ANDROMEDA: Astronomical Data Mediation for Virtual Observatories

José-Luis Zechinelli-Martini, Genoveva Vargas-Solar, Victor Cuevas-Vicenttin

ANDROMEDA: building e-Science data integration tools

José Luis Zechinelli-Martini, Genoveva Vargas-Solar, Víctor Cuevas-Vicenttín

Research Center of Information and Automation Technologies, UDLAP
Ex-hacienda Sta. Catarina Martir s/n, San Andrés Cholula, México
LSR-IMAG, CNRS
BP 72 38402 Saint-Martin d'Hères, France
{joseluis.zechinelli, victor.cuevasv}@udlap.mx, Genoveva.Vargas@imag.fr

Abstract: This paper ANDROMEDA, an astronomical data mediation system that enables transparent access to astronomical data servers. Transparent access is achieved by a global view that expresses requirements of community of users (e.g., astronomers) and data integration mechanisms adapted to astronomical data characteristics. Instead of providing an ad hoc mediator, ANDROMEDA can be configured for giving access to different data servers according to different user requirements (data types, content, data quality, and provenance). ANDROMEDA can be also adapted when new sources are integrated to the community or new requirements are specified.

1 Introduction

In recent years, there has been a prodigious increase in the quantity of available data for astronomical research, due in part to (i) new technological breakthroughs in sensors and storage devices, and (ii) the increasing automation in data acquisition and processing. Consequently, in the years to come, astronomers will be able to generate calibrated data far more rapidly than they can process and analyze them.

New extended surveys like 2dF, SDSS, 2MASS, VIRMOS, and DEEP2 are revolutionizing the way astronomy is done; making available huge amounts of high quality data. To this avalanche of optical data, obtained via medium size telescopes, we must add the large databases that will be produced by the new generation of large ground and space telescopes in x-ray, optical, infrared and radio wavelengths. The nature of this databases (wide wavelength coverage), will represent an unprecedented challenge for the unification of optical observations in infrared, millimetrical and x-ray obtained from space and ground observatories, which also include radio-telescopes such as the GTM, which is now being built by INAOE and the University of Massachusetts in Cerro La Negra, Mexico at 4600 m. of altitude.

We cannot overemphasize the importance and complexity of this challenge. In the next five years, the available databases will grow to the point of tens of thousands of

parameters for hundreds of millions of astronomical objects, i.e., teradatasets. This complexity will be augmented by the existence of measurement errors, deviations and tendencies in the data, and above all the greatest difficulty is the absence of homologation between the different databases. Each one of which will have not only its own set of parameters but also its own access software, which will make extremely difficult the cross match between them [10].

In order to advance under the growing burden of the large amounts of data, we need applications to manage, ask queries, visualize, and analyze the entire space of large databases in an intelligent and automatic manner: a *Virtual Observatory* (VO). This is even more important for countries such as Mexico, which do not have access to the large international observational facilities [10]. The VO is a system in which the vast astronomical archives and databases around the world, together with analysis tools and computational services, are linked together into an integrated facility. Several VO projects are now funded through national and international programs, and all projects work together under the International Virtual Observatory Alliance (IVOA) to share expertise and develop common standards and infrastructures for data exchange and interoperability [4].

For the first time data acquisition and archival is being designed for online interactive analysis. Shortly, it will be much easier to download a detailed sky map or object class catalogue, than to wait several months to access a telescope that is often quite small. The several surveys that have been completed or are under way will yield together a Digital Sky of interoperating multi-terabyte databases. In time, more catalogues will be added and linked to the existing ones. Query engines will become more sophisticated, providing a uniform interface to all these datasets [7].

The objective of our work is to provide data integration mechanisms that can help astronomers to avoid the tedious task of manual data integration. Such mechanisms can be tuned for accessing specific collections of data servers according to different information requirements. This paper presents our approach for integrating astronomical data and the design and implementation of ANDROMEDA (Astronomical Data Resources Mediation).

The remainder of this paper is organized as follows. Section 2 introduces our approach for integrating astronomical data. Section 3 describes ANDROMEDA, its architecture, data servers and implementation issues. Section 4 describes an experimental validation that we conducted with real astronomical data. Section 5 compares related works with our solution. Finally, Section 6 concludes the paper and discusses future work.

1 Astronomical data integration

Data integration is a well known problem in the database domain. It essentially consists in combining data coming from different sources into a single and homogeneous

data set structured according to a so called mediation (global) schema. The mediation schema provides a homogeneous and generic view of a set of databases contents. Most data integration solutions are based on the descriptions of databases contents and their semantic relationships with the mediation schema.

Databases describe their content by exporting their schema. In our work each information source exports a local view of the data it contains, the *exported schema* expressed as XML schema. Figure 1 illustrates a tree representation of the exported schema of the Sky Server photometric data server. Elements and attributes are denoted by nodes. Attributes are circled, keys are underlined and an asterisk denotes multi-valued cardinality (multi-valued node). This schema defines that a *photoObj* element is composed of two complex elements: *coordinates* and *magnitude.* As well as a series of simple elements and attributes of which only the *id* attribute is presented in the figure. The *coordinates* element is composed of *ra* and *dec* elements. The *magnitudes* element consists of a sequence of *magnitude* elements. In turn, a *magnitude* element is composed of a *modelMag* and *modelMagError* elements as well as a *magnitudeName* attribute. The primary key of the *photoObj* element is the *id* attribute.

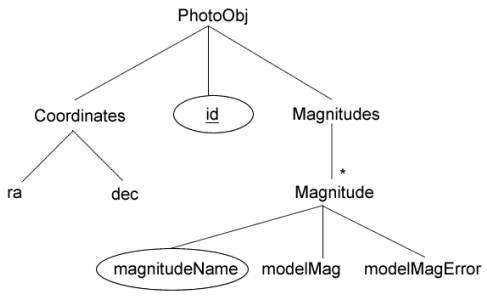

Fig. 1. Sky Server photometric data server exported schema

Data integration is possible as long as meta-information describing semantic correspondences[1] among data sources and mediation schemas (attribute equivalence), transformation functions and mediation query expressions enabling the population of the mediation schema, are available. Most approaches imply that such information and particularly mediation query expressions are manually defined. This is particular cumbersome when the number of sources increases. Therefore, inspired in [1, 5, 6], we propose an approach for automatically generating mediation queries for integrating astronomical data.

Astronomical data integration is executed in a series of phases:

- Identify for each data source which data elements are relevant. The relevant elements can be found provided it is established a priori which elements in the ex-

[1] A semantic correspondence establishes a relationship between two elements that refer to the same object of the real world.

ported schemata correspond to elements in the mediation schema. Documents that contain only the relevant elements should then be obtained and organized according to a so called *mapping schema*. The result of the first phase is a series of couples <mapping schema, intentional specification>. Each tuple associates a mapping schema (expressed as a tree) with its intentional specification expressed as an XQuery statement.

- Identify the operations (join, union) that can be used for integrating the results from the different sources and populated the mediation schema.
- Generate a mediation query expressed as an XQuery statement that intentionally expresses the content of the mediation schema in terms of local sources.

1.1 Mapping schemata generation

The objective of this phase is to identify for each exported schema a *mapping schema* (i.e., the set of nodes that are relevant for populating the mediation schema) and to compute an XQuery expression that specifies how to transform data from the exported schema to the *mapping schema*. For example, consider the exported and mediation schemata presented in Figure 2. The mediation schema is represented by the tree at the top, while the exported schema corresponding to the Sky Server spectra data is located at the bottom. The dashed lines represent semantic correspondences between elements.

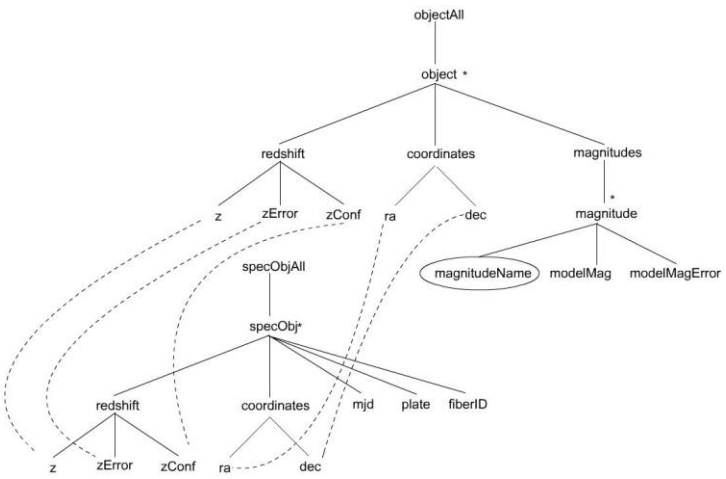

Fig. 2. Mediation and exported schemata

Looking for relevant nodes for building a mapping schema
A relevant node verifies at least one of the following conditions:

1. The node semantically corresponds to a node in the mediation schema.

2. The node does not have a corresponding node in the mediation schema, but its maximum cardinality is strictly greater than one; and at least one of its descendants has a semantic correspondence with a node in the mediation schema.
3. The node is key or foreign key of a relevant node in the exported schema.
4. The node does not have a corresponding node in the mediation schema, but it is the root of the exported schema and there is at least one node of this schema which semantically corresponds to a node in the mediation schema.

Accordingly, in order to find relevant nodes in an exported schema, the mediation schema tree is traversed first in preorder to find all the nodes that have a semantic correspondence with a node in the exported schema or that are `no-text` nodes. These nodes are used to create a new tree with a structure resembling that of the mediation schema. This procedure is carried out by the following recursive algorithm:

```
Input: Mediation schema tree
Output: Mapping schema tree

traverseTree(TreeNode currentNode, TreeNode medNode,
   Hashtable nodeCorresp, Hashtable nextCorresp) {
   //Make a copy of the mediation schema node to add
   //it to the tree if appropiate
   TreeNode nodeToAdd := medNode.copy();
   //Add all no-text nodes to the tree
   if( ~medNode.isTextNode() ) {
      currentNode.add(nodeToAdd);
      //Save the node pairs that have semantic
      //correspondences in a hashtable
      TreeNode corresp := nodeCorresp.get(medNode);
      if( corresp ≠ nil)
         nextCorresp.put(nodeToAdd, corresp);
   }
   //Add only the text nodes that have semantic
   //correspondences
   else if( nodeCorresp.containsKey(medNode) ) {
      currentNode.add(nodeToAdd);
      TreeNode node := nodeCorresp.get(medNode);
      nextCorresp.put(nodeToAdd, node);
   }
   else
      return;
   //Traverse the children of the current node
   //in preorder
   for each n ∈ medNode.children()
      traverseTree(nodeToAdd, n, nodeCorresp,
      nextCorresp);
}
```

Figure 3 shows the resulting *mapping schema* for the Sky Server example. The *objecAll*, *object* and *redshift* nodes are first added to the new tree since they are `no-text` nodes. Then the children of the *redshift* node: *z*, *zError*, and *zConf*; are added because they are `text-nodes` with semantic correspondences with nodes in the mediation schema (see semantic correspondences in Figure 2). Next, the *coordinates* element is added, since it is a `no-text` node as well as its children *ra* and *dec*, having correspondences with nodes in the mediation schema. Finally, the `no-text` nodes *magnitudes* and *magnitude* are added, because their children do not have semantic correspondences with nodes in the mediation schema.

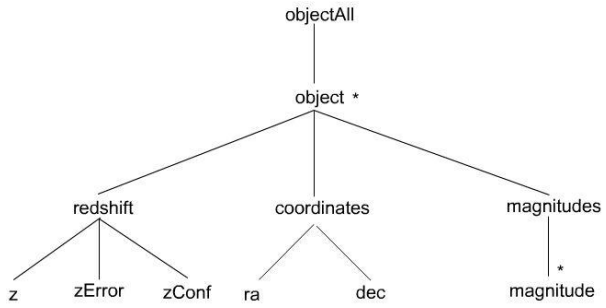

Fig. 3. Generated Sky server mapping schema after the first step of the algorithm

Next it is necessary to traverse the new tree in postorder to eliminate all the remaining `no-text` leaves of the tree that do not have semantic correspondences with nodes in the mediation schema. For this purpose, the hash table `nextCorresp` has been created to keep track of all the nodes that have a semantic correspondence with a node in the mediation schema. The algorithm follows:

```
removeNodes(TreeNode mappingNode, Hashtable
  nextCorresp) {
   //Traverse the children of the current node
   //in postorder
   for each n ∈ mappingNode.children()
      removeNodes(n, nextCorresp);
   //If the node is a leaf and does not have a semantic
   //correspondence with a node in the mediation schema
   //remove it from the tree
   if( mappingNode.isLeaf() and
    ~nextCorresp.containsKey(mappingNode) ) {
         TreeNode parent = mappingNode.getParent();
         parent.remove(mappingNode);
   }
}
```

By applying the algorithm to the tree in Figure 3, the children of the *redshift* element, are kept in the tree because they have semantic correspondences with nodes in the

mediation schema. The same principle applies to the children of the *coordinates* node. However, the child of the *magnitudes* element, *magnitude*, is a leaf node and does not have a semantic correspondence with a node in the mediation schema, so it is removed. After the removal of the *magnitude* node, its parent, the *magnitudes* node becomes a leaf node itself and it is removed for the same reason as its child. This algorithm yields the tree presented in Figure 4.

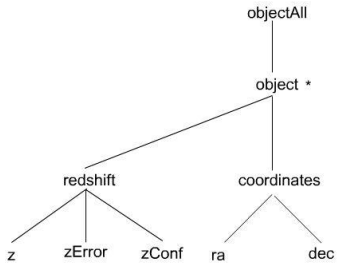

Fig. 4. Final Sky Server mapping schema

Intentional expression of a mapping schema
Finally, this mapping schema tree is traversed for creating its intentional description as a XQuery expression. The algorithm to generate this expression is presented in appendix A. The derived mapping XQuery expression for this example looks as follows:

```
<objectAll>{
   for $var1 in
   doc("SkyServerSpectraData.xml")
   /specObjAll/specObj return
   <object>
      <redshift>
         <zConf>{$var1/redshift/zConf/text()}</zConf>
         <zError>{$var1/redshift/zErr/text()}</zError>
         <z>{$var1/redshift/z/text()}</z>
      </redshift>
      <coordinates>
         <dec>{$var1/coordinates/dec/text()}</dec>
         <ra>{$var1/coordinates/ra/text()}</ra>
      </coordinates>
   </object>
}</objectAll>
```

1.2 Candidate operations derivation

Once the mapping schema and its associated XQuery statement have been created for each source, it is necessary to define which operations should be used for integrating them. The join and union operators are frequently used for integrate the data from dif-

ferent sources [1, 5, 6]. The result of this phase it the set of candidate operations, that is, all possible combinations of mapping schema expressions, that can be used for populating the mediation schema by integrating data from a set of sources.

In the case of astronomical data servers, usually no primary keys can be used for uniquely identifying the same object. In our solution we consider a spatial join, which identifies entries of the same object in different data servers based on their position (right ascension and declination coordinates). However due to the variability on the position measured by each survey, which is worsened by different error magnitudes for each survey, object positions are approximated and thus the same object can have different positions in different data servers. We propose a special method to determine whether two objects found on different surveys refer to the same physical object. The method consists of computing the distance between two objects from different surveys and compares it against a threshold supplied by the user:

$$dist\ (obj_{\ i}, obj_{\ i+1}) < \varepsilon \quad \forall i \in \{1..n\}$$

Whenever the distance is less than the threshold, the two objects are considered to be the same. The advantage of this solution is that it is easy to implement, since it only requires a condition in the `where` clause of the query and it can be extended to an arbitrary number of data sources. It has the disadvantage that the error magnitudes of each survey are not taken into consideration. Methods that are more sophisticated have been proposed, for example the Xmatch algorithm for Sky Query. In principle, the Xmatch or a similar probabilistic algorithm can be adapted to our system.

1.3 Mediation queries generation

The objective of this phase is to generate a set of mediation queries expressions that specify the mediation schema intention. In our case only consider the spatial join described earlier and the resulting mediation query is an XQuery statement. Given a mediation schema, this statement is generated by traversing the mediation schema in preorder. The basic idea is to merge the nodes from each mapped source according to the mediation schema.

However, not all nodes should be handled in the same way. Since a valid XML document should have only one root element, this is taken from the mediation schema. The *coordinates* elements require special treatment since the same object has approximately the same position in different data servers. In order to provide a global view over the data, it is appropriate to associate the same position coordinates values for each integrated object. A straightforward approach is to take the average of the measures of the object's position for all the sources involved. Certain elements involve difficulties as well, for example, some multi-valued nodes are grouped within a single node (e.g. *magnitude* elements in a *magnitudes* element). In this case the multi-valued nodes from the different sources should be grouped together; otherwise the syntax of the mediation schema would be violated. The complete algorithm is presented in appendix B. The generated mediation XQuery expression for the integration

of the exported schemata of the SkyServerPhoto and SkyServerSpectra according to the mediation schema in Figure 2 follows:

```
<objectAll>{
for $var1 in
doc("SkyServerPhotoMappedData.xml")/objectAll/object
for $var2 in
doc("SkyServerSpectraMappedData.xml")/objectAll/object
where ( andromedaf:distance(
number($var1/coordinates/ra/text()),
number($var1/coordinates/dec/text()),
number($var2/coordinates/ra/text()),
number($var2/coordinates/dec/text()) ) < 0.08 )
return
   <object>
      <magnitudes>
         {for $var3 in $var1/magnitudes/magnitude
          return $var3}
      </magnitudes>
      <coordinates>
         <ra>{(number($var1/coordinates/ra/text()) +
            number($var2/coordinates/ra/text()) div 2}
         </ra>
         <dec>{(number($var1/coordinates/dec/text()) +
           number($var2/coordinates/dec/text())
           div 2}
         </dec>
      </coordinates>
      <redshift>
         <zConf>{$var2/redshift/zConf/text()}</zConf>
         <zError>
            {$var2/redshift/zError/text()}
         </zError>
         <z>{$var2/redshift/z/text()}</z>
      </redshift>
   </object>
}</objectAll>
```

2 ANDROMEDA

ANDROMEDA is an astronomical data mediation system for integrating astronomical data sources used in the Mexican Virtual Observatory built at INAOE. Users specify their information needs through a mediation schema created with their expert knowledge and expressed using XML Schema language. The mediation schema includes the data elements of interest that can be retrieved from one or many of the astronomical data sources (e.g., the right ascension and declination coordinates). Queries are ex-

pressed by specifying the right ascension and declination coordinates of the centre of the circular area, the radius of the circle, a tolerance value for the spatial join, and the data sources. The coordinates are given in decimal format according to the J2000 epoch convention and the radius and tolerance values in arc minutes (see Figure 5). When the user clicks the data button, an HTTP request is created using the GET method to pass the parameters to the mediator, which generates and returns the results. ANDROMEDA has been implemented in Java and it uses SAXON 8.0-B for executing XQuery expressions.

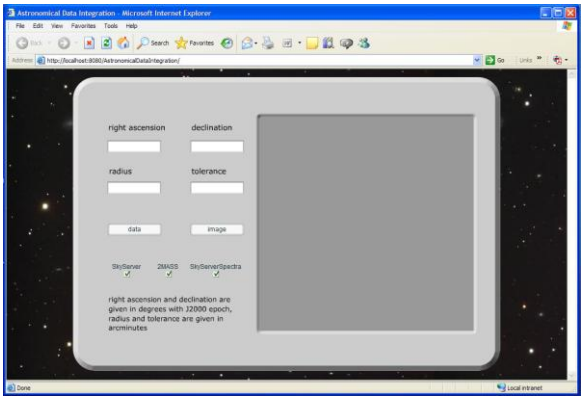

Fig. 5. ANDROMEDA user interface

2.1 Architecture

Figure 6 illustrates the architecture of ANDROMEDA. The system follows a wrapper-mediator architecture [11]. In this architecture, the mediator provides a uniform user interface to query integrated views of heterogeneous information sources while wrappers provide local views of data sources in a global data model [5]. In ANDROMEDA users express their queries following a mediation schema that provides a unified global view of the local data sources and describes information requirements provided by an expert. The ANDROMEDA mediator adopts the semi-structured data model as pivot model.

The key aspect in ANDROMEDA is the automatic generation of meta data describing mediation queries used for integrating data (populating completely or partially de mediation schema). Such data are essential during the integration process, especially when the number of sources is important: the more semantic knowledge is available the easier and more precise is data integration with respect to application requirements.

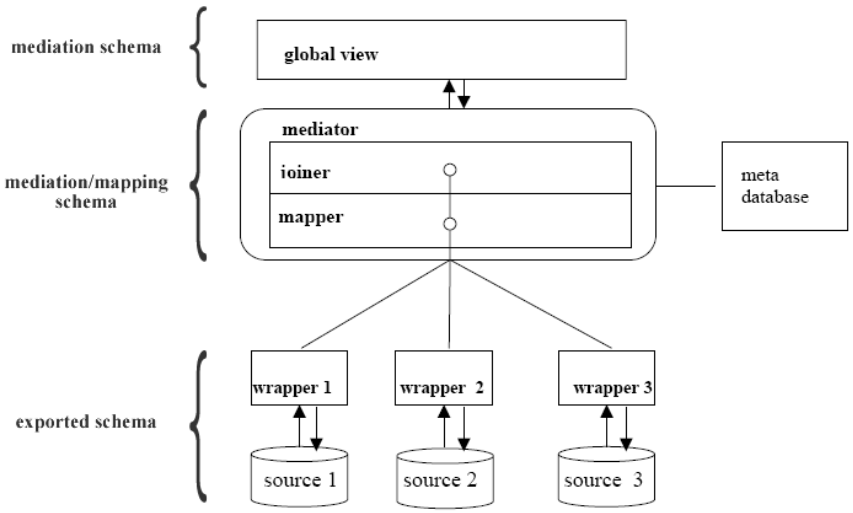

Fig. 6. ANDROMEDA general architecture

2.2 Data Servers and wrappers

The current version of ANDROMEDA works with three data sources, although the system is designed in such a way that can be extended to an arbitrary number of data sources. These are the Sky Server [7] photometric data, Two Micron All Sky Survey [8] photometric data, and Sky Server spectrographic data[2].

The sources integrated by the current version of ANDROMEDA are accessible via the WWW using the standard HTTP protocol. Each of them provides an HTML form in which the user can specify the area of the sky of interest providing the coordinates of the centre of a circle and its radius. The results can be obtained in several formats, including HTML, text and XML.

In order to obtain data from a source, the wrapper creates an HTTP request with the coordinates and radius parameters provided by the user. Then a connection is established to send the request and retrieve the data from the response. Since the sources of our implementation return data under the XML format, an XSL stylesheet provides the means to transform the data to the format required by the exported schema defined using XML Schema.

[2] Although both, the photometric and spectrographic data of the Sky Server [8] are accessible by a single interface, for purposes of design and testing we considered and implemented them as separate data sources.

2.3 Mediator

The mediator gives a global view of a set of data sources while making transparent their heterogeneity by automating data integration. The mediator transforms each user query expressed with respect to the mediation schema into a set of sub-queries expressed on the exported schemata; it dispatches sub-queries to sources, retrieves partial results and integrates them into a final result expressed according to given user requirements.

When the user sends a query, the `Mapper` creates the mapping schemata for each of the registered sources according to the mediation schema. Then a query is dispatched to each of the local sources and the results are transformed to its corresponding mapping schema and saved in temporary XML files. Finally the mapped results from the local sources are integrated by an XQuery expression generated by the `Joiner` and the final results are presented to the user.

2.4 Meta database

In order to integrate data, the system requires an appropriate meta-database, which initially contains descriptions of exported schemata, the mediation schema, and the semantic correspondences between them (see Figure 7). The meta-database stores also a set of mediation queries that are automatically generated and can be combined for totally or partially populating the mediation schema.

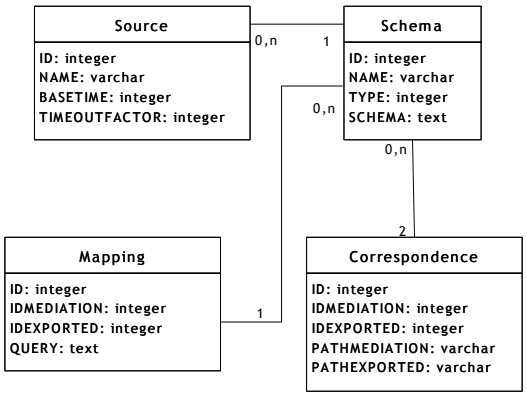

Fig. 7. UML diagram of the meta-database schema

A schema is identified by its id, a name and its full XML Schema file. Semantic relationships among schemata are described by semantic correspondences that are also stored in the meta-database. The meta-database of ANDROMEDA was implemented using MySQL Database Server.

3 Experimentation

In order to have a realistic experiment to validate our ANDROMEDA, the INAOE provided us with a list of objects in the SDSS of particular relevance in their research. The objective was to obtain the data for each of these objects available from both Sky Server and 2MASS. For the spatial join procedure, we used a value of 0.0014 degrees (aprox. 5 seconds of arc) as it was suggested. Since the position given is exactly that measured by SDSS, we only needed to give a radius similar to the spatial join tolerance itself. In this case we used a radius of 0.1 arc minutes. For the list contained 409 objects, matches were found for 322 of them, and within these 322 objects, multiple possible matches were found for 71 objects. The total execution time was approximately 20 minutes.

For determining the global query execution time of our solution we tried to extensively integrate as much objects as possible from SkyServerPhoto, TwoMASS [9], SkyServerSpectra, by defining increasingly larger regions. The evaluation was done in a Dell Inspiron 8600, Intel Pentium M 1.50GHz, 512MB RAM. Of course the first observation is that our algorithm is polynomial and that execution time increases with respect to the area of the specified regions (see Figure 8).

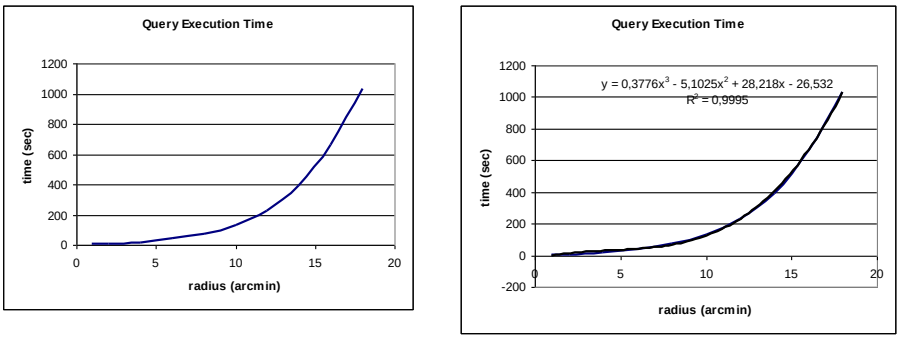

Fig. 8. Query execution behavior

Other important information concerning our experiment is shown in Figure 9. For a 2 arc minutes radio, results contain respectively 143 objects from Skyserver photo (139983 bytes), 10 objects from TwoMass (6595 bytes) and 2 from SkyServerSpectra (555 bytes). Only two objects could be integrated (those whose distance was less than a 0.0014 threshold) which represented (3284 bytes). The largest radio that could be processed under the specified conditions was 18 arc minutes, representing 60 integrated objects corresponding to (96570 bytes). Our execution time is reasonable even if it will be penalizing for large sky areas it can remain interesting even if the number of sources increases. We are currently running extensive experimentations for integrating data according to the set of queries proposed by J. Gray for SkyServer. Such queries were initially specified for the SkyServer multidatabase and thus no integration is considered. We are adapting them for testing ANDROMEDA.

radius (arcmin)	time (msec)	time (sec)	retrieved objects				file sizes (bytes)			
			SkyServerPhoto	TwoMASS	SkyServerSpectra	Global	SkyServerPhoto	TwoMASS	SkyServerSpectra	Global
1	6599	6,599	143	10	2	2	139983	6595	555	3284
2	5598	5,598	508	22	2	2	496980	14439	555	3284
4	17655	17,66	1616	79	4	4	1580806	51678	1043	6502
6	40198	40,2	2755	144	10	8	2695023	94134	2506	12939
8	74888	74,89	3859	234	18	12	3775458	152957	4460	19367
10	131949	131,9	5173	332	27	19	5060981	216997	6647	30616
12	228839	228,8	6700	444	37	24	6554558	290199	9084	38663
14	394297	394,3	8538	594	49	37	8352454	388212	11989	59540
16	663905	663,9	10539	777	60	47	10309689	507790	14661	75632
18	1034667	1035	12746	961	74	60	12468276	628033	18059	96570

Fig. 9. Query results

Finally, it is important to note that providing transparent access to astronomical data sources and enabling data integration must be evaluated under qualitative criteria. It is always interesting to automate a process that is in general done manually.

4 Related work

Several approaches have been proposed for astronomical data integration and the construction of the Virtual Observatory. SkyQuery [3] is maybe the most representative prototype of a federated database application, implemented using a set of interoperating Web services. In SkyQuery a portal provides an entry point into a distributed query system relying on metadata and query services of the database SkyNodes. Such database consists of individual databases representing a particular survey located at different sites along with their wrappers. SkyQuery receives queries expressed in SQL, it locates the referenced SkyNodes and submits the query to every SkyNode. The final result is computed by a applying a probabilistic spatial join to partial results.

Recently, the SkyQuery architecture is being redesigned (Open SkyQuery project [2]) for the emerging VO standards such as the VOTable, the Astronomical Data Query Language (ADQL), the VO Query Language (VOQL) and the VO Registry services. A limitation in SkyQuery is that it does not provide a unified global view over the set of data sources, so the user needs to be familiar with the schema of each data source in order to build the query and interpret results. Yet, when the number of sources increases it is difficult to imagine that a user will easily remember details about each server. Even if the number of sources is reduced, astronomical data formats and are highly heterogeneous. Even values referring to the same astronomical objects can vary depending on the type of instrument used for observing the sky.

Most existing astronomical automatic data integration approaches concern homogeneous relational data servers. However, astronomical data integration is still done manually especially when heterogeneous data are involved. The challenge is to provide an integrated view over a set of astronomical data sources, while making trans-

parent to the users the heterogeneity of these sources. In addition, it is desirable to have the ability to easily incorporate new data sources.

5 Conclusions and future work

The methodology and algorithms presented in this paper provide a reasonable solution to the integration of distributed, heterogeneous and autonomous data sources in the domain of astronomy. In addition, the inclusion of new data sources can be achieved with relative ease. Our prototype implementation demonstrates that our approach provides a viable method for astronomical data integration. The system provides users a unified global view over the set of data sources, while it hides the specificities of each source. The particularities of astronomical data, mainly the possibility to identify objects by their position and to query data sources by specific areas of the sky, proved to be more helpful than challenging. Still the query evaluation algorithms can be improved by using a formal XML algebra. There is also room for improvement in the spatial join procedure, which could be modified to take into consideration the specificities of each data source in order to lead to more reliable cross identification of objects among different surveys.

References

[1] Bouzeghoub M., Farias Lóscio B., Kedad Z., Soukane A., Heterogeneous Data Source Integration and Evolution. *In Proceedings of the 13th International Conference, DEXA 2002*, Aix-en-Provence, France, September, 2002,

[2] Budavári T., Szalay A., Malik T., Thakar A., O'Mullane W., Williams R., Gray J., Mann B., and Yasuda N. Open SkyQuery - VO Compliant Dynamic Federation of Astronomical Archives, *In Proceedings of the ADASS'03*, 2003

[3] Budavári, T. Malik, A. Szalay and A. Thakar. T. SkyQuery – A Prototype Distributed Query Web Service for the Virtual Observatory, *In Proceedings of the ADASS'02*, 2002

[4] Hanisch R. J. and Quinn P. J. *The IVOA*. http://www.ivoa.net/pub/info/TheIVOA.pdf

[5] Kedad, Z., Xue, X. Mapping generation for XML data sources: a general framework, *In Proceedings of WIRI 2005*, in conjuction with ICDE'05, 2005

[6] E. Métais, Z. Kedad, I. Comyn-Wattiau and M. Bouzeghoub, "Using Linguistic Knowledge in View Integratio: toward a third generation of tools", *International Journal of Data and Knowledge Engineering (DKE)*, 1 (23), North Holland, 1997.

[7] Szalay A., Gray J., Kunszt P., and Thakar A. Designing and Mining Multi-Terabyte Astronomy Archives: The Sloan Digital Sky Survey. *In Proceedings of SIGMOD*, 2000.

[8] Szalay A., Kunszt P., Thakar A., Gray J., and Slutz D. The Sloan Digital Sky Survey and its Archive. *In Proceedings of the ADASS IX*, eds. N. Manset, C. Veillet, D. Crabtree, (ASP Conference series), 2000

[9] 2MASS *Explanatory Supplement to the 2MASS All Sky Data Release*. http://www.ipac.caltech.edu/2mass/releases/allsky/doc/explsup.html

[10] Terlevich R., López López A. and Terlevich, E. *El Grupo de Ciencia con Observatorios Virtuales del INAOE*, 2003. http://haro.inaoep.mx/ov/archivos/ObservatorioVirtual.pdf.

[11] Wiederhold, G., "Mediators in the architecture of future information systems", *Computer*, pages 38-49, 1992.

Appendix A

Mapping schema query generation algorithm.

```
createQuery(TreeNode mappingNode, HashMap nextCorresp){
//The buffer will keep track of the expression
buffer :=  “”;
//Use a stack to traverse the tree in preorder
Stack nodesStack ;
//Take the root node as the parent of the key node
TreeNode keyNodeParent := keyNode.getParent();
rootElementName: = mappingSchemaNode.getName();
//Add the root element to the XQuery statement and take //its path
buffer += "<" + rootElementName + ">{ ";
keyNodeParentPath : = keyNodeParent.getPath();
//Keep track of the variables in the statement
varNumber := 1;
//Generate the desired indentation
indent :=  "     ";
//This expression refers to the node that contains
//the key
buffer += indent + "for $var" + varNumber + "in doc(\""
+ exportedName + "Data.xml\")" + keyNodeParentPath + " return";
//Put this node in the stack
nodesStack.push(mappingNode);
//Keep track if the node being processed is the root
isRoot := true;
//Keep track of when the first multi-valued node has //been found
firstMultiValuedFound := false;
//Store the elements that need to be closed in the //statement
Stack tagsToClose;
//Keep track of the node the current variable refers to
SchemaNode currentVarNode := keyNodeParent;
//Use a stack to keep the scopes of each variable
//related node
Stack varNodes;
varNodes.push(currentVarNode);
//Traverse the tree in preorder
while( ~nodesStack.isEmpty() ) {
   //Take the next item from the stack
   o := nodesStack.pop();
   nodeName := null;
   TreeNode node = null;
   //Check if this item is a node or a level mark
   //If it is a node then take its name
   if( o ≠  “⊥”) {
      nodeName :=  node.getName();
   }
   //If it is a level marker close the current tag
   //if needed, reduce the indent and change the
   //current variable node
   else {
      if( ~tagsToClose.isEmpty() ) {
         closeTag := tagsToClose.pop();
         indent -= “     “;
         buffer += indent + closeTag;
         if( closeTag.indexOf( "}") ≠ -1 ) {
            varNodes.pop();
            currentVarNode = varNodes.peek();
```

```
         varNumber--;
      }
   }
   continue;
}
//Check if the node being processed is not the root
if( ~isRoot) {
   //If the node is a text node put this element
   //in the statement and make a reference to
   //its contents
   if( node.isTextNode() ) {
      buffer += indent + "<" + nodeName + ">";
      TreeNode exportedNode:=nextCorresp.get(node);
      nodePath := exportedNode.getPath();
      expression := "{$var" + varNumber +
      t := currentVarNode.getPath().length();
      nodPath.substring(t);
      if( ~node.isAttribute() )
         expression += "/text()";
      expression += "}";
      buffer += expression;
      buffer += "</" + nodeName + ">";
   }
   //Check if the node is a multi-valued node
   else if( node.cardinality() > 1 ) {
   //if the first multi-valued node has already been
   //found, then create a new variable
   //with a for expression
   if( firstMultiValuedFound ) {
      varNumber++;
      oldNodePath := currentVarNode.getPath();
      TreeNode corresp := nextCorresp.get(node);
      if( corresp = null )
         error("no matching node found");
      currentVarNode: = corresp;
      varNodes.push(corresp);
      nodePath = currentVarNode.getPath();
      forExpression := indent + " { for $var" +
      varNumber + " in $var" + (varNumber-1) +
      nodePath.substring( oldNodePath.length()) +
      " return";
      buffer += forExpression;
      buffer += (indent + "<" + nodeName + ">";
      tagsToClose.push("</" + nodeName + ">" +
      indent + "}");
   }
   //If this is the first multi-valued node in the
   //tree then just add the element tags
   else {
      buffer += indent + "<" + nodeName + ">";
      tagsToClose.push("</" + nodeName + ">");
      firstMultiValuedFound := true;
   }
}
//For no-text and not multi-valued nodes just
//add the element tags
      else {
         buffer += indent + "<" + nodeName + ">";
         tagsToClose.push("</" + nodeName + ">");
      }
```

```
        }
        isRoot := false;
        //If there is next level in the subtree add a
        //level marker and increase the indent
        E := node.children();
        if( |E| > 1 ) {
           nodesStack.push("⊥");
           indent.append("     ");
        }
        //Add all the children of the current node
        //to the stack
        for each e ∈ E
        nodesStack.push(e);
    }
    //Close the root element
    buffer += "}</" + rootElementName + ">";
    return buffer;
}
```

Appendix B

Mediation query generation algorithm.

```
createJoinQuery(TreeNode mappingSchemas[],
  TreeNode mediationSchemaNode){
  //Use a buffer to keep track of the generated statement
  buffer := ‟”;

  //Add the root element to the expression
  buffer += "<" + mediationSchemaNode.getName() + ">{";
  vars := 0;

  //Save the paths of the root elements of each
  //mapping schema

topPaths[];
  //Save the paths of the ra and dec elements of
  //each mapping schema
  raPaths[];
  decPaths[];
  for i := 0 to mappingSchemas.length {

   vars++;
     mappingSchemas[i] =
     mappingSchmas[i].getFirstChild();

   file := mappingNames[i] + "MappedData.xml";

   topPaths[i] := mappingSchemas[i].getPath();
     buffer += "for $var" + vars +
     "in doc(\"" + file + "\")" + topPaths[i];
     TreeNode ra :=
     getDescendantByName(mappingSchemas[i], "ra");
```

```
    raPaths[i] := ra.getPath();
    TreeNode dec := getDescendantByName(

  mappingSchemas[i], "dec");

  decPaths[i] := dec.getPath();
  }
  //Add the clause that carries out the spatial join
  buffer += createWhereClause(mappingSchemas, topPaths,

raPaths, decPaths));
  buffer += "return";
  //Use as stack to traverse the tree in preorder in
  //a non-recursive manner
  Stack nodesStack;
  nodesStack.push(mediationSchemaNode.getFirstChild());

//Use a stack to keep track of the elements
  //that need to be closed
  Stack tagsToClose;
  //Keep track of the indent the statement should have
  indent = "     ";
  //Keep track of wheter the first multi-valued node
  //has been found
  boolean firstMultiValuedFound := false;
  //Traverse the tree in preorder
  while( ~nodesStack.isEmpty() ) {
     //Take the next item from the stack
     o = nodesStack.pop();
     String nodeName := null;
     SchemaNode node := null;
     //Check if this item is a node or a level marker
     if( o ≠ "⊥" ) {
        //If it is a node then take its name
        nodeName = node.getName();
        //If it is the coordinates element add the
        //associated special expression
        if( nodeName = "coordinates" ) {
           buffer += createCoordinatesExp(indent,
           topPaths, raPaths, decPaths));
           continue;
        }
        if( node.cardinality() > 1 ) {
           //If it is a multi-valued node check if
           //the first multi-valued node has already
           //been found.
          //If that is the case generate new
          //variables with for expressions

               if( firstMultiValuedFound ) {

           for i := 0 to mappingSchemas.length {

              TreeNode nodei = getDescendantByName(
                mappingSchemas[i], nodeName);
                //Add the for expression only
```

```
                  //if the node is found in the

                  //mapping schema
                 if( nodei ≠ null) {
                   nodeiPath = nodei.getPath();
                   buffer += indent + "{for $var" +
                   (vars+1) + " in $var" + (i+1) +
                   nodeiPath.substring( to
                   Paths[i].length + " return $var" +
                   (vars+1) + "}";

                 vars++;
                 }
              }
              continue;
           }
           firstMultiValuedFound := true;
        }
        //For text-nodes add expressions to take their
        //values if they are found in the mapping schemas
        if( node.isTextNode() ) {
           for i := 0 to mappingSchemas.length {
           TreeNode textNode = getDescendantByName(
           mappingSchemas[i], nodeName);
           if(textNode ≠ null) {
              textNodePath = textNode.getPath();
              textNodeExp = "{$var" + (i+1) +
              textNodePath.substring(
              topPaths[i].length()) + "/text()}";
              buffer += indent + "<" +
              nodeName + ">" + textNodeExp + "</" +
              nodeName + ">";
            }
         }
      }
      //For no-text and not multi-valued nodes add
      //only the element tags
      else {
         buffer += indent + "<" + nodeName + ">";
         tagsToClose.push(indent+"</"+nodeName+">";
      }
   }
   //If a level marker is found close the current tag
      //and reduce the indent
      else {
      if( ~tagsToClose.isEmpty() ) {
         closeTag := tagsToClose.pop();
         indent -= "     ";

               buffy.append(closeTag);
      }
      continue;
   }

   //If there is a next level in the subtree

   //add a level marker and increase the indent
```

```
      E := node.children();

      if( |E| > 1 ) {
         nodesStack.push("⊥");

         indent += "     ";
      }

      //Add all the children of the current node
      //to the stack
      for each e ∈ E
         nodesStack.push(e);
      }
      //Close the root element
      buffer += "}</" + mediationSchemaNode.getName()
      +        return buffer;
}
```