Journal of Physics: Conference Series **664** (2015) 062043 doi:10.1088/1742-6596/664/6/062043

Re-engineering SAM or Changing the Engine in the Train While it is Running

R. Illingworth[1], **M. Mengel**[1], **A. Norman**[1]
[1]Fermi National Accelerator Laboratory, Batavia IL, USA

E-mail: anorman@fnal.gov

Abstract. In the last few years at Fermilab we re-architect-ed our SAM[1][1] family of data catalog and file transfer tools – including major changes – while continuing to transfer over 1 Pb/month of data to multiple existing experiments and bring new experiments on board. This work was done with less than 3 FTE-years of effort, and the changes made include major ones, such as changing interprocess communication protocols, migrating database back-ends, removing and replacing major components, and supporting new file delivery methods. This paper will summarize the approaches we have used to do this, including using design patterns like the Facade, Adapter, and Command patterns, and assisting experiments one at a time with client migration. This process has allowed us to modernize our infrastructure with reasonable costs in both calendar time and developer effort, while continuing to provide the operating service to our customers with minimal interruptions.

1. Overview

The SAM system for experiment data handling is a venerable system combining a file transfer and caching system with a metadata catalog. However, to support D0 operating on the wider Grid, additional software, known as SAMGrid had been written, which while functional, relied on obsolete and unsupported software components, and required a fairly large amount of care and feeding. What was needed was to update SAM using current technology, make needed improvements, and streamline the overall software suite, to make it more maintainable. The difficulty was that with a small staff, rewriting major portions of the package would take quite some time, and we needed to be in operation during that time window; so we came up with a design to let us replace SAM piece-wise while keeping it in operation, and embarked on a project to upgrade the system while people were using it.

2. Before the Upgrades

At the beginning of the process, the architecture of SAM and SAMGrid looked like figure 1, which made it problematic to run on the Grid, due to:

- the many network connections CORBA needs to make from remote sites, which often have firewalls, etc.
- the amount of software that needs to be brought along to the job
- security, or lack thereof, in the CORBA layer as implemented

[1] Sequential Access with Metadata

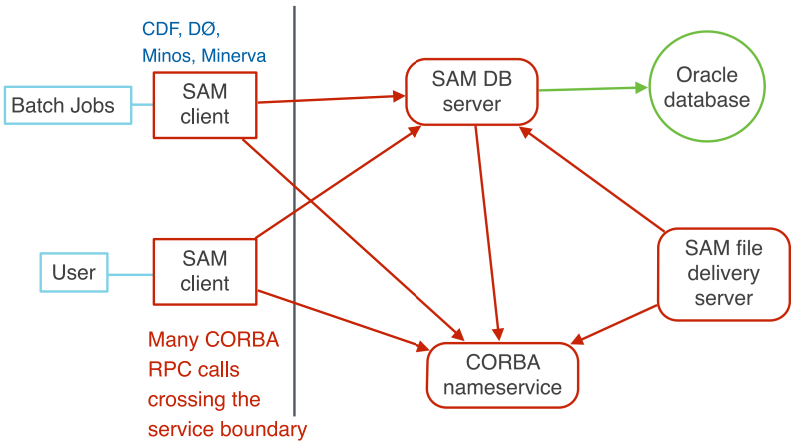

Figure 1. SAM Architecture: before

This original architecture was also difficult to modify, because CORBA has very rigid interface definitions, and changes like adding an optional parameter to a method involve updating not only the server, but all of the clients.

3. Design Considerations

The design to let us migrate SAM a little bit at a time involved what a software architect would call the Facade Pattern[2]: where an interface layer is put up, and all access to the system is done through that interface, and then changes can be made behind that Facade, without affecting how clients use it. So to simplify Grid operation, and to provide the Facade behind which to rework the system, we decided on an HTTP/REST[3] interface, with Adapters for the CORBA services. Using an HTTP interface provides logging, caching, and load-balancing capabilities in off-the-shelf components. Also, the same grid proxy certificates used for file transfer, etc. can be used to provide security for connections from jobs to SAM. While this change required clients to change their scripts once, after that we could make changes behind the Facade without changes on their part. We also took the opportunity to simplify some interfaces and remove others.

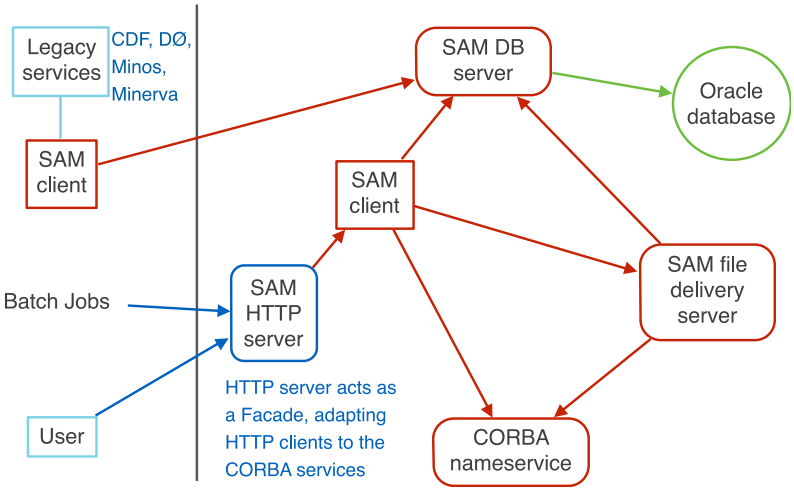

Figure 2. SAM Architecture: initial Facade

4. First stage

In the first stage, we put up an initial SAM-Web instance, which was basically a CherryPy Python webserver running in a python instance with the CORBA SAM interface code loaded as well. Then web services providing the SAM calls were trivially setup. In use, the client speaks HTTP to the SAMweb instance, which speaks CORBA to the existing infrastructure as in figure 2

This initial implementation was very easy to construct, and met all of our goals for Grid operability. It also facilitated us making changes to the overall architecture piecemeal, as each individual web call could be re-implemented to access the database directly and so replace the corresponding CORBA interlace completely.

Switching existing users over to this interface was actual work for the clients, as they had to switch what python and/or C++ calls they made do use a web-client based API to talk to SAM. We only had two experiments, Minos and Minerva that had significant effort here of existing scripts and programs to convert, and they were able to run in a mixed mode where some scripts still used CORBA directly, and others used CORBA via SAM-Web, and everything still all worked together while they migrated.

5. Transition

The next phase of the rework was to get rid of the data path through the CORBA DB server to the database, and have SAMweb talk to the database directly. This was also an opportunity to make the database code largely database independent, by using a database agnostic interface, which can implement the Bridge design pattern[2]

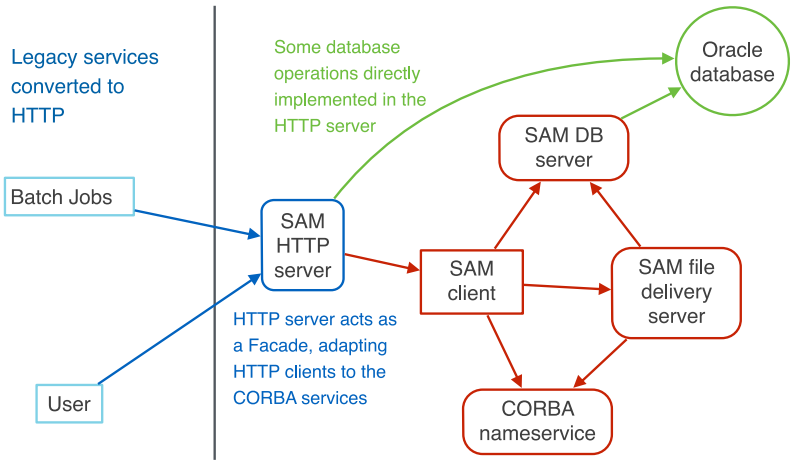

Figure 3. SAM Architecture: Transition

Since SAM supports a simplified query language, a toolset which supports complex query construction was needed, and SQLAlchemy was chosen. With this added to the system, the transition architecture looks like figure 3.

Now individual methods can either directly call the database via SQLAlchemy, or continue to use the old CORBA interface, during the transition period. To complete the removal of the CORBA database server, the SAM Station was modified to start making HTTP calls to the SAMweb interface, rather than CORBA calls to the DB server, to do its database operations.

6. New Query Language Implementation

SAM provides a simple query language that insulates users from having to deal with SQL, and the complexities of the underlying metadata and location catalogs.

- The query language allows selecting data files based on their metadata properties
- More advanced features include selecting files based on file provenance (child or parent files)
- As part of the updating project (and new facade), we replaced it with an improved version that was no longer dependent on the Oracle database

With the direct database API chosen, we were able to take the old SAM "dimensions" query language used to specify file datasets based on metadata, and rewrite it with a proper parser-generator in Python, which is used to build a parse tree, and then converted to SQLAlchemy query objects. as in figure 4.

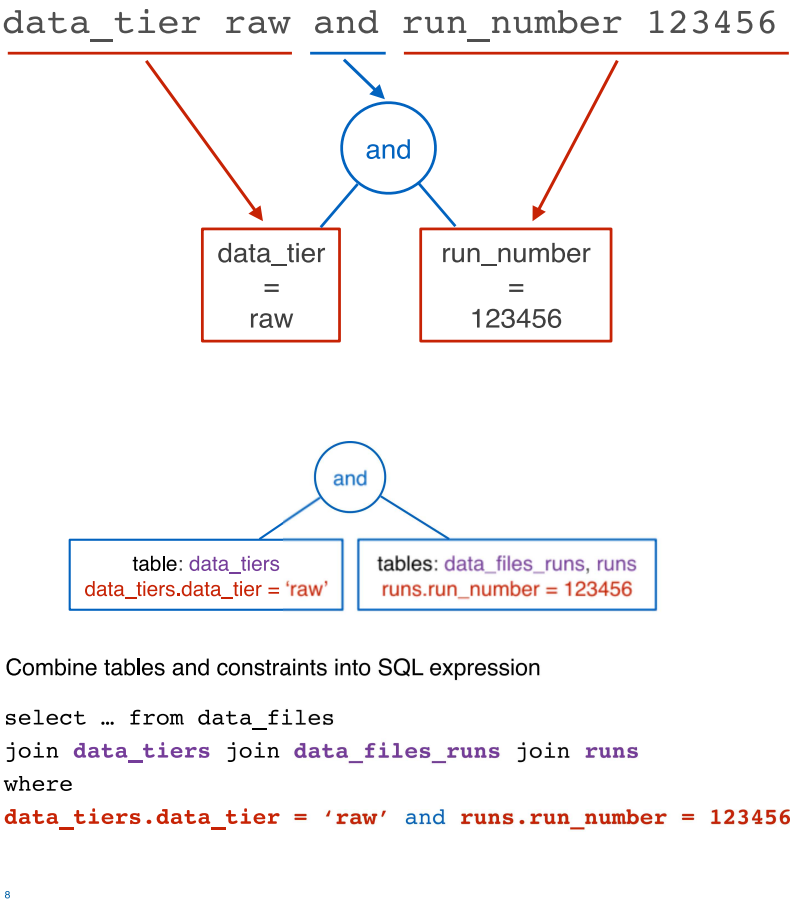

Figure 4. Parsed Dimensions Query

This approach of building up SQL expressions from small pieces:

- gives a consistent syntax, (which the old implementation did not have) allows end users the flexibility to create complex queries without knowing the internal database schema
- allows us to alter the implementation, schema, or even the database engine without affecting the end users

Users of the SAM-Web interface got the new and improved query notation, while users of the CORBA interface had the old one; but we had software in place to automatically migrate existing definitions to the new syntax, and a large subset of dataset queries work identically in both notations.

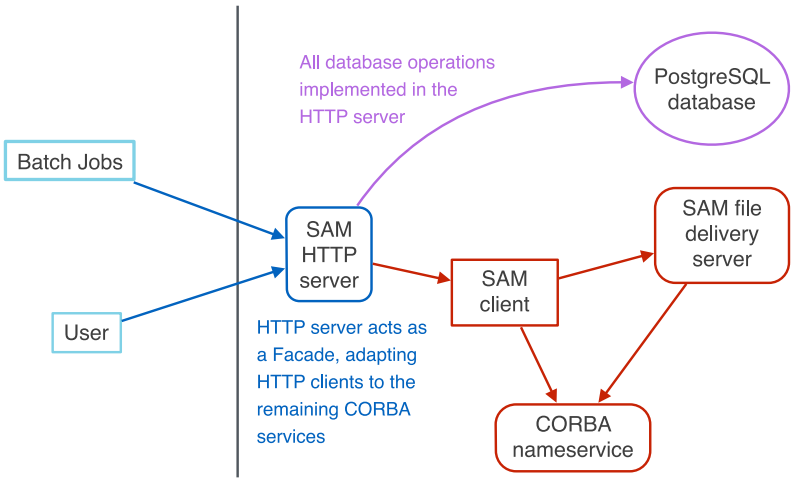

Figure 5. SAM Architecture: Database Migration

7. Database Migration

The next major change we made behind the scenes was to convert from Oracle, which the old SAM software used exclusively, to PostgreSQL. This makes the entire SAM infrastructure Open Source. Once all the uses of the old CORBA database servers were removed in the SAMweb frontend, and in the SAM station, and all of the database access went through SQLAlchemy, we were able to switch database engines and rework a few queries for perofmance, with minimal code changes. The architecture at this point is described in figure 5. This opportunity was also taken to drop a few dozen tables from the database schema that were in the initial SAM design, but which had been used either never, or extremely rarely.

Other than a short outage (the longest was 3 hours) to actually transfer the data from Oracle to PostgreSQL, the users basically did not see this change.

This also puts us in a position to try any database that SQLAlchemy supports (thirteen currently and counting)

8. Final State

The last transition (still going on) is to get rid of the CORBA path from SAM-Web to the SAM Stations. This involves replacing the remaining CORBA server components with HTTP capable services. Once this is done, the SAM-Web service and the SAM Stations can communicate via HTTP, and the CORBA interfaces can be dropped entirely, and our architecture will look like figure 6.

When this transition is complete, we will have a much smaller, leaner system which no longer relies on CORBA, and is largely database independent.

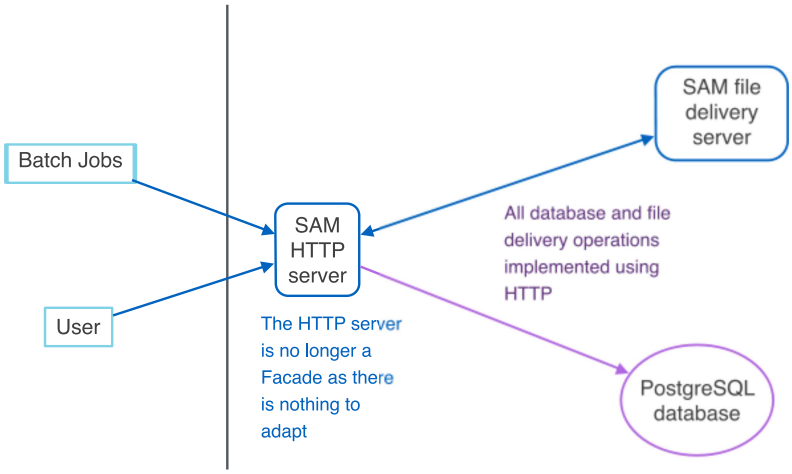

Figure 6. SAM Architecture: Final

9. Other benefits

The new architecture has other benefits. We have been able, for example to stand up multiple SAM-Web instances for a single experiment, and route certain categories of work to each instance to load balance, again completely transparent to the end users; the routing of requests to SAM-Web instances is done by the front-end Apache server.

10. Time line

- August 2011 SAM-web facade coding started
- October 2011 prototype could run simple SAM projects
- November 2011 Limited user testing
- March 2012 Started New & Improved Dimension parser
- July 2012 New Dimension parser roll-out
- August 2012 put behind Apache for production for Minerva & Nova
- January 2013 File Transfer Service moves to SAMweb interface
- April 2013 6 experiment instances
- June 2013 NOvA starts running SAM projects for MC
- September 2013 NOvA offsite SAM usage rollout SMU, Nebraska, etc.
- October 2013 improved DCache/SAMStation integration
- November 2013 Postgresql coding changes begin
- February 2014 Database conversions begin
- September 2014 PostgreSQL migration completed for all current experiments
- 2015-16 Last remaining CORBA components removed

Interruptions:

- up to 3 hours for actual database conversions
- misc 30 second outages when new versions started

11. Conclusions

This approach has let us

- make major structural changes
- without impacting users
- while maintaining operations
- minimizing developer effort

and is highly recommended to others needing to migrate legacy systems.

Acknowledgments

The author acknowledges support for this research was carried out by the Fermilab scientific and technical staff. Fermilab is Operated by Fermi Research Alliance, LLC under Contract No. De-AC02-07CH11359 with the United States Department of Energy

References

[1] Illingworth R A 2014 *Journal of Physics: Conference Series* **513** 032045
[2] 1994 *Design Patterns: Elements of Reusable Object-Oriented Software* (Addison Wesley)
[3] 2007 *RESTful web service* (O'Reilly Media)