International

Virtual

Observatory

Alliance

IVOA SkyNode Interface
Version 1.01

IVOA Working Draft 24 June 2005

This version:
1.0 http://www.ivoa.net/Documents/WD/SNI/SkyNodeInterface-20050624.doc

Latest version:
http://www.ivoa.net/Documents/latest/SkyNodeInterface.html

Previous versions:
none

Working Group:
http://www.ivoa.net/twiki/bin/view/IVOA/IvoaVOQL

Editors:
William O'Mullane, Masatoshi Ohishi

Authors:
IVOA VOQL Working group

Abstract

This document describes the minimum required interface to participate in the IVOA as a queryable VONode as well as requirements to be a Full OpenSkyNode, part of the OpenSkyQuery Portal.

Status of this document

This is an IVOA Working Draft for review by IVOA members and other interested parties. It is a draft document and may be updated, replaced, or obsolete 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".

Acknowledgments

This working draft has been developed based on discussions at various IVOA meetings and continuing emails on the mailing list. The editor expresses his appreciation for many valuable contributions from the VOQL WG.

Contents

1 Introduction

At the Cambridge IVOA meeting (2003) there was general agreement to the ADQL (Astronomical Data Query Language) concept. This implied an SQL like language but passed as an XML document. VOQL (Virtual Observatory Query Language) was seen as a higher layer built on ADQL and web services. ADQL is defined in [1].
SkyQuery (www.skyquery.net) is a successful implantation of distributed astronomical query system. This is implemented using some proprietary Microsoft Technology but for the most part uses SOAP Services. Several people have expressed in interest in extending SkyQuery and making it more open. Currently to be a SkyNode one has to implement a set of web interfaces some of which would require .NET technology. The original

prototype has proved the idea but now it is time to bring this in line with VO efforts and to make it properly platform independent.
OpenSkyQuery (http://www.openskyquery.net/) opens up the SkyQuery protocol to enable other databases and servers to become “Full SkyNodes”.
It has been demonstrated that a single WSDL (Web Services Definition Language) file may be implemented in at least Java and C#, i.e. on Linux as well as Windows. It has also been demonstrated that a single client may interact with both implementations of the service.
In this document we describe the astronomical data query interface for the virtual observatory. Any IVO site implementing the Query Interface (hereafter called as QI) would be considered a “Basic SkyNode”. More advanced nodes would participate in distributed queries through the portal at OpenSkyQuery.NET. This was demonstrated at the AAS in January 2005. JVO have also implemented a portal which works with Basic SkyNodes.

2 Open SkyQuery Portal, IVOASkyNode, ADQL and VOQL – brief architectural discussion.

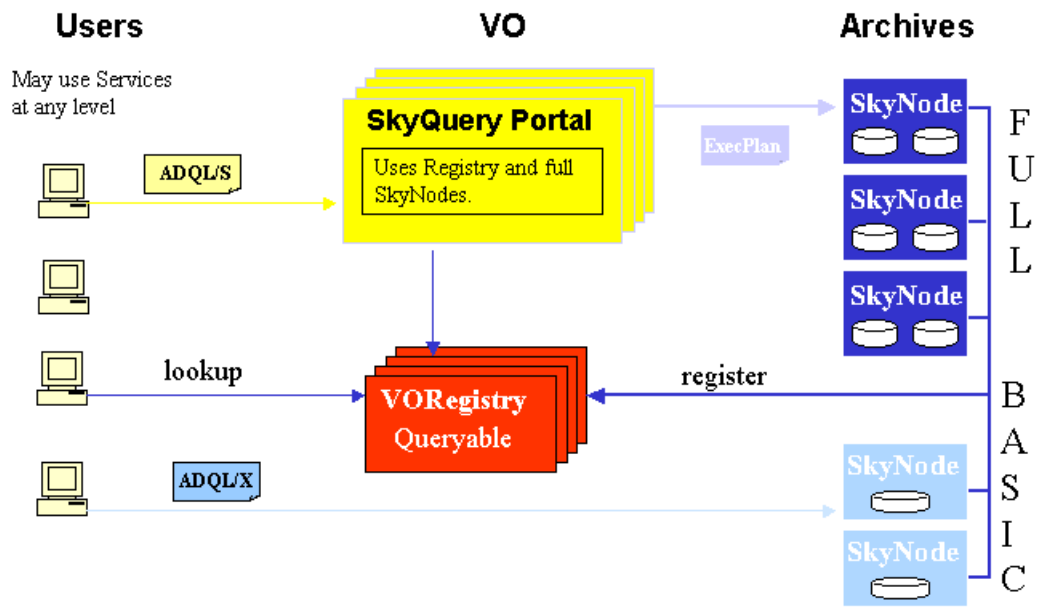

Figure 1. Architecture

The Astronomical Data Query Language (ADQL) is the language used by the International Virtual Observatory Alliance (IVOA) to represent astronomy queries posted to VO data services (SkyNodes). ADQL has two forms: ADQL/s (string form) and ADQL/x (XML representation corresponding to ADQL/s). ADQL/x is used to transport

queries to IVOA SkyNodes. Different SkyNodes may not support all features of the Language (see the matrix below in section **4.1**). Hence ADQL would be passed from the SkyQuery Portal to the SkyNodes or it may come directly from a client or the VOQL portal. All nodes and the portals should be accessible via SOAP services. The boxes on the nodes in Figure 1 represent SOAP methods.
The Open SkyQuery Portal would accept a form of string based query like the current SkyQL (www.skyquery.net), which is in fact ADQL/s (as shown in Figure 1). A parser converts this into ADQL/x.,. The portal uses the registry to resolve server names for individual SkyNodes, makes the Execution Plan (see Section 5 below), and passes it on to the first node in the plan setting up an execution sequence as in the current SkyQuery portal.
Finally the Virtual Observatory Query Language (VOQL) is an ambitious language at a higher level than ADQL. A VOQL portal would take VOQL programs. This would need all the work of the SkyQuery portal and more to make it function.
In summary we may see three layers of VOQL:

- VOQL1 WebServices : ADQL and VOTable to exchange information between machines
- VOQL2 Federation : SQL-like query language and federation system i.e. combination of SkyQuery , JVOQL and VO standards.
- VOQL3 VOXQuery: future high level VO language with query and procedural abilities

3 Standard Interface

QI-1 SkyNodes SHALL implement the services outlined in the support interfaces specification (GetAvailability) [2]

QI-2 All errors in the services SHOULD be returned as SOAP exceptions. There is no perceived need at this time to have special exceptions although this may become useful in the future. In the initial version then all exceptions may simply be a useful message in a generic exception.

4 IVOA SkyNode Interface

4.1 Feature Matrix

The following matrix tries to categorize features and SkyNode/ADQL Levels.
Basic SkyNode here is the minimum IVOA SkyNode Interface – this is useful in itself as it allows one to send queries to a system using ADQL. This is also just one step up from cone search. A matrix has been used as any feature on their own may be useful, i.e., a node which can do XMATCH is already useful even if it may not participate in the portal because it lacks other features.
It is assumed large survey data would be served in Full SkyNodes as footprint services should make queries more efficient on the large surveys.
This table is meant as a summary, more explicit requirements are detailed below.

Feature	Basic SkyNode	Full SkyNode	Optional
ADQL – circle	X	X	
Functions	X	X	
Performance Query		X	
Takes exec plan does Xmatch		X	
Footprint – Region intersect			X
VOStore			X

4.2 IVOA SkyNode Interface

The SkyNode interface requires several methods to work. Several of these methods are for Metadata – in this case we need to know about Tables and Columns that may be included in a query. To facilitate interoperation we will also need to be able to get UCDs and units for each column. ADQL also exposes standard ways for querying metadata but it seems sensible to have explicit Web Services calls for this also.
Another service will need to take an ADQL document and return a VOTable of data. The current incarnation of ADQL syntax supports only queries which would return data i.e. there is no provision for Data Manipulation Language which would have no data to return.
Below we split the requirements in two sections – the minimum set required to be a SkyNode called Basic and more advanced requirements aiming toward complex portal creation. It is assumed nodes will exist in the spectrum between these extremes. The Functions interface allow the node to specify precisely what it supports.

4.2.1 Basic Sky Node

QI-3 SkyNodes SHALL register with the registry with type=”OpenSkyNode”
We need to add a new SubClass of Capbability and VOResource for this – there will be other metadata we need which is specific to OpenSkyNode, i.e.,

- Compliance :is it a Basic, Partial, or Full SkyNode
- MaxRecords : if there is a limit to the number of records returned
- Latitude and Longitude : GPS coordinates of the Node ;
- PrimaryTable : the main table in the archive
- PrimaryKey : the key column of the primary table.
- AcceptsUCDs : True or False

Check for this schema on
http://www.ivoa.net/twiki/bin/view/IVOA/IvoaVOQL#SkyNode_Registry_Schema_Extentio.
At time of writing http://www.ivoa.net/xml/OpenSkyNode/OpenSkyNode-v0.1.xsd is in use (although it lacks to accept UCDs).

Specifying AcceptsUCDs means that a query may be specified using UCDs – this is not a requirement and shall probably not become one, specifying the metadata simply allows experimentation with this idea. UCDs do not carry unit information so although it makes sense to use a UCD to request a column it is unclear if this is useful in specifying the predicate of a query – here one would need to know the unit and type of the native SkyNode column to specify the query correctly.

QI-4 SkyNodes SHALL implement the “Tables” interface, which returns a list of MetaTables for all tables that may be used in ADQL passed to this endpoint. The MetaTable SHOULD have the TableName, a description, the Primary Key, approximate Row Count, Rank, XmatchRegion (can this be Xmatch-ed and region searched) and a set of Relations which list ForeignKey and Table pairs for related Tables.

QI-5 SkyNodes SHALL implement the “Columns” interface, which returns information about the columns of a given table name. This SHALL return an array of MetaColumns. This structure SHALL contain the Column Name, DataType, Unit, ByteSize (number of bytes), Precision (number significant digits), Description, Rank, and UCD. DataType should be the type that would appear in the resulting VOTable if this column was requested.

QI-6 SkyNodes SHALL implement the “Column” Interface which takes a column name and a table name, and returns the MetaColumn information for that column.

This is needed for more interactive applications.

QI-7 SkyNodes SHALL implement the “Formats” interface. This takes no parameters and returns a list of formats which this Node supports for Query Results. Hence this MAY return VOTable, DataSet and ASCII.

QI-8 SkyNodes SHALL implement the “Functions” interface, which returns an array of MetaFunctions this structure SHALL contains the function name parameter list and comments for all extra functions this node supports. This also implies a new ADQL subclass which contains these functions.

QI-9 SkyNodes SHALL implement the "PerformQuery" interface, which takes an XML document including an ADQL query and an optional string parameter called “format”. Format MUST be one of the strings listed from a call to the "Formats" interface. This returns a document including a single appropriate IVOAResult, which is the result of processing the query. IVOAResult SHALL be an abstract class with multiple subclasses, one for each format. SkyNodes SHALL support at least VOTable format.

QI-10 SkyNodes SHOULD accept the VOQL equivalent of Standard SQL-92 Metadata [3] queries. These queries include (expressed here as SQL but would be ADQL/x coming to the node):

```
Select * from tables
Select * from colums
```

We need to decide how much of the entire set is relevant and required. Since most databases handle this we could just say all of it. The syntax may be a little different in each database i.e. in SQL-Server this is

```
Select * from information_schema.tables
```

It is noted that Oracle does not support Information Schema (as far as the author can tell).

QI-11 Built-in Functions

The set of required Skynode functions remains under discussion. Functions which are available may be listed using the Functions interface. The finalization of this list is work which the VOQL working group must complete.

4.2.2 Rank

Both for tables and columns it has been suggested some ordering mechanism is needed, e.g., how do we indicate to a user the most important columns, how do we get RA and DEC to appear near each other in a column list. In this version of the specification Rank has been introduced for Column and Table MeteData.
Rank is a grouping ordering Integer value, it would be ordered in reverse, i.e., highest number first no number = 0. This would also allow a sort of grouping for Low Medium High if we were so inclined by using number ranges e.g. 0-999=Expert, 1000-1999 = intermediate, 2000-2999 = Novice.
This does not solve the problem of specifying groupings, e.g., how do identify columns of interest to radio astronomers, cosmologists, optical astronomers. Perhaps this is not an issue.

4.2.3 Full SkyNode requirements

QI-12 Full SkyNodes SHALL implement QueryCost() interface which takes a simple ADQL query to return the object density per square degree for a set of criteria.

A particular survey with uniform density may choose to not run this as a full query but rather perform the query on a much smaller area or use statistics or heuristics. Basically this does not have to be 100% accurate it is an indication only. Better that it completes quickly.
On the other hand, theoretically, in some databases if the full investigative query were run then it will cause the data for the real query to be cached.
The notion here is to find out the volume of data which would be returned from the specified query. This allows nodes to be ranked in the correct order for a Xmatch.

QI-13 Full SkyNodes SHOULD accept complex Shapes in their queries as defined in the ADQL syntax (using the Region specification).

QI-14 Full SkyNodes SHALL be able to perform cross matching (Xmatch) between their survey and table of data provided in VOTable format. This will occur in a plan statement.

An example of the ADQL/s (which would be expected in ADQL/x format):

```
select ...
  from PhotoObj o, #upload up
  where XMatch(o, up, 3, 1)
```

Description of XMatch

An example for the cross-matching algorithm is a probabilistic calculation that minimizes the chi-square parameter as defined by:

$$\chi^2 = \frac{1}{2}\sum_n \alpha_n \left[(x - x_n)^2 + (y - y_n)^2 + (z - z_n)^2\right] - \frac{1}{2}\lambda\left[x^2 + y^2 + z^2 - 1\right]$$

where x, y, z are the Cartesian coordinates corresponding to the ra and dec specified by the user, **a** is a weighting parameter calculated from the astrometric precision of the survey, and λ is the Langrange multiplier in the minimization to ensure that the (**x,y,z**) is a unit vector. The code for spGetMatch is included in the Appendix to this document.

We compute four cumulative quantities at each cross-identification step – these are

$$a = \sum \frac{1}{\sigma_i^2}, \quad a_x = \sum \frac{x_i}{\sigma_i^2}, a_y = \sum \frac{y_i}{\sigma_i^2}, a_z = \sum \frac{z_i}{\sigma_i^2}.$$

The best position is given by the direction of (a_x, a_y, a_z). The log-likelihood at that point is given by

$$\chi^2 = a - \sqrt{a_x^2 + a_y^2 + a_z^2}$$

This is divided by the number of surveys considered up to that point, and compared to the tolerance. If a Tuple's log-likelihood exceeds this threshold, it is killed. This cross-identification process is fully symmetric, the particular order of matching does not matter. The cross-matching is applied to each node recursively by the portal when it runs the query execution plan.

A particular node may decide to implement this differently however if the chi-square parameters are not returned than other nodes wishing to use the above algorithm will not function correctly.

In dealing with tables the obvious approach is to load the table into a temporary table in the database. Here again other approaches may be adopted by the implementer. The interface is all that matters.

QI-15 Full SkyNodes SHALL implement an ExecutePlan() web method, which takes an ExecPlan and passes the relevant part of the plan to the next node. From that node it shall receive a VOData result. If there are no more nodes in the plan it simply executes the ADQL query and returns the resulting VOResult. The return type shall be specified in the plan at each step. The logical node and physical replicas SHALL be sent

with the plan. The ExecPlan structure containing the portalURL the plan came from and which is used for logging, the format the output is required in, the index of the current PlanElement, and an ordered array of PlanElements.

The PlanElement structure SHALL contain the statement (ADQL document) the Target for this element (a logical node name) and an ordered array of hosts (physical nodes which should behave as the logical name – there may be only one). The order of the hosts SHOULD be in most preferable first, this is of course form the portal perspective – a node MAY decide to re-rank the nodes for itself or simply send a quick query to all mirrors and take the first which responds. The PlanElement SHOULD contain the format of output required from this Node.

QI-16 Full SkyNodes SHOULD implement the footprint service. This would take a region specified in the region XML and return a new region which is the intersection of the survey and the given region. The Region specification has also the ability to deal with Spectral and Temporal footprints, this implies a node should be able to deal with these entities. Currently we know how to deal with regions (just about) but have not really dealt with temporal nor spectral overlaps.

The latest WSDL file is based on ADQL1.0, and it is located at
http://www.ivoa.net/xml/SNI/SkyNode-v1.0.wsdl

5 Changes from previous versions

- None. This is the first release.

6 References

[1] IVOA VOQL Working group; IVOA Astronomical Data Query Language http://www.ivoa.net/Documents/latest/ADQL.html

[2] IVOA Grid & Web Services Working Group; Support Interfaces; http://www.ivoa.net/internal/IVOA/IvoaGridAndWebServices/VOSupportInterfaces-0.22.pdf

[3] http://www.contrib.andrew.cmu.edu/%7Eshadow/sql/sql1992.txt

Appendix SPGETMATCH stored procedure

```
CREATE  PROCEDURE spGetMatch(@r float, @w float, @eps float)
-----------------------------------------------------------------------
--/H Get the neighbors to a list of @ra,@dec pairs in #upload in
photoPrimary
--/H within @r arcsec . @w is the weight per object.
--/U
--/T The procedure is used in conjunction with a list upload
--/T service, where the (ra,dec) coordinates of an object list
--/T are put into a temporary table #upload by the web interface.
--/T This table name is hardcoded in the procedure. It then returns
--/T a matchup table, containing the up_id and the SDSS objId.
```

```
--/T The result of this is then joined with the photoPrimary table,
--/T to return the attributes of the photometric objects.
--/T @r is measured in arcsec
--/T @w is the weight, it is 1/@sigma^2, where @sigma is in radians
--/T @eps is the chisq threshold
--/T <samp>
--/T <br> create table #x (pk int,id bigint,a float, ax float, ay
float, az float)
--/T <br> insert into #x EXEC spGetMatch  2.5, 0.0000001, ...
--/T </samp>
-----------------------------------------------------------------
AS
BEGIN
        SET NOCOUNT ON;
        SET IMPLICIT_TRANSACTIONS ON;
        DECLARE @pk int, @a float, @ax FLOAT, @ay FLOAT, @az FLOAT,
          @qx float, @qy float, @qz float, @b FLOAT, @rr float,
          @FETCH_STATUS INT;
        DECLARE ObjCursor CURSOR
           FOR SELECT  pk, xmatch_a, xmatch_ax, xmatch_ay,
xmatch_az
          FROM #upload;
        SET @FETCH_STATUS = 0;
        CREATE TABLE #x (
                pk  int,
                id bigint,
          chisq float,
          a float,
          ax float,
          ay float,
          az float
                );
        OPEN ObjCursor;
        WHILE (@FETCH_STATUS = 0)
          BEGIN
            WHILE(@FETCH_STATUS = 0)
              BEGIN
                FETCH NEXT FROM Objcursor INTO @pk, @a, @ax, @ay,
@az;
          SET @b  = @a*SQRT((@ax*@ax + @ay*@ay +
@az*@az)/(@a*@a)+0.00000001);
          SET @qx = @ax/@b;
          SET @qy = @ay/@b;
          SET @qz = @az/@b;
          SET @rr = @r /60;
                SET @FETCH_STATUS = @@FETCH_STATUS;
                IF (@FETCH_STATUS != 0) BREAK;
                INSERT INTO #x
                SELECT @pk as pk,
              objID as id,
              @a+@w-
sqrt(power(@ax+@w*cx,2)+power(@ay+@w*cy,2)+power(@az+@w*cz,2)) as
chisq,
              @a  + @w as a,
              @ax + @w*cx as ax,
              @ay + @w*cy as ay,
              @az + @w*cz as az
                     FROM dbo.fGetNearbyObjXYZ(@qx,@qy,@qz,@rr)
              WHERE
                 @a+@w-
sqrt(power(@ax+@w*cx,2)+power(@ay+@w*cy,2)+power(@az+@w*cz,2)) <
@eps
               END;
          END;
```

```
        CLOSE ObjCursor;
        DEALLOCATE ObjCursor;
        SELECT * FROM #x
--  SET IMPLICIT_TRANSACTIONS OFF
END
```