International

Virtual

Observatory

Alliance

Proposal for a *Simulation Database* Standard

Version 0.5

Note 2009 May ...

This version:
0.5-20090504
Latest version:
http://www.ivoa.net/Documents/latest/NOWHERE
Previous version(s):

Author(s):
Gerard Lemson
Hervé Wozniak
Laurent Bourgès
Rick Wagner
Claudio Gheller

Abstract

We propose a standard protocol for a “Simulation Database (SimDB)”. A SimDB implementation is a web service that gives access to a databasde containing metadata describing simulations. SimDB has been developed in the IVOA Theory Interest Group with assistance of representatives of relevant working groups. Whilst the theory *interest* group can not create working drafts and

promote documents onto the recommendation track, we feel this document has sufficient detail that this can be done in short order.
This Note gives an overview of SimDB, and contains the MUSTs , SHOULDs etc, but leaves many of the details to other formal and normative documents, in particular where it concerns the data model and its representations that forms the core of the proposal.

Status of This Document

This is a Note. The first release of this document was 2009 …. This is a preparation of a specification document, but, coming from the theory interest group, can not be a working draft on the recommendation track. Because of this we have taken the liberty to add hints here and there in the document to working groups with suggestions or requests for feedback. In the final specification these must be taken out.

This is an IVOA Note expressing suggestions from and opinions of the authors. It is intended to share best practices, possible approaches, or other perspectives on interoperability with the Virtual Observatory. It should not be referenced or otherwise interpreted as a standard specification.

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

Acknowledgements

...

Contents

1 Introduction

We here describe a proposal for an IVOA standard, the *Simulation Database* (SimDB). Work on this standard started under the header *Simple Numerical Access Protocol* (SNAP) in the theory interest group since the Victoria interoperability meeting, 2006. Recent developments have made us decide to split SNAP in two separate tracks, SimDB and SimDAP, which stands for *Simulation Data Access Protocol*. The main part of this *Note* deals with SimDB only: what it is supposed to be, what its current state is, and what further work is needed and the possible organisation of that work.

Work on this specification has been organised via a googlecode SVN repository in the *volute* project originally created by Norman Gray for the Semantics Working group. The SimDB project in particular can be followed under https://volute.googlecode.com/svn/trunk/projects/theory/snapdm . In particular follow the current document is available from https://volute.googlecode.com/svn/trunk/projects/theory/snapdm/doc/note.

SimDAP deals with extracting subsets of data from simulation results that have been discovered through a SimDB. Its dependence on SimDB is through its dependence on aspects of the SimDB *data model*. The progress of the SimDAP specification can be traced also via the googlecode volute project: https://volute.googlecode.com/svn/trunk/projects/theory/snap (see also [3]).

We think the SimDB effort has evolved far enough that it can be moved onto the recommendation track. After summarising its main features, we identify some issues related to this next step, namely that it is hard to find a single working group that could be responsible for tracking all aspects of the specification. We provide a number of possible solutions to this, but leave it to the TCG and the Exec to decide on this. In the mean time the current group of developers will keep working on this, with the goal of having working prototype/reference implementations available by the October 2008 interop.

The organisation of thie Note is as follows. First we give an “executive” summary of the proposal. Then we describe the data model on which the specification is based. There we address explicitly some “meta-meta-design” issues for this specification: how do we go about designing the abstract (i.e. implementation neutral) data model, and how do we use it to derive usable representations of it.

We also explicitly state our position on using the results of prior standardisation efforts of relevance, both from the data model and other working groups. Then we define the protocol itself. This has two parts, one which we will callSimDB/TAP, which is based on a relational database representation of the data model. The other we will call SimDB/REST and is based on an XML representation of the model.

We end by describing some prototype implementations.

2 Summary

SimDB is a protocol describing an online web service providing access to a database that contains *metadata* describing numerical computer simulations of astrophysical systems and related resources. The protocol defines the structure of this database and how queries can be sent to it. We do this by defining SimDb as a kind of specialised TAP/ADQL service.
We also define a serialisation to XML of specific types of resources in the database. These can be used in a registration protocol and for retrieving full descriptions of identified resources using simple, non-ADQL requests.

For this specification we follow the original SNAP idea, which limited its focus to simulations that produce a representation of 3+1D space, and to various post-processing products of a similar nature (we will assume to include the latter when referring to “simulations” in the rest of this note. In later versions this restriction on the type of simulations is likely to be relaxed.

1. The structure of the SimDB is based on a (logical) data model, fully specified in UML2. This data model is part of the specification, though it has no normative aspects for the protocol. It instead defines the concepts that we use when discussing the structure of the SimDB database, without referring to specific implementations.
2. From the UML data model we derive[1] physical representations for use in their respective SimDB service contexts:
    a. A relational database schema expressed according to the TAP specification.
    b. An XML schema, defining valid XML documents containing SimDB meta data descriptions for use in messaging.
    c. A set of UTYPEs identifying elements of the model in case this model is to be expressed in VOTables or other non-SimDB-standard representations. One context for these is as the results of the ADQL queries.

[1] For SimDB this derivation relies on standardized set of rules. We believe it us ultimately the domain of the DM working group to come up with such rules.

    d. A human readable HTML document describing all the individual model elements in detail.
3. These physical representations are to be used in the service interface specification of SimDB instances. These are
    a. A TAP/ADQL-based querying of the metadata repository as a relational database. SimDB *is a* TAP service, mandating /sync ADQL only and fixing the data model
    b. A RESTful web service interface, using standard HTTP methods (GET, PUT, POST, DELETE, etc.) to provide mechanisms for maintaining the actual entries in a SimDB. SimDB resources would be uploaded, retrieved, or modified using the XML format defined by the XML Schema derived from the data model.
    c. **@@TBD needs discussion@@** An OAI-PMH compliant publishing interface, to allow harvesting of SimDB records. Like the Registry, this permits SimDB instances to acquire records published in other SimDBs.

The protocol aspects will be fully defined in this Note, but often references will be made to external documents containing details. For example the data models and their representations are defined in external documents. Whilst under development the most recent versions of these can be found under the Volute GoogleCode[2] project, more precisely in the SVN repository under theory/snapdm[3] (we will refer to this repository as “Volute” in the rest of this Note).. The precise locations of documents will be given where required. Eventually the documents will be uploaded to the IVOA Wiki.

This concludes the overview of the normative/standard aspects of the SimDB as far as it has currently been designed. We do want to mention the following aspects of the SimDB effort that will simplify much of the further work:

1. Under Volute we have XSLT scripts that derive the physical models directly from the UML model according to predefined mapping rules.
2. We also derive Java classes with JPA and JAXB annotations to make it easy to implement a SimDB from the specification.
3. From this we are developing a full SimDB implementation using code generation from the UML model only.
4. We propose an implementation path to transform an existing, “legacy” relational database containing simulation metadata to the SimDB specification.

The following groups have committed to creating a reference implementation of a SimDB, some of these are meant to be ready by the October 2008 interoperability meeting.

- GAVO (Gerard Lemson)

[2] http://code.google.com/p/volute/
[3] http://code.google.com/p/volute/source/browse/#svn/trunk/projects/theory/snapdm

- UCSD (Rick Wagner)
- INAF/Trieste (Patrizia Manzato) @@ **TBD correct? ITVO maybe?** @@
- INAF/Catania (Ugo Becciani) @@ **TBD correct? ITVO maybe?** @@
- VO-France (Franck LePetit, Laurent Bourges)

3 SimDB data model: design overview (descriptive)

SimDB is a protocol giving access to a database. The structure of that database is prescribed by a data model. This data model is represented in different ways in different parts of the SimDB protocol. For example TAP requires a relational data model specified according to the metadata standards defined in the TAP protocol [6]. We define UTYPEs for linking to the elements of the data model, and we specify a set of XML schemas that together define the serialisation of particular objects in the data model to XML.

These representations are all derived from an implementation-neutral version of the data model. *That* specification is fully defined in UML and is what we will refer to as the *SimDB data model*, or SimDB/DM. The UML diagram will ultimately be uploaded to the appropriate place on the IVOA Wiki. Until then it is available on the Volute repository (see [4]) as an XMI [10] document.

A possible problem is that one needs the UML drawing tool MagicDraw[4] (community version 12.1 is used) to read that file. Though the DM working group has mandated that data models are to be represented as UML (interop, Cambridge 2003), there is no agreement on a tool or standard representation of such documents. And though XMI is supposed to provide a tool neutral representation, we have not checked that other tools can read the diagrams. JPG and similar representations will be made, but these are not complete. In particular documentation of the individual elements is not visible.

To partially remedy this situation we provide an HTML description of the full model in [5]. From this the model one could in principle draw the UML from scratch for it contains all relevant information. This HTML representation also contains the UTYPEs for each of the elements in the model and is part of the normative specification of SimDB (see below).

We now describe the main structure of the UML model, detailing some aspects that have provoked discussions in the past such as the use of characterisation-like elements and registry-like resources and curation. In Appendix A we describe the UML syntax we have used.

3.1 "20 questions"

We have followed the approach to data modelling suggested in [9] and applied there to the design of the SDSS database for the SkyServer web site. According to that approach one tries to gather O(20) science questions that a system should be able to answer and one designs the system in such a way that this becomes possible. We have polled some scientists with the question that if they

[4] http://www.magicdraw.com/

were presented with a database of simulation metadata, what questions would they want to ask it to find interesting simulations. The following list gives some of the answers:

- What system/object is being simulated?
- What physical processes are included?
- How is the system being represented in the simulation (particles (Langrangian), (adaptive) mesh (Eulerian)), both, other?
- How are the physical processes implemented ?
- Characterise the numerical approximations (.e.g. resolution, softening parameter)
- What observables are available for the system/object, possibly as function of time[5]? As it is a spatial system, at least *simulation box*size, center-of-mass position.
- What observables are available for the constituents, i.e. what is the schema of the objects from which the simulation built. E.g. particles in N-body simulation, grid cells in an adaptive mesh simulation, particle groups in a cluster finder?
- Per snapshot, per simulation object type, per variable:
  - Characterise the *possible* values
  - Characterise the result
- Are post-processing results available?
- Are services/applications available for accessing the results?
- Which code ran the simulation?
  - Which *version* of the code ?
  - Is software available?
- Who ran the simulations*?*
- What were values of input parameters?
- How were initial conditions created?

3.2 Usage scenario

3.2.1 Millennium Run, simulation, post-processing and publication

The *VIRGO consortium* ran the **simulation**. For this they used a particular version of the **simulation code** *Gadget*. This simulation code needs a certain number of **input parameters** to be set. The simulation code approximates **physical processes** (Gadget supports *gravity* and *hydrodynamics*, *star formation*, *black hole formation* etc) using **numerical algorithms** (*TreePM*, *SPH*). The simulation code allows one to choose which physical processes one want to include (for Millennium only *gravity*). The processes act on **simulation objects** (here *point particles*) that on their own, or in aggregate represent **real world objects**. (here *Large Scale Structure*) The simulation objects have a number of properties (here [*id, x, y, z, vx, vy, vz, mass*]) that are evolved forward

[5] Re: Rick Wagner's example of certain properties only being calculated after a certain stage in the simulation is reached.

in time by the simulation code. The code allows me to choose which properties I want to calculate.
The simulation produces **results** that consist of collections of the simulation objects, one for each type of simulation object that is included. In this simulation each result corresponds to a **snapshot** taken at a particular time. The results are stored in multiple **files** (512 per snapshot). These files have a complex **format**, the particles are ordered using an index based on a space-filling curve, and the files contain hash tables that act as indexes to optimise retrieval of spatial sub volumes.
The Millennium simulation contained about 10 billion particles in a box of size 500Mpc/h. It had 64 snapshots, including the initial conditions which were calculated using a separate code. Each snapshot was about 400GB.

The snapshots produced by the simulation were used as input for various types of **post-processing**. For example a **cluster finder** using the *friends-of-friends* (FOF) algorithm was run on each of the snapshots. This produces *FOF groups*, which are *dark matter halos* with more complex properties (*radius*, various types of *mass*, *velocity dispersion* etc). The results are stored again in 512 files per snapshot.
These files with FOF groups are used as input for the *SUBFIND* code which detects substructures in the FOF groups, which correspond to bound “sub-halos”. These sub-halos evolve in time by merging with each other. The resulting merger trees are detected using a further post-processing algorithm which uses the particle data and the sub-halo files.
These merger trees are the input for a simulation code that implements galaxy formation using so called semi-analytical algorithms. These algorithms model physical processes such as gas cooling. star formation and evolution, supernovae feedback etc.
All the post-processing results are also stored in a **relational database**, which is published through a **web application** that allows users to submit SQL queries to the database.

@@TBD some more examples@@

3.3 *Domain model*

The SimDB data model (SimDB/DM form now on) is inspired by the *Domain model for Astronomy* proposed in [7]. We will explain the logic behind that and the current model using a restricted version of the domain model, adjusted for the current domain, that of theoretical simulations. The model is shown in Figure 1. In that figure elements in orange are represented also in the SimDB data model, possibly with a different name. The purple classes are not part of SimDB/DM, but are used to explain other features.

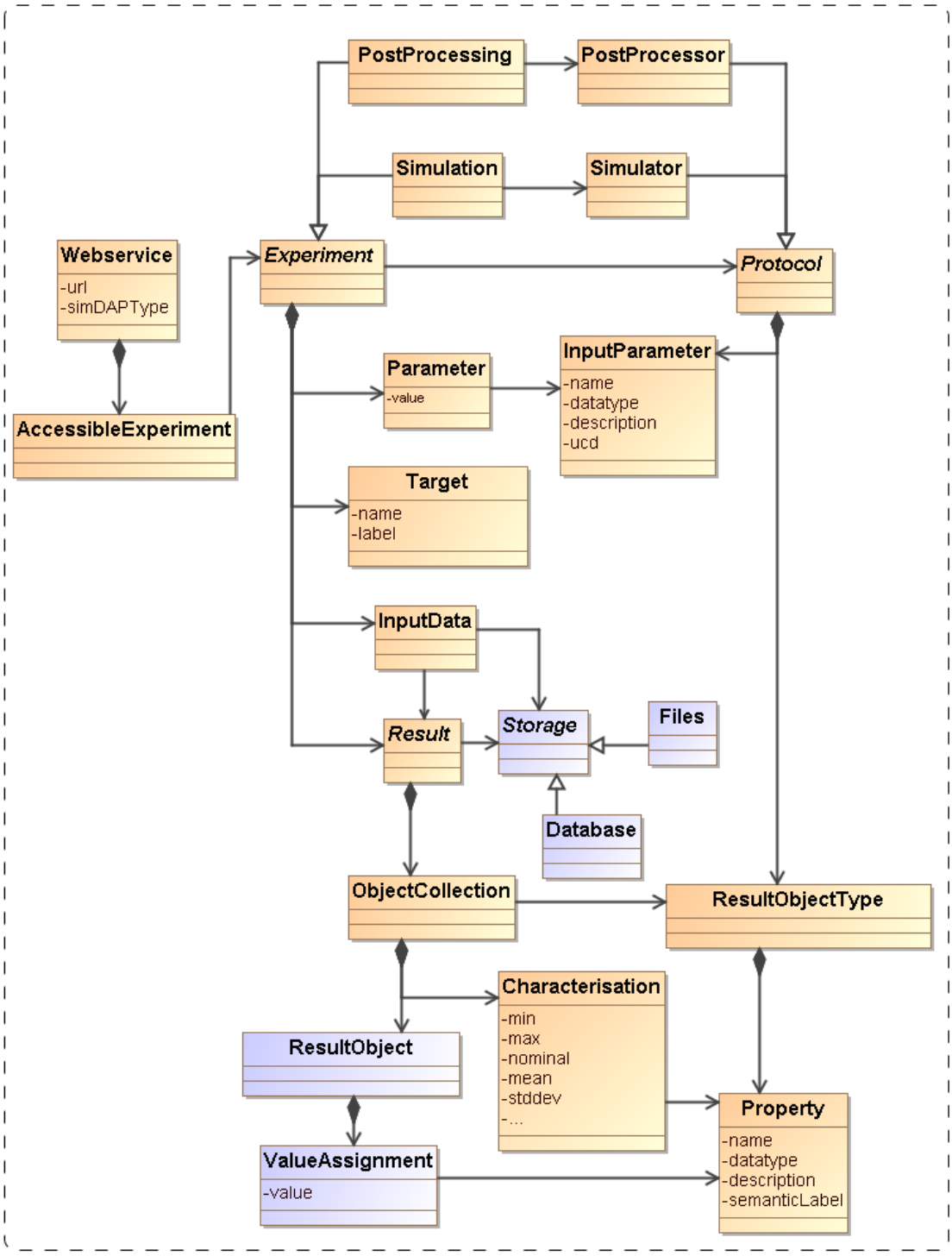

Figure 1 Schematic model used to explain the logic of the SimDB model and the domain model it is based on.

We start the description by assuming the existence of one or more **Files** that a publisher thinks may be of interest to the community because they contain astronomical data. Instead of in files the data might also reside in a **Database**,

and to be generic we introduce a **Storage** base class that abstracts the actual physical location of the data.
Registering that files exist somewhere is not of great interest without providing information about the *contents* of the files. The philosophy that we follow is that the files are of potential interest because they contain the **Results**[6] of an (astronomical) **Experiment**, and accordingly their contents must be explained by describing the experiment that gave rise to it. Only in this way can one make scientific use of the files or other storage resources.
The abstract **Experiment** is made concrete by adding some examples of experiment types that are important for the current model dealing with **Simulations** and simulation **PostProcessing**.
In our model, **Experiment** represents the actual *running* of an experiment; to describe the *design* of the experiment we introduce the concept of **Protocol**. This separation between design of experiment and the execution is a *normalisation* that reduces redundancy in the model. See section 4.5.1 below for a discussion of this concept. We mirror the concrete subclasses of **Experiment** by adding concrete subclasses to **Protocol** such as **Simulator**, which represents simulation codes according to which **Simulations** are run, and **PostProcessor** corresponding to **PostProcessing** runs.
The **Protocol** class contains **InputParameters**. An **Experiment** using a particular **Protocol** only needs to indicate the *values* for these parameters. In this way a single instance of the **Protocol** can be reused by many **Experiments** performed according to it.
The **Protocol** also defines the possible structure of the results of the experiments. In our model **Results** contain **ResultObjects.** These objects have a given type, represented by the **ResultObjectType** contained by **Protocol**. The **ResultObjectType** defines the **Properties** that these objects have.
For example the results of N-body simulations may contain particles having properties position, velocity, mass and possibly othes. Adaptive Mesh Refinement (AMR) simulations produce results that are collections of mesh cells of various sizes, positions and contents. Similarly post-processing codes such as halo finders produce “halos” and “semi-analytical” galaxy formation codes produce galaxies.
In general a single result can contain objects of different types. For example a Smooth Particle Hydrodynamics (SPH) simulation may contain dark matter particles, star particles and gas particles. And in general the codes allow one to configure which of these exactly are chosen in a given experiment.

[6] We do not assume that in reality the relation between the conceptual Result and the concrete Storage elements can be modelled by a single reference. Especially for the largely non-standardised world of simulations a single result can be distributed over many files, but it is also possible for one file to contain multiple results. In the current SimDB model we do not attempt to model such relations explicitly. We delegate the responsibility for accessing the physical results to (web) services and this issue is more explicitly addressed by the SimDAP protocol.

One aspect of the experiment that is not determined by the protocol is *why* the experiment was performed. In the model we introduce the **Target** concept for this, which represents real world objects or processes that are being simulated. For example, with the same N-body simulator one may simulate a galaxy merger or the evolution of large scale structure of the universe.
As discussed above, the actual way in which results are stored in files or databases is hard, if not impossible to model. Instead we assume that **Webservices** of various kind may be used to access the results of simulations and other SimDB products.
Some of these will be standardised in the SimDAP specification, but custom services may also be introduced. The model allows one to describe the experiments and their results, which should allow users to discover results of interest, after which the web services can be called for actually accessing these.

4 SimDB/DM: details (normative)

The actual SimDB follows the logic explained in the previous section, filling in details and adding some more concepts. A single image of the model is too large to fit in a readable fashion in this Note. We will provide cut-outs of relevant pieces in the following sections. For full details we again refer to the UML model in [4] and the HTML documentation in [5].

The capitalised keywords “MUST”, “REQUIRED”, “SHOULD”, and “MAY” as used in this document are to be interpreted as described in the W3C specifications (IETF RFC 2119 [13]). Mandatory interface elements are indicated as MUST, recommended interface elements as SHOULD, and optional interface elements as MAY or simply as a non-capitalised “may”.

4.1 Packages

We subdivide the model in packages (see 9.2A.3). These subdivide the model in subsets of classes and data types that belong together. This subdivision is generally based on collection hierarchies. We use this subdivision to show the model in detail and give a short description of each package. See for the packages and their dependencies.

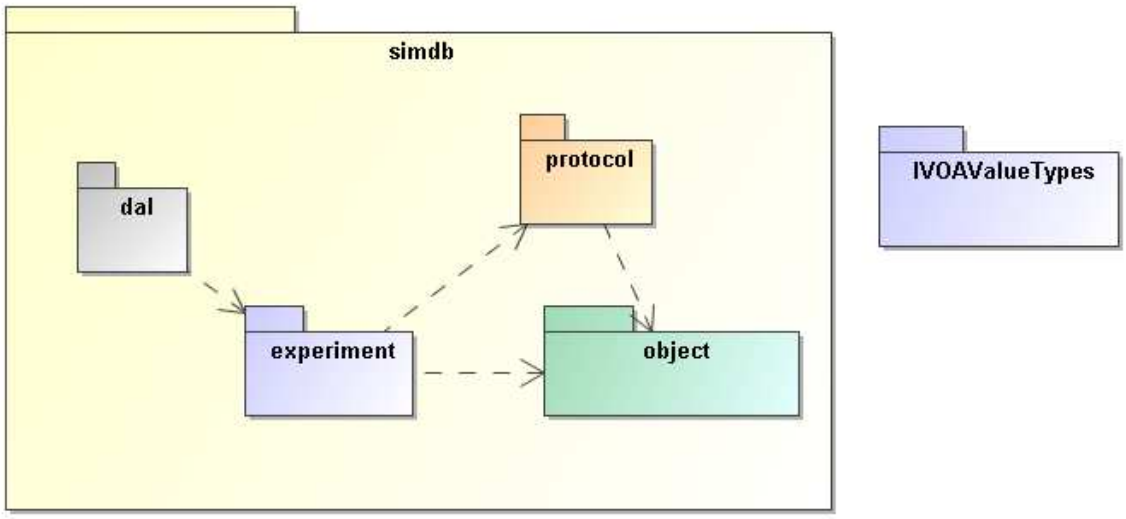

Figure 2 The packages of the SimDB data model and their relationships. The IVOAValueTypes package is defined in the formal UML profile and contains the predefined primitive types (see).

4.1.1 simdb

This is the root package of the SimDB data model. It contains the *Resource* Class (see below), which plays the same role as the Resource class in the Registry Resource model []. Only Resources can be directly registered in a SimDB, other types of objects can only occur in the context of a Resource object’s hierarchy.

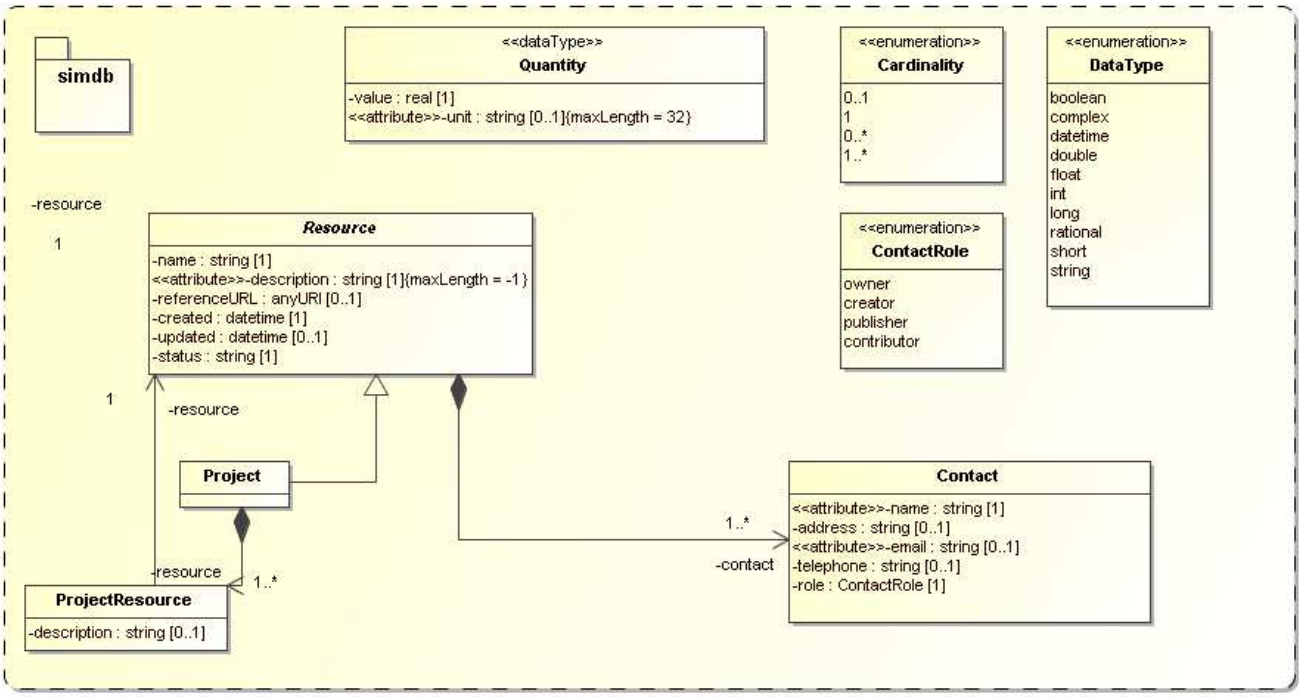

Figure 3 The "simdb" package.

4.1.2 object

This package contains Class definitions that describe objects and their properties. It is used to describe objects in simulations, both the ones that are used, such as the particles in N-body simulations, but also astronomical objects that are the result or target of the experiments.

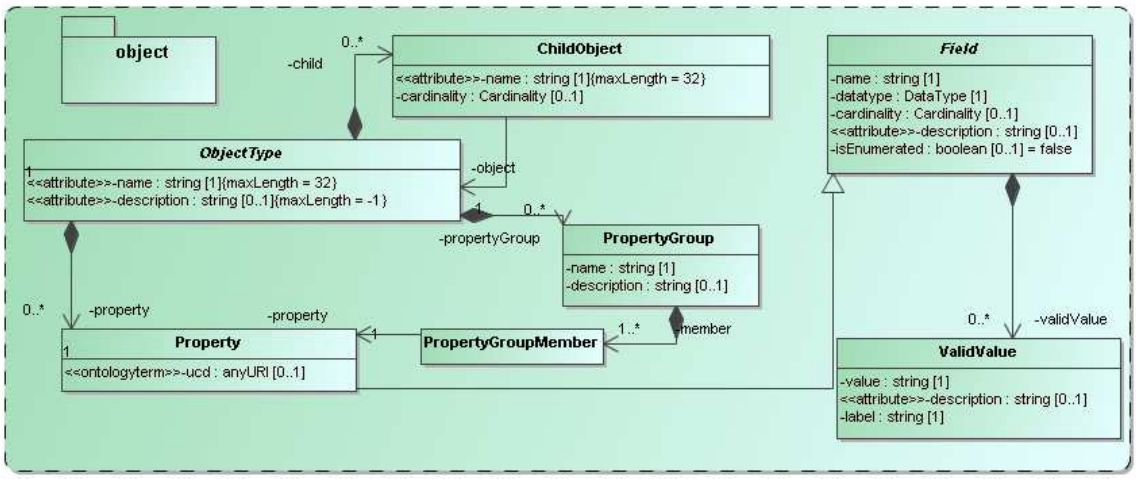

Figure 4 The "objects" package.

4.1.3 protocol

The protocol package contains the *Protocol* class and subclasses such as *Simulator* and *PostProcessor*. The package furthermore contains classes that describe features of these resources such as *InputParameter*, *RepresentationObjectType* or *Algorithm*. See below for more details about some of these classes.

(The image for this package is too large to fit easily on a page so we refer to the online resources.)

4.1.4 experiment

The experiment package contains the *Experiment* class at its root and subclasses of it such as *Simulation* and *PostProcessing*. It also contains classes that describe further features such as parameter settings and in particular the *Snapshot* class which represents the results of the experiments. In later sections we describe some more details of these classes.

(The image for this package is too large to fit easily on a page so we refer to the online resources.)

4.1.5 dal

This package, named after the Data Access Layer working group, contains a few classes that aim to model web services that give access to the results of experiments. As mentioned above we do not explicitly model how these results are stored, but defer such considerations to the actual services that allow download and/or manipulation of these resources. These services may implement protocols such as SimDAP, but they may also be custom services. The main feature that is important for the SimDB is that for a given experimental result one should be able to find services that allow their retrieval.

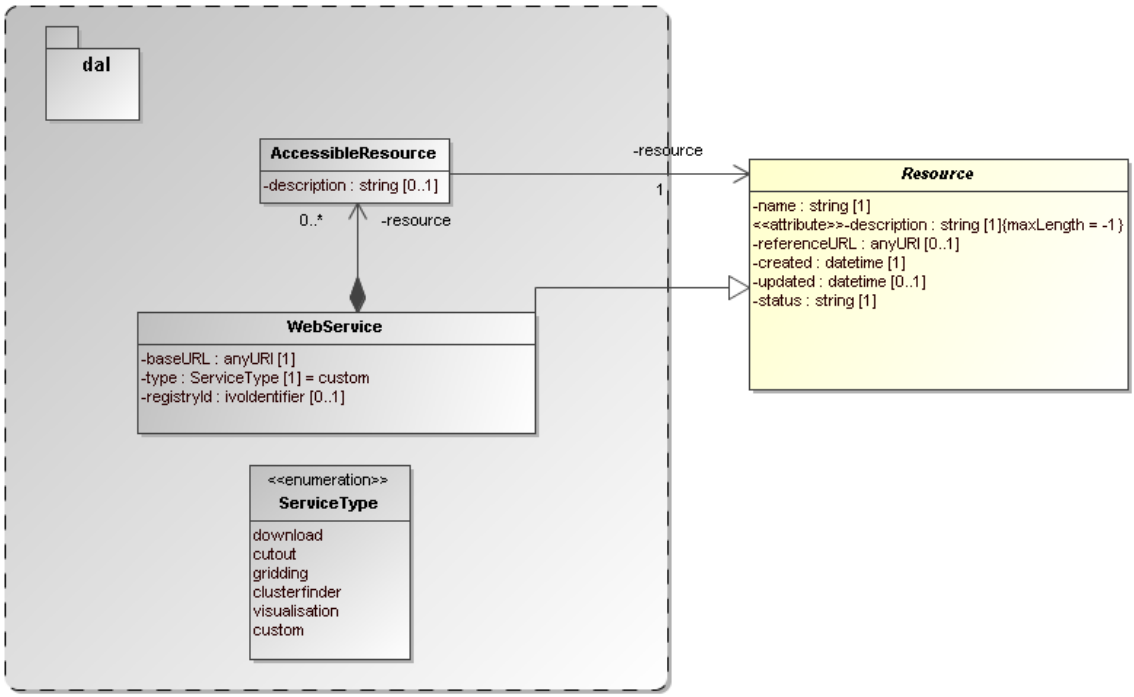

Figure 5 The dal package, with a class representing web service for accessing other SimDB resources.

4.2 SimDB/Resource

At the root of the SimDB data model is an abstract class called *Resource*, in the rest of this document we will refer to this as SimDB/Resource, a notation we will follow with other concepts as well. *Resource* represents the different types of highest level meta-data objects to be stored in a SimDB. Concrete examples of this are represented as subclasses. First *Experiment* (SimDB/Experiment), which represents different types of experiments that have been performed (run/executed/...) and have produced the results that SimDB users may be interested in. Examples of SimDB/Experiment-s are *Simulations*, but also the various PostProceissing operations transforming simulation results into other products such as halo catalogues, density fields etc.

The second major type of SimDB/Resource is the SimDB/Protocol. This concept represents a *formally prescribed way of doing an experiment*. It is derived from the concept with the same name in the domain model, which itself was inspired by the concept with the same name in Chapter 8.5 in [3]. In the SimDB/DM this concept has concrete representations in the computer programs that are being used to run simulations and post-processing etc. As such it defines the possible input parameters, possble algorithms, the kind of results that can be produced by the code. Every SimDB/Experiment must indicate which SimDB/Protocol was used and for example provide values for the input parameters, indicate which physics was used

This separation between *Protocol* and *Experiment* is an important feature of the model. It is directly taken over from the domain model presented in [7], and is related to the Measurement-Protocol pattern in [8]. That pattern says that when one does a *measurement* (of some property) it is important to remember the *protocol* by which the measurement was made ([8], p65). In [7] this was expended to experiments, which in general consists of large numbers of "measurements", all done in similar ways. Whereas the term measurement seems to be more applicable to observations, it is simple to generalise the concept a bit and apply it to the *calculation* of properties during a simulation. Actually this is similar to the CalculatedMeasurement in [8].
An important reason to introduce this separation here is to avoid having to redefine the parameters and other aspects of a simulation code each time a simulation is run.

4.2.1 Connection to Resource Registry's VOResource

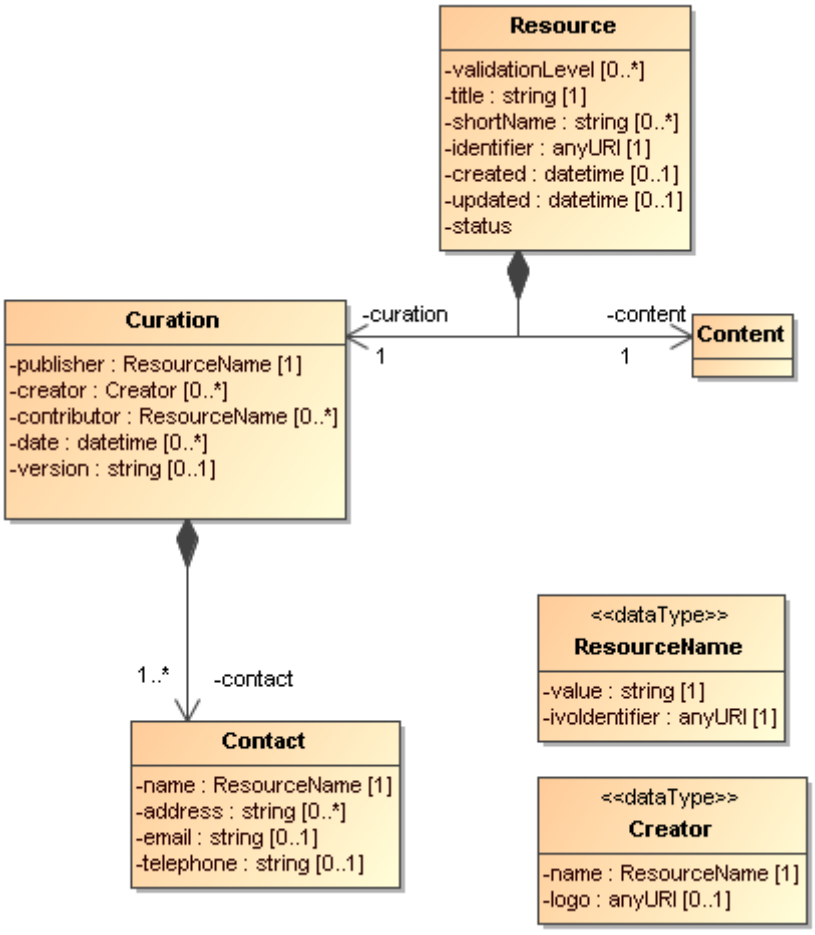

Figure 6 UML rendering of the Resource complexType from [16].

In Figure 6 we present a UML rendering of the *Resource* complexType as as inferred from the Resource Registry VOResource XML Schema [16]. Comparing that model to SimDB/Resource in Figure 3 we can see that these two models for

Resource are related, but not identical. In data modelling terms, it is not true that a SimDB/Resource *is a* Registry/Resource (or *vice versa*). *Curation* is modelled differently and arguably with less detail in SimDB[7], but the main difference is in the *Content*. SimDB provides a very detailed and specialised model for the *Content* of Simulations and related resources, by modelling provenance, motivation and results characterisation. This higher level of detail gives rise to a higher level of granularity in the types of resources stored in a SimDB, which in general will be to fine grained for registration in a Registry. This is similar to the case of a single image, which is not a Registry/Resource, whereas a SIAP-compatible *service*, providing access to many images, is.
A SimDB service itself will have to be registered (see chapter 7 for that discussion), i.e. a SimDB service *is a* Registry/Resource. In discussion with Ray Plante (IVOA Interoperability meeting May 2007, Beijing) on this issue it was proposed that some part of the contents could also be registered in a Registry directly, i.e. we should be able to identify Registry/Resource-s in SimDB. Considerations to decide on how to make this identification would be for example that all data products resulting form a well defined (and published) scientific project could qualify. To represent such a possibility for now we have introduced another subclass of SimDB/Resource: SimDB/Project. This is not much more than an annotated aggregation of other SimDB/Resources, with some additional attributes describing the motivation etc. The metadata of a SimDB/Project is not the same as that of a Registry/Resource, however we propose that we should be able to define a transformation (possibly implemented again in XSLT) to transform a SimDB/Project and produce a Registry/XML representation. Some more thoughts on this subject will be given in chapter 7.

4.3 *Object types, object collections and characterisation*

The world of simulations is very heterogeneous, and in contrast to a model for images or for spectra, we can not predict what type of results a simulation or post-processing product will deliver. Simplifying a bit, “all” images contain pixels at a given sky position, measuring a flux; spectra contain pixels representing a given wavelength, again measuring a flux. SimDB model we do not assume very much about the type of products produced and hence these have to be explicitly described in the model. In the domain model [7] results are represented as in the diagram in Figure 7, which is a slight modification of the original:

[7] **TBD Should we follow the Registry’s Curation model for SimDB resources?**

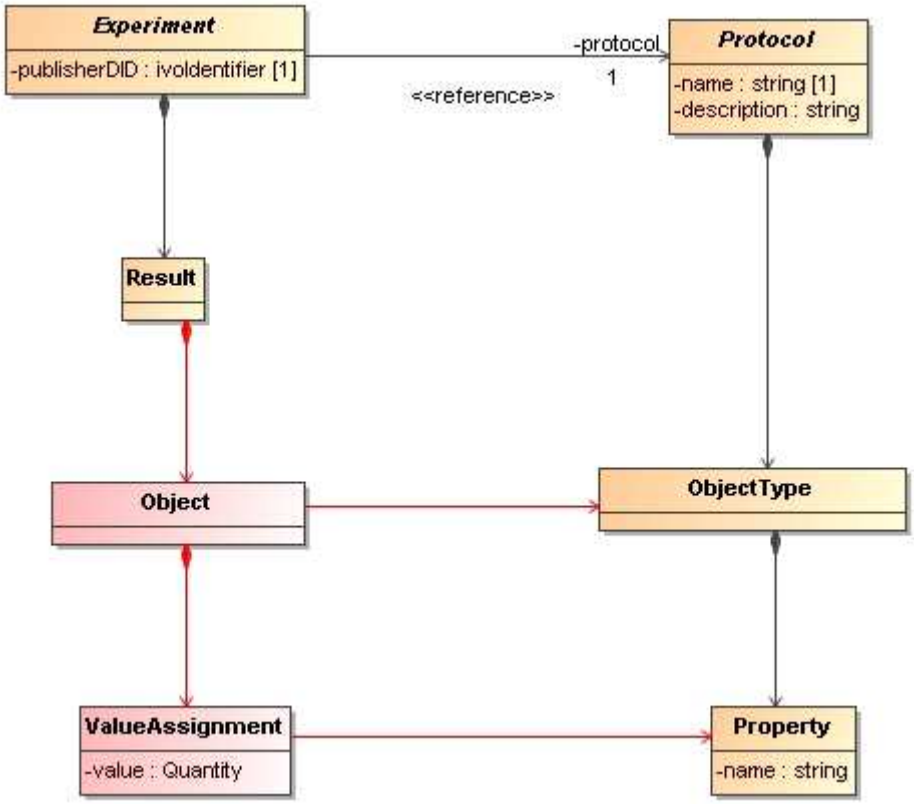

Figure 7 Domain model for results.

Results consist of Object-s. Objects are structured according to an ObjectType. The possible object types are predefined in the protocol by lists of Property-s. Objects are instances of object types, similar to the way objects are instances of classes in common object oriented programming languages, and assign values to the properties.

In SimDB (and the domain model) the ObjectType concept is supposed to represent the “atomic building blocks” of a result. In the model a result should be seen as a collection of these objects. So for example an image is a collection cof pixels, with properties location, flux, time. The bbjects in a source catalogue (result of a "source extraction experiment") are the sources, with whatever properties the source extractor calculates. In simulations a result may consistsof a collection of N-body particles, with properties (x, y, z, vx, vy, vz, mass) or a collection of (adaptive mesh refinement) grid cells, with properties (position, size, density, temperature). Howevere the actual result is stored, in principle one must be able to identify such collections of objects and their properties.

This is a model that *could* describe the actual data products. Objects, instances of object types could be stored in a database for example. However, the current data model is not supposed to be that fine grained. In this respect it is different for example form the spectrum data model, which *does* contain the objects (“pixels”) themselves. The SimDB/DM is a model for metadata, and a detailed

model for the data is beyond its scope. Nevertheless it may be useful to have some indication of the actual results, albeit not in all detail.
We have therefore looked for a way to *summarise* the detailed results. This is very similar (in our opinion) to the approach taken in the Characterisation data model and we use similar names. In fact it follows the proposal for a Characterisation domain model **@@TDB add link@@** that may be discussed further in the DM working group (private discussions with Mireille Louys and Francois Bonnarel).
The following diagram shows our approach to characterising the contents of the results (which in the SimDB/DM are called Snapshot-s).

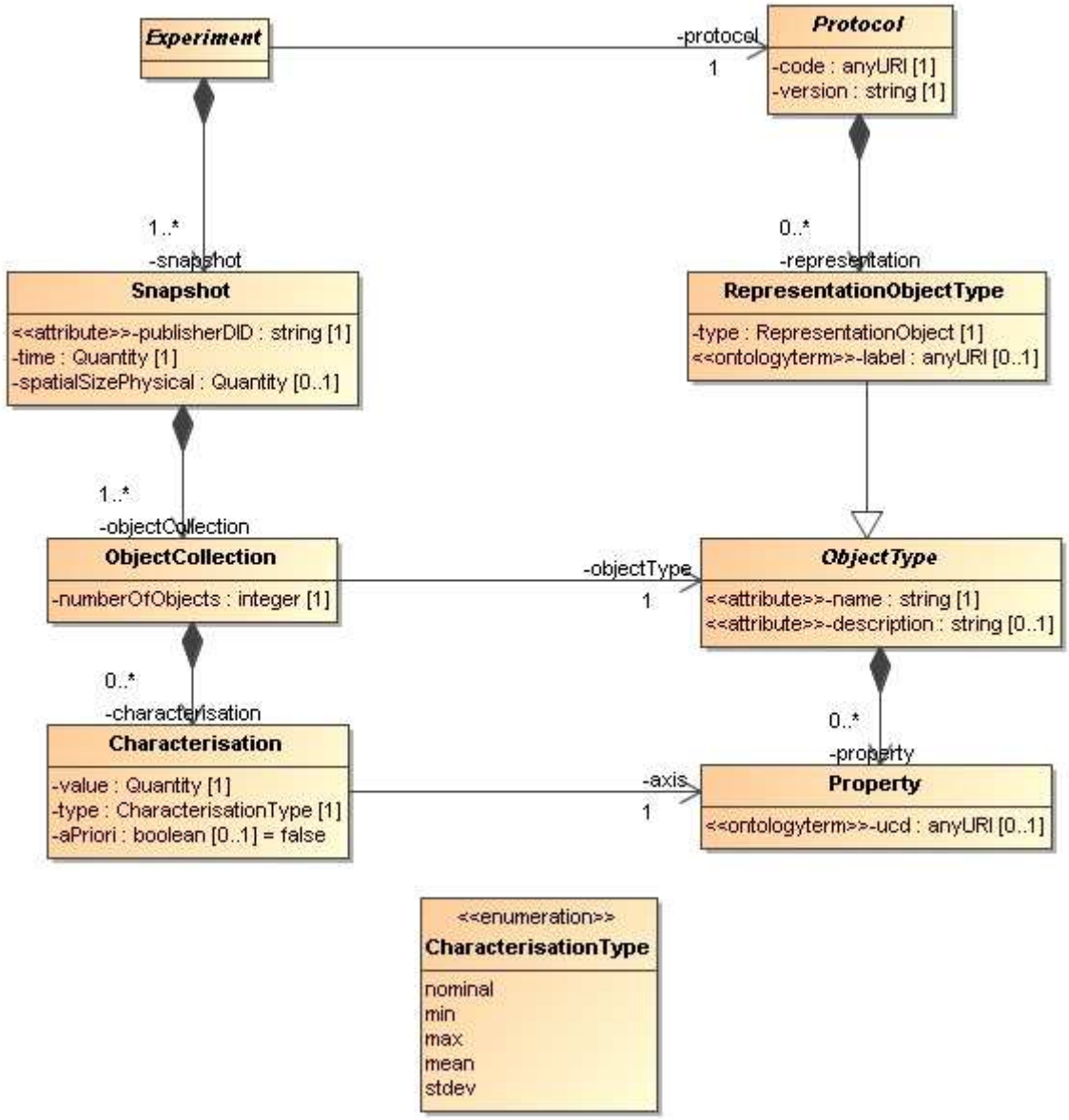

Figure 8 Characterisation in SimDB/DM

A snapshot has a collection of ObjectCollection-s, each of which represents a

collection of objects of a single ObjectType. The object collection has a collection of Characterisation-s, each of which characterises a property of the object. The interpretation is that from the collection of objects of a given object type, we can extract collections of value(assignment)s, one for each property of the object type. It is these collections that we want to summarise. Precisely *how* this summary is made is defined by the *type* attribute on the Characterisation class. It takes a value in an enumeration, examples of which are *min*, *max*, *mean*. We expect this list to change, maybe become a semantic vocabulary, as it corresponds to various statistics one could derive form the collections of properties.
@@TBD We want to discuss the issue of characterisation in the SimDB/DM further in the context of the corresponding discussion in the DM WG@@.

4.4 SKOS vocabularies

In the SimDB data model, observables, object types, properties, parameters that play a role in a given simulation have to be defined explicitly, for the world of simulations is too large to define all possibilities in the model itself. This in contrast for example to the spectrum data model **@@TBD add reference@@**, where we know that a flux is determined for a wavelength interval, or a model for images where a flux is determined for a spatial pixel. In principle the publisher of a SimDB/Resource has all freedom to name and describe these entities. For other users to understand the meaning of them, we have where appropriate, added an attribute corresponding to a semantic label. This is similar to the situation in VOTable, where FIELD-s can be given a UCD (or UTYPE) that allows users to understand the meaning of a column in the table.
In SimDB we need to generalise this concept as UCDs are not sufficient for our puspose. For example target object types are not covered by the list of UCDs and the same for other elements in our model. The Semantics WG is defining semantic vocabularies for use in the VO, for example of astronomical objects. We try to anticipate their results by introducing a special type of attribute in our UML profile that corresponds to a concept in a given ontology.
Technically, we have defined a sterotype <<ontologyterm>> that can be assigned to an attribute in the UML model. Attributes with this stereotype must define a value for the tag "ontologyURI". This value must be a URI that points to a SKOS XML document defining a list of (top) concepts that are the valid values for the attribute.
Concretely, in our model we have assigned this sterotype to the following attributes and assigned the indicated URIs. Currently the URIs point to a location in the snapdm project in volute. Eventuallly they should link to a location decided by the IVOA/Semantics group.

Table 1 Attributes in the data model that represent a concept in a SKOS vocabulary, i.e. have sterotype <<ontologyterm>>.

attribute	URI
Algorithm:label	IAUT93.rdf
Physics:physicsLabel	http://ontology.of.physical.objects
Property:ucd	UCD.rdf

RepresentationObject:label	IAUT93.rdf
TargetObjectType:label	IAUT93.rdf
TargetObjectType:…	http://some.way.to.point.at.identified.objects
TargetProcess:astroJournalSubject	AAkeys.rdf

4.5 Issues for the model

4.5.1 Normalisation

The current version of the SimDB/DM is rather more *normalised* [9] than most of the other data models in the IVOA. We explain this concept based on a particular choice we made during the modelling process, and then we discuss the consequences of particular choices.

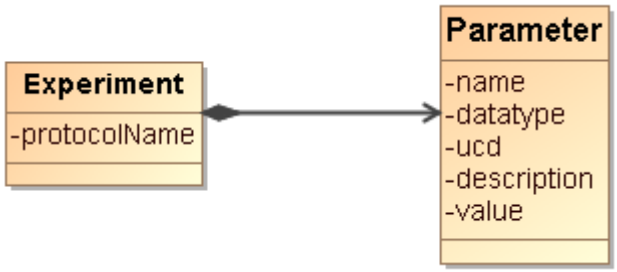

Figure 9 Non-normalised model for experiments and parameters.

At an earlier stage, the model was less normalised in the design of the input parameters of an experiment, as is illustrated in Figure 9. There was no separate protocol class, only an attribute *protocolName* on the **Experiment** class indicated the protocol by which the experiment was run. Also, the input parameters on the experiment were completely contained in a collection of **Parameter**-s. The **Parameter** class contained all the details, including *name* of the parameter, *description*, *ucd* etc. It also contained the *value* of the parameter in the experiment.

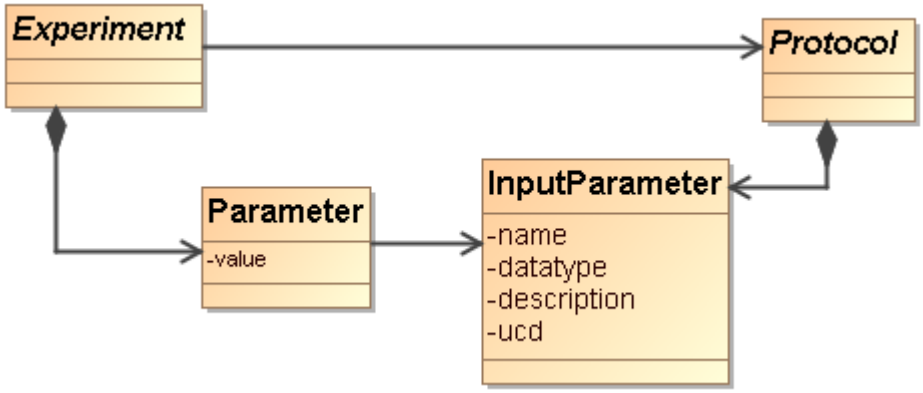

Figure 10 Normalised modelling of experiments, protocols and their parameters.

Currently the model treats parameter definitions and settings as in Figure 10. In this normalised design, the *Protocol* is given a class of its own, and it contains the input parameter collection. The *InputParameter* class does *not* contain a *value*, only the definition of the parameter: *name*, *datatype*, *ucd*, *description*. The values assigned to parameters in a given experiment are captured with the *ParameterSetting* class, contained in a collection off *Experiment*.
The motivation for this change of model was that a SimDB instance will in general contain many simulations (experiments) run with the same simulator code (protocol). In the old model, each experiment has to define the collection of input parameters with all details. In the new design this only has to be defined once, on the appropriate protocol. This clearly is less redundant, which is one important design goal of a normalised modelling approach. At the same time it provides an explicit identity to input parameters, which allows us to ask explicit questions about all parameter settings for a given, identified parameter. In the old model this is only indirectly possible, using equality of the name of input parameters for all experiments having the same *protocolName*. Now we can ask for all experiments with the same protocol reference, and look for parameter settings with the same input parameter reference. This is arguably a more "correct" model of reality.

There are therefore advantages to normalisation, but there are also disadvantages. We need to realise these and make choices that optimise the usability of the data model. One of the main disadvantages is that *references*, which naturally have to be introduced when normalising a model, are more difficult to deal with than most of the other modelling elements, particularly in some physical representations (see below). When defining a new experiment, one will have to find the input parameter that one needs to set, and instead of simply giving name/value, one needs to represent the reference to the parameter. For this one may have to extract the protocol as stored in the SimDB and find the appropriate identifiers of the input parameters. In this sense an *Experiment* definition becomes less self-contained, it depends on the details of the *registered Protocol*. This protocol is registered separately and necessarily at an earlier point in time.
This puts strong requirements on SimDB implementations to maintain referential integrity, something which will be even harder to achieve if we were to allow cross-SimDB referencing. In one advanced usage scenario the UC San Diego version of SimDB registers the Enzo[8] simulator, whilst the Italian SimDB allows registration of simulations that used it and reference the remote protocol[9].
Similarly a query language needs to be able to handle with this level of indirection. For example in a relational database one needs to write joins between ParameterSetting and InputParameter. For expert SQL users this is not a problem, but is something to get used to. For simpler query languages, those not allowing joins, like TAP/Param, asking meaningful queries becomes very difficult. One way around this problem could be to add some view definitions to

[8] http://lca.ucsd.edu/portal/software/enzo
[9] We actually do not support this scenario in the current version of SimDB.

the model. In relational databases, views are predefined, named SQL queries that can be treated as if they were tables when querying the database. It is quite straightforward to define some SQL queries that as it were denormalise the model and put the input parameter definition back under the experiment together with the value. This way one may protect users of the database from the high level of normalisation. @@***TODO discuss this further*** @@

4.5.2 Quantities and Units

At various locations in the SimDB data model numerical values can be defined, for example in parameter settings or the characterisation of properties of representation object type collections. Often these numerical values will need to have a unit. The IVOA has two ways of dealing with units. Either units are fixed explicitly for properties/parameters in protocol or data model, sometimes depending on the small list of possible UCDs. Alternatively units are epxlicitly stated, for example in VOTable. At the moment we suppport the second mode, especially because, as is true for VOTable, we do not know what kind of property is being used. To this end we introduce a value type in the model, `Quantity`, which contains a value and a unit, and which is the data type of various value attributes, for example in *NumericalParameterSetting* or *Characterisation*. In the XML schema this is translated to a complexType with 2 elements, in the relational database schema to two columns, one with the value, one with the unit. It is in the use of the relational schema that we anticipate problems with this approach, especially in the query protocol to SimDB. Consider the typical science question: return all N-body simulations with particle mass roughly $10^{10}M_{sun}$. In SimDB this would be need to be translated in an ADQL query which contains the unit column explicitly. Allowing users freedom of registering SimDB resources using any units they desire can lead to resource containing, for the same observable "N-body particle mass", values with a whole range of units. To provide reasonable support to users requires the SimDB implementation to be able to do the automated transformation. But in the We propose in SimDB to use ADQL/TAP as the query interface. If units are stored explicitly users can phrase queries using these,

An alternative approach is to mandate stating values for properties with a given UCD (or other semantic label) always with the same units. This would solve the query problem but poses others. For one it may be very (too?) unnatural for users to be forced to use meters for cosmological simulations, or megaparsec for simulations in the solar system. Related to this is the probably contentious discussion of what units to assign to what UCD. One might choose SI or cgs units, but these are not always very useful or natural. **@@NB This is similar to the recent proposal by Alberto Micol for "Standardising units and formats (and ref frames?) in transmission"**
http://www.ivoa.net/forum/dal/0905/1270.htm @@

4.5.3 Linking services and experiments

When studying the SimDB/DM and comparing it to the explanation in section 2.1, one will notice that there is no concept of storage for the results in the proposed model. The reason is that whereas we can define the concept of a Result as collections of Objects quite satisfactory, we have shied away of trying to model the precise way these results are actually stored in files or a database. There are simply too many possible and actual ways in which results can be stored in a file system. In general, a result, or snapshot in our model, can not be modelled with a simple reference to a file or table in which it is stored. Large results may be split up over many files, stored as structs, or in arrays. We have therefore decided not to open this can of worms (or reopen it, see the Quantity data model [20]).

Instead we assume the existence of web services that allow users access to the results of SimDB experiments. Some of these services may implement a standard protocol as defined by SimDAP, or they may be custom services. The precise way to relate experiments to services and what can be inferred about how to call them is the task of the SimDAP protocol.

In the model actually web services are related to resources in general. This can be used to represent services that gives access to a set of experimental results from some project, or that can for example visualise any result of experiments performed according to a fixed protocol, for example generic Gadget-format visualisers.

4.5.4 Constraints

The UML profile (see 9.2Appendix A) allows the definition of various types of constraints. For example we have constraints on the lengths of (string) attributes. These are defined using tags on the <<attribute>> stereotype. Other constraint on that stereotype are uniqueGlobally and uniqueInCollection. The latter can for example be used to state that the names of properties on an object type should be unique for that object type. The former states that the attribute should be unique in the

Issues:

- Can we make statements like: "A SimDB implementation MUST enforce all the rules from the UML model"? Or do we have to stick to statements in terms of the physical representations?
- *DM WG may want to start thinking how to express constraints in a DM*

5 SimDB/TAP (normative)

A SimDB "is a" TAP [6] service, with certain restrictions. What this means is that the main way by which SimDB is supposed to allow users to query for simulations and related resources is through ADQL queries submitted through HTTP requests. SimDB uses the TAP protocol to define *how* these queries should be sent. SimDB is a *special* TAP service in that the TAP metadata are predefined and based on the SimDB/DM. One consequence of this is that all SimDB instances will understand the same ADQL queries.
Here we define the subset of TAP functionality that must be supported and define the SimDB TAP_SCHEMA.

5.1 SimDB as TAP service

A SimDB MUST implement the TAP protocol exactly as it is defined in the corresponding specification, but with some following changes. Here we provide a few consequences of this requirement (introduced by "Following TAP …"), and list explicitly all deviations of that specification (introduced by "In contrast to TAP …").

Following TAP, a SimDB/TAP service "MUST be represented as a tree structure of web resources each addressable via a URL in the HTTP scheme"[10].
In contrast to TAP, SimDB mandates some extra structure on this structure. A SimDB service itself MUST be accessible through a registered web resource. We will refer to this address as %SimDB%.
The TAP functionality MUST be accessible through %SimDB%/tap. I.e. it is this URL that corresponds to the "tree structure of web resources" in the TAP specification. This structure is mandated to provide a uniform treatment of the different protocol features, TAP and REST.

Following TAP, a SimDB MUST implement synchronous ADQL querying of the database following the method defined in TAP. As a consequence ADQL querying will be available under

```
%SimDB%/tap/sync?REQUEST=ADQLQuery&ADQL=…
```

@@TBD check compatibility with TAP@@

In contrast to TAP, a SimDB MAY (i.e. "NEED NOT") implement asynchronous ADQL queries. In fact a SimDB MAY support all the other query configurations including (a)synchronous Parameter queries. In all cases the TAP specification MUST be followed regarding request format, execution, result and error handling etc.

[10] TBD Should we allow HTTPS as well?

Following TAP, a SimDB MUST [11] support metadata querying of the TAP_SCHEMA in all manners defined in TAP. In particular ADQL queries to the TAP_SCHEMA tables MUST be supported.

In contrast to TAP, the metadata for a SimDB service are prescribed and as a consequence each SimDB service MUST give the same reply to the same TAP metadata query. This reply is completely specified by the SimDB/TAP metadata described in 5.2.

Following TAP, the result of a query MUST be a VOTable constructed according to the description in section 2.9 of the TAP specification.

@@TBD more cases where TAP is followed and/or deviated from? @@

5.2 SimDB/TAP_SCHEMA

TAP expresses its content metadata in various ways. One of these is through a set of tables in the TAP_SCHEMA, containing information about schemas, tables, columns, foreign keys and possibly other elements that assist the user in defining queries.
Generic TAP implementations are free to define their database, i.e. there is no fixed data model. SimDB/TAP is special in that it completely[12] defines the contents of the schema, i.e. SimDB/TAP prescribes the relational model of the SimDB database.
This relational model is derived from the SimDB data model. SimDB/DM is an *object* model, but we follow standard object-relational (OR) mapping algorithms to derive from this a relational database model. This model is completely described using the various metadata formats in the TAP specification.
The formal definition of the SimDB/TAP_SCHEMA is presented as a VOTable serialisation of the TAP_SCHEMA tables in Appendix D.

The main features of the OR mapping are summarised in

Table 2 Object-relational mapping from SImDB/DM to the TAP_SCHEMA

Object model element	**Relational data model element**
objectType	1 table (or view!) per object type. Must contain all attributes as columns (see below), where appropriate including attributes inherited from base class(es). Must include some extra columns such as identifiers and timestamps.
simple attribute	column of mapped ADQL data type (see Table 3)
structured attribute	one column for each attribute of structured

[11] TBD since the metadata is prescribed, iso a MUST we might use a SHOULD ?
[12] TBD We should decide whether we want to allow extensions

	datatype, with a particular naming convention.
reference	foreign key (FK) column of same type as identifier column in referenced table.
collection (parent-child)	FK column named “containerId” from child to parent
inheritance	Identifier column of the subclass is also a FK column to the base class’s table. On the table for the root base class a VARCHAR column named DTYPE stores the Class name of the object actually stored in the row.
package	Ignored. Could be mapped to schema, but we have decided not to do so.
…	…

The mapping of data model primitive types to ADQL types is preformed according to the prescription in Table 3.

Table 3 Mapping of SimDB//DM primitive data types to TAP's ADQL data types.

Primitive data type (from IVOA Value types)	**ADQL Type**
boolean	BOOLEAN
integer	INTEGER
real	DOUBLE
datetime	TIMESTAMP
string	VARCHAR(L) Length L=256 if no length constraint is given, otherwise the maxLength or length constraint is used.
…	

There are typically different ways to map classes in inheritance hierarchies. Some advocate to have a separate table for each concrete class and store all attributes, including the ones that have been inherited., others have a table for each class and only. We have decided that for each Class in the model there should be a table to query. This may be a view, SimDB

6 SimDB/REST (normative)

SimDB defines a REST [11] interface for retrieving or uploading XML representations of complete SimDB/Resource-s.

The representations of the data model that are retrieved or uploaded MUST be XML documents that are valid according to the XML schema defined in 6.1. The actual actions and their protocol is defined in the subsequent subsections.

6.1 HTTP

6.1.1 GET

A SimDB MUST implement an HTTP GET request for an identified resource. The format of the URL MUST be

```
%SimDB%/rest/get?resourceId=:ID
```

Here `:ID` is the IVO identifier of the requested resource. `%SimDB%` is the root of the SimDB service as defined in 5.1.
The SimDB service MUST return an XML document that serialises the indicated SimDB/Resource according to the XML schema defined in the next section 6.2

@@TBD the form of the URL is rather simplistic. Maybe more REST-like format is better? @@

6.1.2 POST

A SimDB MAY implement an HTTP POST for uploading a SimDB/Resource.

@@TBD Needs further details. The POST may be structured in a properly REST-like form. But we may postpone this to a later version @@.

6.2 SimDB/XML

SimDB defines an XML serialisation of the data model using a set of XML schemas. These schemas define valid SimDB/XML documents that are used in the REST part of the SimDB protocol.
The schema documents are part of this specification and will eventually be uploaded to the IVOA Wiki pages. Until then they can be retrieved from the Volute repository:
http://volute.googlecode.com/svn/trunk/projects/theory/snapdm/specification/xsd/.

The XML schemas are defined using a fixed mapping from the data model in UML to complexType and simpleType definitions in XML schema (see [18] and links from there). We furthermore define a set of root elements based on the concrete SimDB/Resource classes, these define the valid documents. Consequently the schemas are described in one set of documents containing the type definitions, and another single document containing the definitions of the root elements. This follows the approach in the design of the XML schemas for the IVOA Registry.

Similar to the object-relational mapping the following table gives a summary of the mapping of UML to XSD.

Table 4 Mapping of UML to XML Schema concepts.

Object model element	XSD model element
objectType	complexType
attribute	element of type according to the data type of the attribute
attribute	element[13] of appropriate data type.
reference	element of type base:Reference[14] . That type
collection (parent-child)	element of appropriate (mapped) type on the container type.
inheritance	extension
dataType	complexType.
enumeration	simpleType restricting xsd:string with an xsd:enumeration element fror each of the literals in the UML enumeration.
package	(target)Namespace and separate XSD document.
package dependency	Import of XML schema corresponding to package one depends on.
model	XSD document with a root element for each of the concrete root entity classes, i.e. those classes that are not contained in another class (either direct or through a base class).
…	

7 SimDB/VOSI (normative)

SimDB is a web based VO service protocol and hence it should comply with the *IVOA Support Interface* [17] specification. SimDB does not define a SOAP

[13] Note, attributes map to elements, not to attributes. This is so that all attributes are treated uniformly, as structured data types must be mapped to complexType-s and attributes of such a type MUST be mapped to elements.

[14] See http://volute.googlecode.com/svn/trunk/projects/theory/snapdm/specification/xsd/base.xsd

binding and therefore the “REST binding” ([17], section 2, item 3) of that specification applies.
TAP
Capabilities and *Availability* MUST be spe
@@TBD need input on how to complete this section. @@

8 Registration of SimDB implementations

SimDB implementations must be registered like all IVOA standard services. But also part of the contents of a SimDB may be registered in their own right. This will not be true for all SimDB/Resources. But for example web services and Projects may be registered. Indeed the original motivation for introducing the SimDB/Project into the model was for this purpose.

8.1 Registration of SimDB services (normative)

A SimDB implementation MUST be registered in an IVOA registry.
This means that an XML document must be provided that describes the service as a VOResource [16]. We must decide on the form of this document. Can we reuse an existing type of VOResource, for example VODataService? Or must the registry model be updated with a new type of “SimDB” resource?

As SimDB is a TAP services we likely can follow the way TAP services are registered, but it would be good to be able to query a registry for all SimDB implementations.

@@TODO discuss with Registry WG@@

8.2 Registration of SimDB/Resources (future)

Certain types of SimDB/Resources MAY be registered in their own right in a Registry. In particular SimDB/WebServices SHOULD (MUST?) be represented in a Registry. The Resource metadata may be more comprehensive than the metadata in the SimDB/DM.
Also other SimDB/Resources may be registered individually. In particular if these represent “something large”, a comprehensive result from a scientific project. Individual simulations may not qualify, protocols may though. **@@TBD how should protocols be registered?@@** .
In particular SimDB/Projects can be registered. This concept represents exactly the result of a larger project, a complete collection of resources. **@@TBD how should projects be registered?@@** .

8.3 SimDB as extension registry (far future)

IF we conclude from the previous section that SimDBs harbour resources that are qualified to be registered in a Resource Registry, it may become possible to view SimDB as an extension registry.

To do so will put requirements on SimDB. In particular it has to implement the registry interface and must for example be able to deliver SimDB/Resources in a representation appropriate for a Registry.
This likely should be postponed till later version.
@@TBD can we leave it at this, or are their more interesting statements to make about this possibility?@@

9 Extras (some normative, some not)

9.1 SimDB/UTYPE (normative)

UTYPEs are strings that “point into a data model”.

We have defined a list of UTYPEs for the model. These can be found in the HTML documentation of the model in [5]. When representing components of the data model in a VOTable (for example), these SHOULD be used, in particular when the VOTable contains results of ADQL queries to SimDB/TAP.
We use a particular rule to derive the UTYPEs from the UML model. We present this rule in a quasi-BNF form for each of the elements that have a UTYPE:

```
utype             :=    [model-utype | package-utype | class-utype |
                        attribute-utype | collection-utype |
                        reference-utype | container-utype
model-utype       :=    <model-name>
package-utype     :=    model-utype “:/” package-hierarchy
package-hierarchy :=    <package-name> [“/” <package-name>]*
class-utype       :=    package-utype “/” <class-name>
attribute-utype   :=    class-utype “.” attribute
attribute         :=    [primitive-attr | struct-attr]
primitive-attr    :=    <attribute-name>
struct-attr       :=    <attribute-name> “.” attribute
collection-utype  :=    class-utype “.” <collection-name>
reference-utype   :=    class-utype “.” <reference-name>
container-utype   :=    class-utype “.” “CONTAINER”
identifier-utype  :=    class-utype “.” “ID”
```

9.2 Harvesting (future)

@@TBD Here we could describe, IF we want to make it part of the specification, how remote SimDB-s might be harvested to replicate (some of) their resources. @@

References

[1] R. Hanisch, *IVOA Document Standards*, http://www.ivoa.net/Documents/latest/DocStd.html

[2] This document, at web address http://code.google.com/p/volute/source/browse/trunk/projects/theory/snapdm/specification/SimDB-note.doc

[3] Gheller C., Wagner R. *et al*, *Simulation Data Access Protocol (SimDAP)*, http://code.google.com/p/volute/source/browse/trunk/projects/theory/snap/SimDAP.html

[4] SimDB data model UML diagram obtained from MagicDraw : http://volute.googlecode.com/svn/trunk/projects/theory/snapdm/specification/uml/SimDB_DM.xml

[5] HTML representation of the SimDB data model in [4] http://volute.googlecode.com/svn/trunk/projects/theory/snapdm/specification/html/SimDB.html

[6] TAP specification, until further notice we will use version 0.5 on the TAP page of the data access layer working group : http://www.ivoa.net/internal/IVOA/TableAccess/TAP-0.5.pdf (**NB started out using 0.4, need to check that changes in 0.5 are properly represented where applicable**).

[7] *A Unified Domain Model for Astronomy* Lemson, G., Dowler, P, Banday, A.J., 2004 ... http://www.aspbooks.org/a/volumes/article_details/?paper_id=861

[8] *Analysis Patterns* Fowler, M.

[9] Some links to web pages on data model normalisation http://www.datamodel.org/NormalizationRules.html http://en.wikipedia.org/wiki/Database_normalization

[10] XMI ...

[11] *Architectural Styles and the Design of Network-based Software Architectures* Fielding, R. T. 2000, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[12] *IVOA Spectral Data Model* Jonathan McDowell *et al*, 2007 http://www.ivoa.net/Documents/latest/SpectrumDM.html

[13] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels,* IETF RFC 2119. http://www.ietf.org/rfc/rfc2119.txt

[14] *OMG Unified Modeling Language (OMG UML), Infrastructure* Version 2.2 http://www.omg.org/docs/formal/09-02-04.pdf

[15] Bob Haniisch et al, *Resource metadata for the virtual observatory* http://www.ivoa.net/Documents/latest/RM.html

[16] Ray Plante et al 2008, *VOResource : an XML Encoding Schema for Resource Metadata* http://www.ivoa.net/Documents/REC/ReR/VOResource-20080222.html

[17] Guy Rixon *IVOA Support Interfaces Version 1.00* …
http://www.ivoa.net/Documents/WD/GWS/VOSI-20081023.pdf
[18] XML schema, http://www.w3.org/XML/Schema
[19] Carlos Rodrigo et al, *S3 : proposal for a simple protocol to handle theoretical data (microsimulations)*
http://www.ivoa.net/Documents/latest/S3TheoreticalData.html
[20] Jonathan McDowel et al, *Data Model for Quantity*
http://www.ivoa.net/internal/IVOA/IvoaDataModel/qty23.pdf

Appendix A UML syntax

For a particular community it is possible to create a domain specific extension of UML by defining a so called *Profile*[15]. Such a profile can restrict the set of available syntactic elements to a subset of those available to UML as a whole. But also one can assign new meanings to existing elements by defining *stereotypes* for example, with associated properties (*tag definitions)*. It is also possible to predefine classes and data types (see below) that can be reused by the data modeller.

We have an initial implementation of a UML profile as created by MagicDraw available under this link. The profile is also contained in the UML file containing the SimDB data model. Here we give a list of the main elements that we use and give a short motivation for their inclusion in the language. It is our opinion that the DM working group should be ultimately responsible for a profile such as this, as it gives the possibility of defining a domain specific language for all IVOA data modelling efforts, thus giving some uniformity to those disparate efforts.

A.1 Element

All elements mentioned below are specialisations of UML Element.

Stereotypes

- **<<utype>>**: every modelling element can declare itself to represent an element in another data model by assigning this stereotype.
  Tags:
  - **utype [string]**: this holds the actual UTYPE that points to the other modelling element that is represented here.

A.2 Model

This is the root of the complete model, contains all packages, classes etc. Also contains any imported profile.

Stereotypes

- **<<model>>**
  If the designer wants to annotate the model with the tags in this stereotype (s)he must explicitly associate this stereotype to the Model.
  Tags:
  - **author** : Indicates the author(s) of the model.
  - **title** : provides a long title to the model. The name of the model should be short

[15] See for example http://advanceduml.wordpress.com/the-unified-modeling-language/profiles/ for a tutorial on profiles.

A.3 Package

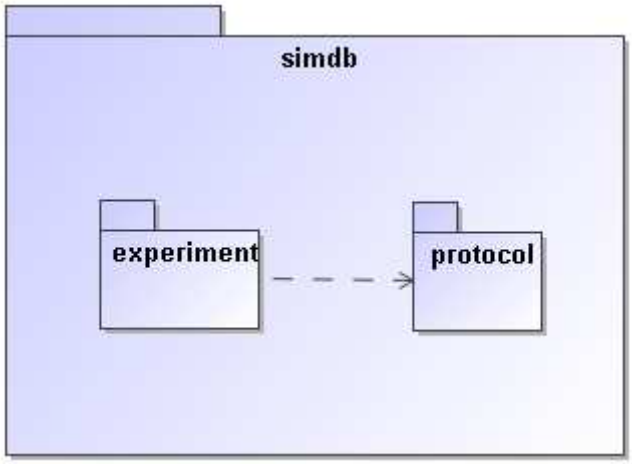

Figure 11 This figure shows a package "simdb" that contains two other packages. Of these the experiment package depends on the protocol packages, which is indicated by the dashed arrow.

A package groups related elements such as class definitions and possibly sub packages. Packages can depend on each other (indicated by the dashed line), which means that elements in one package can use elements in the target package in their definition. This relation is transitive. A package is similar to an XML namespace and in fact we map UML packages to XML namespaces in 6.1.

A.4 Class

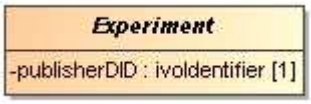

Figure 12 A Class is a rectangular box, with the name of the class in boldface.

Classes are the fundamental building blocks of a data model. A Class represents a full fledged concept and is built up from properties and relations to other Classes. An important feature of Classes as opposed to DataTypes (see below) is that objects have their own, explicit identity.

Properties:

- **isAbstract**
  Indicated by *italicised* name of the object. Implies that no instances can be made of the class, one needs sub classes for that.

A.5 ValueType

A ValueType represents a simple concept that is used to describe/define more complex concepts such a Classes. ValueType-s are, in contrast to Classes not

separately identified. They are identified by their value. For example an integer is a value type; all instances of the integer value 3 represent the same integer.
In this profile ValueType-s are only represented using specialised examples. Attributes (see below) must have a ValueType as their datatype.

A.6 PrimitiveType

PrimitiveTypes are the simplest examples of ValueTypes. They are represented by a single value only. A set of PrimitiveTypes is predefined in the IVOA profile (see Figure 13).

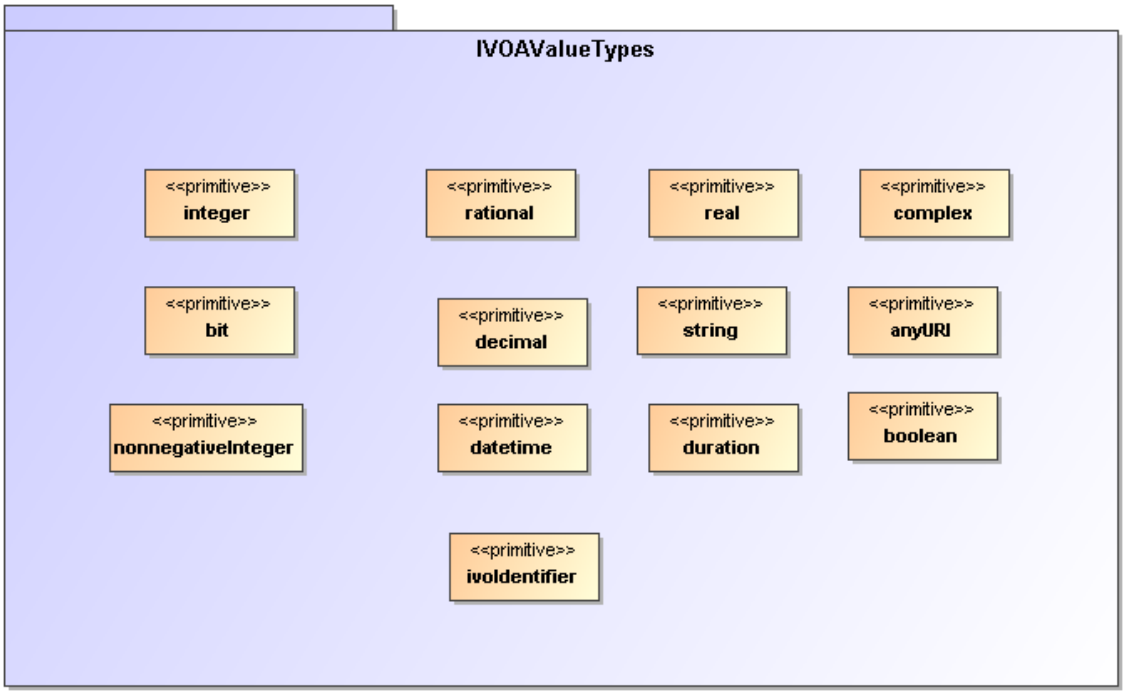

Figure 13 The PrimitiveTypes that are predefined in the IVOA profile.

A.7 DataType

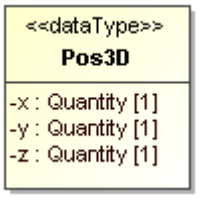

Figure 14 Example of a structured datatype:Pos3D represents a position in 3D space and is defined using x, y and z attributes. The DataType symbol is distinguished form the Class by the <<dataType>> stereotype.

A DataType is a ValueType that has more structure than a single value. This structure is modelled using Attributes, just as on ObjectTypes.

A.8 Enumeration

Figure 15 An enumeration is indicated by a box with the name of the the enumeration and the list of literals.

An Enumeration is a ValueType that is defined by a list of valid values. These are the only values that instances of this data type can assume.

A.9 Attribute

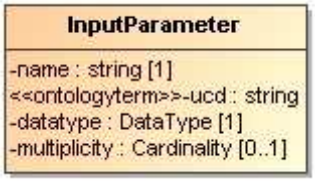

Figure 16 An attribute is indicated by a line with a name, a datatype, an indication of the multiplicity and possibly a stereotype.

An Attribute is a *Property* of a type (object type as well as structured data type). An attribute's data type is always a Value type, not an object type. For object type properties one should use References

Properties

- data type
- multiplicity/cardinality: indicates the cardinality of the attribute (assumed to be 0..1, or 1. This is a relational bias based on normal form and the assumption that most databases do not allow storage of arrays in single columns.)
-

Stereotypes

- **<<attribute>>**
  To assign further properties such as the tags this stereotype attribute must be explicitly assigned.
  Tag definitions
    - **length [integer]**: Constraint indicating that an attribute must have a specific fixed length. Is relevant only for attributes of type string.

- **maxLength [integer]**: Constraint indicating that an attribute may at most have the indicated length. Is relevant only for attributes of type string. Is used in mappings to TAP to indicate the length of the corresponding column. Thist would seem to be very much an application specific feature and therefore belong to logical modelling. But this profile can be used for that purpose, hence it is included.
- **uniqueGlobally**: Constraint indicating that only one instance of the type of the Class owning this attribute can have a given value. Globally should be read to mean globally in a given instance of the model, i.e. a database for example that stores instances of the model.
- **uniqueInCollection [boolean]**: If true, indicates that the value of the attribute can not be shared by the same attribute of any other instance of the Class owning this attribute that is in the same collection, i.e. has the same container object. In SimDB/DM an example is given by the name attribute of the InputParameter class. For a given Protocol

- **<<ontologyterm>>**
  There are many instances in the data model where we need to describe elements of the SimDB/Resource-s explicitly, because we do not have implicit information based on the context. Examples are the various properties of object types, the target objects and processes etc. Apart from a name and a description we then frequently add an attribute which is supposed to "label" the element according to an assumed standard list of terms.
  We model this using the <<ontologyterm>> stereotype. Attributes with this stereotype are assumed to take their values form such a predefined "ontology"[16].
  Tag definitions:
  - **ontologyURI**
    A URL locating a standard (RDF|SKOS|OWL|???) document containing a list of terms from which the value for this attribute may be obtained. It is our opinion that the Semantics working group should be responsible for the definition of relevant ontologies (or semantic vocabularies, or thesauri, or ...) required for a given application domain, though the contents should be decided in cooperation with domain experts.

A.10 Inheritance

[16] Possibly this should be a vocabulary, that at least is intended, and the stereotype might have to be called <<skosterm.., with tag definition named *skosVocabulary*.

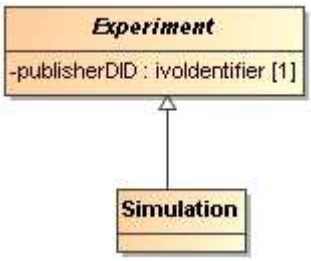

Figure 17 Inheritance is indicated by a line with an open arrow from a subclass to its base class.

Indicates the typical *"is a"* relation between the sub-class and its base-class (the one pointed at). In this profile we do not support multiple inheritance.

A.11 Collection

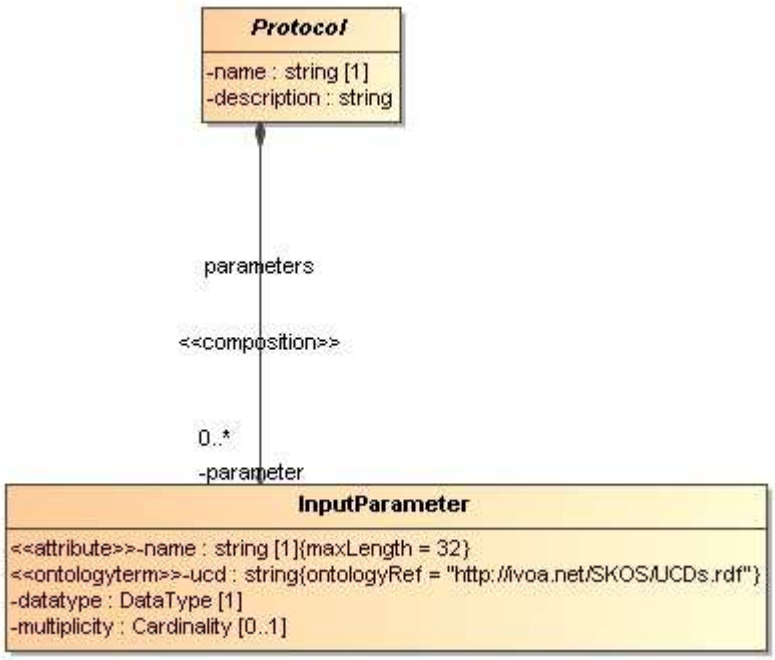

Figure 18 The line with the closed circle on one end and an arrow on the other indicates a composition relation, or collection, between the parent (on the side of the circle) and the child, on the other side.

This relation indicates a *composition* relation between one, parent object and 0 or more child objects. The life cycles of the child objects are governed by that of the parent.
In UML a composition relation is represented by a binary association *end*.

A.12 Reference

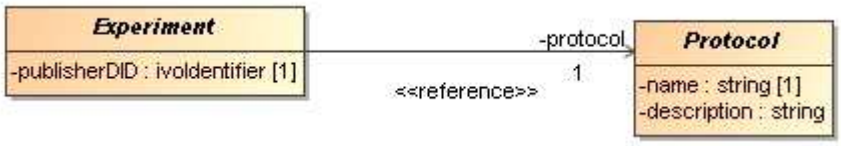

Figure 19 A Reference is represented by a line connecting a class with another, referenced, class with an arrow on the referenced class. Note, the <<reference>> sterotype indication is not required.

This is a relation that indicates a kind of *usage*, or *dependency* of one object on another. It is in general shared, i.e. many objects may reference a single other object. Accordingly the referenced object is independent of the "referee". In our profile the cardinality can not be > 1.
For implementing the Reference in UML we use a shared, navigable binary association end.

A.13 Subsets

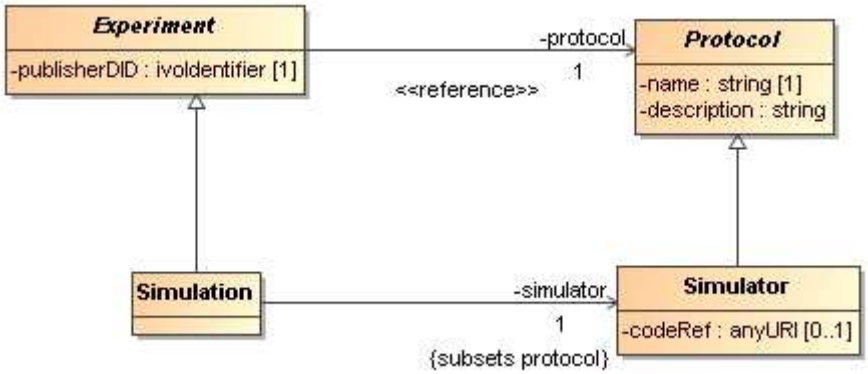

Figure 20 The subsets property can only be assigned to a relation between two objects that are both subclasses of Classes that have an equivalent relation. It is indicated by the {subsets ...} annotation to the relevant association end.

The “subsets” property, when associated to a Reference or Collection (in UML to the corresponding association end), indicates that a relation overrides the definition of a relation of the same name defined on a base class. It does so by specifying that the class at the end point of the relation should be a subclass of the class at the endpoint of the original, sub-setted relation.

A.14 … (?)

@@TBD What did I miss? @@

Appendix B Data model usage by SimDAP and S3

The SimDB data model can also be used in other contexts than SimDB implementations. Here we describe how SimDAP uses SimDB/DM and how S3 can be represented in SimDB/DM.

B.1 SimDAP

@@TBD Rick?@@

B.2 S3

S3 [19] is a simple protocol for querying and retrieving results of "micro-simulations". S3 supports three types of requests ([19], section 2):

1. **Metadata**
2. **Data query**
3. **Retrieve file**

The result of a **Metadata** query is a VOTable describing the service and a list of PARAM elements, one for each parameter accepted by the **Data query** request. These parameters will generally correspond to parameters of the model, but can also contain extra service specific parameters.

The **Data query** finds models corresponding to particular (ranges of) values for these parameters, selected ones of which can be downloaded using the **Retrieve file** request.

Most of this protocol can be described using SimDB/DM concepts. First one can argue that an S3 implementations is a SimDB/WebService giving access to results of a single SimDB/Protocol, possibly always a SimDB/Simulator, or maybe another type. The SimDB/Protocol is described using its set of available SimDB/InputParameters, obtained in the **Metadata** request. The **Data query** part searches for SimDB/Experiments executed with the protocol and having particular SimDB/ParameterSettings.

Each S3 Experiment presumably produces a single Result. The Result type does not yet exist in the SimDB/DM, where its place is taken by the Snapshot class, which is a special type of Result. Introducing the generic Result class has been discussed before precisely for this purpose of supporting more general simulation types.

Note that SimDB/TAP already supports querying for the input parameters of any protocol, and for experiments (and their results) where these parameters have been set to particular values, thought S3's query interface is arguable much simpler. It also allows one to register an S3 service, albeit so far only as a "custom" WebSevice linked to the particular protocol.

So we see that a SimDB implementation can already support most functionality of S3. Some features of S3 could be added to SimDB to complete this support. For example a new "S3" literal in the ServiceType enumeration and a more generic Result class could be easily added. SimDB/WebService might be declared (using a reference) to implement a (rather than "be a") SimDB/Protocol in its own right, which allows it to define its own extra parameters.

By construction, SimDB/TAP offers more generic query capabilities than S3's **Data query**. This comes at the cost of more complex syntax especially due to need to support multiple Protocols, but S3 could be built on top of SimDB/TAP.

@@TBD feedback form S3 authors please!@@

Appendix C An intermediate representation for data models

The UML language is not very familiar to most IVOA developers. To define it one requires moreover a graphical design tool such as MagicDraw. The XMI representation of these, though XML, is very complex to read and interpret, as it is meant to be able to epxress UML models in all their generaility. As shown in Appendix A we use a much more limited set of UML modelling elements. We have therefore defined a much simpler XML representation of this profile, which moreover is much more easily read and interpreted.

The latter was important for our code generator (VO-URP) in various ways. It allows the XSLT scripts used to generate XSDs, TAP metadata, DDLs, Java code, HTML documentation etc to be much simpler than if they had to be written against the XMI files. It also forms a nice representation for providing runtime metadata objects for the generated classes.

As first step in our simulation pipeline we generate this intermediate representation. The document itself is structured according to an XML schema that represents the UML profile rather directly and that we here shortly describe. This schema is located here.

This transformation is implemented in the xmi2intermediate.xsl XSLT script. The latest version of the intermediate representation for the SimDB data model can be found in this location, all other generation scripts work on this intermediate representation, not on the XMI document itself. Variations in tool-generated XMI or different versions of XMI can now be supported by appropriately adjusting the xmi2intermediate.xsl script. In principle this XML document is expressive enough, and simple enough to allow for simple text editing.

Appendix D SimDB/TAP_SCHEMA

The following VOTable document provides the contents of the complete SimDB/TAP_SCHEMA. We have chosen this representation over for example the

@@TBD This is a place holder section that is waiting to be filled once the TAP_SCHEMA is fixed. @@

@@TBD Is a filled VOTable representing the TAP_SCHEMA tables an acceptable form to pre-define the SimDB/TAP_SCHEMA?
I choose it because it is likely more detailed than the VODataService representation. @@

@@TBD Shall we reproduce the VOTable here, or is it sufficient to have a link to the online version?
@@