Astronomical Data Analysis Software and Systems XIII
ASP Conference Series, Vol. 314, 2004
F. Ochsenbein, M. Allen, and D. Egret, eds.

Flexible Storage of Astronomical Data in the ALMA Archive

Holger Meuss, Andreas Wicenec

European Southern Observatory

Simon Farrow

University of Manchester, Institute of Science and Technology

Abstract.

The requirements for the archiving of ALMA observation data are challenging: Not only are the expected rates of observation and monitor data extremely high (0.5 TeraByte/day), there is also the need to archive metadata about projects, proposals, observations, scheduling blocks, etc. in a flexible way that allows for changes in the structure of these data over the years.

The ALMA archive is divided conceptually into three parts: (1) The BulkStore for the very observation data, (2) the MonitorStore for monitor data collected by all instruments, and (3) the XMLStore for metadata about observation and monitor data. The entities in the three distinct stores are highly interrelated.

We will give an overview over the architecture of the ALMA archive with a special focus on XML storage. XML (eXtended Markup Language) was chosen not only as format for communicating data in the ALMA computing infrastructure, but also for archiving data, since it provides the required flexibility needed by the ALMA archive: XML is designed to represent semistructured data, i.e. data whose structure is irregular, changing over time or even unknown. This makes it the format of choice for software that has to work over many years, when changes in the underlying data structures are unavoidable.

1. Introduction

The ALMA Archive stands in the center of the ALMA dataflow: Most ALMA subsystems exchange data in the form of XML documents (for metadata) or binaries (for monitor and observation data) via the Archive (Wicenec, A. et al. 2004). XML is the format of choice for representing and exchanging data in the ALMA computing infrastructure. This makes it easy to change the structure of data, a circumstance unavoidable especially in the first years of operations. This flexibility has to be paid by the higher effort for storing XML persistently: Storage technologies for XML data are by far less mature as they are for relational data as elaborated in the following sections.

2. The Architecture of the ALMA Archive

As illustrated in Figure 1, the interface of the ALMA Archive provides three access ways for storing (1) observation data, (2) monitor data, and (3) metadata. Internally, these three kinds of data are distributed to two storage areas:

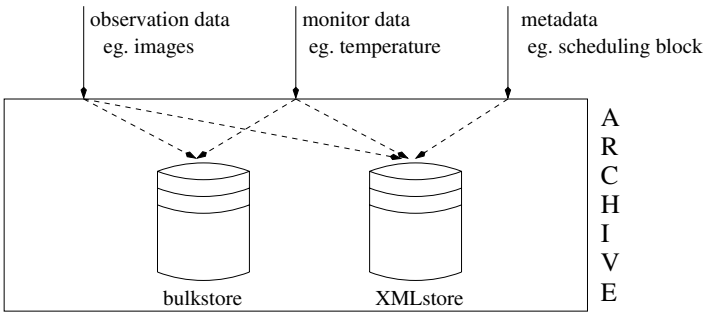


Figure 1. The ALMA Archive

observation and monitor data go to the bulkstore, whereas the metadata is sent to the XMLstore. Note that observation data as well as monitor data always have a header containing metadata, which is stored in the XMLstore, too.

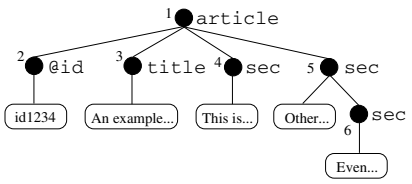
The bulkstore essentially consists of NGAS, a scalable file store (Wicenec, A. et al. 2002) allowing for efficient storage and retrieval of mass data on magnetic disks.

All metadata like scheduling blocks, observing proposals, etc. as well as the metadata for the very observations is stored in XML format in the XMLstore. Since the observing tool, supporting investigators in preparing an ALMA observation, is required to run locally, i.e. disconnected from the ALMA computing environment in general and the ALMA archive in special, two versions of the XMLstore are implemented: one in the central archive with full functionality, e.g. recovery, and the local archive, a lightweight XML database running on laptops of investigators preparing an observation. Data from the local and central archive are synchronized on the investigator's demand.

3. XML and Semistructured Data

The data model for semistructured data (Abiteboul, S. et al. 1999) has been developed to overcome the limitations of relational data: semistructured data may have a schema that is irregular, changing over time, unknown, or not existent at all. Therefore semistructured data can live without a schema, containing a sort of a semantic description in each data field itself.

Semistructured data is modeled as a labeled directed graph, or, for simpler cases, as labeled tree. The left part of Figure 2 illustrates this with the example of an article having an id, a title and a sequence of possibly nested sections. Content is represented in the labels of leaf nodes. In contrast to a relational modeling, different articles may have a varying number of sections, may have an optional author etc. We can also observe that the tree contains some kind of semantic description in the form of the node labels.




```
<article id='id1234'>
  <title> An example for XML </title>
  <sec> This is the first section </sec>
  <sec Other sections may follow...
    <sec> Even nested ones. </sec>
  </sec>
</article>
```

Figure 2. Semistructured data as a tree (left) and XML (right)

In the last years, the eXtended Markup Language (XML; W3C 1998), an instance of the semistructured data model, pervaded all areas of software industry as a means to represent and exchange data in a flexible way. Figure 2 shows on the right a small XML document that corresponds to the tree on the left side. Every XML document can be modeled as semistructured data: An element (`<sec>...</sec>`) corresponds to a node in a tree, with the children of the node being represented as nested subelements.

There are two notions of schema for semistructured data and XML: *descriptive* schemas are structural summaries describing the structure of a collection of XML documents in the form of a collapsed tree. Structural summaries can be easily adapted, if documents are changed or added to the collection. Structural summaries, as illustrated in Figure 3, are often modeled as DataGuides (Goldman, R. and Widom, J. 1997): nodes in the original collection are represented as one common DataGuide node if they are reachable by the same sequence of node labels (e.g. the two `sec` nodes 4 and 5 of figure 2). In real-world cases, a DataGuide is smaller by several orders of magnitude than the original document collection. *Normative* schemas are essentially context-free grammars describing the hierarchichal nesting structure of XML elements (DTD or XML Schema). An XML document is called *valid* if it conforms to the structure described by an XML schema or DTD.

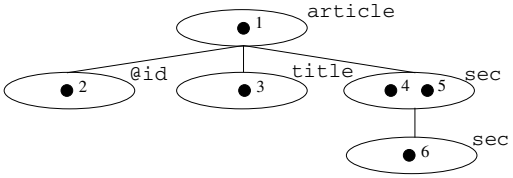


Figure 3. A DataGuide for the example in Figure 2

4. Querying and Indexing XML in the ALMA Archive

The World Wide Web Consortium (W3C) is defining query languages for XML for different application scenarios, e.g. XQuery. All query languages are based on XPath (W3C 1999), a language for specifying locations in XML documents. XPath's syntax is derived from expressions used in operation systems for referring to files, as can be seen in the following examples:

```
article/sec
article[title="Interesting"]/sec
article//sec
```

The first of the above examples points at all nodes being `sec`-nodes and children of an `article`-node. The second example selects `sec`-nodes being children of `article`-nodes that have a `title`-child reading "Interesting". The last example selects all `sec`-nodes that are descendants of an `article`-node.

XML documents can be stored in either a native XML database (Tamino, Xindice, etc.) or an (object-)relational database enhanced by XML capabilities. We chose to use Xindice for the local archive and DB2 with XML Extender for the central archive. DB2 provides the possibility to store XML documents as CLOBs in columns with dedicated parts of the documents being redundantly stored in additional "side tables". XPath-like expressions are used to specify which parts of documents are copied into side tables. A side table may be equipped with a standard relational index for improving on access performance.

This approach of DB2 has a major flaw: side tables can only be accessed by SQL queries, whereas we would like to query the original documents with XPath queries. IBM plans tackle this problem in future releases, but for the moment we are going to implement a workaround based on DataGuides in order to compute the (non-trivial) translation of an XPath query into the corresponding SQL query involving side table accesses: Instead of using a DataGuide as descriptive schema, we store in its nodes information about which parts of the documents are stored in which side tables and their datatypes: Each node of the DataGuide is annotated with the name of the side table and column where the respective content was moved to. Given an arbitrary XPath query, we can use this DataGuide to check whether the XPath query can be evaluated with the help of side tables and how the corresponding SQL query should look like.

References

Abiteboul, S., Buneman, P., and Suciu, D. 1999: *Data on the Web: From Relations to Semistructured Data and XML.* Morgan Kaufmann, 1999.

Goldman, R. and Widom, J. 1997, *DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases.* In 23rd int. conf. on Very Large Databases (VLDB'97)

Wicenec A. et al. 2002, in ASP Conf. Ser., Vol. 281, ADASS XI, ed. D. A. Bohlender, D. Durand, & T. H. Handley (San Francisco: ASP), 95

Wicenec A. et al. 2004, this volume, 93

W3C 1998, *Extensible Markup Language (XML) 1.0.* Recommendation of the World Wide Web Consortium (W3C), 1998.

W3C 1999, *XML Path Language (XPath).* Recommendation of the World Wide Web Consortium (W3C), 1999.