International

Virtual

Observatory

Alliance

Ontology of Astronomical Object Types

Version 1.0

IVOA Working Draft 2007 Feb 19

This version:
http://ivoa.net/Documents/WD/Semantics/AstrObjectOntology-20070219.html
Latest version:
http://www.ivoa.net/Documents/latest/AstrObjectOntology.html
Previous version(s):

Editors:
S. Derriere
A. Preite Martinez
A. Richard

Author(s):
L. Cambrésy – cambresy@astro.u-strasbg.fr
S. Derriere – derriere@astro.u-strasbg.fr
P. Padovani – ppadovan@eso.org
A. Preite Martinez – andrea.preitemartinez@iasf-roma.inaf.it
A. Richard – richard@astro.u-strasbg.fr

Abstract

The Semantic Web and ontologies are emerging technologies which enable advanced knowledge management and sharing. Their application to Astronomy can offer new ways of sharing information between astronomers, but also between machines or software components and allow inference engines to perform reasoning on an astronomical knowledge base.

This document presents the current status of an ontology describing knowledge about astronomical object types, originally based on the standardization of objects types used in the SIMBAD database. Specifically, this ontology of defined concepts is designed to enable advanced reasoning on astronomical object types. The possibilities offered by such a system are semi-automatic or fully-automatic applications such as checking the semantic consistency of databases entries, providing new means of building or refining queries and suggesting object types matching a description.

Status of this document

This is an IVOA Working Draft for review by IVOA members and other interested parties. It is a draft document and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use IVOA Working Drafts as reference materials or to cite them as other than "work in progress"..

A list of current IVOA Recommendations and other technical documents can be found at http://www.ivoa.net/Documents/.

Acknowledgments

The Active Galaxy Nuclei section of the ontology is currently being made in collaboration with Paolo Padovani of ESO, the Young Stellar Objects and diffuse matter sections with Laurent Cambrésy of the CDS and the variable star and emission nebulae sections with Andrea Preite Martinez of INAF-Roma.

Contents

1 Introduction

Until now, the experiments on ontologies regarding astronomy have focused on primitive concepts ontologies (i.e. non-defined concepts). With this work, we are exploring the possibilities of defined concepts ontologies in the field of astronomy (cf. section 2 for a presentation of the components of an ontology.)

Ontologies are structures representing and formalizing knowledge. They can be used to guarantee the consistency of knowledge shared between men and machines as well as between machines. Their use ranges from basic classification in the case of primitive concepts ontologies to advanced inference and reasoning in the case of defined concepts ontologies.

This possibility of automated consistency checks and inferences is what interest us most. Indeed a few ontologies have been built to represent part of the astronomical knowledge, but since they lack formal definitions of the concepts, they allow very little reasoning. While this can be sufficient in some cases, it tremendously limits the application of the ontology. Though it is much more difficult, we are willing to build such definitions to set-up a semantic layer allowing to automate operations usually performed by humans since it is the human who has the knowledge to do these operations.

To experiment on these possibilities, we are building an ontology of astronomical object types along with some applications. This ontology is first based on the standardization of object types[1] used in the SIMBAD[2] database. These choices are motivated mainly by the possibilities offered by an astronomical knowledge engine coupled to databases, like consistency checks of the semantics of the database entries or advanced queries.

Last but not least, ontology-based systems are little dependent of the evolution of the ontology. This means that when the astronomical knowledges evolves, one just has to update the ontology accordingly and the systems exploiting it will take the changes into account, unlike dedicated systems for which each change can impact the whole system.

This document covers the following points: the basics of ontologies, the ontology construction process, a global[3] description of the ontology of astronomical object types in its current state, its applications and, to conclude, some perspectives.

1 Objects and object types in SIMBAD refer to a categorization of the nature of astronomical sources, not to objects and types as in object-oriented programming.

2 http://simbad.u-strasbg.fr/

3 A complete description of the ontology is available separately as a Javadoc-like document.

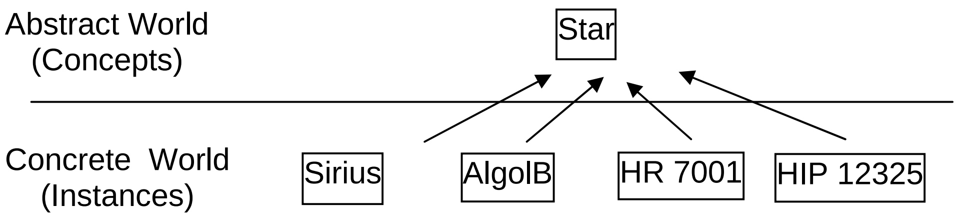
2 Ontology Components

The following sections will explain the basics of ontologies and description logics. For a thorough introduction to Description Logics and their use in ontologies, one can look into [Napoli, 2004], the first chapter of [Staab and Studer, 2004] and [Napoli, 1997] .

2.1 Concepts and Instances

Ontologies are often defined as a representation of a conceptualization. Thus, their most fundamental components are *concepts* (also called *classes*). A Concept is an abstract object which defines the common features of a group of concrete objects. The concrete objects are called *instances* or *individuals*.

e.g. All the stars are instances of the same concept Star.

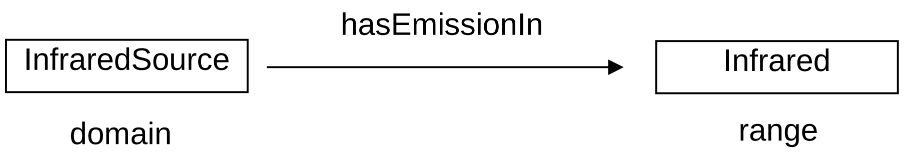


A concept can be defined as the union of other concepts

2.2 Properties

A *Property* (also called *role*) represents a binary relationship between two concepts or unions of concepts. The *domain* of a property is the concept to which the property can be applied and the *range* of a property is the concept where the property takes it value.

e.g. : To represent that infrared sources (concept *InfraredSource*) have an emission in the infrared part of the electromagnetic spectrum (concept *Infrared*), one can introduce the property *hasEmissionIn*, defined as follows:



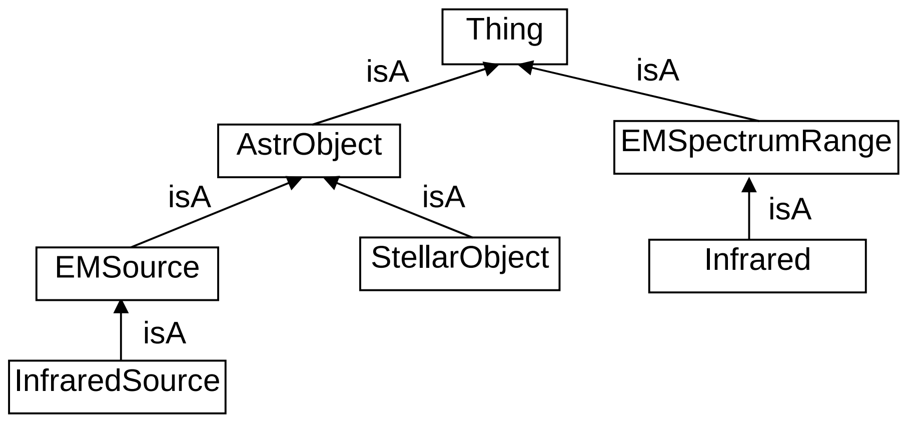
2.3 Subsumption relationship

Both concepts and properties are organized into a hierarchy by the subsumption relationship. It can be roughly summarized as a kind of a “is a” relationship, meaning that children are more specific than their parents.

- Concept subsumption
  If A and B are two concepts, A is subsumed by B (B subsumes A)

if and only if all the instances of A are instances of B
e.g. the concept *GiantStar* is subsumed by the concept *StellarObject*

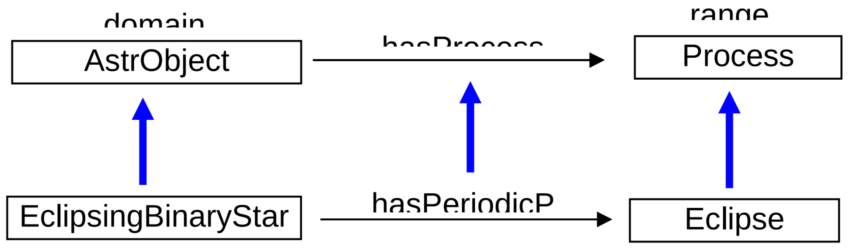
The universal subsumer is called *Thing* or *TOP* and is always found at the top of a subsumption hierarchy



N.B. A common mistake is to mistake the subsumption relationship for a "part of" relationship and build a hierarchy that is really a hierarchy of components (i.e. the concept *Vehicle* subsumes the concept *Car* but does not subsume the concept *Wheel* because "a car is a vehicle" but "a wheel is a part of a vehicle, not a kind of vehicle")

- <u>Property subsumption</u>
  If A and B are two properties, A is subsumed by B (B subsumes A)
  If and only if domain(A) is subsumed by domain(B)
  AND range(A) is subsumed by range (B)

  e.g.



(A → B means "A is subsumed by B")

2.4 Concepts definitions

In a formal ontology, concepts can be either *primitive* (i.e. non-defined) or *defined* by necessary and sufficient conditions and/or constrained by necessary conditions. These conditions are expressed as restrictions on properties.

e.g. “An electromagnetic source is an astronomical object which has an emission in some part of the electromagnetic spectrum” can be translated as :

EMSource ≡ AstrObject **and** hasEmissionIn **some** EMSpectrumRange[4]

This means that any instance which verifies the conditions “AstrObject **and** hasEmissionIn **some** EMSpectrumRange” is an instance of EMSource and that this condition is true for every instance of EMSource.

One of the consequences of this is that subsumees inherit their subsumers' necessary conditions (which is consistent with the “more specific kind of” meaning of the subsumption relationship.)

3 Ontology construction

3.1 Implementation choices

The implementation of an ontology is a decisive matter since the different implementations offer different capabilities and limitations. A detailed explanation of the following implementation choices is available in Annex A

- *The language of representation*

Since we wanted to build an ontology of defined concepts, we needed a formalism that would allow this. Description Logics[5] is an adequate and mature means of representing ontologies. Furthermore, the Web Ontology Language[6] (OWL) is based on description logics and is probably the most widespread language for describing ontologies. So we decided to describe our ontology using Description Logics and to implement it in OWL-DL at best or in its recent evolution OWL1.1[7] if expressiveness beyond OWL-DL was needed. Both of these flavors are well-supported by existing reasoners and are the best compromise between complexity and expressiveness.

- *The reasoner*

After testing the possible reasoners, we chose to use RACER 1.7.23 as our reasoner since it is by far the best compromise. Though discontinued now, the years of research and development on it make it at least as valuable and reliable as its commercial counterpart RacerPro as well as any other inference engine.

[4] For legibility purposes, the description logic syntax used in this document is the Manchester-OWL syntax (cf. http://www.co-ode.org/resources/reference/manchester_syntax/)
[5] http://wiki.eurovotech.org/twiki/bin/view/VOTech/DescriptionLogics
[6] http://www.w3.org/TR/owl-guide/
[7] http://owl1_1.cs.manchester.ac.uk/

- *The ontology editor*

The last choice to make for the implementation is to select a graphic editor to build and edit the ontology. We settled for Protégé-OWL [Horridge et al., 2004], developed by the University of Stanford, which is currently both the most complete and most intuitive graphic editor for ontologies.
Though the editor is well documented, we set up a page of advice[8] to ensure people willing to use Protégé would not be bothered by some minor problems we were ourselves confronted with.

- *Naming conventions*

To be sure we had a unified syntax for the names in the ontology, we made the following choices :

- The characters allowed are uppercase and lowercase letters only.
- Java-like naming: use uppercase letters and no spaces.
  (e.g. PlanetaryNebulaShell)
- Concept names begin with an uppercase letter, property names begin with a lowercase letter.
  (e.g. PlanetaryNebulaShell / hasEmissionIn)
- At least during the construction phase, acronyms and shortened names are strongly discouraged to avoid risks of mistakes or ambiguity.

3.2 Limitations and issues

The sheer nature of an ontology and the implementation choices imply some limitations one has to be aware of when constructing the ontology.

- *Conditions on concepts must be always true:*
  This is one of the greatest problems: since concepts describe what all of their instances have in common, the conditions constraining or defining them must be always true. Specifically, conditions that are “usually true” or “true in most cases” or “true 95% of the time” are not allowed. However it is important to notice that a statement is considered “true” if the considered knowledge says so: if the knowledge evolves, so will the ontology.

- *Cardinality is allowed, qualified cardinality is allowed but discouraged:*
  Cardinality describes a restriction on the number of times a property has the concept as its domain. Qualified cardinality also precises the range of the property.
  e.g. hasComponent **maximum** 2 (cardinality)
  hasComponent **maximum** 2 StellarObject (qualified cardinality))

  Qualified cardinality is rather CPU-heavy, therefore it is strongly advised to replace it by existential restrictions every time it is possible.

- *Intervals and enumerations are acceptable:*

[8] http://wiki.eurovotech.org/twiki/bin/view/VOTech/ProtegeAdvice

Still, both tend to degrade the performances and are therefore to be used wisely.

  e.g hasMeasurement **some** {SpectralTypeO,SpectralTypeB,SpectralTypeA}

- *Restrictions on values are impossible:*
  You can describe a concept C has being the domain of a property *but* you cannot describe C as having a given value for a property.

  e.g. You can describe a concept Star as having a temperature, but you cannot describe this concept as having a temperature of n Kelvin.

- *Restrictions with variables are impossible:*
  There are no variables in description logics. Therefore some relationships cannot be expressed, like for instance relationships between components of a given compound object

  e.g. you can express that each of both components of a double star has a gravitational link with an instance of the same concept as the other but you cannot express that they are linked one with the other.

- *Complexity must not be too high:*
  If the structure is difficult to manipulate for the reasoner, like if there are too many restrictions that are CPU-heavy (qualified cardinality, enumerations...), even if the ontology is well-made, its exploitation in applications will be jeopardized since the reasoning time will be too long (cf. 3.4.4 Overall complexity test)

- *Definitions must be adequate:*
  Definitions and restrictions in general must fit the use of the ontology. For instance, if an application never manipulates data on the components of a galaxy, defining galaxies via their components will be useless at best and will degrade the overall performance of the application at worst. (cf. Note in section 3.4.2)
  It is important to remember that a usable ontology is not a universal description. Indeed, it is impossible to have a perfect representation and even if it were possible, the complexity would be so high that the structure would be impossible to use and maintain.

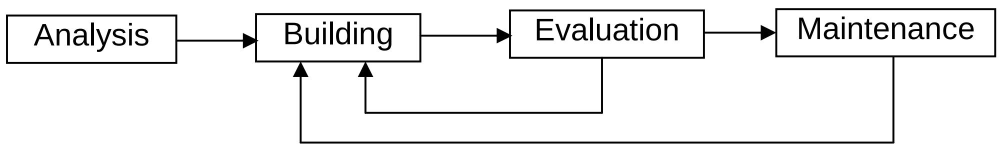
- *Size must be manageable*
  An overly detailed ontology, or covering too wide a field, is likely to become illegible, hard to manage and would yield unrealistic reasoning times.

- *Naming issues*
  This is a minor problem since it has no impact on the correctness or the use of the ontology. Still, it is better to have names describing as clearly as possible concepts and properties. Furthermore, even if the end-user will never see the ontology, it will be much easier to maintain if it is easy to read. The only problem with naming is that most of the time names are

ambiguous or misleading and finding a name which naturally evokes a given concept or property is a very difficult task.

3.3 Construction cycle

There is no unified procedure for building ontologies. Still, it always comes down to an iterative process like the following one. [Staab and Studer, 2004, [Uschold and King, 1995]



- Analysis :
  - What does the ontology conceptualize?
  - What will it be used to do?
  - Identifying the concepts.
- Building the ontology
  - Defining the concepts.
  - Building the subsumption hierarchies.
  - Adding annotations.
- Evaluation
  - Consistency checks.
  - Efficiency tests
  - Going back to building step for adjustments if needed
- Maintenance
  - Tests in real use
  - Update/evolution as needed (going back to the building step)

3.4 The building process

3.4.1 Analysis

We aim to build an ontology to be used as a knowledge layer over existing tools such as the SIMBAD[9] database of astronomical objects. More precisely, we want to have a semantic tool which would be able to perform automatically operations such as :

- Building advanced queries on astronomical databases or registries.
- Checking and validating the objects' classification in the SIMBAD database.
- Making proposals to enhance the classification on SIMBAD objects when new identifiers or measurements are added.

The idea to rely on an ontology comes from the possibilities of automatic

9 http://simbad.u-strasbg.fr/

reasoning allowed by the existing reasoners and APIs. The shortcoming is that to be able to exploit these tools we have to build an ontology of *defined* concepts (i.e. have as many concepts' definitions as possible.)

As for what the concepts of the ontology will be, since we planned to use the ontology first with the SIMBAD object types[10], we decided to first try and represent these objects as concepts and then see if some concepts were lacking or inadequate and eventually adjust the structure. This choice of representation is adequate for the following reasons:

- Since we want to perform operations on astronomical objects and their types, it is best to have a representation (including the definitions of the concepts) that is as close as possible to that use.
- There are around 150 object types in SIMBAD, which makes an amount of defined concepts low enough to keep the ontology core manageable.

3.4.2 Building

As exposed previously, the building process is iterative. Basically it can be broken down to this :

- Finding conditions to constrain the concepts, fully defining them if possible.
- Introducing the properties and/or concepts needed to build the conditions.
- Building the subsumption hierarchies of concepts and properties, taking into account both the conditions expressed on the concepts and the unexpressed knowledge we may have of these concepts.
- Adding the annotation properties we need for the applications.

e.g. To describe the concept DoubleStar, one can try to describe its components :

- a double star is an astronomical object
- a double star is a system of objects
- a double star is composed of exactly 2 objects
- both of the components are stellar objects

Fortunately, these conditions are not only necessary but also sufficient. Therefore, a possible definition of DoubleStar is:

DoubleStar ≡ AstrObject **and** hasComponent **exactly** 2
and hasComponent **only** StellarObject

This is not the only definition of a double star and one must keep in mind that depending on the uses of the ontology, other definitions could give better results and that having multiple definitions can also be either a good or a bad thing. (e.g. our definition of *DoubleStar* is worthless if we never manipulate the components of systems)

[10] Objects and object types in SIMBAD refer to a categorization of the nature of astronomical sources, not to objects and types as in object-oriented programming.

Having the previous definition, we need to make sure we have already declared the property *hasComponent* and the concepts *AstrObject* and *StellarObject*. If we have not, we must declare them before inputting the definition of *DoubleStar*.

The subsumption hierarchies can be either constructed by describing which concept/property subsumes which, or they can be inferred by a reasoner. Our choice was to build them ourselves and then run the reasoner to check if there was no inconsistency or lack in our structure.

Last, we add annotation properties to our concepts. These annotations have no impact on the reasoning but can be used to put labels on the different objects. These labels can be either human-readable text (e.g. names, descriptions) or information we want to link directly to the object, for example to use them when accessing the ontology via an API (e.g. SIMBAD database codes).

3.4.3 Consistency Check

An important point is to be sure of the consistency of the ontology since an inconsistent ontology would yield questionable results. Fortunately, this very tedious task is well performed by some reasoners, thus we only have to launch an automated procedure and wait a few seconds for the results. Obviously, given the importance of the consistency and the convenience of automated tools, we test the consistency after each set of changes we make, even if the changes are supposed to be purely cosmetic.

3.4.4 Overall complexity test

Testing the ontology is done in two steps. First, we make sure that the complexity of the structure is not going to be problematic. One way to evaluate this is to ask the reasoner to classify the ontology. Indeed, classifying the ontology is the first thing the inference engine will do before executing any request.
The time taken for this operation depends on three factors:

- the complexity of the logic used
- the size of the ontology
- the completeness of the description of the subsumption links

If this test takes too much time, it is likely that the ontology will not be usable in real conditions. If such is the case, corrections are to be made. Since usually the ontology size cannot be reduced, the general idea is to write simpler restrictions on properties. This means using a less complicated logic if possible. For instance, using existential restrictions instead of qualified cardinality restrictions helps keeping the complexity lower for the reasoner. Therefore, such (re-)writing is strongly advised when possible.
e.g.

With qualified cardinality:

PlanetaryNebula
≡ CompoundObject
and hasComponent **exactly** 1 PlanetaryNebulaCentralStar
and hasComponent **exactly** 1 PlanetaryNebulaShell

Without qualified cardinality:

PlanetaryNebula
≡ CompoundObject
and hasComponent **some** PlanetaryNebulaCentralStar
and hasComponent **some** PlanetaryNebulaShell
and hasComponent **exactly** 2

3.4.5 Real-use test

Once this overall complexity test is performed with adequate performance, we check the ontology's performance in real use. This is done by testing the applications exploiting the ontology and evaluate the performance, both in terms of execution speed and results quality. The analysis of the results help us fine tune the ontology to our exact needs.

4 Ontology structure

4.1 Concepts

4.1.1 Two different kinds of concepts

As exposed previously, our goal being to build an ontology of astronomical object types, we need to create a concept for each of them. But we also wish these concepts to be defined so we can use a reasoner on them.

Therefore, we need to create all the concepts needed to write definitions for these concepts. To be exact, we need ranges for the properties we use in our definitions and these additional concepts are the ranges of the properties. But then, since they are only ranges, we do not need to define them.

So in conclusion, our concept hierarchy is made of two kind of concepts :

- Concepts representing astronomical object types, which we want defined.
- Concepts that are only ranges of properties, which we will keep primitive11.

4.1.2 The problem of compound objects

Though we are limited by the lack of variables in description logics (cf. section 3.2), we can describe most of the relationships between compound objects and their components. This is interesting because these relationships can take part into a definition.

Still, one problem is that, when we refer to the SIMBAD list of object types, we find that some compounds are not astronomical objects

e.g. PartOfCloud, Region, Void.

Furthermore, when we describe the components of a given astronomical object, we may want to introduce components which are not astronomical objects

[11] These concepts could be mapped to another ontology where they would be defined.

themselves.

e.g. When describing galaxy components, we may want to introduce the concepts of Halo, Disk or Bulge.

And these non-object components may themselves have some components.

e.g. The Halo of a Galaxy has Star and GlobularCluster among its possible components.

To represent correctly these relationships, we have introduced the following concepts and properties :

- *AstrObject:*
  subsumes all the concepts representing astronomical objects[12].

- *CompoundObject:*
  subconcept of AstrObject which subsumes all the concepts representing astronomical objects which are composed of at least two distinct astronomical objects

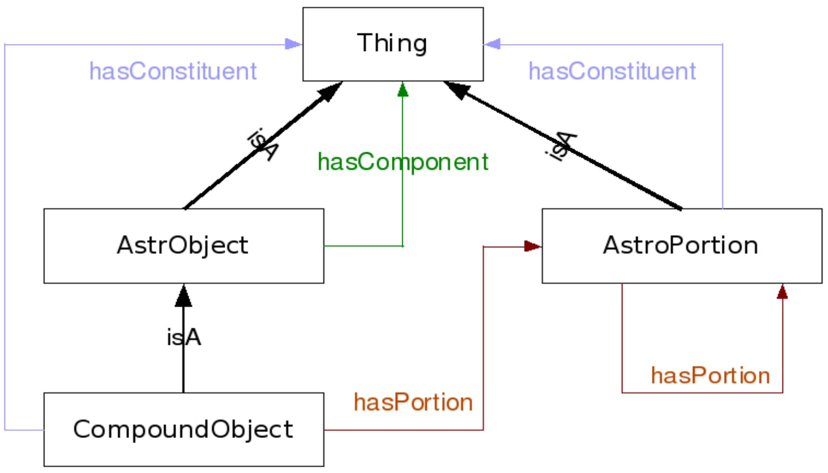
- *AstroPortion:*
  subsumes all the concepts representing portions of astronomical objects which are not astronomical objects themselves[13].

- The following properties:

property name	*domain*	*range*
hasConstituent	CompoundObject OR AstroPortion	AstrObject
hasComponent	CompoundObject	AstrObject
hasPortion	CompoundObject OR AstroPortion	AstroPortion

hasConstituent is used to link a CompoundObject or AstroPortion to any of its constituents (which are necessarily astronomical objects).
hasComponent is used to link a CompoundObject to any of its direct components (which are necessarily astronomical objects). An important corollary is that the sum of all the components is a definition of a given CompoundObject.
hasPortion is used to link a CompoundObject or an AstroPortion to any of its Astro Porti on.

With this system, we should



12 Wl types like *PlanetaryN*

13 Inc *rtOfCloud*.

be able to describe all relationships between objects, portions of them and their components.

e.g. We can describe that a galaxy has a halo which has a globular cluster among its components, which itself includes a double star which is composed of a giant and a white dwarf[14]:

Galaxy (CompoundObject) hasPortion Halo (AstroPortion)
Halo hasConstituent GlobularCluster (CompoundObject)
GlobularCluster hasConstituent DoubleStar (CompoundObject)
DoubleStar hasComponent Giant (AstrObject)
DoubleStar hasComponent WhiteDwarf (AstrObject)

4.1.3 The description of astronomical objects

As evoked in section 4.1.1 we are to write definitions, or at least necessary conditions, of our concepts representing astronomical object types. More precisely, we aim at defining the concepts in the AstrObject and AstroPortion branches. Indeed, as seen in section 4.1.2 the AstroPortion section of the ontology takes part in composition relationships of astronomical objects. For this reason we are likely to need definitions on them to get better inferences on astronomical objects -not to mention that some AstroPortion are actually referred to as object types in the SIMBAD list.

Within these definitions we use primitive concepts as range for the properties. These concepts are introduced when we need them. They are organized in several branches of the concept hierarchy, each branch corresponding to a point of view used to describe astronomical objects.

- *EMSpectrumRange*
  Set of ranges in the electromagnetic spectrum
- *Measurement*
  Measured observational parameters/properties
- *Morphology*
  Geometry or morphology of astronomical objects
- *Process*
  Phenomenon or associated process
- *AtomicElement*
  Atomic elements

Of course these sections and their content will evolve with our needs. Namely, if we need new concepts or even a new top-level concept corresponding to a new descriptive point of view, we will add then (of course the consistency of the ontology must be preserved when such changes happen).

4.1.4 Global schema

To summarize what has been developed in the previous sections, currently the concept hierarchy is organized around the following top-level concepts which are:

- AstrObject

[14] Between parenthesis is the most specific subsumer of the concept between AstrObject, CompoundObject, AstroPortion.

- AstroPortion
- AtomicElements
- EMSpectrumRange
- Measurement
- Morphology
- Process

These sections can be split in two categories: AstrObject and AstroPortion subsume the astronomical object types and their constituents while the other sections are ranges of properties used to define the concepts of the AstrObject and AstroPortion sections.

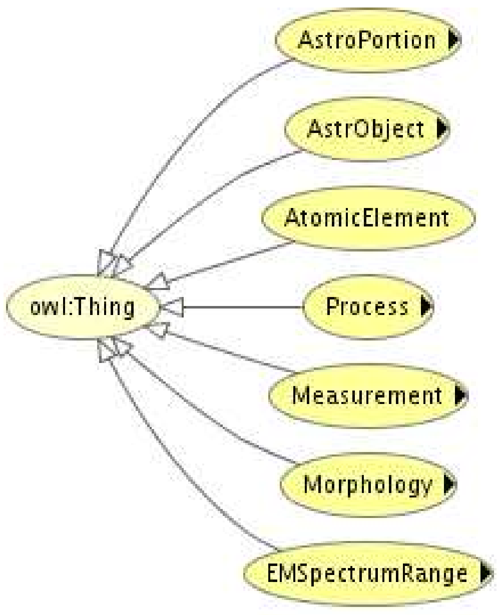
4.2 *Overview of the concept hierarchy*

We now present a graphic overview of the concept subsumption hierarchy. For legibility reasons the different subsections of the hierarchy are shown separately.

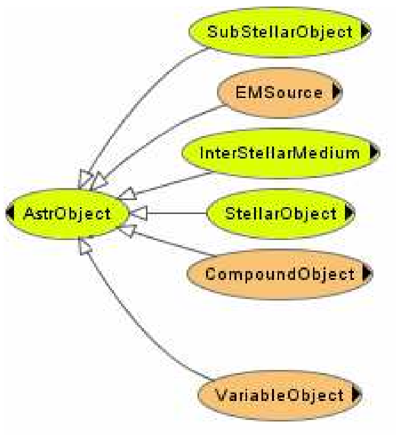
Color used:
– Yellow: concepts for which we have necessary conditions but no definition
– Orange: concepts for which we have at least one definition
– White: concepts from other branches than the one considered which can be either defined or not but have been colored white to enhance legibility.

A complete documentation of the ontology is available separately in a Javadoc format.

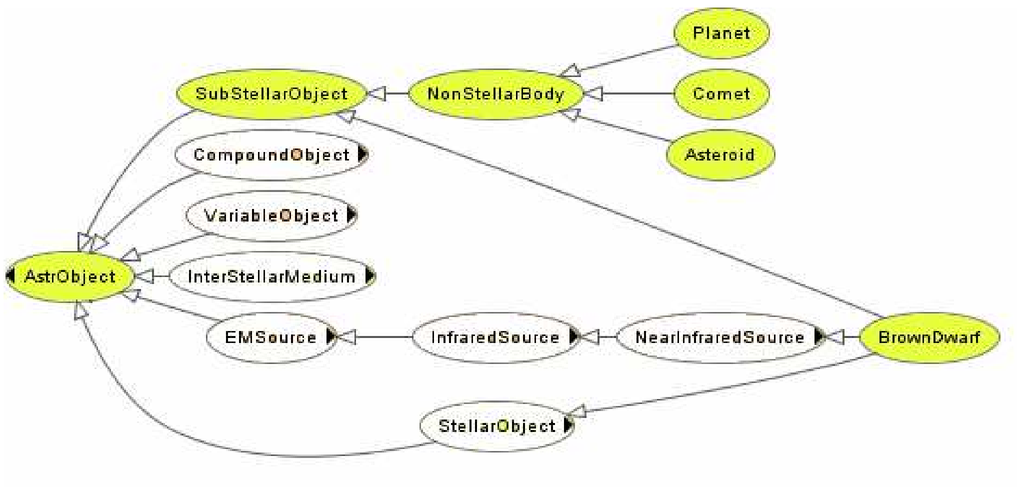
4.2.1 Top-level concepts



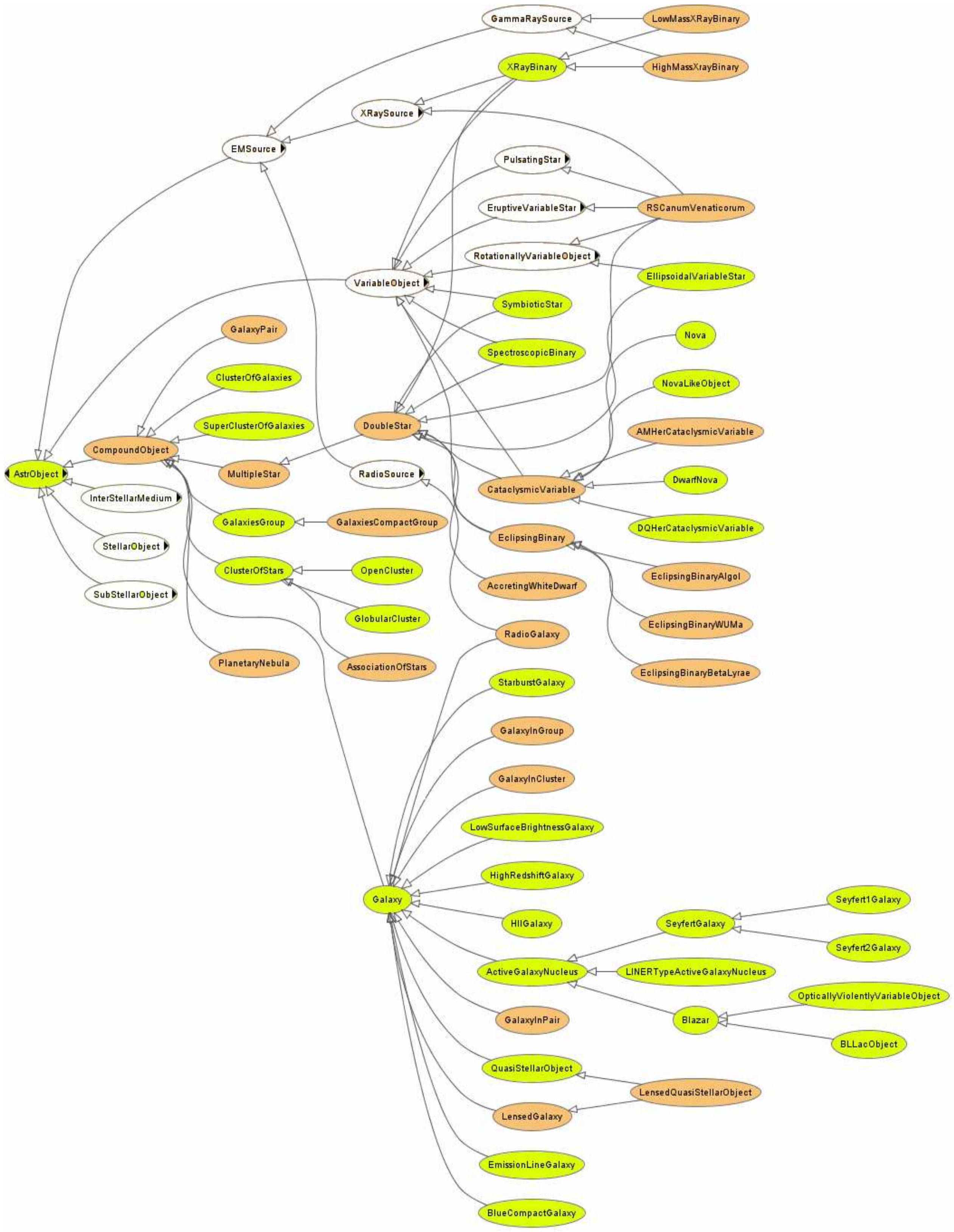
4.2.2 The AstrObject section



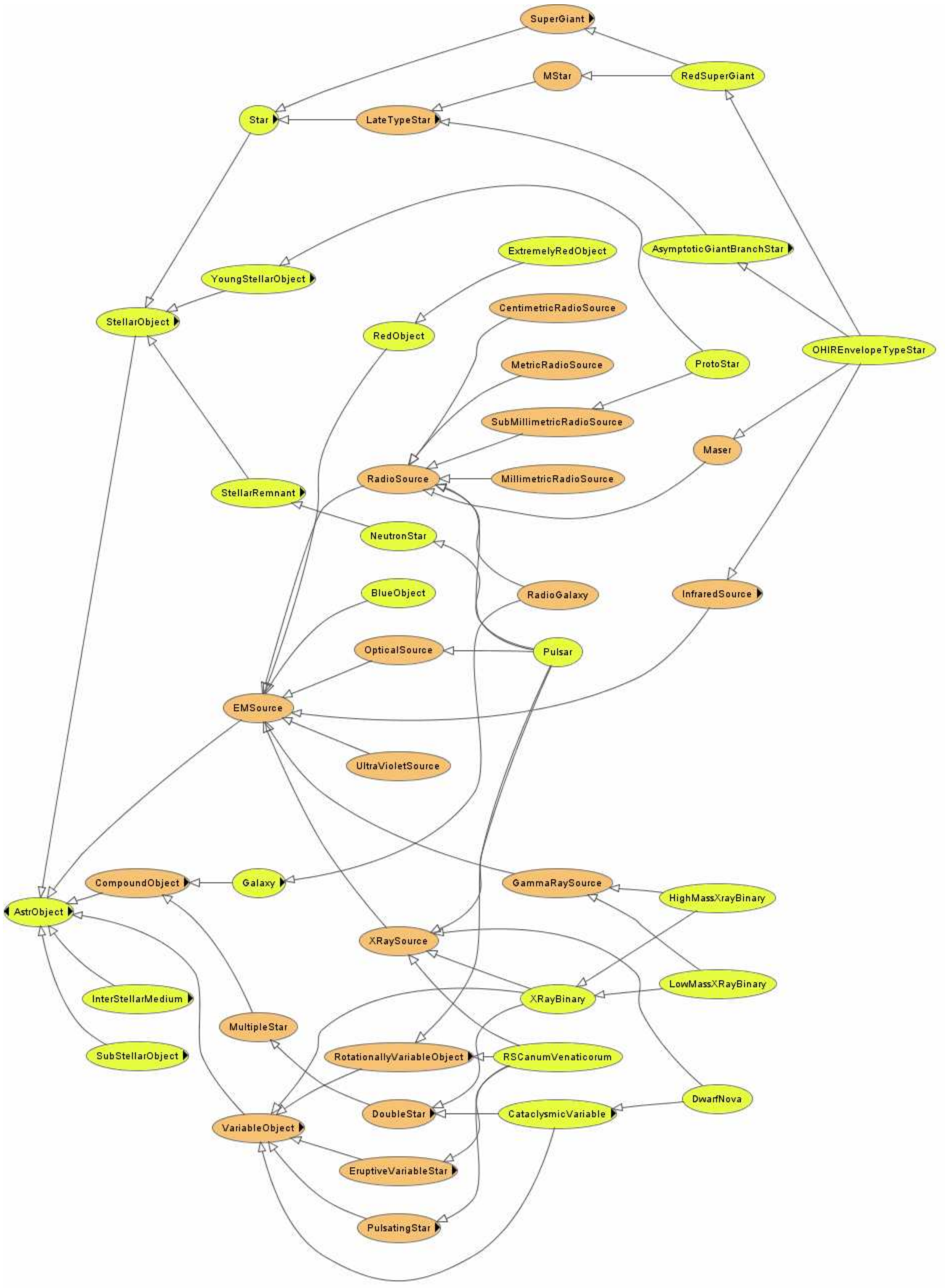
- **The Substellar subsection**



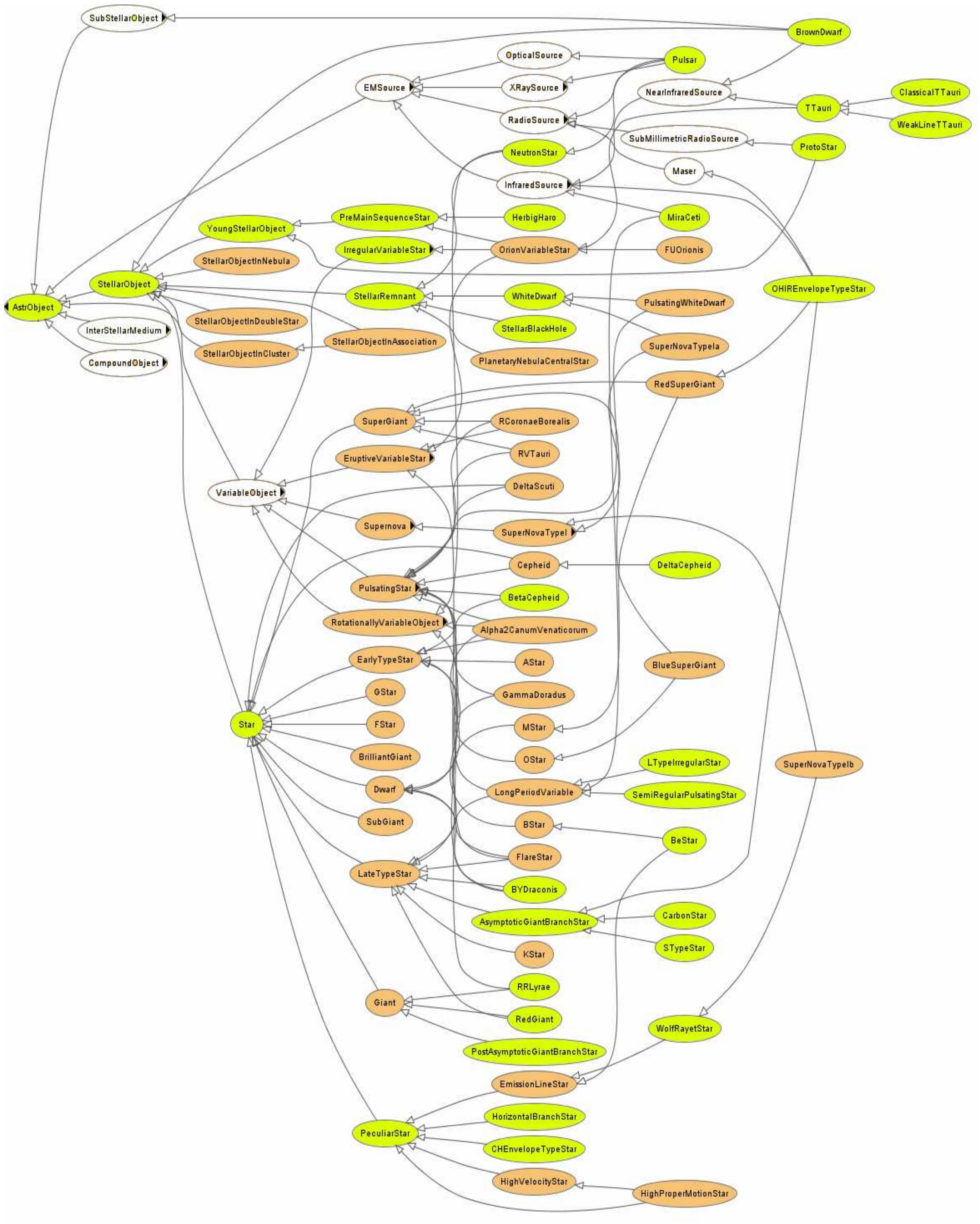
- **The CompoundObject subsection**



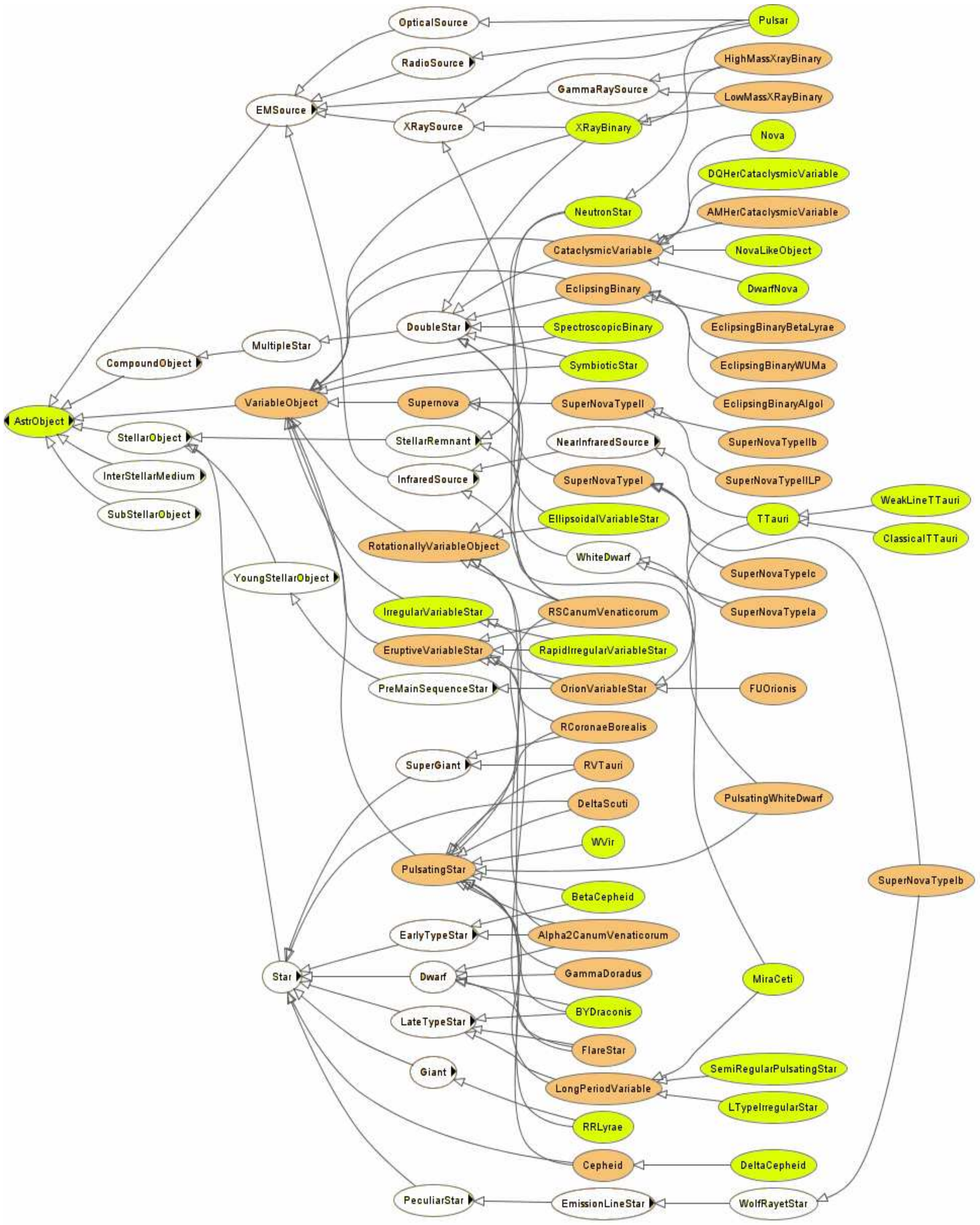
- **The EMsource subsection**



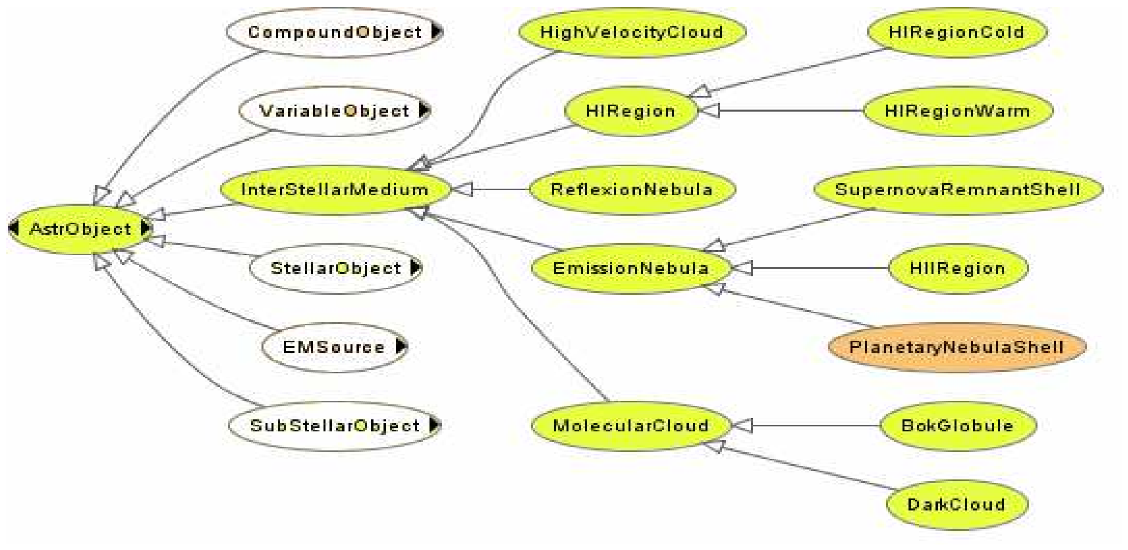
- **The StellarObject subsection**



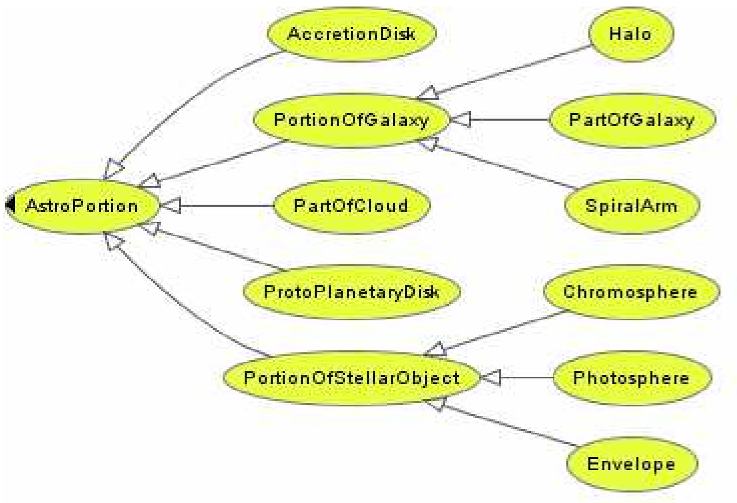
- **The VariableObject subsection**



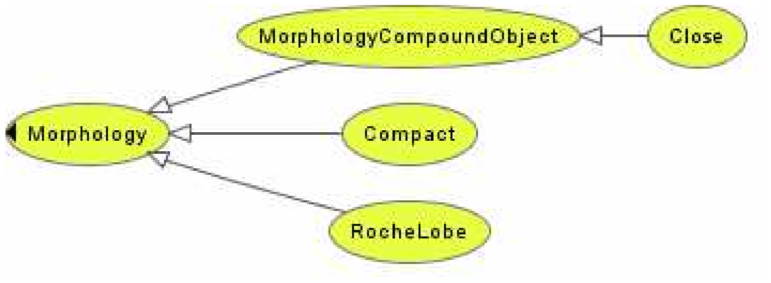
- **The InterStellarMedium subsection**



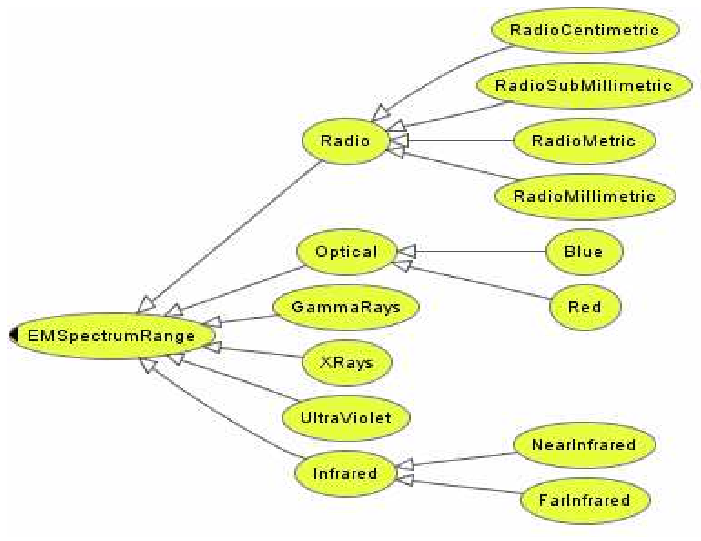
4.2.3 The AstroPortion section



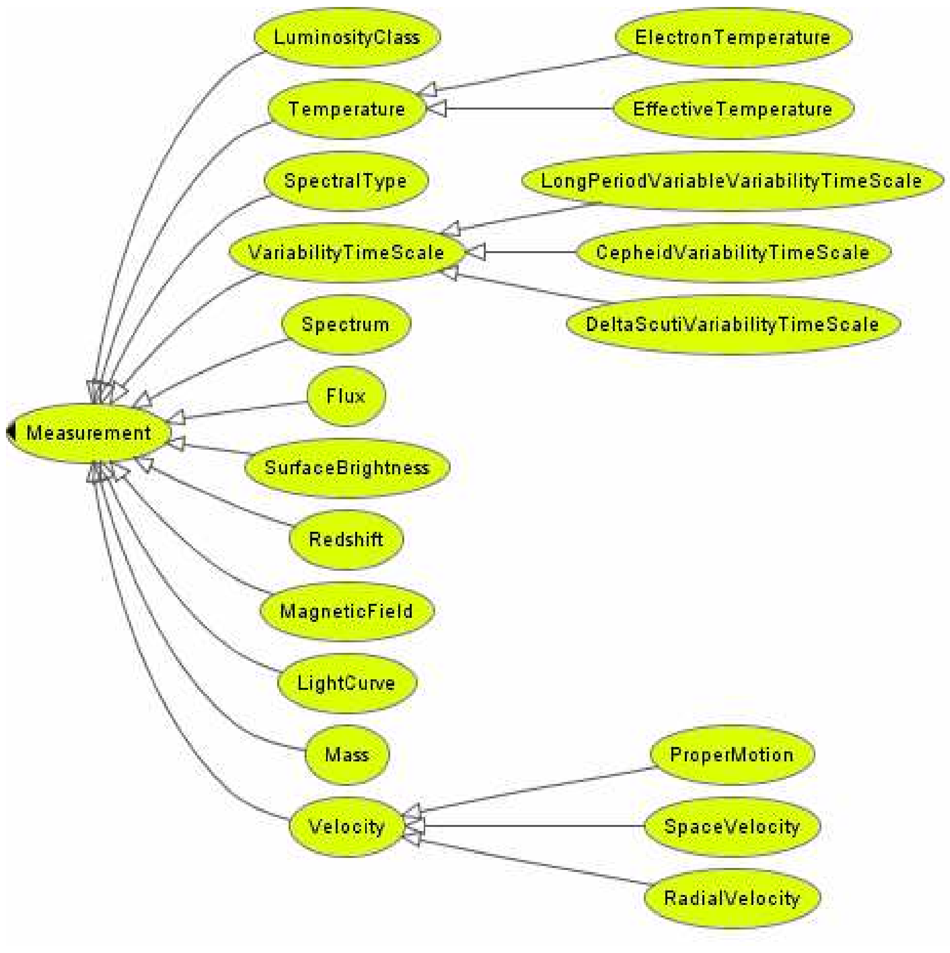
4.2.4 The Morphology section



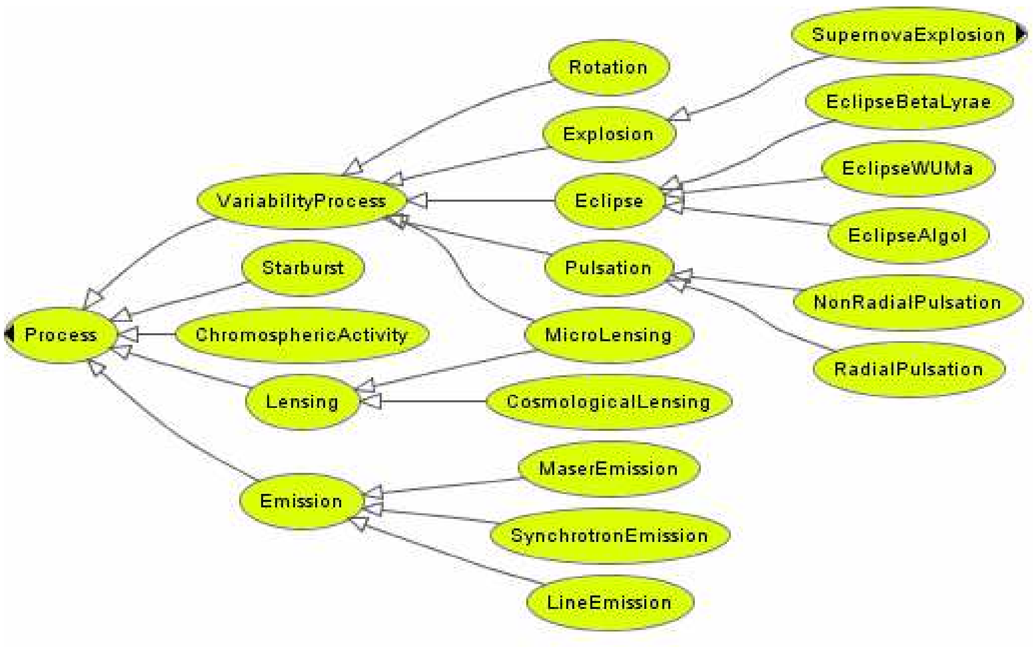
4.2.5 The EMSpectrumRange section



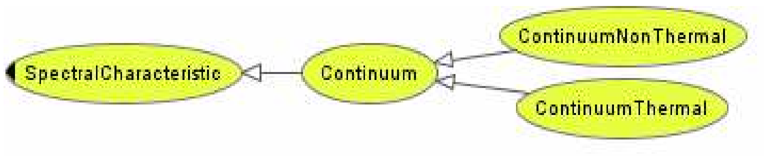
4.2.6 The Measurement section



4.2.7 The Process section



4.2.8 The SpectralCharacteristic section



4.3 The properties

4.3.1 Description of properties

We already introduced the hasConstituent, hasComponent and hasPortion properties in section 4.1.2. Other properties were introduced to describe astronomical objects via not only their constituents but also their emission, the processes they are subject to, the measurements made, their morphological features or their spectral characteristics.

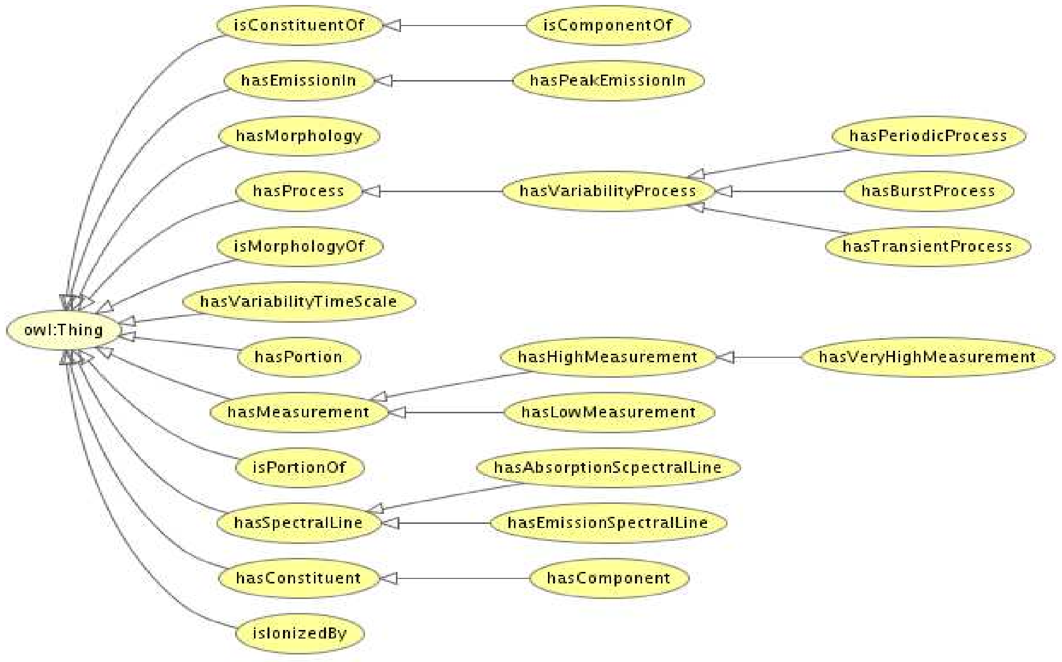
The following list describes the current properties, it is likely to evolve to fit our need as we write new definitions.

name	***domain***	***range***	***inverse***	***transitive***
hasEmissionIn	AstrObject	EMSpectrumRange	none	no
hasPeakEmissionIn	AstrObject	EMSpectrumRange	none	no
hasMeasurement	AstrObject	Measurement	isMeasuredFor	no
hasHighMeasurement	AstrObject	Measurement	none	no
hasLowMeasurement	AstrObject	Measurement	none	no
isMeasuredFor	Measurement	AstrObject	hasMeasuremen t	no
hasProcess	AstrObject	Process	none	no
hasVariabilityProcess	VariableObject	ProcessVariability	none	no
hasBurstProcess	VariableObject	Explosion	none	no
hasPeriodicProcess	VariableObject	Eclipse OR Rotation OR Pulsation	none	no
hasTransientProcess	VariableObject	ProcessVariability	none	no
hasPortion	AstrObject OR AstroPortion	AstroPortion	isPortionOf	no
isMorphologyOf	Morphology	AstrObject	hasMorphology	no
hasMorphology	AstrObject	Morphology	isMorphologyOf	no
hasConstituent	CompoundObject OR AstroPortion	AstrObject	isConstituentOf	yes
hasComponent	CompoundObject	AstrObject	isComponent	no
isConstituentOf	AstrObject	CompoundObject OR AstroPortion	hasConstituent	yes
isComponent	AstrObject	CompoundObject	hasComponent	no
hasSpectralLine	AstrObject	AtomicElement	none	no
hasEmissionSpectralLine	AstrObject	AtomicElement	none	no
hasAbsorptionSpectralLine	AstrObject	AtomicElement	none	no
hasVariabilityTimeScale	VariableObject	VariabilityTimeScale	none	no
isPortionOf	AstroPortion	AstrObject OR AstroPortion	hasPortion	no

Note that properties hasConstituent and isConstituentOf are transitive to allow descriptions closer to reality since when considering astronomical objects the

following rule is always true:
if A is a constituent of B and B a constituent of C then A is a constituent of C.

4.3.2 An overview of the property subsumption hierarchy



4.3.3 Annotations

The annotations do not have any impact on the ontology but are useful for:

– improving the legibility
– adding extra information (eventually usable via an automated process.)

The most common annotations in OWL are RDFS comments and labels. But one can define annotation properties with specific names and namespaces to fit his needs. Currently we use the following annotation properties:

- *rdfs:comment*
  General comment about the attached OWL item (concept, property...)
- *misc:description*
  Text definition of a concept, as complete as possible
- *simbad:name*
  Standard name in SIMBAD's object classification
- *simbad:shortCode*
  Short code in SIMBAD's object classification
- *vizier:kwd*
  VizieR registry keyword (used by the registry request builder application, cf. section 5.2)

Currently, annotation properties to which are attached multiple values are written as a multiple identical annotation properties, each one having one single value attached.

5 Applications

5.1 OWL API

5.1.1 Jena framework

To build applications exploiting the ontology, we need an API allowing us to access and manipulate directly an ontology written in OWL. Only a few exist and nearly all of them are based on the Jena framework. Jena is a Java framework for building semantic web applications. It is open source and provides -among various programming toolboxes- an OWL API.

Since it is reliable, mature and offers a good compatibility with most of the other RDFS/OWL APIs, Jena was our first choice of API to build our applications. We since switched to the Protégé-OWL API.

5.1.2 Protégé-OWL API

On the one hand, a shortcoming of the Jena Framework for OWL exploitation is that it is a general RDF/RDFS framework. Thus Jena lacks specific primitives for OWL-based applications. On the other hand, the Protégé-OWL API provides nearly every function needed to exploit an OWL Ontology which results in a faster and simpler programming. Moreover, since this API is powering the Protégé ontology editor, it benefits from the same development support as the editor and is not likely to be forsaken any time soon. So after considering the pros and cons of the different APIs, the Protégé-OWL API is our final choice for our programming needs.

It is worth noting that these APIs being Java-based, this implies at least the core of the applications to be coded in Java.

5.2 Registry request builder

Our first application exploiting the ontology of astronomical object types is a request builder for querying astronomical registries. This was presented at IAU XXVIth GA, Prague 08/2006 during Special Session 3

5.2.1 Why an ontology-based request builder?

The idea is to be able to have a tool able to build advanced queries in the VO registry. Since the ontology is one of astronomical object types, the queries will be performed on the <subject> element of the Registry scheme, which may contain a description of astronomical object types.

The idea of such a tool comes from the limitations of existing registry querying methods. Obviously, when querying on the <subject> field of registry entries, one must use existing keywords in the query in order to have some results. 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.

For example, if we consider the registry entires coming from VizieR[15]:

- Double stars do not have an associated VizieR keyword, so one cannot query directly on them.
- But specific double stars like eclipsing binaries or cataclysmic variable do have associated VizieR keywords.
- Moreover, if one queries for the keyword associated with multiple stars, the query result will only be the entries featuring this keyword, which means that multiple stars with specific keywords like cataclysmic variables will be ignored.
- There is no tool for automatically retrieving more specific or more general keywords.

5.2.2 Implementation method

Since the subsumption relationships in the ontology corresponds to the knowledge needed to retrieve more specific or more general keywords, our request builder searches through the ontology to automatically propose adequate keywords for the user to use.

To make that possible, we have used annotations in the ontology. Concepts corresponding to existing <subject> keywords were annotated with those keywords. Currently the VizieR registry keywords have been implemented (cf. vizier:kwd annotations in the ontology)

The general idea is that the request builder will be fed a query subject and will refine or broaden the query by adding keywords found during a search within the subsumed or subsuming concepts ontology, the original query subject being the starting point of the search.

The registry request builder is being developed in Java/JSP using the Jena OWL API, an Apache Tomcat server. It also uses the free RACER 1.7 reasoner server to set-up the ontology during the startup.

5.2.3 The search for keywords

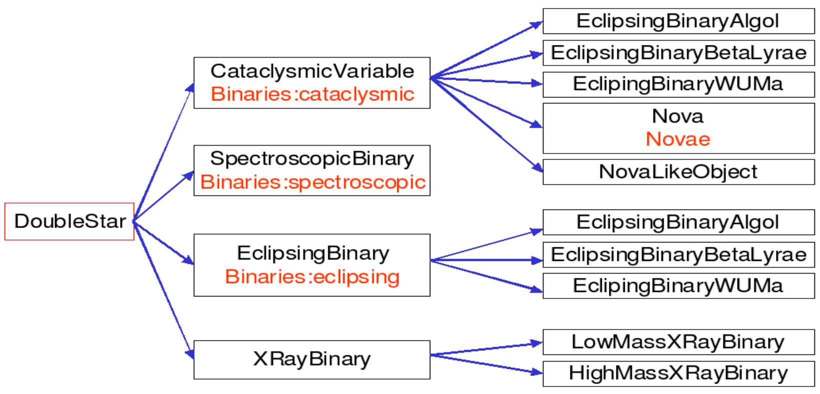
The search for keywords is done in two times: first search through the subsumed concepts to look for more specific keywords. After this step, if no keyword has been found for the query another search is performed, this time to get the most specific subsumer.

For the following examples, we consider the VizieR keywords which will be written in red on the graphs.

- First search : more specific keywords. The user wants to make a query

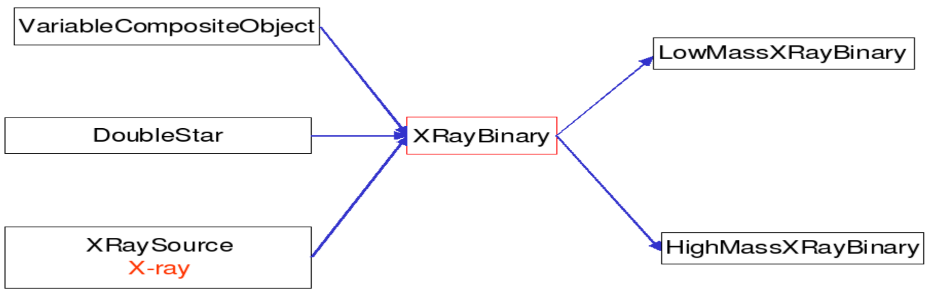
[15] http://vizier.u-strasbg.fr/viz-bin/VizieR

about double stars. Hence the tool searches for VizieR keywords attached to subconcepts of DoubleStar (DoubleStar included)



There is no keyword attached to DoubleStar but the tool will retrieve the ones attached to SpectroscopicBinary, EclipsingBinary, CataclymicVariable and Nova and will suggest them to the user.

- Second search : more general keywords. The user wants to make a query about X-ray Binaries. The tool searches within subconcepts without success. Since XRayBinary does not have an attached keyword either, the tool searches the superconcepts

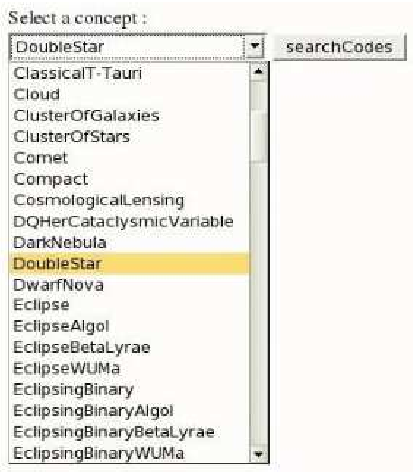


In this case, the tool suggests the keyword corresponding to XRaySource since it is the closest to the original query subject.

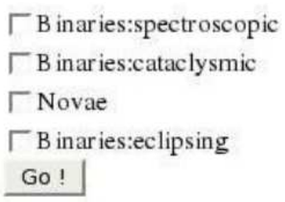
5.2.4 Operation overview

- The user selects an astronomical object type on which he wishes to query the registry from a list of all the object types represented in the ontology.

- The search for keywords is performed



- The request builder outputs a list of suggested keywords corresponding to the wishes of the user.



- The request builder builds a query corresponding to the checked boxes and sends it to the registry.

5.2.5 Benefits of the registry request builder

Such a tool offers the comfort to solve the problem of finding adequate keywords via an automated process. Moreover, the process is highly reliable since it is based on knowledge formalized with the help of experts from the user community itself. Finally, the tool is independent from the knowledge evolution. For example, if a new object type was to be added to the existing list or another changed, all there is to do is to modify the ontology accordingly, not the request builder.

6 Perspectives and challenges

The future of this work is divided in two main orientations: completing and improving the ontology and developing applications.

6.1 Completing the ontology

Currently, the ontology includes the greatest majority of SIMBAD's object types, plus a few more added during the construction. But from the 190 concepts of the AstrObject and AstroPortion branches only 86 have at least one definition, meaning that there are still two thirds to define. Moreover, existing definitions might be improved or backed by additional ones to fit better the applications thus improving performance.

We are willing to establish collaborations with experts who could help us define the astronomical object types, like we are already doing for the Active Galaxy Nuclei section with Paolo Padovani of ESO, or Young Stellar Objects with Laurent Cambrésy of CDS.

6.2 Other use cases

Besides the registry request builder, the applications we are working on are a couple of tools for the SIMBAD database:

- A multiple object type auto-completion and validation tool
  The idea is, for a given SIMBAD entry relative to an astronomical object, to identify the concept this entry is instance of. Then use the ontology to check if the SIMBAD object type(s) associated with this entry are consistent and suggest missing SIMBAD object types if there are some (the SIMBAD object types appear as annotations[16] on concepts in the

16 *simbad:name* and *simbad:shortCode*

ontology). Such a tool could help ensuring that items added to SIMBAD are consistent an well-described by the SIMBAD keywords.

- A cross-identification validation tool
  The idea is to check if the SIMBAD object types derived from the various identifiers associated to a given entry in SIMBAD are consistent. Checkable inconsistencies include complex relationships like the presence of all the necessary components of a system (like a double star). This tool would help guaranteeing the consistency of already existing entries (since they must be checked manually if an inconsistency is assumed), so it is a good complement to the previous tool.

Of course these are not the only possible applications, but they are relevant use cases for the ontology since it is originally built on SIMBAD list of object types and designed to be used as a semantic layer on top of astronomical object databases or similar structures.

Appendix A - Implementation choices

○ *The language of representation*

Since we wanted to build an ontology of defined concepts, we needed a formalism that would allow this. Description Logics[17] is an adequate and mature means of representing ontologies. Furthermore, the Web Ontology Language (OWL) is based on description logics and is probably the most widespread language for describing ontologies, which comforted ourselves in our choices to describe our ontology using Description Logics and to implement it in OWL.

After choosing to implement in OWL, we chose what OWL flavor is best for us:

flavor	***logic***	***decidable***	***comments***
OWL-Lite	SHIF(D)	yes	least expressiveness of the OWL flavors, least resource-consuming
OWL-DL	SHOIN(D)	yes	more resource-consuming but a lot more expressive
OWL-1.1[18]	SHROIQ(D)	yes	revision of OWL-DL, adds qualified cardinality restrictions and more expressiveness on roles, even heavier resource-wise but still decidable
OWL-Full	beyond SHROIQ(D)	no	No limit on expressiveness, only subsets are decidable

OWL-Full is inadequate since we need a decidable logic to use a reasoner. OWL-Lite, though very attractive in terms of performance, allows far less expressiveness than we need. In fact, to match our expressiveness needs, we chose to implement in OWL1.1 at most and OWL-DL at best. And since we try to keep the complexity as low as possible so currently we still are within the boundaries of OWL-DL.
Last but not least, we need a logic which is supported by a reliable reasoner and this is the case with OWL-DL since most reliable reasoners like RACER, Pellet or FaCT++ implement logics corresponding to OWL-DL. Even better : in RACER's case, nearly all of OWL-1.1 is supported.

○ *The reasoner*

Various efficient reasoners are available for description logics. All are based on different description logics and their implementations are summarized in the following table :

[17] http://wiki.eurovotech.org/twiki/bin/view/VOTech/DescriptionLogics
[18] http://owl1_1.cs.manchester.ac.uk/

reasoner	***test version***	***logic***	***implementation***	***License***	***comments***
RACER	1.7.23 and 1.7.24	SHRIQ(D)	CommonLISP	free license	discontinued since 1.7.24 (authors went commercial with RacerPro)
RacerPro	1.9	SHRIQ(D)	LISP	commercial	DIG-only interface is free but not as flexible as the original RACER
FaCT++	1.1.3	SHOIQ(D)	C++	GPL	difficulties with large scale hierarchies
Pellet	1.3	SHOIN(D)	Java	MIT (GPL-like)	Theoretically supports SHOIQ(D) but difficulties with cardinalities and larger scale hierarchies

To determine which is best for our needs, we performed various tests[19]. The tests compared the performances of the different reasoners for the following tasks :

- Checking the consistency of the ontology
- Classifying the ontology (i.e. inferring subsumption relationships for both concepts and properties from the constraints on the concepts)

The tests led to the following conclusions :

- RACER/RacerPro is currently the best reasoner in terms of both performance and reliability if a higher expressiveness is needed, especially regarding cardinality.
- RacerPro being commercial, prices and possible incompatibility with some OWL APIs (e.g. Jena) may be a problem.
- RACER 1.7.23 is known to be compatible with all APIs
- RACER 1.7.24 is a debugged revision of RACER 1.7.23. Specifically, it handles properly complex description logics expressions like anonymous concepts as ranges, which RACER 1.7.23 reports as inconsistent.

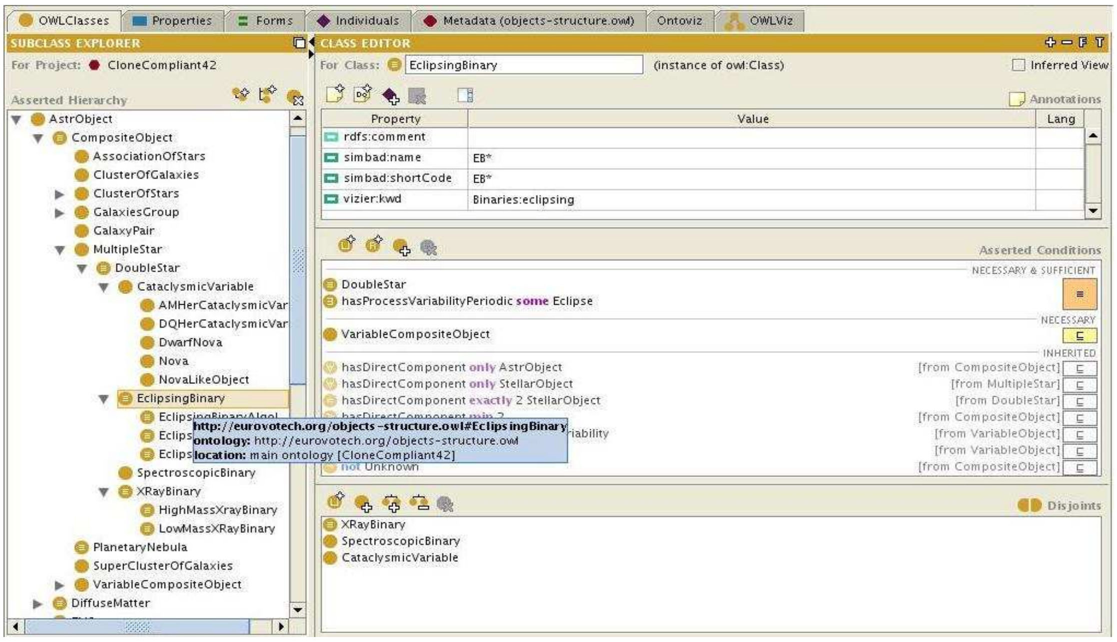
Thus, we chose to use RACER 1.7.24 as our reasoner since it is by far the best compromise. And even if it is discontinued now, the years of research and development on it make it at least as valuable and reliable as its commercial counterpart.

○ *The ontology editor*

The last choice to make for the implementation is to select a graphic editor to build and edit the ontology. We settled for Protégé-OWL [Horridge et al., 2004], developed by the University of Stanford, which is currently both the most complete and most intuitive graphic editor for ontologies.

[19] http://wiki.eurovotech.org/twiki/bin/view/VOTech/InferenceEngineTests

- *Protégé view of concepts*



For Project: CloneCompliant42

- AstrObject
  - CompositeObject
    - AssociationOfStars
    - ClusterOfGalaxies
    - ClusterOfStars
    - GalaxiesGroup
    - GalaxyPair
    - MultipleStar
      - DoubleStar
        - CataclysmicVariable
          - AMHerCataclysmicVar
          - DQHerCataclysmicVar
          - DwarfNova
          - Nova
          - NovaLikeObject
        - EclipsingBinary
          - EclipsingBinaryAlgol
          - Eclips
          - Eclips
        - SpectroscopicBinary
        - XRayBinary
          - HighMassXrayBinary
          - LowMassXRayBinary
    - PlanetaryNebula
    - SuperClusterOfGalaxies
    - VariableCompositeObject
  - DiffuseMatter

http://eurovotech.org/objects-structure.owl#EclipsingBinary
ontology: http://eurovotech.org/objects-structure.owl
location: main ontology [CloneCompliant42]

For Class: EclipsingBinary (instance of owl:Class)

Property	Value	Lang
rdfs:comment		
simbad:name	EB*	
simbad:shortCode	EB*	
vizier:kwd	Binaries:eclipsing	

NECESSARY & SUFFICIENT
DoubleStar
hasProcessVariabilityPeriodic **some** Eclipse

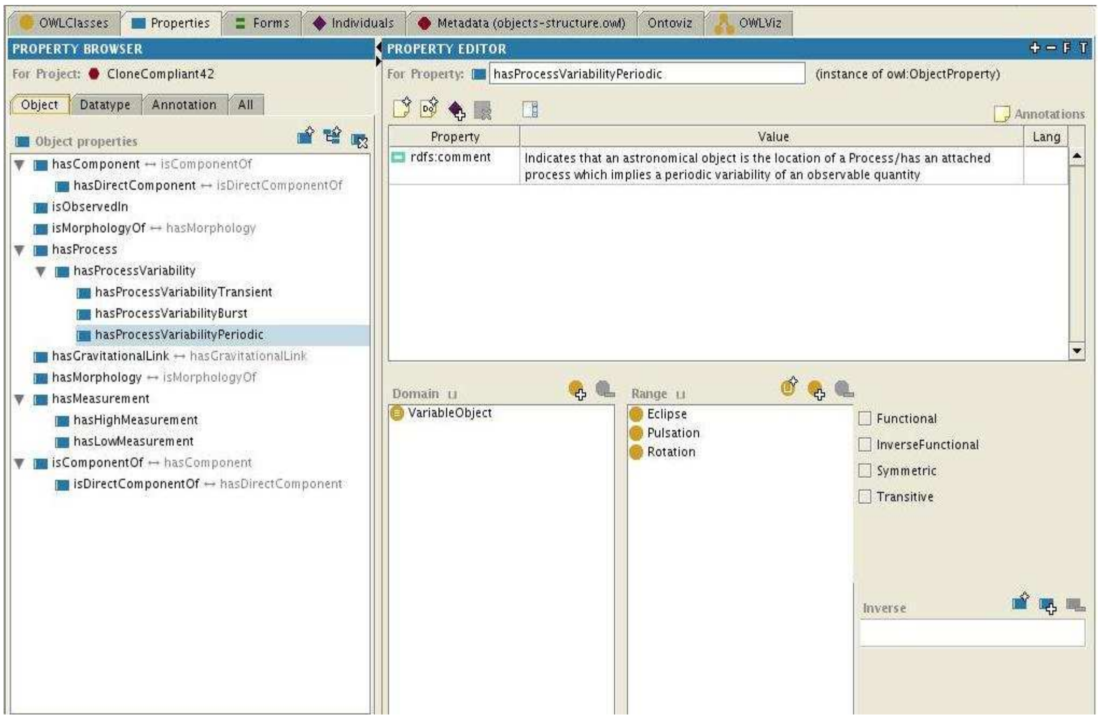
NECESSARY
VariableCompositeObject

INHERITED

hasDirectComponent **only** AstrObject	[from CompositeObject]
hasDirectComponent **only** StellarObject	[from MultipleStar]
hasDirectComponent **exactly** 2 StellarObject	[from DoubleStar]
hasDirectComponent min 2	[from CompositeObject]
…iability	[from VariableObject]
	[from VariableObject]
not Unknown	[from CompositeObject]

Disjoints:
XRayBinary
SpectroscopicBinary
CataclysmicVariable

Protégé view of properties



For Project: CloneCompliant42

- hasComponent ↔ isComponentOf
  - hasDirectComponent ↔ isDirectComponentOf
- isObservedIn
- isMorphologyOf ↔ hasMorphology
- hasProcess
  - hasProcessVariability
    - hasProcessVariabilityTransient
    - hasProcessVariabilityBurst
    - hasProcessVariabilityPeriodic
- hasGravitationalLink ↔ hasGravitationalLink
- hasMorphology ↔ isMorphologyOf
- hasMeasurement
  - hasHighMeasurement
  - hasLowMeasurement
- isComponentOf ↔ hasComponent
  - isDirectComponentOf ↔ hasDirectComponent

For Property: hasProcessVariabilityPeriodic (instance of owl:ObjectProperty)

Property	Value	Lang
rdfs:comment	Indicates that an astronomical object is the location of a Process/has an attached process which implies a periodic variability of an observable quantity	

Domain: VariableObject

Range: Eclipse, Pulsation, Rotation

☐ Functional
☐ InverseFunctional
☐ Symmetric
☐ Transitive

Though the editor is well documented, we set up a page of advice[20] to ensure people willing to use Protégé would not be bothered by some minor problems we

20 http://wiki.eurovotech.org/twiki/bin/view/VOTech/ProtegeAdvice

were ourselves confronted with.

Appendix B - Changes from previous versions

From TN v1.0 :

- New properties and ranges (4.1.3, 4.3.1)
- Concept and overviews updated (4.2, 4.3.2)
- Change of API to Protégé-OWL API (5.1.2)
- Change of reasoner from RACER 1.7.23 to RACER 1.7.24
  (including the description of a RACER 1.7.23 serious bug) (Appendix A)

Glossary

defined concept
Concept which is defined by at least one set of necessary and sufficient conditions

domain (of a property)
Concept to which a property can be applied.

Jena
Java framework for building Semantic Web applications. It provides a programmatic environment for RDF, RDFS and OWL, SPARQL and includes a rule-based inference engine. Jena is open source and grown out of work with the HP Labs Semantic Web Programme. (http://jena.sourceforge.net/)

primitive concept
Concept which is not defined by at least one set of necessary and sufficient conditions.

property (role)
Binary relationship between two concepts or unions of concepts (since you can define a concept as the union of other concepts).

Protégé
Protégé is a WYSIWYG ontology editor developed by the University of Stanford (http://protege.stanford.edu/). It features a version dedicated to OWL ontologies: Protégé-OWL revolving around an API partially compatible with Jena: the Protégé-OWL API

range (of a property)
Concept where a property takes its value.

subsumption
Relationship between concepts or properties. It can be roughly summarized as a kind of a “is a” relationship, meaning that children are more specific than their parents.

References

[Horridge et al., 2004] M. Horridge, H. Knublauch, A. Rector, R. Stevens, C. Wroe A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools Edition 1.0. University Of Manchester, 2004

[Napoli, 1997] A. Napoli Une introduction aux logiques de descriptions. Rapport de recherche RR 3314, INRIA, 1997.

[Napoli, 2004] A. Napoli Description Logics (DL): general introduction. In : Summer School on Semantic Web and Ontologies, Aussois, June 23, 2004.

[Staab and Studer, 2004] S. Staab and R. Studer *Handbook on Ontologies*. Springer, Berlin, 2004.

[Uschold and King, 1995] M. Uschold and M. King Towards a Methodology for Building Ontologies. Uschold M. Towards a Methodology for Building Ontologies Workshop on Basic Ontological Issues in Knowledge Sharing, held in conduction with IJCAI-95, 1995.