Abstract

Saada transforms a set of heterogeneous FITS files or VOTables of various categories (images, tables, spectra . . .) in a database without writing code. Databases created with Saada come with a rich Web interface and an Application Programming Interface (API). They support the four most common VO services. Such databases can mix various categories of data in multiple collections. They allow a direct access to the original data while providing a homogenous view thanks to an internal data model compatible with the characterization axis defined by the VO. The data collections can be bound to each other with persistent links making relevant browsing paths and allowing data-mining oriented queries.

1. Introduction

Back in 2003, the Survey Science Consortium (SSC) of the XMM-Newton ESA mission was looking for a database system able to store the information extracted from the data files generated by the reduction pipeline. A specificity of that pipeline is to compute cross correlations between X-ray sources and a set of about 200 archival catalogues and to pack them within the dataset delivered by the SSC to ESA and to the scientific community. The SSC database system developed at Strasbourg has been designed to make the most from these correlation links. It must store the links in a persistent way; it must restore the uniqueness of archival sources correlated several times with different X-ray sources. It must
allow the users to filter their selections of X-ray sources either with their intrinsic characteristics or with the characteristics of correlated archival sources. In other terms, the system must support a hierarchical model of linked data. The first solution adopted was based on an object oriented database management system (namely O2) now withdrawn. We then decided to build our own tool based on open software. Considering that these features could be interesting for other teams or missions we decided to offer it to the community. This tool is named Saada (Système Automatique d’Archivage de Données Astronomiques). Saada is a tool generating local databases (SaadaDBs) containing multiple collections of heterogeneous data.

The paper starts with an overview of Saada. In section 3 we explain the way heterogeneous data are stored within a SaadaDB. Section 4 adopts a user point of view explaining briefly how to run the software. It is followed by a look inside a SaadaDB.

2. Saada at a Glance

The purpose of Saada is to make as easy as possible the implementation of archives containing a large variety of data products.

The name Saada is related to both the project as a whole and to the database installer. The generic name for a database created by the Saada installer is a SaadaDB which can be later renamed adequately. In this paper, the term Saada refers to the global feature of the project whereas SaadaDB refers to the features of a particular database created with Saada.

A SaadaDB is a standalone database including a storage system, a Web interface and a data loader. It doesn’t need to be connected to some remote service. It can be deployed on a laptop or on a big server. It can even work without network. Saada can be used on Linux, MacOS, or MS Windows.

A SaadaDB has the ability to integrate images, spectra, tables or any other files possibly linked to each other. It is managed with either a graphical tool or by scripts and doesn’t require writing code. The data loader extracts keywords values from input data files (FITS or VOTables) and stores them within a
relational database. Data can be seen either as tables of native values or as
instances of a data model compliant with the VO.

Data stored in the SaadaDB can be accessed in different ways:

1. As a relational database whose tables have been filled with data extracted
from the input files by the SaadaDB data loader. This database can
be accessed by any application having a connectivity with a relational
database management system (RDBMS).

2. As a VO resource providing a data access through one of the following
protocols (SIAP [1], SSAP [2], CSP or TAP [3]).

3. As a Web application.

![Figure 1: SaadaDB modules](image)

Figure 1 sketches the main modules constituting a SaadaDB:

- The relation database (RDBMS), entirely managed by the SaadaDB, can
  however host data not managed by that SaadaDB.

- The repository is a file folder used by the SaadaDB as a storage area.

- The WEB interface, based on the Saada API, runs as a simple JEE [4]
  application (servlets only) hosted by a Tomcat server.

- VO interfaces are managed by servlets which are actually part of the WEB
  interface.

Data can be queried in a classical way (sky search, keyword filtering) or with
more accurate queries as shown in the example below.
Select all XMM detections located near ABEL 426
and having a flux greater than 1e-13
and correlated with
at least one Simbad source
having an obj_type
containing the string ‘Radio’

This example uses the computed correlation links mentioned in the introduction to filter data. The exact syntax is detailed in section 4.5.

3. Storing Collections of Heterogeneous Data

3.1. Input data files - Data categories

The basic job of the data loader consists in extracting keywords values from the input files and to arrange them in the database in a way that makes as easy as possible to reply to user requests (fig 2).

Saada distinguishes two file categories:

1. Those from which one can extract data, currently FITS files and VOTables.
2. The flat-files from which no data is extracted. They are just stored as file references.

Saada data model supports five categories of data whose definition matches the common meaning in astronomy (see table 1). Data files are manually affected to one category. As the structure of the input files can be rather complex (multi-extension files) a selection must be done on the set of keywords to be loaded. The keywords of the primary header (FITS) or of the first data table
(VOTable) are always taken. The data-loader can also take the keywords of another FITS extension, which can be either automatically detected or given by the administrator. By default, the extension loaded is the first matching the specified category.

For the TABLE category, row values are also stored in a sub category named ENTRY. The row storage works exactly the same way to that of the others categories, taking the column definition instead of the keywords. Pixels or table data are never stored for the others categories. They can however be read to extract the boundaries of the space or energy coverage.

3.2. Saada Collections

In a SaadaDB, data are distributed in separate collections created by the administrator. A collection is an abstract container, identified by a name. Collections creation and the choice of the collection where to store a given dataset are completely free. It is a design issue that must be sorted out independently of the products input format. The collection boundaries match the natural scope of the SaadaQL queries (see 4.5).

Collections are split in six sub-containers, one per category plus the one for the ENTRY. All have the same twofold internal structure:

1. The class level storage made of the tables containing the values extracted from the file (except for the FLATFILEs).
2. The collection level storage containing one table with computed values compliant with the Saada data model.

This data model includes the most relevant parameters in three axis (Space, Time, Energy) of the VO characterization model and some observation pa-
rameters. It is a subset of the ObsCore model [6]. It is always possible to add
by hand user-defined columns(see [4.1]).

All class level tables are joined within the collection level table making the
retrieval of all fields from both levels easy.

3.3. From Heterogeneous Native Data to Homogeneous Mapped Data
3.3.1. Storing Heterogeneous Data at Class Level

The way native data are stored at class level is a bit tricky. By default,
data products having the same format are all put in the same class level table.
There is one different class level table for each format identified in the set of
loaded files. This rule is actually a bit more flexible since the administrator
can force different products to be stored together in the same class. In this
case, a class merger adds new columns to the table and when a type conflict
occurs (same keywords with different data-type) the type is downcasted toward
the more general one (from boolean to text). The class level tables use a name
given by the administrator or by default the name of the first loaded file. The
choice of merging data classes is made by the administrator. This operation
mode might be relevant in some cases such as storing in one class data from the
same origin (e.g. a reduction software).

3.3.2. Storing Mapped Data Stored at Collection Level

Mapping class level storage to data model is very challenging because there
are rarely one to one links between a keyword and a data model field:

- The vocabulary used to name a given quantity is very heterogenous;
- The quantities stored in datafiles are not always complete (e.g. missing
  unit). The missing data must then be searched in keyword description or
  in the comments included within the data file or even in the documentation
  of the software used to generate the product;
- A data model quantity can result from a computation of multiple values
  (e.g. applying a threshold on spectrum flux defined with WCS keywords);
- Relevant information can be spread over multiple extensions or resources.
To be accomplished automatically with a good success rate, this operation should be based on a knowledge database identifying the origin of the data product and returning the correct mapping rules. This issue is likely one of the major justification of the Virtual Observatory.

In order to minimize the risk of mismatch when setting the data model fields, the data mapper of Saada is currently limited to basic identifications. Data model quantities are either searched among native keywords or given as constant values. Sometimes they must be formatted (date) or a unit conversion must be applied (energy range).

Saada offers two different ways to do this mapping:

1. The Autodetection mode: The data loader explores the keywords to find out the relevant quantities. It can furthermore interpret WCS keywords or column definitions of a data table or even image pixels. This mode is fine for classical products, but it can lead to irrelevant values when applied to more exotic data.

2. The Mapped mode: The administrator can set a data model field with a constant value or with the name of a keyword from which the value will be taken. If the keyword is not found, the field is no set, but the process continues.

The two modes are run in parallel with priority level rules set by the administrator.

3.4. The Saada Relationships

In a SaadaDB, data collections can be linked to each other by persistent relationships. A relationship is a named entity joining the data of one given category in a given collection to the data of another category in another collection (or the same). The links in a relationship can be qualified with numerical values, the qualifiers, which are part of the relationship definition. Links are reported on the WEB interface and can be used to refine a data selection. They are also used by the download facility to pack associated data with the selected ones.
4. Saada in Action

4.1. Creating a SaadaDB

A graphical installer achieves the creation of a new SaadaDB. The administrator must set some local resources (location, RDBMS...), a global unit system for the spectral range and a global space frame. Columns can also be added to the collection level model. All functionalities used by the future SaadaDB are checked at creation time.

4.2. Managing a SaadaDB

All management operations can be controlled from the graphical tool. The main functionalities are summarized in the table.

Most of these operations can also be run by `ant` utility tasks. The XML description of the tasks can be downloaded from the graphical tool. These scripts allow easy repetition of a management sequence on a wider scale.
<table>
<thead>
<tr>
<th>Collection Data</th>
<th>Creating, removing and describing collections. Loading data, removing data, indexing data, editing data-loader filters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta data</td>
<td>Editing UCDs [7], units or description attached to the data columns.</td>
</tr>
<tr>
<td>Relationship</td>
<td>Creating, populating, indexing removing relationships.</td>
</tr>
<tr>
<td>VO resources</td>
<td>Editing VO registry records for specified data collections, creating a TAP service or creating an ObsTap resource.</td>
</tr>
</tbody>
</table>

Table 2: Admintool Features

![Figure 4: Web Interface](image)

4.3. The Web Interface

The Web interface [8] is packed within the distribution. It must be deployed (or re-deployed) after the database schema [1] has been modified. It can be deployed from the graphical administration tool, or more safely by a script invoking the tomcat deployer. It is a Rich Internet Application (RIA) built with JQuery and allowing an easy browsing of the database content. Very complex queries can be setup with an editor stacking individual constraints edited within the graphical interface. The final query string can be refined by hand.

---

The interface proposes a shopping cart feature where the user can store any sort of data in a downloadable ZIP ball. Following a request made by the GBOT project [9], the shopping cart content can be forwarded toward a third-party server. This feature is used to feed a reduction pipeline with data selected from the WEB interface. Finally, data can be sent from the Web interface to other VO clients by using a Web profile SAMP connection [10].

4.4. Deploying the VO Services

Saada supports four VO protocols served each by one (or more) specific servlet(s). Simple protocols are restricted to their relative Saada category (ENTRY for the cone search, IMAGE for SIAP and SPECTRA for SSA). Publishing a simple service with Saada does not require any action. The service can be connected once the Web interface is deployed just by building proper URLs.

The fourth protocol is TAP. It can endorse any data table. There is only one TAP service attached to a SaadaDB. The service is set from the graphical administration tool by selecting the data collections to be published. The TAP_SCHEMA (see this A&C issue) is automatically updated. Joins between collection storage and class storage tables are taken into account. The TAP implementation uses the CDS/ARI library developed by G. Mantelet [11].

4.5. The Query Language

Saada has its own query language, SaadaQL, where the clause `SELECT columns FROM table` is replaced with `SELECT category FROM class IN collection`. The returned columns are not specified since the SaadaQL query always returns a set of identifiers 2 which are used by the cache to provide the actual data (see 5.2). SQL WHERE statement is replaced with four clauses with each one having its own meaning. They are implicitly ANDed.

The query example given in section 2 is rephrased below with the SaadaQL syntax.

---

2Saada relies on an internal identifiers mechanism encoding the location of any record
### Table 3: SaadaQL Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WherePosition {...}</td>
<td>List of search positions</td>
</tr>
<tr>
<td>WhereAttributeSaada {...}</td>
<td>SQL WHERE statement filtering on collection level fields and on class level fields if the scope of the query is restricted to one class.</td>
</tr>
<tr>
<td>WhereRelation {...}</td>
<td>List of patterns applied to the vectors formed by the links starting from the queried collection</td>
</tr>
<tr>
<td>WhereUCD {...}</td>
<td>Filter expressed with Unified Content Descriptor (UCD), a formal vocabulary for astronomical quantities</td>
</tr>
</tbody>
</table>

Select ENTRY From * In CATALOGUE
WherePosition {
  isInCircle("49.94666+41.51305",1,J2000, ICRS)}
WhereAttributeSaada {
  _ep_8_flux > 1e-13}
WhereRelation {
  matchPattern { CatSrcToArchSrc,
    AssObjClass{SimbadEntry},
    AssObjAttSaada{ _obj_type LIKE '%Radio' }}}}
5.1. The Storage System

In theory the Java layer could run with any RDBMS through JDBC [12] but in reality, each system having its own SQL dialect, the application must remain aware to the RDBMS it is connected to. That is why Saada only supports three DBMS:

1. PostgreSQL (PSQL) 8+: high performances, high scalability;
2. MySQL 5.1+: no real advantage compared to PSQL, but widely used in many places;
3. SQLite: limited concurrency features, but no installation, just a library packed within the Saada distribution.

In order to remain RDBMS agnostic Saada does not use specific extension such as PGSphere for PSQL. The sky search is based on Healpix [13]. All coordinates are stored with their pixel number (level 14 hard coded). A few SQL procedures are also used.

5.2. Java Objects and Persistence

Data access is a bit more sophisticated than a simple JDBC callback. It is based on a tight coupling between the SQL storage and the Java classes. Each

\[\text{Availability of multiple users to access data at the same time}\]
data record is modeled by a Java class. Basically the Java super-classes models
the collection level storage whereas the class level storage is modeled by business
classes (sub-classes). Business classes are built on the fly by the data loader and
dynamically linked with the application. The instances of the data classes are
stored in a cache using the SoftReference mechanism [14]. This allows the cache
to use the whole space available in the memory heap while leaving it available for
the garbage collector. When a record is loaded in memory, only the collection
level attributes are queried. The class level attributes are all retrieved at the
first attempt to access one of them.

The cache mechanism is very efficient to feed the Web interface, which dis-
plays on the same page a lot of heterogeneous objects (objects of different
classes). It is however bypassed when the query result must be downloaded
in a VOTable or in a ZIP archive. In this case, the data are read in streaming
mode and the response is formatted on the fly.

5.3. Table Indexation

By default, the data loader removes all indexes of the concerned tables before
to start and builds them again after the job is complete. All columns are
indexed by default. This behaviour can be disabled avoiding unnecessary index
computation. Indexes can be individually managed from the administration
tool.

5.4. Relationship Indexation

When the project started, the SQL system couldn’t efficiently process queries
using filters on linked data, especially when they contain numerical predicates
associated with a cardinality constraint such as shown by the example below:

\[
\text{Select all XMM detections correlated with} \\
\text{more than 3 Simbad source} \\
\text{located at less than 3 arcsec}
\]

We took over this limitation by developing a query engine dedicated to the
processing of relationship patterns. It is based on hash maps stored in files and
loaded on demand in memory.
<table>
<thead>
<tr>
<th>Data Ingestion</th>
<th>Rate</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catalogue ingestion</td>
<td>500 to 5000 rows/sec</td>
<td>Depends on the column number.</td>
</tr>
<tr>
<td>Spectrum ingestion</td>
<td>1 to 5/sec</td>
<td>Depends on the keyword number.</td>
</tr>
<tr>
<td>Images ingestion</td>
<td>1 to 3/sec</td>
<td>Includes the vignette generation.</td>
</tr>
<tr>
<td>Flat files ingestion</td>
<td>few 1000/sec</td>
<td></td>
</tr>
<tr>
<td>Source selection 1arcmin around a position</td>
<td>100ms</td>
<td>Done on a catalogue of 5,000,000 sources</td>
</tr>
<tr>
<td>Sources selection combing filtering on both collection and class level</td>
<td>2sec</td>
<td>require a join between both tables (5,000,000 X 1,000,000 rows)</td>
</tr>
<tr>
<td>Source selection by filtering on relationship links</td>
<td>2sec</td>
<td>The actual relation contains 10,000,000 links</td>
</tr>
</tbody>
</table>

Table 4: Order of magnitude of the performance of a SaadaDB (measured on the XCatDB)

5.5. Performance

The flexibility of Saada prohibits to define performance criteria which are both simple and relevant. A SaadaDB can host a lot of collections with very different sizes. Data collections can host various number of classes, they can be linked with relationships (see 3.4) having a very different number of qualifiers and links. The data ingestion rate depends on the complexity of the collection data mapping. The global performance of a SaadaDB is also driven by the efficiency of both underlying RDBMS and JDBC driver. Taking this into account, the values given below, issued from the experience with the XCatDB [15], must be considered as orders of magnitude.

The volumetry limitation also depends on the RDBMS. We consider that Saada easily supports data collections containing a few tens of millions of entries. Beyond, some tuning must be done such as a replication at collection level of the most used class level attributes.

---

4Core CPU at 2.4GHz, 16GbRAM, PSQL 8.4, Scientific Linux 6
6. Future and Prospects

After a first beta release \cite{16}, the concept of Saada has been adapted to the bio-computing paradigm in the Bird project led by Hoan Nguyen Ngoc \cite{17}. Both projects are still developed and supported, but separately.

The project evolution is mainly driven by user requests. The versioning is managed in nightly build mode.

Saada has undergone a major evolution in 2014 with the convergence of the inner data model toward the Obscore data model. The query engine now supports the search by regions. In parallel, a part of the code has been refactored and the website has been redesigned. The foreseen evolutions will be focused on both data loader and VO interfaces. The flexibility of the data loader will be improved with the possibility of taking data from more than one FITS extension or one VOTable resource and also with the implementation of arithmetic operations for the mapping between the native data and the collection level data. In collaboration with the CDS, we are working on the design of the knowledge base mentioned in \textsection{3.3.2}. The support of the VO interfaces will also be improved with the implementation of new standards such as SIAP-V2 and Datalink and with the possibility for the administrator to do a fine-tuning on the content of the VO responses. Looking further ahead, we are thinking about improving the modularity and the flexibility of the WEB interface.

7. Conclusions

The initial goal of Saada was to pack into a single multiplatform tool two basic functionalities: data management and the WEB interface deployment. The VO interface quickly became the third pillar of the tool. This ambitious way to integrate a large set of features probably contributed to give an image of a product leaving little room for the user setup. That, combined with difficulties encountered by some users to install PostgresQL or MySQL, slowed down the usage of Saada. These matters have been overtaken by the development of the script mode and the support of an embedded database in addition to dozen of
other improvements. Since then, Saada has showed it robustness especially with the XCatDB hosting millions of data files from the XMM-Newton mission. It remains one of the few tools able to transform a set of data files into a database, to publish it in the VO and to provide a Web interface with a few clicks. Among other projects and thanks to its ability to handle heterogeneous data collection, Saada has been chosen by the CDS to host data attached to the Vizier catalogs.

References


Acknowledgements

We would like to thank A. Nebot Gomez and F. Grisé (Strasbourg Observatory) for reviewing this paper, Taro L. Saito (Treasure Data, Inc) for developing and supporting the SQLite JDBC driver and all both contributors and users of Saada.