AIDA, AN ARCHITECTURE FOR DISTRIBUTED ACCELERATOR DATA AT SLAC

G. White, S. Chevtsov, C. P. Chu, D. Fairley, E. Grunhaus, R. Hall, P. Krejcik, G. McIntyre, D. Rogind, R. Sass, H. Shoaee, M. Zelazny, SLAC, Menlo Park, California, U.S.A

Abstract

Rapid development of scientific software applications for a large instrument like an accelerator, in an established and evolving environment, is made difficult by the diversity of interfaces, protocols, and hosts, of the source data. Additionally, analytical applications deal mainly with complex data structures, such as synchronized beam data for a whole beamline, rather than individual control points. AIDA (Accelerator Integrated Data Access) is a distributed 3-tier system that allows Matlab, Java programs, or scripts, to interoperate with EPICS Channel Access, legacy control systems, relational databases such as Oracle, accelerator modelling systems, EPICS and SLC Archivers, and other data servers, in ways oriented to scientific users. It also includes a Google-like web interface for search and plots. At SLAC, AIDA provides a uniform, fast, interface to 4.5 million named elements in 14 lower level systems, over two control systems, for about 70 utilities and 20 large scientific applications. This approach was found to be key to the rapid commissioning of LCLS at SLAC. We present the first public description of the developed AIDA system since its early thinking at ICALEPCS 2001.

INTRODUCTION

AIDA is a complete, fast and robust system for helping physicists and programmers interface to controls and scientific data systems. It can get or set simple EPICS Process Variable (PV) data [1] (floats, waveforms etc), or complex "structured" data like archived values with timestamps, model parameters like transfer matrices, Oracle DB query result tables, Beam Position Monitor beamline orbit data, synchronized magnet control etc, in the speed required for online scientific applications. However, although the data may be complex, the application programing interface (API) is very simple, using a java-bean like "get" and "set" programming model, of named data items like EPICS PVs, plus `name=value` pair parameters. This makes AIDA also popular among SLAC scientists for offline diagnostics and analysis, since its easy to use from Matlab.

Typically the server side had to do some significant processing to compile that data before returning it. An AIDA network is implemented in CORBA, with Java client API and Java and C thin-server sides. It includes an authorization scheme for controlling access, and complete error messaging and logging.

The use cases, programming examples, and architecture, of AIDA are described.

Primary Use Case Model

Aida processes between 35000 and 120000 unique queries per day of normal machine operation, and knows 4.5 M named entities across 2 control systems. Its use is split primarily between production quality matlab applications for commissioning applications, and high level Java applications intended for long term controls and optimization, such as configuration (in the SLAC version of EPICS SCORE, in accelerator model applications, beam energy management and diagnostics etc).

For LCLS commissioning, the control system was a hybrid of the legacy SLC control system (Fortran and C on VMS hosts and RMX front ends), and a new EPICS control system being phased in (which is based on Linux and Rtems).

Since AIDA knows all of the controls, archiver systems, model, and backend databases of both of these systems, and has a simple Java API which is easy to use from Matlab, it was the central point for rapid commissioning of LCLS.

DATA SOURCES

There are presently 23 production data sources with which an AIDA client my interact. The primary ones are:

- EPICS Channel Access. Get and Set is supported, but not monitor. Monitor is presently not considered worth it for the primary use cases, which are applications oriented as opposed to controls displays, feedback etc
- EPICS Archiver
- Legacy SLC controls database. This database acts actually like a memory system, so setting the database is like setting the device values in the field
- SLC Archiver
- XAL Accelerator Model data
- DIMAD Accelerator Model data
- Oracle Relational Database. This AIDA server can issue canned SQL queries (including parameter substitution), so it acts as a trivial way for applications to get data from Oracle in a well managed robust oracle connection.
- Timing synchronized BPM Orbit data
- Timing synchronized magnet control
- RF data and control

Note that there are cases of 2 data sources for a single data type, such as XAL and DIMAD modelling. In these cases, AIDA presents the same syntactical interface with the same options, so the user largely doesn't have to know the source. Basic semantics are kept uniform too.

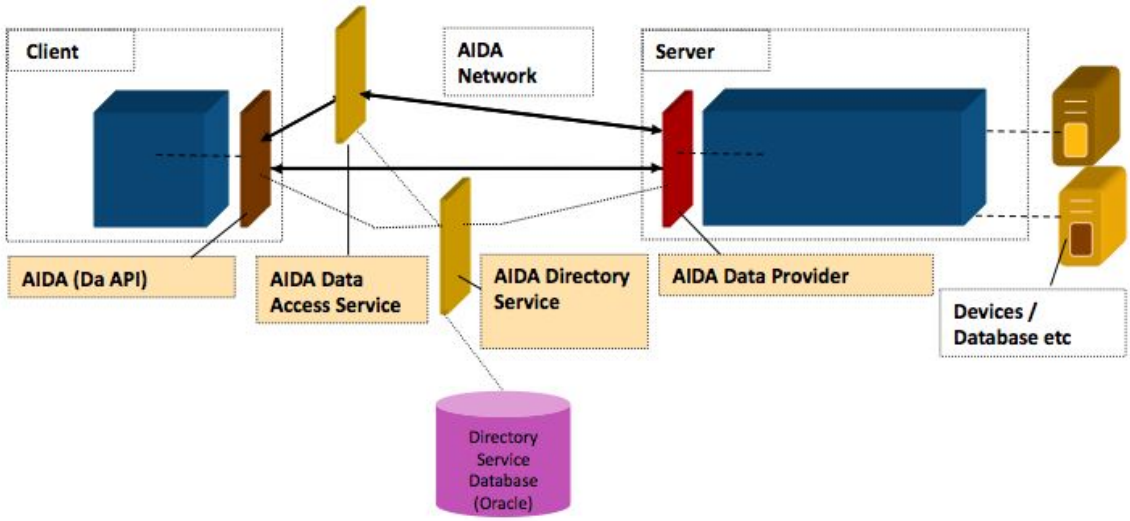

Figure 1: Schematic of an AIDA Network. For clarity, only a single data server is shown, but in a real AIDA network there will be many of these. The AIDA components are shown as thin boxes. The server side shows a thin AIDA front end to a data server, which may itself interface to back end data systems such as IOCs.

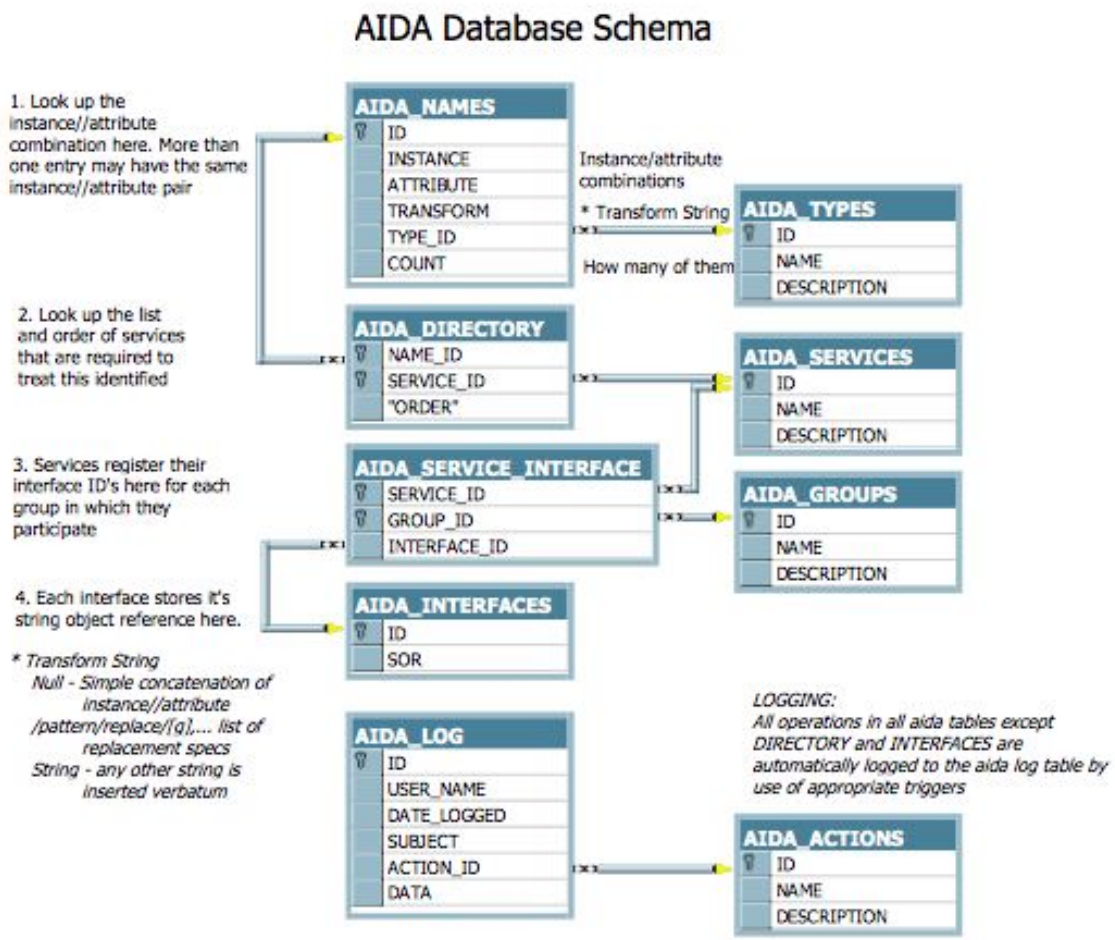

Figure 2: The picture shows the Aida Directory Service Database schema entity-relationship diagram (ERD). The table AIDA_NAMES contains the entity names (instance and attribute columns, akin to EPICS PV and field), and the transform (metadata that describes, if necessary, what string to ask the data provider to get the entity named). The other tables describe which server or servers are involved in each entity, and CORBA metadata.

ARCHITECTURE

AIDA is a 3-tier service oriented architecture; the client side with the API, a middle tier of 2 Aida system servers, and the backend controls and data servers themselves (see Fig. 1). The middle tier is comprised of an AIDA Name and Directory Service (see below), which keeps track of all the named data entities in the AIDA network, and a so called "Da" Server, which mediates all requests for data.

An AIDA network is implemented in CORBA. The Directory Service and Da Server implement special APIs, but all data servers implement a single IDL (Interface Definition Language) defined protocol. In this way, servers can be brought up very quickly, and all data communications are uniform and easily learned. Presently, there are 23 real production data servers, and 4 test servers, giving 21+2+4 = 27. There is also a mirror system of 27 servers in an orthogonal development application level network, so 54 servers in total. In each network, 19 of the servers are Unix based (in 3 hosts), and 8 are VMS (in 1 host).

Everything Aida knows, that is, the name of each data entity (akin to a PV) and how to get or set it, is in AIDA's Directory Service, whose database is implemented in Oracle (see Fig. 2). The data entity names in the Aida Directory service are automatically added or removed by batch processes that monitor the contributing data systems and update the directory service as changes occur in those data systems. Names can also be added or removed by hand. This makes AIDA very useful for tracking such things as EPICS PVs, since it knows all PVs, and can get the name or value of any subset according to a regular expression.

PERFORMANCE

AIDA was designed to get simple data fast, but incorporate facilities for getting structured, typed data too. It was considered appropriate that such heterogenous data, such as the transfer matrices for a whole beamline, would be allowed to take longer, since it is typically required by scientific analysis rather than controls per se. AIDA achieves a basic round trip of ¡2ms for a single simple type (Fig 3), and median 3 ms for dynamically serialized structured data (Fig 4).

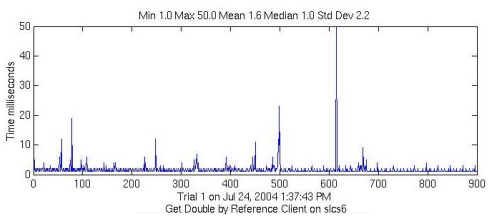

Figure 3: AIDA performance measurements of 900 acquisitions of a double value. Large excursions are repeated at the same point in successive tries (not shown), so are attributable to systematics of the measurement, probably garbage collection.

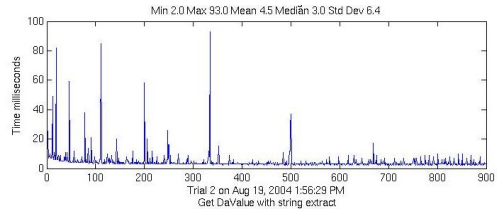

Figure 4: Performance of 900 acquisitions of structured data of 15 history data points (a small amount of history shown so as measurement is not dominated by the server's acquisition, but by the serialization/deserialization across the AIDA network). This shows that a 8 millisecond roundtrip can be expected for a first acquisition, but many acquisitions (of different values) will settle on 3 ms, due to caching, code path and Java VM warmup.

CONCLUSIONS AND PLANS

A uniform, fast, programming and Matlab scripting interface to all process variables and data in the hybrid control systems at SLAC, proved key to rapid commissioning applications development for the LCLS. This unified approach allows data and control component systems to be added or removed, without greatly affecting high level applications and analytical code. Plans for the long term future of AIDA are being formally prepared, and include creating 2 production level AIDA networks, one for office based analytics with open access, and one high availability network specifically for controls. More information can be found at [2].

REFERENCES

[1] R. Dalesio et al, EPICS, `http://www.aps.anl.gov/epics/`.

[2] G. White, et al, `http://www.slac.stanford.edu/grp/cd/soft/aida/`