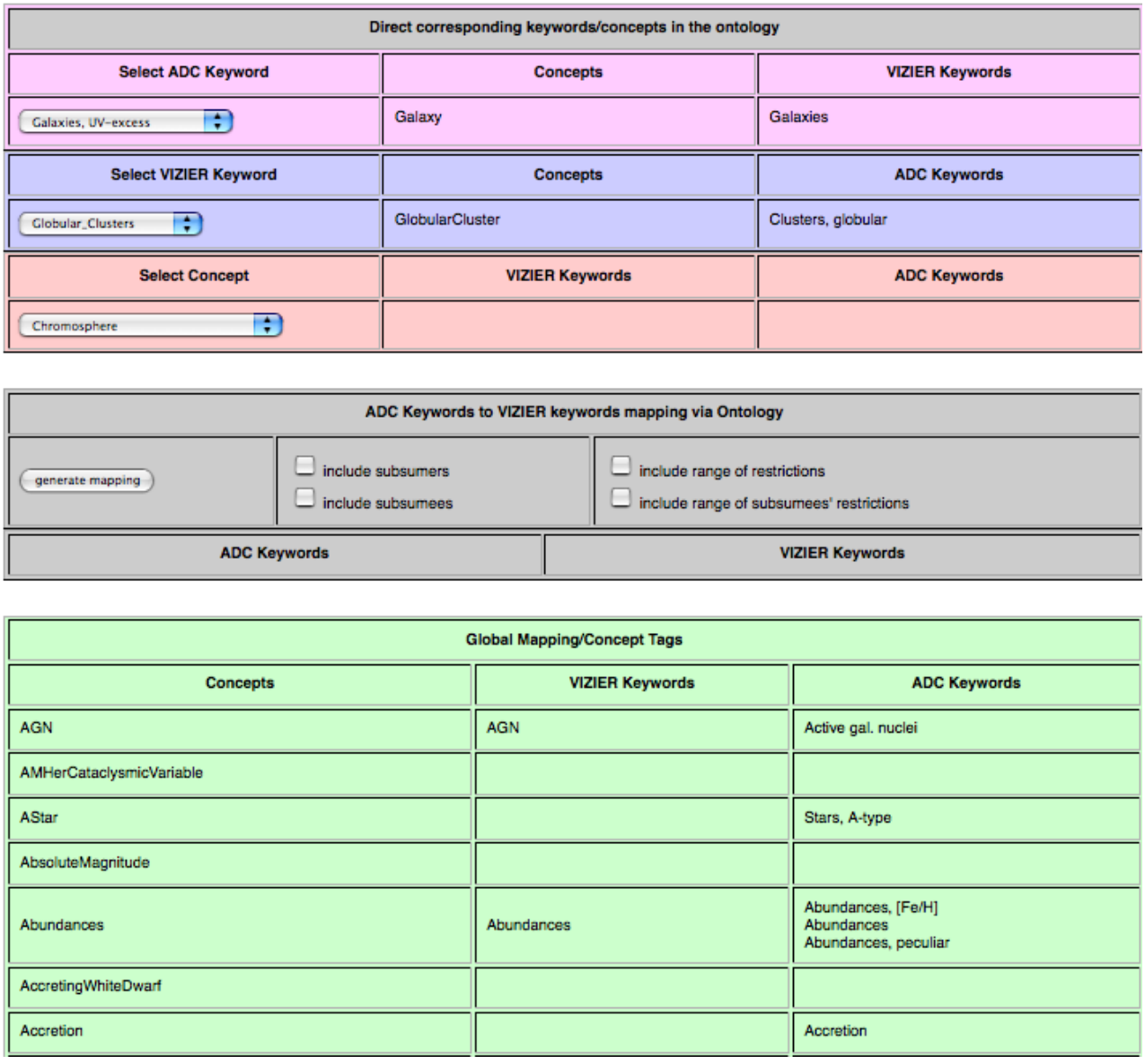
Figure 4: Screenshot of an example of hierarchy for the VizieR keywords, with only the concepts annotated with such keywords being represented on the graph.

Keyword mapper

Following the direction of the annotation viewer, a keyword mapper has been developed. Its goals are first to map keywords with regard to the ontology and second to output these mapping in various formats so that they can be either used or compared.

This last option is especially interesting since some keyword mappings already exist so comparing them with the results obtained with the ontology-based mapping may lead to enhancements in ontology annotations or the mapping actually used by other applications.

The main specifications of this prototype applications are:
- Mappings can be outputted on screen as well as text files.
- It can map any two sets of keywords that exist as annotations within the ontology
- The mapping strategy can be parametrized to exploit any of the ontology elements



Direct corresponding keywords/concepts in the ontology

Select ADC Keyword	Concepts	VIZIER Keywords
Galaxies, UV-excess	Galaxy	Galaxies

Select VIZIER Keyword	Concepts	ADC Keywords
Globular_Clusters	GlobularCluster	Clusters, globular

Select Concept	VIZIER Keywords	ADC Keywords
Chromosphere		

ADC Keywords to VIZIER keywords mapping via Ontology

generate mapping

include subsumers
include subsumees
include range of restrictions
include range of subsumees' restrictions

ADC Keywords	VIZIER Keywords

Global Mapping/Concept Tags

Concepts	VIZIER Keywords	ADC Keywords
AGN	AGN	Active gal. nuclei
AMHerCataclysmicVariable		
AStar		Stars, A-type
AbsoluteMagnitude		
Abundances	Abundances	Abundances, [Fe/H] Abundances Abundances, peculiar
AccretingWhiteDwarf		
Accretion		Accretion

Figure 5: Screenshot of an example of mappings using a version focused on mapping Astronomical Data Center (ADC) and VizieR registry keywords.