



A unified domain model for astronomy, for use in the Virtual Observatory

Version 0.9.1

IVOA Working Draft
2003-11-04

This version:
0.9.1

Latest version:
[0.8.2](#)

Previous versions:
lots

Editors:
Gerard Lemson, Pat Dowler, Tony Banday

Authors:
IVOA Data Modeling working group.

Abstract

.

Status of this document

This is a Working Draft. The [first release of this document](#) was 11 September 2003.

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

Jonathan McDowell, Ray Plante.

Contents

Abstract.....	1
Status of this document.....	1
Acknowledgments.....	2
Contents.....	2
1 Summary.....	2
2 Motivation.....	4
2.1 Analysis.....	4
2.2 Esperanto.....	4
2.3 Use cases ?.....	6
3 Modeling methodology.....	7
3.1 Universe of Discourse.....	7
3.2 Modeling language.....	8
4 Model: concepts.....	11
5 Model: Packages.....	14
5.1 package Phenomenology.....	15
5.2 package Standards.....	18
5.3 package Types.....	22
5.4 package Values.....	24
5.5 package Products.....	26
5.6 package Protocols.....	27
5.7 package Experiments.....	31
6 Example Instances.....	36
7 XML + XSLT binding.....	36
8 Example.....	37
8.1 RASS BSC <-> CVO model.....	37
9 FAQ.....	37
9.1 What is the domain model for ?.....	37
9.2 How can the domain model be used ?.....	37
9.3 Can other datamodels be used as well ?.....	37
9.4 How can we compare different datamodels ?.....	37
9.5 How can we map a data model to the domain model ?.....	37
9.6 How can a VOTable be mapped to the domain model ?.....	37
10 References.....	37
Appendix A SI specification.....	38
Appendix B Fowler.....	39

1 Summary

In this working draft we propose a data model for evaluation by the IVOA data modeling working group.

The data model we propose here is a *conceptual model* in the sense as defined for example in Fowler [1]:

Conceptual models model the way people think about the world [p.314, [5]]

One place where conceptual models play a role is in the *analysis phase* of standard software development methodologies.

The model can be used in various ways.

- It can be used as the basis for a meta-data repository that archives can use to describe their data products in a common model.
- It can be used as a model for use in describing derived representations of these archival data products.
- It can be used as a model describing the entities (classes and attributes) that can be used in a common query language for these astronomical archives and for the relations between that can be followed from these entities in navigation to related ones.
- It can be mapped to an XML schema, to a Java or C# class library, to a relational database schema, allowing reference implementations for these particular bindings.
- It can simply serve as a formal, common language in “whiteboard discussions” about the structure of particular data products.

In the current version we concentrate on the high level framework for the model. We indicate where details must be filled in for more specialized models.

In the design of this model we aimed to represent the *domain of discourse* defined as follows:

A part of the model deals with the concept of *Quantity* proposed for the VO first by Ray Plante and an active area of discussion in the IVOA Data Modeling working groups (see [11]). This model generalizes the various modeling proposals that have been made so far in the IVOA workinggroup, but places it inside a larger context.

Our model includes more concepts that have been mentioned in these discussions such as *Unit*, *Error* and *Measurement*, but fills in the details and properly models the relations between these concepts.

We provide a binding of the model to XML by providing a schema that can be used to communicate instances of the model.

We believe that it is very important as this is a way of translating such documents to other representations. For example, if someone wishes to send data as VOTables, we provide a way to translate these to and from the model using XSLT transformations.

We also provide a binding to the relational model by giving a default relational schema that can be used for most databases

2 Motivation

Here we motivate why we think a comprehensive, conceptual data model such as the one proposed here is beneficial if not essential for the further development of the VO.

2.1 Analysis

Most standard software development methodologies such as described in [1], [7], [8], [9] introduce an analysis phase in the development cycle, during which a conceptual model is developed for the *problem domain*. We will call such a model a *domain model*, to distinguish it from more specialized data models, designed for particular applications or for implementation purposes.

To build the VO, large amounts of software will have to be written for connecting users to astronomical archives, for federating archives and for providing services on top of these federated archives. In this effort an analysis phase resulting in a conceptual model is therefore not out of place, in fact required.

2.2 Esperanto

We believe that the domain model we're proposing here can play a more fundamental role than just as the conceptual analysis model of more standard software development projects. One of the special requirements that software in the VO needs to fulfill is that it needs to deal with multiple, federated (legacy) astronomical archives, being used by multiple types of users. In fact, one can say that this is the fundamental goal of the VO.

The problem that is currently facing astronomers who wish to combine data from a number of archives can be illustrated using the diagram in Figure 1.

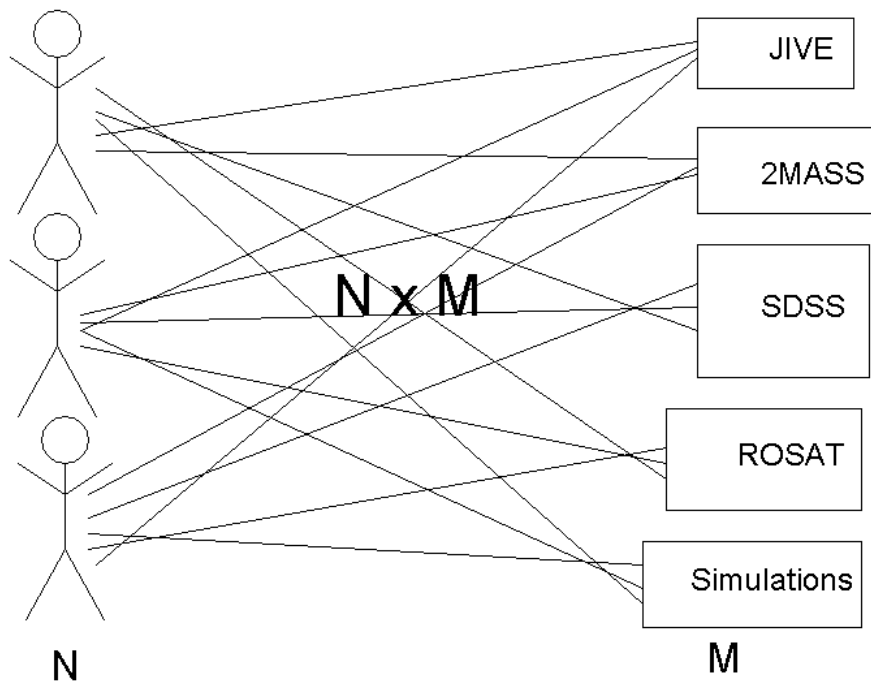


Figure 1: The problem

Many users are interested in many archives. In general, each archive has its own proprietary schema definition and data storage format. This implies that for users to understand the data in each of them they have to learn to understand these schemas and formats. The total amount of effort scales, as indicated, quadratically.

The introduction of a common data model can improve this situation **drastically**, as illustrated in Figure 2. The idea is that users of the VO, possibly through interfaces defined by the VO, will only need to understand the VO's data model but not the individual underlying models of the archives (where they are different from the VO ones). Archives need to be able to describe their data in terms of that same model, need to *map* their schema to it. They will need to be able to answer queries posed in terms of the model and (eventually) return data products that are some predefined binding of the model to the data-exchange language. The total amount of work now scales favourably compared to the old situation.

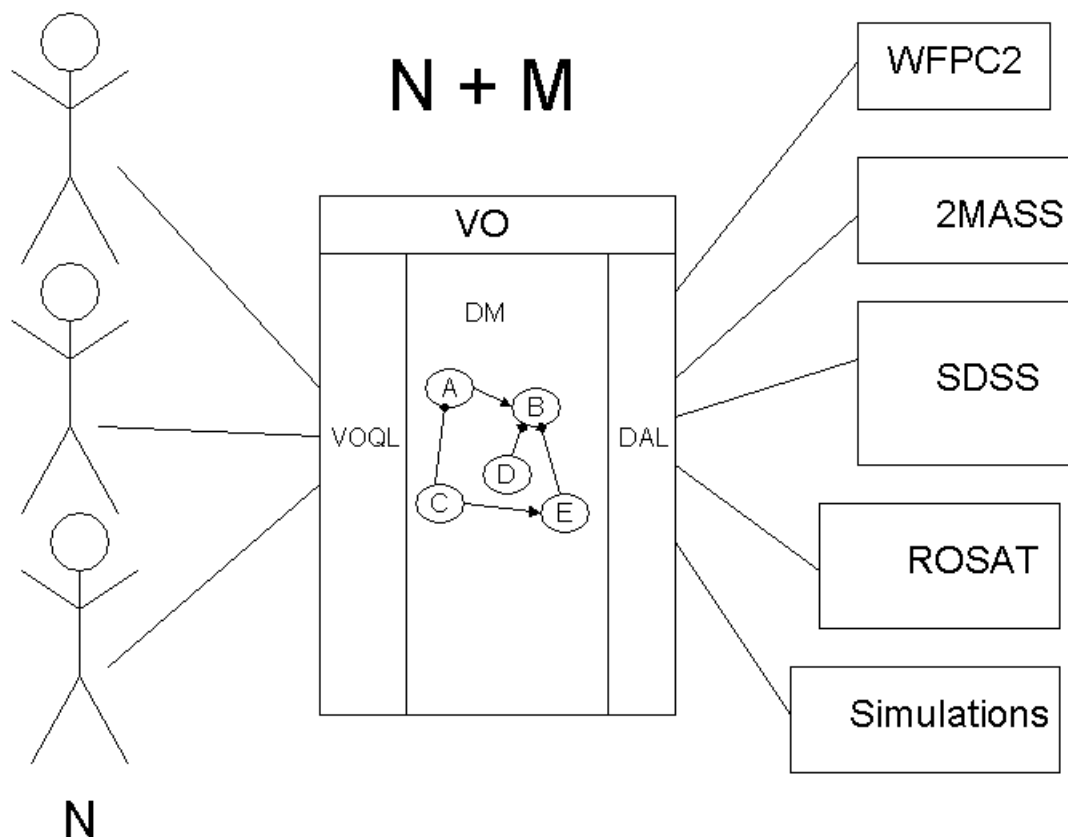


Figure 2: The solution [GL – add some more details on query and archive side]

Another useful metaphor for the use of a common model is the following. We can say that the data in each archive is described in a language that is unique to that archive. Instead of every user being required to learn all the different languages, we could invent a kind of “Esperanto”. The only requirement on the relevant participants in the VO is now that they are able to speak and understand this Esperanto and, where necessary, can translate between Esperanto and their own language. The common domain model plays

this role of “Esperanto” in the interaction between the proprietary model languages of the archives.

Note that in federating services such as cross-matching, archives may also have to be able to be matched to any other archive. Again, if they can do so simply via a common language/model this will improve the scaling for this problem from quadratic ($M \times M$) to linear, M .

This way of using the common domain model is equivalent to an *ontology* as used in discussions for example about the semantic web in [10].

2.3 Use cases ?

Our model is not directly based on explicitly stated, detailed use-cases.

The ultimate use of the IVOA datamodel, in our opinion, is to define, in a formal way, what meta-data are required to describe an archival dataproduct that is published in the VO.

Secondly, any archive publishing data in the VO should be able to answer questions phrased using elements from the data model. The precise syntax of the query language, presumably the future versions of VOQL, is not defined. It is anticipated though that the elements that can be queried and the relations that can be followed are those of the data model. An example of how this might work is provided by the Object Data Modeling Groups’s (ODMG) definition of Object Query Language (OQL) [GL – References !].

Use cases for a conceptual model such as the one we’re describing here are of relatively little use (page 1. in [1]). They can not completely prescribe what ends up in the model, the modeling effort is not completely constrained by them. However we can see some ways in which the model can play a role and these can be seen as use cases enabling further use cases for the VO as a whole.

Meta-data repository

The model as presented here can be used as the basis for a persistent model for a database, be it relational or object oriented. Such a database can be used for storing instances of the model that describe data products in an archive.

Such *meta-data repositories* can be used as a reference implementation through which archives can be officially *published* in the virtual observatory. They also provide a reference implementation for compliance with the future version of VOQL, which is supposed to be expressed in terms of the data model.

Such a meta-data repository can be viewed as a fine-grained registry.

XML Schema

VOQL

3 Modeling methodology

A modeling methodology is a set of rules that help us in making decisions when choices must be made between alternatives during the modeling process. Here we give a motivation and high level description of the methodology we have tried to follow in the design of the domain model.

As defined above, our domain model is a conceptual model, created as the result of the analysis of the problem domain. We have chosen for an object-oriented approach. The reasons for this are ... [GL – Fill in !]. Consequently our model is the result of an object-oriented analysis, which Booch (p. 516 of [9]) defines as “a method of analysis in which requirements are examined from the perspective of the classes and objects found in the vocabulary of the problem domain.”

Our first task therefore is to define our problem domain, in other contexts also called the *universe of discourse* (UoD) (see for example [8]), as it is “.. the world(or universe) that we are interested in talking (or discoursing) about” ([8] p6).

Then we describe how we

3.1 Universe of Discourse

We believe that the UoD for the VO is “the work that astronomers, astrophysicists and support scientists do and the results they have obtained”. Our motivation for this choice is that we believe that users of the VO are ultimately interested in the results of the work done by other astronomers. We believe that users are not “just” interested in getting access to images, simulation results or other physical results of astronomical research, stored in some astronomical archive. We believe they will want to know what is actually represented by these results, *how* these results were obtained, what experiments were executed and how. The latter is what we mean by the term “work”.

When we say that we believe VO users *will* be interested in the experiments that produced the results, we ought to say that they *should* be interested in them. We believe that one of the main tasks of the VO is to enable other astronomers to do rigorous science with the results and services that are made available by their colleagues through the VO. It is obvious that results can only be interpreted through knowledge of the process that produced the results. In many discussions this is summarized using the term “provenance” of the results. We believe the VO has a chance, actually the duty, to formalize the concepts underlying this provenance by including it explicitly in the modeling effort, by making it part of the VO’s Universe of Discourse.

The model that is described in the current paper has been the result of the domain knowledge extracted from the modelers (some of whom are astronomers) and their direct coworkers as well as from literature and other external references. We did *not* have an official use-cases/requirements period. We do not claim that this should not be done. Our claim is that this will lead to refinements of the model as is, but need not fundamentally change the modeling process.

In putting the domain concepts into formal terms our approach has been to be as explicit as possible. Though we see the value of generalization, we are careful in using the *inheritance* relation too soon in the process. In general we introduce a superclass when some other class in the model requires it or when a concept can be *specialized* to, i.e. we prefer the top-down approach if it is the supertype that occurs as a natural concept in the domain.

One consequence of being explicit is that we introduce classes in our model that allow us to describe the “work that astronomers do”. This work has many aspects. The ones we concentrate on here are those that are relevant for the Quantity discussion, and are those aspects in which values are assigned to properties of subjects of astronomical interest. In this section we describe the packages that we have defined and for each package the classes it contains.

3.2 Modeling language

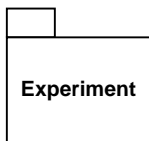
We use a subset of the UML modeling language to express our models.

In particular we have created *Static Structure* diagrams using the Microsoft’s Visio 2002 modeling tool.

We use the following modeling concepts from UML and refer to them in the descriptions in the following section by the italicized names:

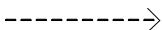
- Package: *package*
Container of subpackages, classes and datatypes. As all our binary associations are navigable and unidirectional, we do not view associations as being directly contained by the package, but by the class that can navigate the association.

Symbol :



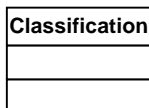
- Dependency: *dependson*
Unidirectional relation only used between packages. Indicates that the contents of one package (may) depend on (that is, can navigate to by means of a reference) the contents of another package. This relation is transitive. The graph created by the dependency relation is a directed, acyclic graph (DAG).

Symbol:



- Class: *class*
A class encapsulates a concept with class properties (in contrast to DataType), which means that ... [GL – fill in.]
A class can be *abstract* or not. In the latter case we say it is *concrete*. This is indicated in the symbol by the name being slanted (abstract) or not. Only concrete classes can have instances.
In the methodology we use we will *concrete* with final in the sense of Java. That is, concrete classes can not have subclasses.

Symbol:

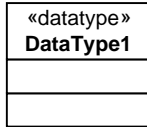


NB

As we are discussing a conceptual model here, it might be better to talk about *type* instead of *class*. It can be argued that a class is an implementation of an (abstract) type. According to Fowler, types correspond more to interfaces, not to implementations. Here and in the following we mix the concepts, the more so as

we do not use the UML *interface* concept here anyway, so the danger of confusion is minimal.

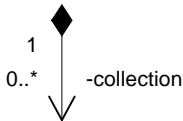
- **DataType: *data type***
 A data type
 Symbol:



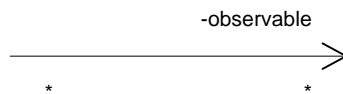
- **Generalization: *specializes, extends***
 This relation indicates an “is-a” relation. It is unidirectional and transitive and points from the “subclass” to the “base class”. The subclass inherits all properties (attributes, collections, references, operations) from the base class. We use single inheritance only.
 Symbol:



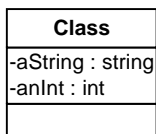
- **Composition: *contains/contained in***
 The way we use composition is as a unidirectional binary association from parent (*container*) to child (*containe*) that is not shared (i.e. has *aggregation* value “composite”). The association end at the child’s end is named.
 Symbol:



- **BinaryAssociation : *reference***
 A reference is a unidirectional binary association directed from *referrer* to *reference*. The association end at the reference side is named and has cardinality 0..1 or 1. The association end at the referrer’s side is unnamed and has cardinality * for shared references, or 0..1 for “owned references”.
 Symbol:



- **attributes: *attributes***
 Attributes are fields on classes whose type is a “simple value type” or a custom DataType. Attributes are always of cardinality 0..1 or 1.
 Symbol (as the middle cell in a Class symbol):



- **stereotype <<key>>**
 Indicates on an attribute or reference that it is, or is part of a unique identifier for the class within its containment context. This means for root classes that the class can be globally (or only within the meta-data repository ?) uniquely identified by the collection of attributes/references that make up the key. For a non-root class it is (at least) unique within its collection. To identify an instance in

a collection uniquely with its key it is assumed that the key of the container (possibly recursively) must be included.

Symbol: prefix to attribute name:

Class
«key» -attribute

4 Model: concepts

Here we give a list of those concepts (nouns) that we believe are relevant for modeling the universe of discourse defined in 3.1. This list is by no means complete. It contains those concepts that we feel belong to the framework for describing the VO's universe of discourse. Each of them contains many specializations and need to be described in more detail using other concepts that have to be added. Instead of writing those down here, where we did go into more detail we have done so directly in the models below. The concepts are somewhat ordered, in that related concepts are near each other, but since the real structure of the model is a graph this can not be represented in a 1-D list. Also, the way to tell the "story" that we're modeling can start at many parts of the model and the current ordering presents only one possible order.

- **experiment ("unit of work")**
Part of the work that astronomers do can be summarized under the term *experimentation*. This does, in the current model, *not* include the work for example of defining and implementing computer code with which results can be obtained. Nor does it include the work of building telescopes for example.
- **result**
The *data* resulting from astronomical work that users of the VO are assumed to be interested in. In the current model this term does *not* include any other "results" of astronomical work such as source code written for general use, telescopes built to do observations, papers written to describe work and results. Question: should these be included in the model explicitly *as results of work*? I think that does not belong to the UoD for the VO. We may, until further notice, assume services and telescopes as given means to produce results. Papers may also end up in the model, but probably not as results either. Result should be confined to *data* objects.
- **protocol**
An experiment should be properly described, that is it should be possible to retrieve (in principle, somehow) what was actually done, why, how. This we call the protocol of the experiment. It has an obvious use when a certain protocol can be followed multiple times. An example of this is a simulator. Just by changing
- **objective** (alternative **observable/goal**)
This is the goal of the experiment, though it is described in the protocol. We can actually describe this in more detail.
- **variable** (related **dependent/independent**)
We claim that the objective of an experiment is ultimately to assign values to certain properties, by measuring them or simulating/calculating them. An objective is ultimately a collection of variables, to which values must be assigned. The variable is tuple of properties to which a value is to be assigned for a given objective.
- **measurement**
A **result** ultimately consists of values that have been assigned to variables. a measurement is a special kind of "value assignment", namely the common one corresponding to a numerical value. We can generalize this to values that can not be "measured" in the usual sense.
- **value/quantity/category**
This is the ultimate "leaf" in the result hierarchy. The concept **value** corresponds to a value assigned to a property. SI calls this a "value of a physical quantity" (see Appendix A and [2]).

- **uncertainty/confidence/error**
In the scientific process of assigning values to properties, for example by measurements, errors may be made. Errors can have different causes. They may be caused by finite resolution of measuring apparatus and the consequent random error this causes in the resulting value. These can sometimes be associated on a value by value basis. There may be systematic errors that pertain to all values that have been assigned in a specific experiment. There may be correlations between errors for different measurements. However they may be caused, to interpret the values that have been assigned it is imperative that they be accompanied by the confidence/uncertainty associated to them.
- **classification**
There are certain **phenomena** that seem not to be “measurable” in the simple sense of the word. For example, *galaxy type* is a useful property to associate to a source that has been classified as a galaxy. Though there may be more or less quantitative (“measurable”) criteria that ultimately decide to which class a galaxy belongs, the *act* of assigning this value is in general different from those phenomena/properties that get assigned values using **measurements**. We believe there is a qualitative difference here that deserves to be taken into account. The value that results of it is in general a string, the uncertainty will be assigned in a different manner. Hence we believe it should be separately modeled.
- **identification**
We believe that there is a third category of value, namely a name that is assigned to a subject, identifying it with/as a previously known object. The value will be a **name**, taken from a list of allowed standard names.
- **phenomenon**
This concept corresponds to SI’s “quantity in the general sense” (Appendix A, [2]), or to Fowler’s PhenomenonType ([1]). It corresponds to the generalization of “a property that a body or substance can have” ([2]). These we believe are the ultimate concepts scientists are trying to determine, apart from a modification described by **property**.
- **property**
This is a phenomenon assigned to a particular body, or substance. It can also be said that a particular body, substance is defined by the collection of its properties that we can measure/know/be interested in. It corresponds to SI’s “quantity in the particular sense” ([2]).
- **subject** (body, substance, space, ...)
Subject is the concept that unifies the body, substance etc mentioned in the SI definition for “quantity in the general sense”. It is really an anchor concept that somehow corresponds to its collection of properties. Is it more than that ? We can give more meaning to this concept probably only by subclassing it into meaningful examples. One may say that **subjects** are the “things” of which we want to measure properties. We have to see this in a somewhat abstract manner for our domain, astronomy, as some of the things we are interested in are as abstract as “a direction on the sky”. Others the images we analyse looking for sources.
- **unit**
This concept is formally defined by the SI specification ([2]). It defines the measuring rod that is required to interpret numerical values (quantities) assigned to numeric properties. SI defines it as a particular property of a particular,

identified subject. All values of the same phenomenon can be expressed by giving the ratio of that property to the property of the identified subject.

- **reference system**

Just as a unit specifies the “measuring rod” for interpreting numerical values assigned to a property, so the reference system can be thought to specify the zero point. An example is a coordinate system for positions on the sky, another a magnitude system for expressing fluxes.

- **representation**

This concept encapsulates the way values assigned to a phenomenon may/must be represented within the UoD. It should be thought of as a kind of abstract type definition. For example, if we define the phenomenon *sky position*, we need to define what (meta-)data are required to specify this phenomenon completely. An example could be that one requires a coordinate system, a longitude, an latitude each of which requires a unit. *How* this is expressed in a particular language is a different issue that is not dealt with in this concept. but if someone tells us they have a value for a sky position, we should be allowed to expect that we can inquire into the coordinate system, latitude and longitude.

- **standard**

For interoperability to work, or more simply, for us to be able to understand each other’s (meta-)data, it is not sufficient to provide a model, we also need to agree on a number of standard instances. For example, we need to agree on definitions of particular units and how we call them. The same for coordinate systems and also the base phenomena themselves. There may be multiple standards that can be used, for example we may choose to allow both SI and cgs units, but we must agree on which instances are valid.

- **physical artefacts**

Ultimately the data are stored somewhere in a physical datastore. This may be a filesystem or a more formal database system. We need to be able to identify and locate these as these are ultimately the containers containing teh data results users of the VO are interested in. One can claim that the whole goal of the data model is to be able to describe these physical data stores down to some desired level of detail.

-

5 Model: Packages

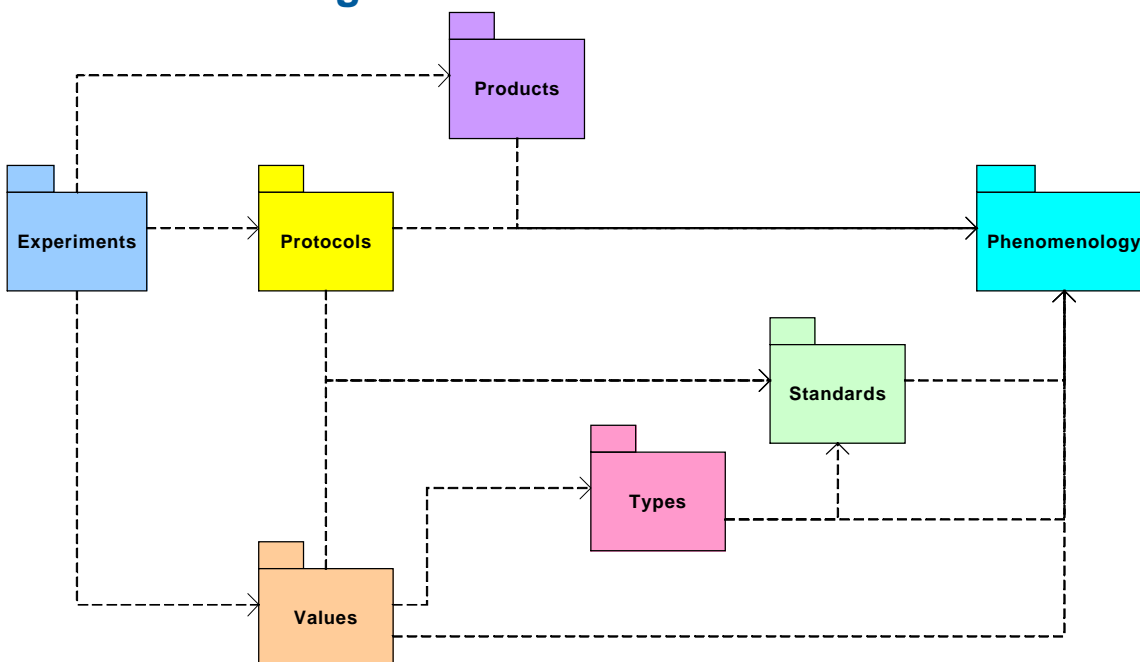


Figure 3: The packages and their dependencies in the model.

We have divided the domain in a number of subsets containing concepts that are more closely related. These subsets have relations between each other. In UML one can represent such subsets as packages and that's what we've done here. In a different binding such as XML, these packages may correspond to namespaces, or in a relational database possibly to a schema.

In the following sections we describe the contents of each of these packages in an order corresponding to a topological ordering of the package-dependency graph in Figure 3.

5.1 package Phenomenology

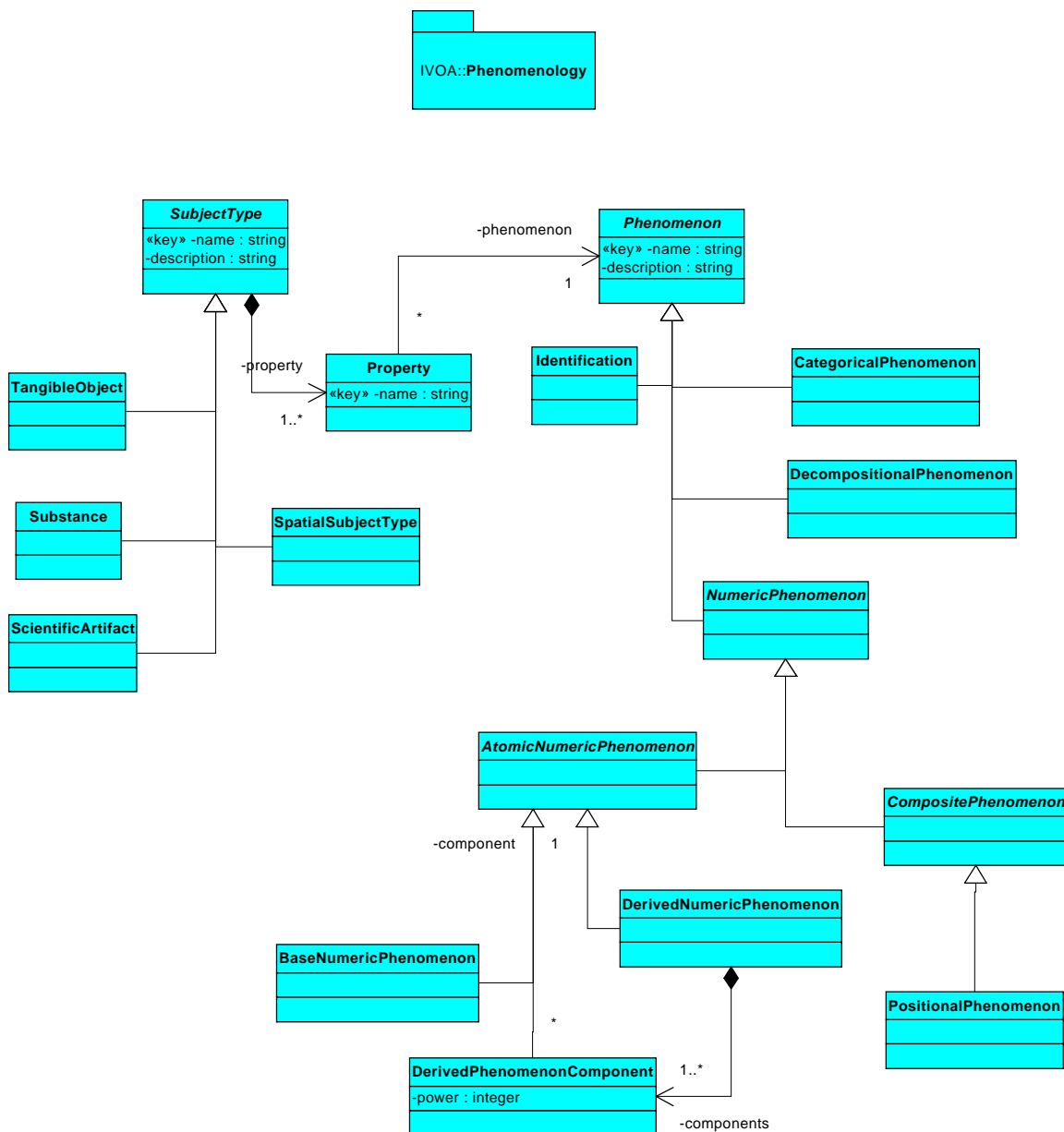


Figure 4: Diagram for *Phenomenology* Package

This package describes the phenomena that can be measured, calculated, simulated etc in the scientific process. We have used the SI model for inspiration in defining the contents of this package [3], see also Appendix A.

abstract class Phenomenon

Models the generalized concept of a property that some subject may have. Examples are *length*, *mass*, *shape*, *galaxy type*. Implements Fowler's [1] PhenomenonType and generalizes SI's "quantity in the general sense" (see [2]).

abstract class NumericPhenomenon extends Phenomenon

Those PhenomenonType-s that correspond to numerical quantities, possibly structured. Examples are *length*, *mass*, *area*, *shape*.

abstract class AtomicNumericPhenomenon

Models a numeric phenomenon that can be expressed using a single value. These include basic phenomena such as length, mass, temperature, but also derived phenomena such as area(=length²) and velocity(=length/time).

class BaseNumericPhenomenon extends AtomicNumericPhenomenon

Models a basic numeric phenomenon, including but not restricted to SI's 7 base quantities, *length*, *mass*, *time*, *electric current*, *thermodynamic temperature*, *luminous intensity* and *quantity of matter*. To these SI associates *Units* that will be described below.

class DerivedNumericPhenomenon extends AtomicNumericPhenomenon

A phenomenon that can be expressed in terms of other quantities, in particular base phenomena. Includes, but is not restricted to, SI's derived quantities [2].

class DerivedPhenomenonComponent

Contained in DerivedNumericPhenomenon. Implements the aggregation of its container class in more basic numeric phenomena. Adds the power with which component quantities appear.

Examples: area = length*length = length²; velocity = length*time⁻¹.

abstract class CompositePhenomenon extends NumericPhenomenon

Models structured phenomena, which for now we assume will always be numeric (as we have not yet seen any other examples). Examples of this are *position*, *shape*. These can be expressed in terms of more fundamental phenomena in some hierarchy.

class PositionalPhenomenon extends CompositePhenomenon

Phenomena that deal with locations in space

class CategoricalPhenomenon extends Phenomenon

description

Implements the phenomenon that models classification properties of subjects. That is, we can discuss, for certain subjects, that they belong to a certain class.

Examples: galaxy type, stellar type.

attributes

- name: string
- description: string

references

note

It may be that someday this will be expressed as another special subtype of numeric phenomenon, for example when we can give a quantitative description of what it means to belong to a certain class. However, the *act* of classification sofar is different from the act of measuring more standard numeric properties. For this reason, and following our methodology stated in 3.1, we create a separate phenomenon for this.

class Identification extends Phenomenon

description

class DecompositionalPhenomenon extends Phenomenon

description

This phenomenon describes how a subject can be described as being built from other subjects.

class SubjectType

description

The "thing" that is being examined, of which properties are evaluated, measured. Examples are astronomical objects, regions (including points) of space, also scientific artifacts such as source, image, which are the subject of analysis experiments.

attributes

contains

- [Property](#)
A subject is really defined completely by the set of its properties. These are by definition the only observable properties of the subject, i.e. those aspects by which the subject can be known, identified.

references

NB

Difference between subclasses and instances (such as source/object/.. etc)

abstract class Substance extends SubjectType

description

class ScientificArtifact extends SubjectType

description

class SpatialSubjectType extends SubjectType

description

class TangibleObject extends SubjectType

description

class Property

description

A property of a subject is the thing through which we know the subject. It is the assignment of a Phenomenon to the subject. The subject is described fully by its complete set of properties. One may say even that the subject is *defined* by its properties.

attributes

- *name: string*

references

- phenomenonType: [PhenomenonType](#)

5.2 package Standards

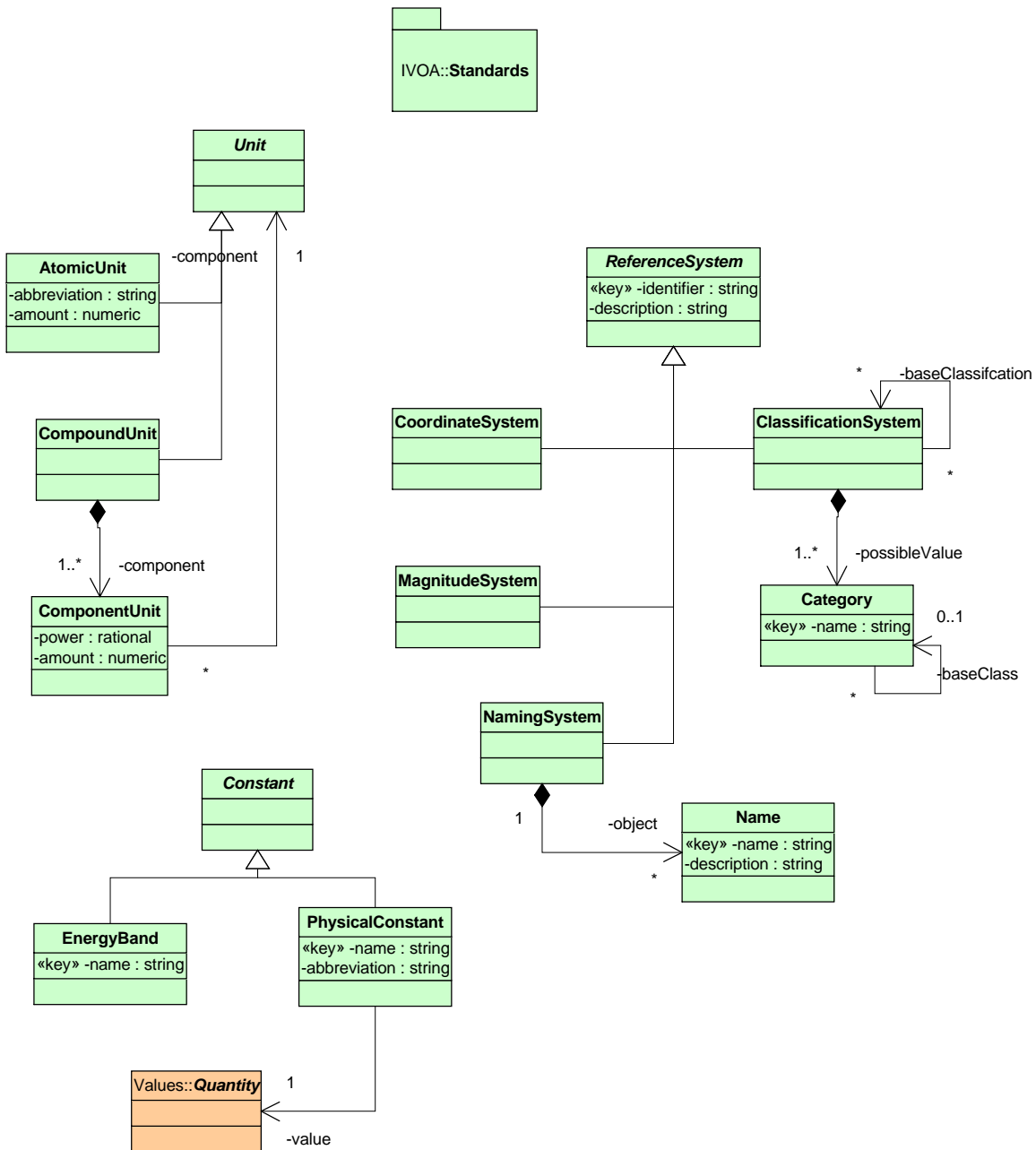


Figure 5: Class diagram for the *Standards* package.

This package describes those concepts for which instances are defined by standard bodies or convention. These instances serve as reference data to be used when creating instances of other concepts. One may compare the classes and instances defined under standards as the “vocabulary” of the “language” whose grammar is defined by the model itself. This package must have a physical representation somewhere as a meta-data dictionary/repository that can be referenced. Examples of the concepts in this package are physical units, standard catalogues, reference systems for coordinates, magnitudes etc.

abstract class Unit*description*

Base class of physical units. We explicitly model simple (atomic) and compound units. But we need a single class that all quantities can refer to.

*description**attributes*

-

references

-

contains

-

class AtomicUnit extends Unit*description*

Implements a basic unit. As defined in the SI standard [2], a basic unit corresponds to a particular, identified Property.

For example, in the SI Standard, the weight of a particular platinum cylinder stored in some vault in Paris is [GL – was ?] the unit with name kilogram for the phenomenon weight. All other weight properties can be expressed as a ratio between the value of that property with the unit.

attributes

-

references

-

contains

-

class CompoundUnit extends Unit*description*

Implements a compound unit, i.e. one that does not correspond to a single property but is derived from more fundamental ones.

Examples: phenomenon: velocity, unit: $m*s^{-1}$;

attributes

-

references

-

contains

-

class ComponentUnit*description*

Implements aggregation of more fundamental units to build a CompoundUnit

attributes:

- power: int

references

-

contains

abstract class ReferenceSystem

description

Similar to the way a Unit gives a measuring stick for expressing values of corresponding numeric phenomena, so a reference system may be interpreted as defining a zero point. Examples are coordinate systems or magnitude systems (both modeled separately)

attributes:

- *references*
- *contains*

class CoordinateSystem extends ReferenceSystem

description

A reference system to be used when providing values for positional phenomena. Instances of this class include for example *galactic*, *equatorial J2000* etc. This class may be modeled more fully using Arnold Rots's space time coordinates model, see [3].

attributes:

- *references*
- *contains*

class MagnitudeSystem extends ReferenceSystem

class NamingSystem extends ReferenceSystem

class Name

class ClassificationSystem extends ReferenceSystem

description

A system for classifying astronomical objects. Examples of instances are

attributes:

- *references*
 - baseClassification : [ClassificationSystem](#)
We allow for nested classification systems by expressing that the current one is a child, that is refinement, of the base classification system. Examples are galaxyType being a child of astroObject, and a parent of spiralGalaxyType.

contains

- [Category](#)

class Category

description

Enumerated type defining possible values in a standard [ClassificationSystem](#).

attributes:

- name : string

references :

- baseCategory : [Category](#)
Mirrors the baseClassification in [ClassificationSystem](#). This category can express through a reference to a category in the base classification which is the parent of the current category. For example we can express that "E5" is a category in the ellipticalGalaxy classification system and is a child of the category elliptical in that classification's parent galaxyType.

abstract class Constant
class PhysicalConstant extends Constant
class EnergyBand extends Constant

5.3 package Types

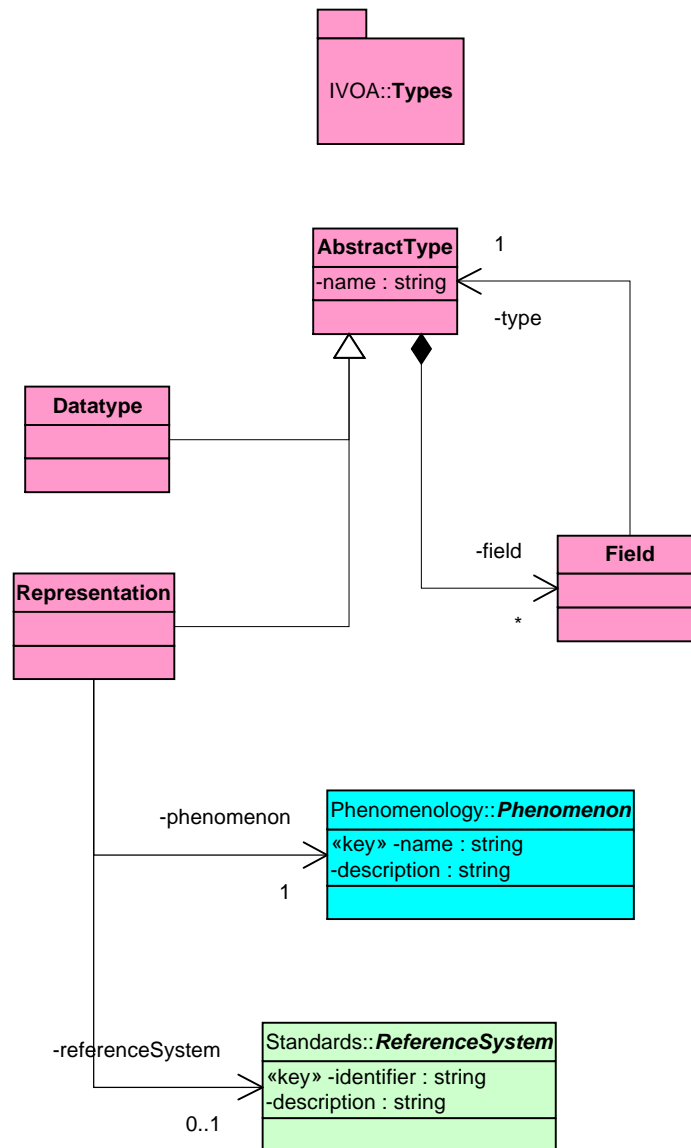


Figure 6: Class diagram for *Types* package

This package defines accepted standard ways to represent values for Phenomena and other standard datatypes.

abstract class AbstractType

This class represents the type for a [Quantity](#) or attribute.

attributes

- name : string
The identifier for this type.

class Datatype extends AbstractType

This class represents the type of an attribute. In our model these are basic *conceptual* types such as the numeric types: *integer*, *real*, *rational*, *complex*. Or others like *string*,

date. We acknowledge that the representation of types in persistent archives will add features to these types, but think that can and should be explicitly modeled in the modeling layer dealing with representations.

class Representation extends AbstractType

This class describes in a formal way what elements are required to provide a value for a given [NumericPhenomenon](#). As such it prescribes all the contents a Quantity instance needs to contain to be a proper value for the given phenomenon. Multiple different representations are possible for a given phenomenon. Again, this is no statement about the representation (sadly the same word) of a Quantity in a persistent archive or message. It does prescribe though the data an archive must be able to provide about values for a given phenomenon.

references

- *phenomenon* : NumericPhenomenon
The phenomenon for which this class is a representation.
- *referenceSystem* : ReferenceSystem
Provides the context/zero-point within which to interpret the other variables. We might want this to be an aggregation of valid reference systems, so that for example, we do not for every possible SphericalCoordinateSystem need to create a representation that indicates it is allowed to use that CoordinateSystem. Instead we might define one and have it implied that all ReferenceSystems that can be transformed to/from that specified one are valid as well. Somewhat akin to an equivalence relation as it were.

class Field

We anticipate that an abstract type may be structured, possibly hierarchical. The way to describe this is, as in standard OO modeling, through fields. These fields are themselves representations of

attributes

- *name* : string
The name by which this field is represented in the type.

references

- *type* : AbstractType
The type of the field.

5.4 package Values

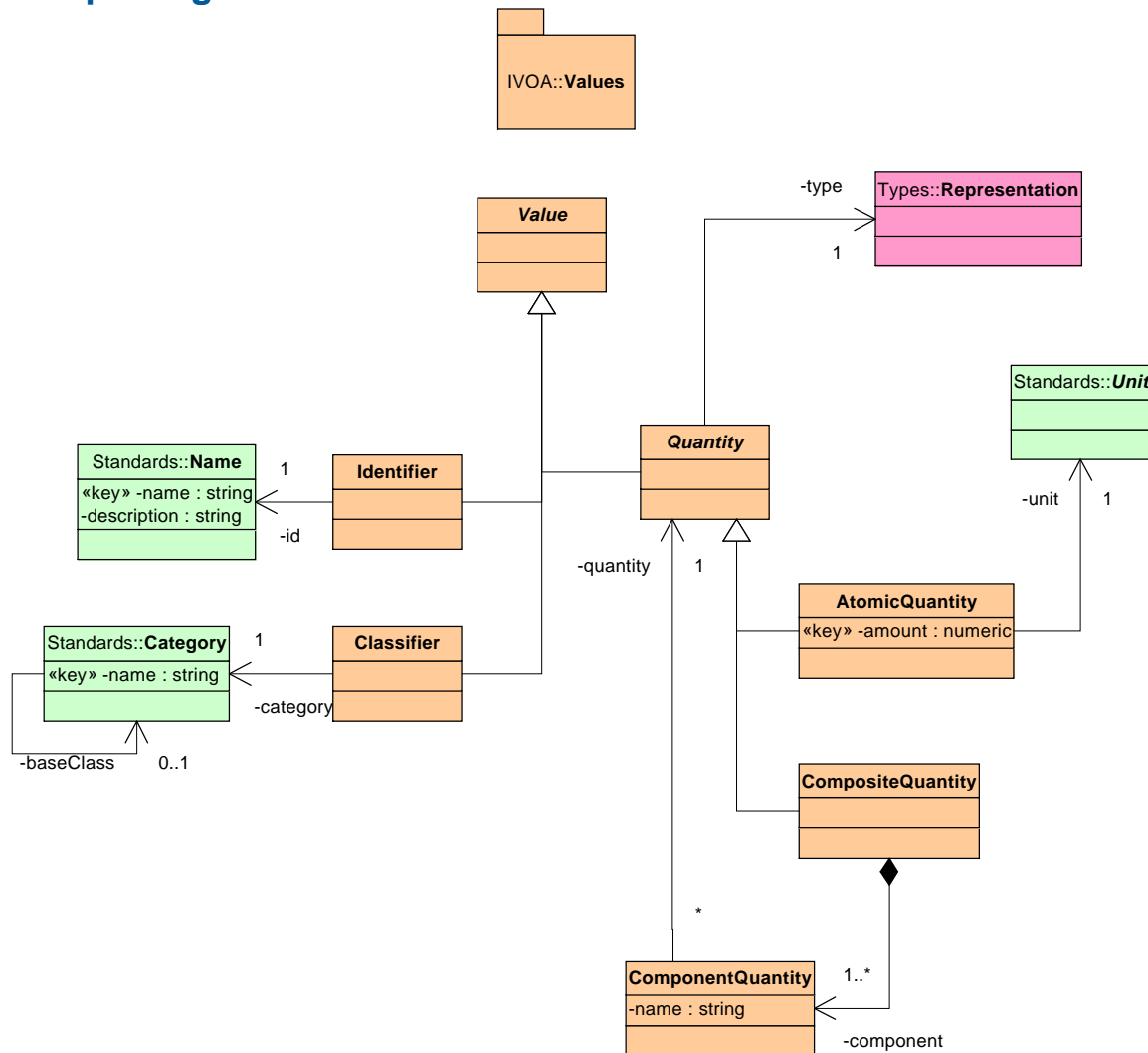


Figure 7: Class diagram for Values package.

This package deals with the actual values that properties can take. This package contains a class encapsulating Ray Plante's Quantity concept that started this whole discussion, though with a particular choice from the range of interpretations that have been put forward in IVOA dm QUANTITY email trails.. In this package we extend the original concept to include also non-numeric values that can be assigned to properties by introducing classification values and identifications. We limit ourselves however to the most basic interpretation where the basic Quantity itself is concerned, namely to a Unit and an amount. In contrast to certain proposals we have provide a different place for Measurement and Error and such.

class Value

description

Baseclass encapsulating the possible values of properties, phenomena etc.

attributes

references
contains

class Quantity extends Value

description

A value based on numeric values, possibly composite. Is the result of a measurement

attributes
references
contains

class Classifier extends Value

description

attributes
references
contains

class Identifier extends Value

description

attributes
references
contains

class AtomicQuantity extends Quantity

description

attributes
references
contains

class CompositeQuantity extends Quantity

description

A composite quantity is an instance of a composite phenomenon.

Examples: ellipsoidal shape, built from a central position, a major axis, ellipticity and position angle. More complex composite quantity definitions than the current one may be required to describe for example a polygonal value for a generic shape.

attributes
references
contains

- ComponentQuantity

class ComponentQuantity

description

Implements the aggregation of the more fundamental quantities making up a composite quantity.

attributes

- name : string

references

- quantity : Quantity

contains

5.5 package Products

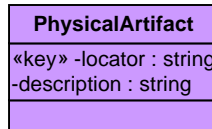
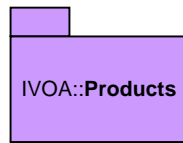


Figure 8: Class diagram for *Products* package.

This package models how results are represented/stored in a persistent archive. We have not paid any amount of attention to this package apart from the root class. We believe other efforts, for example Brain Thomas and Ed Shaya's Quantity models could fill in this package in more detail.

class PhysicalArtifact

This class represents a persistent, physical *representation* of a [Result](#). For example as a file of a certain format, or one or more records in a relational database, or an ASCII table representing a catalogue or

The main aim of this class in the model is that allows a unique identification of results or input data from/to [Experiments](#). It is assumed/prescribed in our model that results and input of experiments are physically represented as such a physical artifact that can be located uniquely, so that in principle users of the model can locate it.

Alternatively one may say that the core goal of the Virtual Observatory is to publish the physical artifacts that are the results of Experiments. Archives will describe their data products as such physical artifacts and users will look for physical artifacts with specific properties as described by the meta-data defined by the rest of the model.

This class forms the boundary between data and meta-data. An instance of this class will allow one to locate the actual physical product (for example an image, or a catalogue). We may want to model this class in more detail by describing the actual storage format and how this represents the particular meta-data entities (variables, properties, errors etc) that are actually stored here. This may be the task of Quantity modeling efforts such as that by Thomas and Shaya. [4]

attributes

- locator: String
A URL that allows one to retrieve the actual physical artifact for which this class is a proxy.
- description: String
A description of the physical artifact

5.6 package Protocols

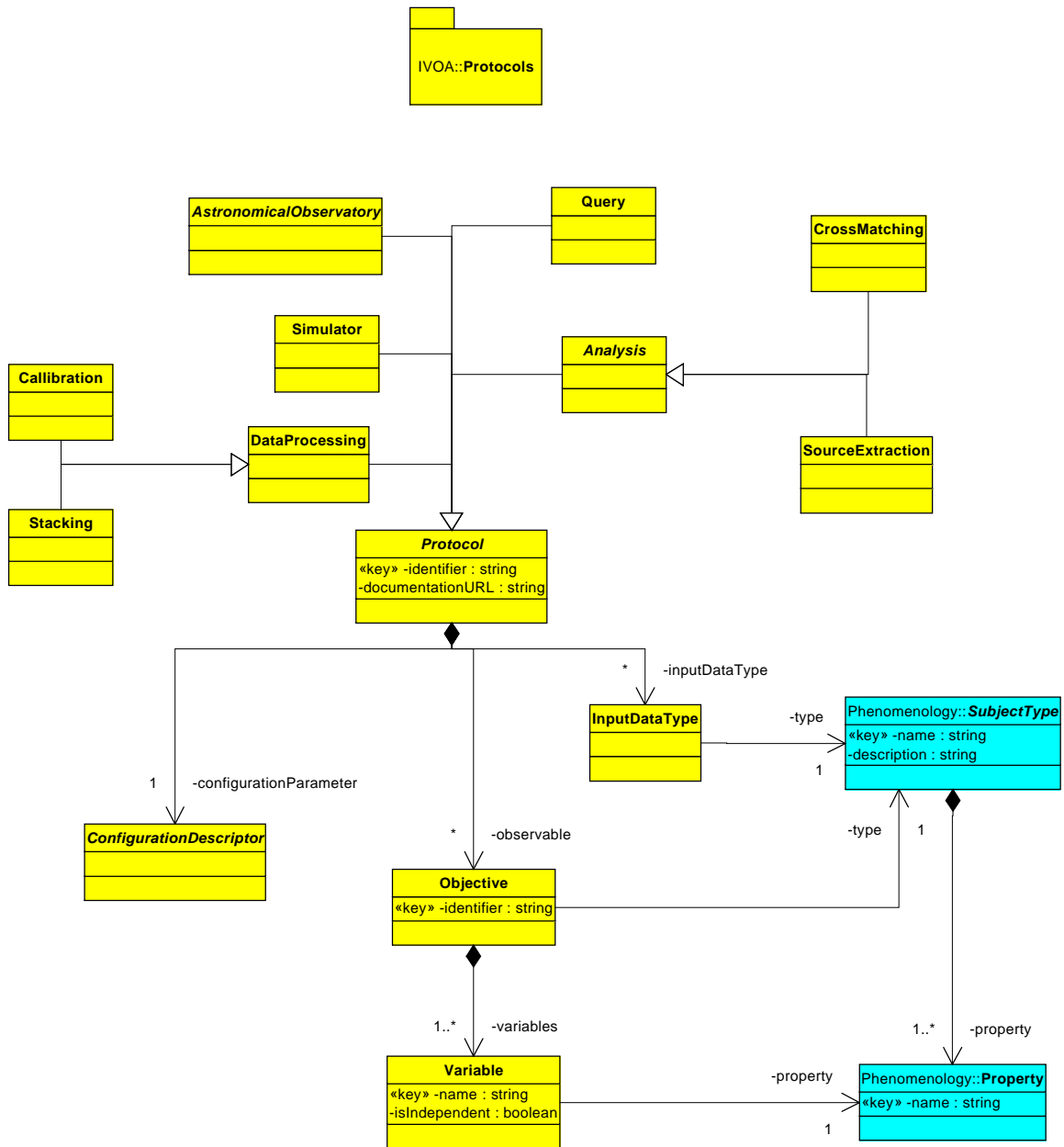


Figure 9: Class diagram for *Protocols* package.

abstract class Protocol

description

Protocol encapsulates a description of *how* a certain experiment can be performed. It can be seen as a blueprint that can be followed to execute a given experiment. It is separate from the experiment execution, which we capture in [Experiment](#), as a given protocol can be used to describe the workflow of many different actual experiments. There are various examples that can illustrate this concept more clearly.

- process/service ala SExtractor [GI – reference]
vs
executing it on a given image using particular parameter settings
- simulator : code with standard configuration files required to call it
vs
simulation: particular initial conditions and parameter settings.
- “observatory” in the sense of a given possible telescope configuration, with a
standard observation proposal
vs
an actual observation with a particular pointing of the telwscope and exposure
time

class Analysis extends Protocol

description

Analysis is a specialization of protocol corresponding to analysing results of a different experiment with the goal of creating new data products. Analysis can be equated with data mining.

class CrossMatching extends Analysis

description

Cross matching is a special kind of analysis in which two or more source/object catalogues are compared to find common occurences/identifications.

class SourceExtraction extends Analysis

description

Source extraction is a specialization of analysis in which an [image](#) is analysed for the occurrence of sources.

class AstronomicalObservatory extends Protocol

description

This is a baseclass describing how various types of astronomical observations can be performed.

class Simulator extends Protocol

description

A simulator is a special kind of protocol in which using approximations of physical processes are expressed in software allowing approximations of physical systems to be evolved in time.

class Query extends Protocol

description

A query is a special kind of protocol in which data products are retrieved from archives or other data products using specified search criteria. This is not analysis, as no new information is produced as in data mining. Only relations that have been previously recognized can for example be traced in join criteria.

Note that according to this definition cross matching and source extraction are not queries !

abstract class DataProcessing extends Protocol

description

Data processing creates new data products out of pre-existing ones, but the goal is different from data mining in that the result is more to make the data products more amenable to analysis.

class Stacking extends DataProcessing

description

Special kind of data processing that produces multi-color stacks of images, appropriately rebinned etc.

[GL – Domain expertise required ...]

class Callibration extends DataProcessing

description

Special kind of data processing that turns raw, uncalibrated data into calibrated data products.

[GL – Domain expertise required.]

class Objective

description

As the name implies, this class encapsulates the ultimate goal that can be obtained by experiments according to this protocol. However, we do give this an interpretation that possibly differs somewhat from the usual interpretation, which we think is in the eyes of the beholder. We try to model objective in such a way that it might attract the interest of people who do not necessarily have the same scientific goal as the producers of the data.

In this model is is assumed that the objective of an experiment is to [assign values](#) to one or more [properties](#) of a [subject](#) that is being investigated, or simulated, or An objective is the set of information about a given subject and if multiple different subject types are investigated a corresponding collection of different objectives may be defined. An actual experiment may achieve a given objective more than once when different subjects of the same subject type are being determined (i.e. observed or simulated or ...).

An example of the latter is that in this model the objective of an experiment that produces an image is the data corresponding to a single pixel, not the complete image. That is, the subject is a direction in the sky from which photons may have hit the given pixel, not the whole region covered by the complete image.

The pixel represents the smallest connected combination of data units (position+flux+time of observation) that can be described and that someone may be interested in. The image is obviously a collection of pixels, corresponding to a collection of subjects, and we do not imply to say that it is not ultimately the whole collection of pixels that was the goal of the experimenter.

We model the objective in terms of the smallest meaningful components. This component is uniquely defined by the subject and kinds of properties, that is the phenomena that are being measured. This ultimately determines whether another user may be interested in the results of this particular experiment.

Notice for example that to obtain information about some contiguous region in the image, a particular source for example, the experimenter will perform another experiment, according to an analysis protocol, to extract that information. This will be described as a new objective, the type of which will be source for example. Or they can analyse the image as a whole and calculate statistics for certain regions. It is only in this phase that objectives are defined that go away from the single pixel.

class Variable

description

An objective is defined formally as the collection of properties of a given subject type it determines. These properties are connected to the objective using this associative *Variable* class.

This class may contain an indication of whether the variable is dependent or independent. This is still being debated. An alternative is that independent variables should really be part of the configuration parameters. [GL – Needs further discussion.]

class InputDataType

description

Many types of experiments (protocols that is) use existing data products to create new ones. This is particularly true for analysis and data processing. This class encapsulates the *kind* of input data that is being analyzed or processed. An example is a source extractor that requires an image.

references

- type : [SubjectType](#)
The kind of data that is being analysed or processed.

abstract class ConfigurationDescriptor

description

Models things like an observation proposal form for an observatory, or a WSDL or an API for a particular service. It is the abstract description of what information is needed to make the process described by the protocol run. It can be viewed as a Java Interface definition for the protocol. It can be an XML schema that prescribes the kind of valid XML files that can be used as configuration.

5.7 package Experiments

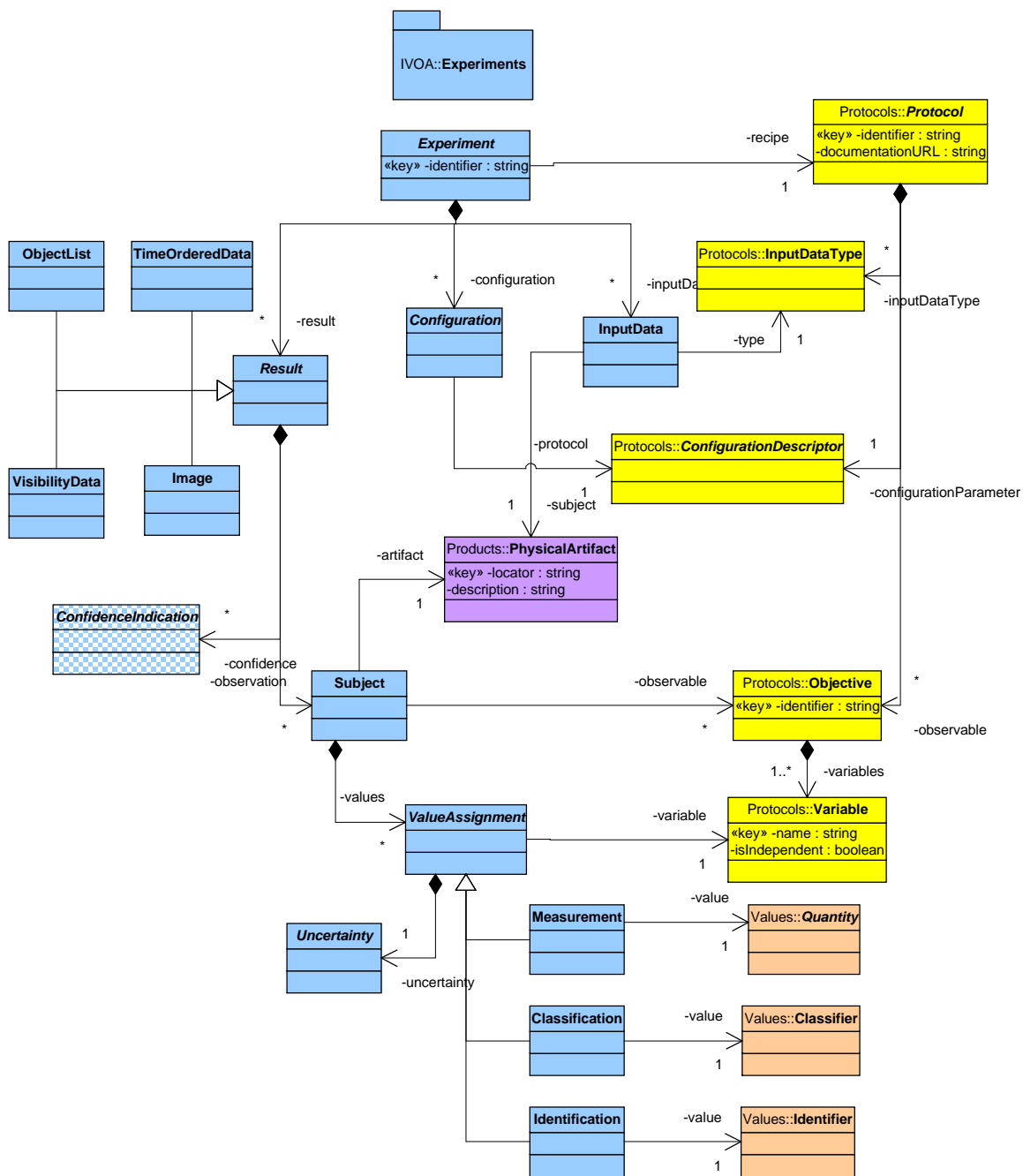


Figure 10: Class diagram for *Experiments* package.

This package contains possibly the central components of the whole data model. It models the actual work that astronomers do and the results of that work. We claim it is central because the ultimate goal of the VO is to provide access to those results in a scientifically proper way.

class Experiment

description

Experiment encapsulates the contact of running, executing an experiment according to a given protocol.

attributes

references

- [Protocol](#) : The protocol according to which the experiment is run.

collections

- [Configuration](#)
- [Result](#)
- [InputData](#)

class Result

description

Result encapsulates a data product that is produced by the experiment containing it. A *Result* is a collection of one or more more fundamental data items which we model using the class [Subject](#), each of which corresponds to a tuple of measurements that are connected in some way. The result is more than the collection of data items though, otherwise we did not need this separate class. It also holds on to measures of confidence that are more complex than can be captured on a data item itself. For example, systematic errors applying to all the measurements in the collection can be placed here. Also correlations between (errors of) individual data items or measurements belong to the *Result*. More concepts applying to the set of data items can in the future be placed here as well.

Ultimately this “connectedness” between the different fundamental data units in a collection will define the result.

attributes

references

collections

- [Subject](#)
- [ConfidenceIndication](#)

class Image extends Result

description

This is a special kind of result that ultimately corresponds to the outcome of a certain kind of astronomical observation (“=” (for example) act of pointing a telescope to a fixed point of the sky and detecting photons on a CCD). Its fundamental data units are those that describe the data that individual pixels on a CCD can deliver. The grouping of pixels in an image defines the result, as well as the particular experiment itself.

Systematic errors pertaining to the image as a whole are the pointing (in)accuracy of the telescope and possible correlated read-out errors of rows of pixels [GL -- I’m shooting blindly here, need input from domain experts to flesh this out in more detail.]

class ObjectList extends Result

description

An object list is a special kind of result corresponding to a for example a catalogue of objects created when doing a source extraction or a cross-match between different catalogues. The subjects correspond to sources and source identifications respectively.

class TimeOrderedData extends Result

description

TimeOrderedData is a special kind of result that corresponds for example to the consecutive pointed observations of a scanning satellite such as ROSAT in its ALI Sky Survey mode, or CMB satellites such as WMAP or Planck. The data can come as an ordered lists of photons, or as a list of voltages, possibly together with a list of pointings containing satellite attitude information.

[GL – Need more domain expertise.]

class VisibilityData extends Result

description

[GL -- Don't know much about this kind of data, but it seems to be relevant to for example radio synthesis observations. Peter Lamb is in charge of modeling this subject.]

abstract class ValueAssignment

description

This is the generalization of the concept of measurement to include assigning values to properties/phenomena that are not explicitly numerical. can not be “measured” in the traditional way.

attributes

references

- variable : [Variable](#)
The variable defined on a protocol to which the value is assigned. Through the variable the value is associated to a particular [property](#) and through that ultimately to a [phenomenon](#).
- uncertainty : [Uncertainty](#)
Some indication into the uncertainty associated to this particular value assignment. This could be called error or confidence and pertains to that part of the uncertainty of a value that is associated to this value assignment alone. This is separate from any other uncertainties associated to the result such as systematic or correlated errors.

class Measurement extends ValueAssignment

description

Measurement encapsulates the concept of assigning a value corresponding to a [numeric phenomenon](#). In actual fact it will assign a [Quantity](#), which is the subclass of [Value](#) corresponding to numeric phenomena.

references

- value : [Quantity](#)
The numerical value that is assigned to the variable reference.
Note. It would be nice to have a more explicit graphical way of constraining members of a baseclass in UML such as for example is possible in ORM [8]. In that case a value reference to [Value](#) could be placed on [ValueAssignment](#), and the current reference could restrict this reference to Value's subclass Quantity.

class Classification extends ValueAssignment

description

Classification assigns a value corresponding to a [categorical phenomenon](#). In actual fact it will assign a [Classifier](#), which is the subclass of [Value](#) corresponding to categorical phenomena.

references

- value : [Classifier](#)
The categorical value that is assigned to the reference *variable* on [Value](#).
Note. See the Note on [Measurement](#).

class Identification extends ValueAssignment

description

This class is a specialization of ValueAssignment that corresponds to the identification of a given source with another source, possibly identified using a Name in a standard catalogue.

[GL – In the model we only assign a name, we may want to generalize this to a pointer to a data item in another catalogue (a PhysicalArtefact).]

abstract class Uncertainty

description

The uncertainty that is associated to a single value assignment. This class may take different forms for different kinds of assignments such as measurement, where a single quantity may suffice, or for classification, where a probability for each possible category may have to be provided.

[GL – This class should be fleshed out further, similar to ConfidenceIndication].

abstract class ConfidenceIndication

description

This is a very loosely specified class that is used to encapsulate any way in which errors, uncertainty, likelihood, confidence etc can be specified concerning one or more value assignments.

abstract class Configuration

description

Configuration contains the parameters settings that govern *how* the experiment was executed. Which parameters can be set is determined by the configuration descriptor of the protocol that was followed in this experiment. This class must be modeled in more detail. The following comparison describes this quite accurately. If the configuration descriptor is defined using an XML schema, the configuration is an XML document that is valid according to that schema.

reference

- configuration : [ConfigurationParameters](#)

class InputData

description

InputData indicates which physical data product(s) were used in the experiment. These data must be instances of the input data type defined under the protocol that was followed in the experiment.

references

- type : [InputDataType](#)
- ...
- subject : [PhysicalArtefact](#)
The actual, localizable physical artefact that was used in the experiment.

class Subject (Observation?)

description

This class encapsulates a single subject for which properties have been determined. A subject is ultimately a tuple of values that corresponds to a single objective of the

protocol that was followed. The experiment can produce multiple subjects for a given objective.

6 Example Instances

- *meter* is a unit for the BaseNumericPhenomenon *length*
- *Sextractor* is a SourceExtraction (subclass of Protocol), whose aim it is to analyse input data of type Image, which is a ScientificArtefact which is a SubjectType. Objective is a *source*, with particular variables.
A SourceExtractor can be said to do two consecutive tasks, first to decompose the image in terms of sources and then to determine properties of the sources.
- An AstronomicalObservation is a Protocol that aims to collect photons from a particular part of the sky. The Subject is the SkyProperties collection (or a SkyRegion), the property is a *luminousIntensity* from a region on the sky. This is the raw information.
- The ROSAT Bright Source Catalogue is a Result of the Experiment to extract sources (the Protocol) from the ROSAT fields (I think) that were themselves the result of The Protocol is described by the published paper (Voges et al). Its Objective was a collection of Sources. A Source has a position, error in position, certain luminousIntensity measures at different wavelengths with their errors, certain flags etc.
-

7 XML Binding

During the IVOA interoperability meeting in Cambridge (2003) we decided that the roadmap towards general acceptance of a data model within the IVOA was as follows:

- whitepaper containing English description of the concepts being modeled.
- A UML diagram with a model design.
- An XML Schema implementation of the model.
- A reference implementation

Here we describe our thoughts concerning the binding of our model as an XML Schema.

7.1 purpose and method

The goal is *not* to describe the UML diagram in XML. For this the XML specification has been invented.

The goal we have set us is to define a schema that supports interchange of astronomical data within the VO and possibly between the VO and clients using XML documents. We therefore wish to design a schema that allows documents to be written using XML elements and attributes that somehow represent the concepts of the VO's universe of discourse and their interrelations.

It will be clear that there can be many different designs for this. The work in this chapter merely gives a reference binding. This binding will not be optimized for particular applications or subsets of the model. Instead the mapping is constrained to be as close as possible to the actual model, to have as little of an impedance mismatch as we can achieve.

There is some literature on the mapping from UML to XML Schema [GL – References !].

Some of these have the goal to define a way to describe an XML Schema using UML diagrams. Our goal is opposite to this. We want a schema that describes our model. These references almost invariably (have to) extend the UML syntax with stereotypes etc to be able to represent all the different syntactic elements that may occur in an XSD document. We take the opposite approach, for example by limiting ourselves to a subset of the elements available in XSD. For example, we will not use the substitutionGroup concept for mapping inheritance. Instead we rely solely on the extension and restriction mechanisms for complexType and simpleType.

7.2 result

8 Example

8.1 RASS BSC <-> CVO model

9 FAQ

9.1 What is the domain model for ?

For defining what meta-data must be provided to publish an archival dataproduct in the VO.

9.2 How can the domain model be used ?

As a meta-data repository to be queried by registries or VOQL.

As an XML Schema in combination with an XSLT transformation that expresses XML documents in terms of the domain model XSD.

9.3 Can other datamodels be used as well ?

Yes, as long as they can be mapped to the domain model.

9.4 How can we compare different datamodels ?

By mapping each to the domain model.

9.5 How can we map a data model to the domain model ?

One can use XSLT to map between schemas.

9.6 How can a VOTable be mapped to the domain model ?

Using a VOTable2DM.xslt file that has to be written.

10 References

- [1] Martin Fowler 1997, *Analysis Patterns*, Addison Wesley
- [2] SI specification on <http://physics.nist.gov/cuu/Units/introduction.html> .

- [3] Arnold Rots, *Space-time metadata*, <http://www.ivoa.net/forum/dm/0117.htm>
- [4] Brian Thomas & Ed Shaya, quantity models ...
- [5] Tom Verhoeff, quotes from Fowler's book
<http://www.wpa.win.tue.nl/wstomv/quotes/analysis-patterns.html>.
- [6] Fu-Chiu John Cheng, *Object-Oriented Design & Analysis, chapter 9*,
<http://www.cse.ttu.edu.tw/~cheng/courses/ooad.htm/F02/ooad09.ppt>.
After book by David William Brown, *Introduction to Object-Oriented Analysis: Objects and UML in Plain English*, Wiley, 2001.
- [7] Bertrand Meyer, 1997, *Object-oriented software construction, second edition*, Prentice-Hall
- [8] Terry Halpin, 2001, *Information Modeling and Relational Databases*, Morgan Kaufman.
- [9] Grady Booch, 1994, *Object-oriented Analysis and Design, second edition*, Addison-Wesley
- [10] Tim Berners-Lee, James Hendler and Ora Lassila, 2001, [The Semantic Web](#), Scientific American
- [11] IVOA Quantity data modeling resources
<http://www.ivoa.net/twiki/bin/view/IVOA/IVOADMQuantityWP>
- [12]

Appendix A SI specification

In the introduction to the SI specification in [2] there is a useful discussion relevant to the discussion about quantity. In particular the following quote has influenced our thinking and has found its place in the model.

Begin quote:"

Some useful definitions

*A **quantity in the general sense** is a property ascribed to phenomena, bodies, or substances that can be quantified for, or assigned to, a particular phenomenon, body, or substance. Examples are mass and electric charge.*

*A **quantity in the particular sense** is a quantifiable or assignable property ascribed to a particular phenomenon, body, or substance. Examples are the mass of the moon and the electric charge of the proton.*

*A **physical quantity** is a quantity that can be used in the mathematical equations of science and technology.*

*A **unit** is a particular physical quantity, defined and adopted by convention, with which other particular quantities of the same kind are compared to express their value.*

*The **value of a physical quantity** is the quantitative expression of a particular physical quantity as the product of a number and a unit, the*

number being its numerical value. Thus, the numerical value of a particular physical quantity depends on the unit in which it is expressed.

For example, the value of the height h_W of the Washington Monument is $h_W = 169\text{ m} = 555\text{ ft}$. Here h_W is the physical quantity, its value expressed in the unit "meter," unit symbol m , is 169 m , and its numerical value when expressed in meters is 169 . However, the value of h_W expressed in the unit "foot," symbol ft , is 555 ft , and its numerical value when expressed in feet is 555 .

End quote.

In our model we have modeled all of these terms explicitly. In particular, the "quantity in the general sense" is the motivation behind our [Phenomenon](#) class, whereas it is explicitly represented as the [AtomicNumericPhenomenon](#). The "phenomena, bodies, or substances" to which the quantities in the general sense are ascribed we model in the [SubjectType](#) class. The "quantity in the particular sense" is our [Property](#) class, which is really the association (*ascribing*) of the phenomenon to the "body or substance".

The physical quantity is loosely represented as our [Representation](#) class, which indicates *how* the quantity can be expressed. The unit is our [Unit](#) class. We have not modeled how the unit is defined in terms of a particular physical quantity, as this falls outside our modeling domain, though it is not hard to see how this could be done (references to [PhysicalArtifact](#) and [Property](#) for example). Instead our units are defined as reference data in the [Standards](#) package.

The definition of "value of a physical quantity" above finally is really the concept we attempted to model in our [Quantity](#) class. It is a value that can be ascribed to a Phenomenon. We did generalize upon the SI model in a few ways. First we assume the existence of more complex numerical/quantitative phenomena, things such as *shape*, *position*. Consequently we allow for quantities that are more complex than a simple value+unit. Therefore the "value of a physical quantity" is really represented by the subclass of Quantity, [AtomicQuantity](#).

A further generalization to this SI specification is ofcourse that we allow for phenomena that are not expressed as numerical quantities, but things such as classification and identification. Part of the motivation for this is described in Appendix B.

Appendix B Fowler

Another motivation for some of the design choices came from the book by Fowler [1]. Fowler discusses what he calls *Analysis Patterns*, which are OO modeling patterns to be used during the analysis phase of conceptual modeling. His chapter 3 deals with *Observations and Measurements*. It discusses a *Quantity* which corresponds to an *amount* and a *Unit*, i.e. very similar to our [AtomicQuantity](#). His motivation for this is the same as we all had, that we need to associate units to numbers to make sense of them and to allow conversion (transformations) between different units.

He discusses units somewhat further, in particular compound units and how these can be expressed in terms of more fundamental ones. This is equivalent to our [CompoundUnit](#).

Fowler then discusses a *Measurement*, which is the assignment of a *Quantity* to what he calls a *PhenomenonType*, as applied to a *Person*. His *PhenomenonType* is the origin of our introducing and naming [Phenomenon](#), though his particular example corresponds more closely to our [AtomicNumericPhenomenon](#). His *Person* (he gives examples from the medical profession) is our [SubjectType](#). In actual fact of course we associate values to *Properties*, which is a *Phenomenon* that has been defined as being applicable to a *SubjectType*. Fowler does not have that distinction, as he does not have to worry about the definition of the subject.

Fowler's *Measurement* is directly equivalent to our [Measurement](#). An interesting aspect of his work is that he also generalizes *Measurement* to *Observation* which also includes *Category Observation*. Our generalization is to [ValueAssignment](#), a very ugly name, but the name "observation" is in danger of being overloaded already too much in our model. His *CategoryObservation* we call [Classification](#) and we introduce [Identification](#) as a further type of *ValueAssignment*.

Another important aspect of our model is also indirectly derived from Fowler's *Observation* pattern, namely our [Protocol](#). He introduces this concept as a way of describing the way in which observations were made. We use his name for describing the way in which an [Experiment](#) can be run/executed/performed. The actual running of the experiment, which includes making observations or generalizations of this, is described by providing values to the various [configuration parameters](#) etc described in the protocol. In Fowler's conception, it is in the protocol that one describes accuracy and sensitivity of the actual measurements, very similar to the way we describe these as part of the *Result* and the *Measurement*, *not* as part of the *Quantity*.

A final interesting discussion that has some relevance for our separating the running of the experiment (in *Experiment*) and the description of how to do it (the *Protocol*) is given in his chapter 3.12. There Fowler discusses different ways of looking onto the concept of *Observation*. He sees various subtypes of this corresponding to *Hypothesis*, *Projection* and *ActiveObservation*. In his example from the medical profession a doctor might *Hypothesize* that an observation of a certain phenomenon might produce a certain result based on other symptoms a patient may have. It may be valuable to store these to be compared to the actual (active) observation that is made later.

A similar concept of *hypothesis* is of course a very important in doing science as well. Often the objective of a certain experiment is to test a certain hypothesis, which might be modeled as a prediction of the result of the experiment when certain input parameters etc are met. We have not modeled this in any detail yet.