International

Virtual

Observatory

Alliance

IVOA SkyNode Interface Version 1.01

IVOA Working Draft 29 September 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:
Maria A. Nieto-Santisteban, William O'Mullane, Masatoshi Ohishi

Authors:
IVOA VOQL Working group

Abstract

This document describes the required interface that must be supported to participate in the IVOA as a BASIC or FULL SkyNode.

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 editors express their appreciation for many valuable contributions from the VOQL WG.

Contents

1 Introduction

Astronomical data (e.g., catalogs) may be published as SkyNodes. Data providers publish the SkyNodes by registering the services with the VO registry, so that they become discoverable by client applications (e.g., Open SkyQuery portal). SkyNodes provide the front-end to the actual databases, using the interface described in this document. At a fundamental level, SkyNodes may be called by client programs. At a higher level, portals (e.g., Open SkyQuery and JVO QueryPortal) provide access to SkyNodes, calling the registry to discover SkyNode services and building an execution plan that coordinates query processing [Figure 1].

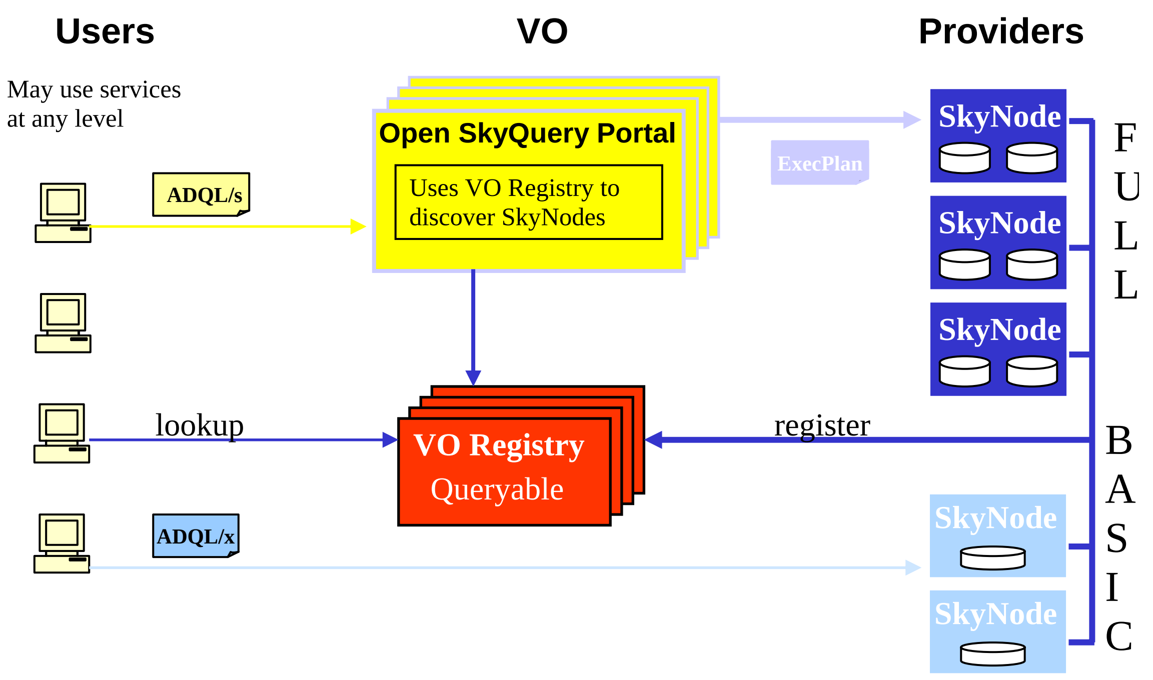

Figure 1. SkyNode query architecture

The Open SkyQuery Portal would accept a form of string based query, which is in fact ADQL/s (as shown in 0). 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.

The SkyNode concept originated with the SkyQuery portal (www.skyquery.net), which demonstrated the use of the Microsoft .NET framework and SOAP Web services to access federated astronomical databases. This document redefines SkyNodes to be SOAP Web services that are independent of the underlying framework and compliant with other IVOA standards. SkyNodes are Web services that may be implemented in at least Java/Axis and C#/.NET. From the perspective of the client, the framework used to implement a SkyNode is irrelevant, as demonstrated by the Open SkyQuery portal (www.openskyquery.net).

Clients interact with SkyNodes using ADQL (Astronomical Data Query Language), which is specified separately in the ADQL Specification Document [1]. ADQL is an SQL-like language used by the International Virtual Observatory Alliance (IVOA) to represent astronomy queries posted to VO data services (e.g., SkyNodes). ADQL has two forms: ADQL/s (string form) and ADQL/x (XML representation corresponding to ADQL/s). SkyNodes are XML Web services that accept ADQL/x queries, posted by clients or translated from ADQL/s by clients or portals (e.g., OpenSkyQuery.net). The ADQL Specification Document distinguishes between the mandatory core of the language and optional extensions that allow higher-level query capabilities.

SkyNode interfaces are defined by WSDL (Web Services Definition Language) documents, which specify the functional interface (methods) provided by SkyNodes. Data providers may chose to implement SkyNodes at the `BASIC` or `FULL` level. The main difference is that `FULL` SkyNodes must implement the ExecPlan method, which implies the capability of performing cross-matches. *There is an important distinction between the methods used to interface with a SkyNode and the ADQL language constructs supported by the SkyNode*. For example, QueryCost() is a method, whereas "SELECT COUNT(a.*) FROM SDSSDR3:PhotoPrimary a" is a language construct.

All SkyNodes (`BASIC` and `FULL`) shall implement the core ADQL specification. `FULL` SkyNodes must also implement specific ADQL extensions listed below (e.g., XMATCH functions). Depending on desired functionality, SkyNodes (`BASIC` and `FULL`) may implement additional ADQL extensions beyond those required at their level.

2 Standard Interface

SkyNodes are implemented as Web services and therefore must be compliant with the IVOA Support Interfaces document.

QI-1 SkyNodes SHALL implement the specifications in the IVOA Support Interfaces document (e.g. GetAvailability method) [2]

QI-2 SkyNodes SHOULD return errors as SOAP exceptions. There is no rceived 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.

3 IVOA SkyNode Interface

3.1 Feature Tables

The following two tables categorize methods and ADQL language features that must be supported at different SkyNode levels. These tables are meant as a summary, more explicit requirements are detailed below. The list of optional features is not complete, and it is likely to evolve.

SkyNode Methods	**Basic SkyNode**	**Full SkyNode**	**Optional**
Get Availability	X	X	
Functions	X	X	
ADQLExtensions	X	X	
QueryCost		X	
ExecutionPlan		X	
Footprint – Region intersect			X
VOStore			X

ADQL-Features	**Basic SkyNode**	**Full SkyNode**	**Optional**
Core	X	X	
Xmatch_chi2		X	
Xmatch_distance		X	
		X	
			X
			X

3.2 IVOA SkyNode Interface

The SkyNode interface includes methods that return Metadata with information about Tables, Columns, columns descriptions as UCDs (Uniform Content Descriptors) and units, functions, etc. When databases contain metadata, they can be exposed through ADQL, however is has been considered important to provide methods for querying the metadata using Web services calls.

Below we split the requirements in two sections – the minimum set required to be a `BASIC` SkyNode and more advanced requirements to be a `FULL` SkyNode. `FULL` SkyNode satisfy all `BASIC` SkyNode requirements. The Functions interface allows the node to specify what built-in functions supports. SkyNodes may implement ADQL extensions, which should be discoverable by calling the ADQLExtensions method.

3.2.1 Basic Sky Node

QI-3 SkyNodes SHALL register with the registry with type=”OpenSkyNode” according to the IVOA registry schema [??]

- Compliance: Basic or Full SkyNode
- MaxRecords: Maximum number of records returned, otherwise -1(unlimited)
- Latitude and Longitude: GPS coordinates of the Node
- PrimaryTable: Name of the main table in the archive
- PrimaryKey: Name of the primary key column (if exists, null otherwise). The primary key shall be a single column.
- AcceptsUCDs: True or False

[At time of writing http://www.ivoa.net/xml/OpenSkyNode/OpenSkyNode-v0.1.xsd is in use (although it does not accept UCDs).<-- check the status of this]

Comment: For columns with units, setting AcceptUCDs=True makes sense only if the units and type of the column are included in the Metadata.

QI-4 SkyNodes SHALL implement the “Tables” interface, which returns a MetaTable that contains the following information about the tables included in the SkyNode:

- TableName: Name of the table (String)
- Description: Text description of table contents (String)
- PrimaryKeyName: Name of the primary key, if one exists, otherwise a string with zero length (String)
- RowCount: Row count, if known, otherwise -1 (Long)
- Rank: Relative importance of the table (Small Int)
- XMatchSupp: Whether the table supports the XMATCH extension (Boolean)
- RegionSupp: Whether the table supports the REGION extension (Boolean)

QI-5 SkyNodes SHALL implement the “Relations” interface, which returns a MetaTable that contains information about relationships between tables to reflect foreign key constraints. This MetaTable shall include

- TableName1: Name of the first table (String)
- ColumnT1: Name of the column in TableName1 (String)
- TableName2: Name of the second table (String)
- ColumnT2: Name of the column in TableName2 (String)

QI-6 SkyNodes SHALL implement the “Columns” interface, which returns a MetaTable that contains the following information about the columns included in the specified table:

- ColumnName: Name of the column (String)
- Description: Text description of table contents (String)
- DataType: Data type that will appear in the resulting VOTable if this column is requested. (String)
- Units: Physical units of the data in the column, using standard IVOA nomenclature (reference?). Zero length string for dimensionless data (String)
- ByteSize: Number of bytes in the column (Small int)
- Precision: Number of significant figures [Is this numerical precision, number of digits to print, uncertainty in the physical measurement…?]
- Rank: Relative importance of the table (Small Int)
- UCD: Universal Content Descriptor. Zero length string if undefined or unspecified. (String)

QI-7 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-8 SkyNodes SHALL implement the "Formats" interface, which takes no parameters and returns a list of supported formats for Query Results. Valid values are VOTable, DataSet, and ASCII.

QI-9 SkyNodes SHALL implement the "Functions" interface, which takes no parameters returns a structure that contains the following information about supported ADQL functions:

- FunctionName: Name of the function (String)
- Description: Text description of table contents (String)
- ParamList: Array of parameter specifications
    - ParamName: Name of the parameter (String)
    - ParamType: Data type of parameter (String)
    - ParamDescription: Text description of the parameter (String)
- OutputType: Data type of function result (String)

This also implies a new entry in the SkyNode WSDL defining the Functions interface.

QI-10 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-11 SkyNodes SHOULD accept the ADQL 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 columns
```

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). [Should be mandatory. Make the metadata tables, even if DB does not provide intrinsic support.]

3.2.2 Rank

Rank is an integer that specifies the relative importance of tables or columns. Larger numeric rank indicates higher importance. Rank may facilitate construction of user interfaces, with precedence given to tables and columns with higher rank.

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.

3.2.3 Full SkyNode requirements

QI-12 Full SkyNodes SHALL implement the QueryCost() method, which returns the number objects in the table and region specified by the query. The results may be used to build efficient execution plans (e.g., a XMATCH). SkyNodes with uniform density tables may estimate QueryCost() statistically, using a subset of the requested region or using heuristics. The QueryCost() estimate does not have to be 100% accurate. The estimate should be insignificant compared to the time to execute the query.

QI-13 Full SkyNodes SHOULD be compliant with the REGION specification defined in the STC Region Specification Document [??].

QI-14 Full SkyNodes SHALL be able to perform a cross match (XMATCH) by participating in an execution plan. This involves accepting data from other SkyNodes, performing a cross-match with local data, and shipping the results.

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

4 Changes from previous versions

- None. This is the first release.

5 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
```