SLAC Control System High-Level and Physics Applications Presumptive Architecture

Compiled by Greg White
December 7, 2003, SLAC. Revised January 23, 2004

This document outlines a possible system architecture for the further development of high-level, "physics" applications of the SLAC Accelerator complex control system. It is intended to be complimentary to the proposals made by Ron Chestnut in "A Possible Path for Control System Evolution," Aug 2003, which deals mainly with lower-level controls.

The systems described here would be hosted on Unix. No plan is given for transition from VMS: the proposal is that new development work would be carried out on Unix leaving the existing VMS system in place. The Unix applications and displays would access VMS systems for control and data resources, but would not supplant them, at least in the medium term. In the long term, after the infrastructure has matured, it may be that high-level applications would be developed to replace the ones on VMS. That depends on whether the lower-level controls and messaging systems now on VMS, are ported to other platforms.

1 Overview

1.1 Four Tier Model

Processes will be arranged in the classic three-tier model plus one (see Figure 1). From the top down, these are:

1. **Presentation Layer**. Applications will, to some extent, be separated from the display technology they use. The Control Room Applications use-type will use presentation technology appropriate to the language they use (if C++ then Qt, if Java then Swing). This document proposes that we create a presentation layer utility, for now called SVGPlot, to help applications create physics oriented displays quickly through an XML interface (see below). We also propose that summary displays, like luminosity plots, and weakly configured ad-hoc displays, such as history plots, are presented through web clients, using XHTML, JSP, JavaScript, and other HTML embedded scripting, including a an SVGPlot browser plug-in.
2. **Application Layer** - the applications themselves. There will be four kinds. Firstly, "classical applications" (for want of a better name), which are approximately speaking programs which use no particular framework API to access data. Languages will be C, C++, and Java, probably mostly Java. Secondly, data I/O, framework oriented applications, e.g. XAL and other applications using XML I/O. These should be developed as Web Services. Third, Matlab applications will be supported at least to the extent that an extensive data access system is provided for Matlab. Lastly, configurable displays, accessed on the web, will be implemented in a J2EE (Java 2 Enterprise Edition) compliant commercial Application Server (AS).
3. **Data Layer**. These are the processes and APIs from which applications access data in the Control System (CS), and includes EPICS Channel Access, Oracle, and two AIDA portals (see below), one using the basic AIDA API, and one which wraps AIDA as a Web Service to help implement XML based I/O such as applications based on XAL.
4. **AIDA Data Layer**. The Accelerator Independent Data Access (AIDA) will provide a language and platform independent mechanism for accessing data of different control systems through a uniform API.

1.2 Use Type Summary

Three approximate classes of application "Use-types" are identified for now (see shaded areas in Figure 1), though clearly the real distinction between these is something that will evolve:

1. **Control Room Applications**: These are core control system applications – highly interactive, stable, fast, and largely state-less (they don't have a fixed life-time). Users: operators, and control room physicists. Use Cases: on-line model based optimization such as steering, lattice diagnostics, correlation plots, history analysis etc.
2. **Sandbox.** Easy access to Matlab analysis facilities, but not so much need for speed or stability. Statelessness is not a requirement. Still, we intended to make access to control system data, such as BPM, magnet, history, and model data, very easy and through a unified interface. Users: accelerator physicists and others. Use Cases: offline model based analyses like bump calculation, algorithm verification etc. Unlike SPEAR-3, Matlab will not be a core control room technology for 24/7 operations.
3. **Summary and ad-hoc displays**. Non-EPICS summary displays and weakly-configured displays such as status displays, "SIP", plus history plots, luminosity plots, updating BPM orbit etc, will be generated as Web applications, optionally "in-browser". This is to share core technology for presentation of data between control room summary and extra-control room displays, by doing both with web technology. Users: operations, software group and, if appropriate, out-side WWW users.

The following pages detail the architectures to support each of these use-types. Lastly, systemic considerations like platforms, skills, and risk are outlined.

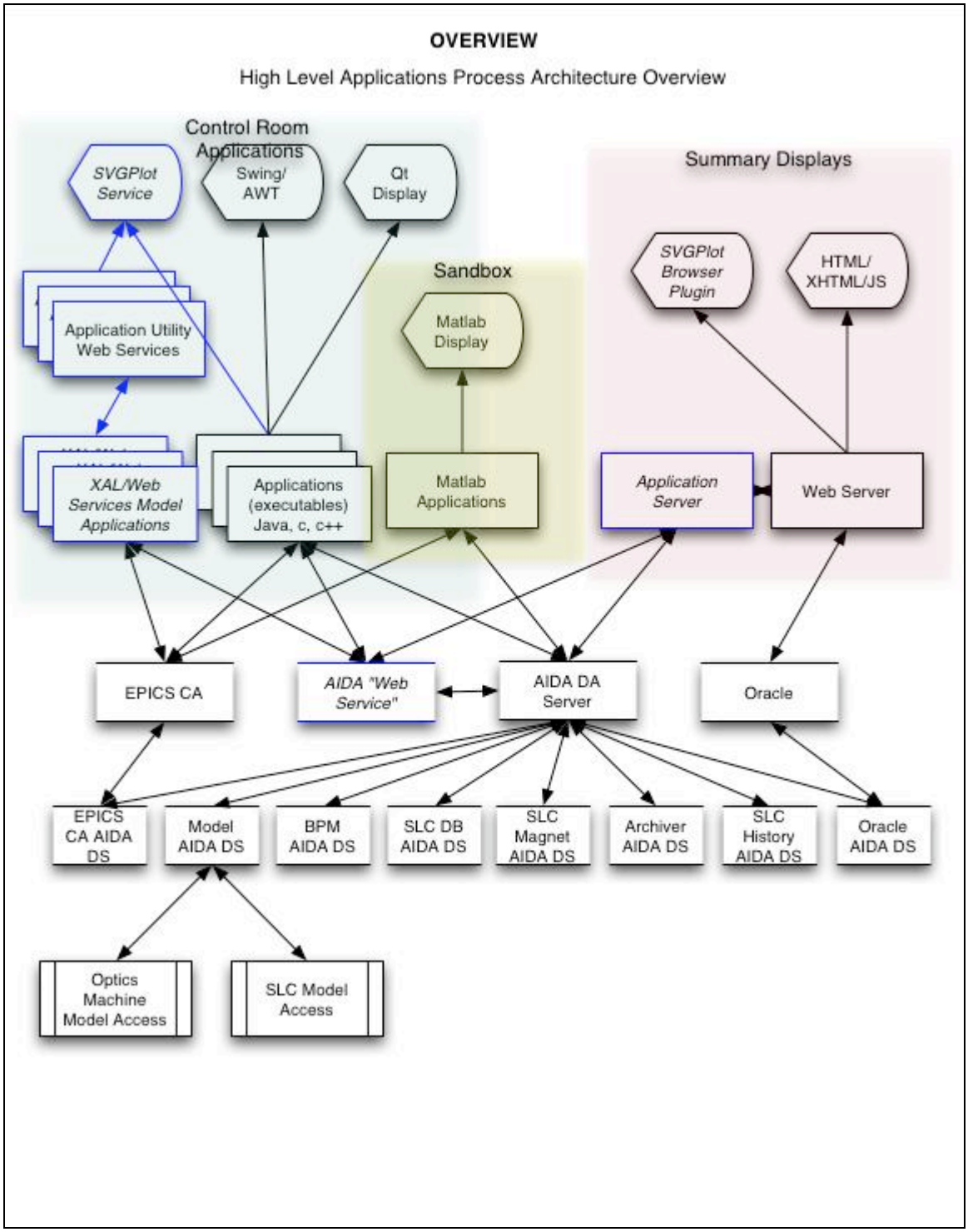

Figure 1: Overview showing how applications are implemented differently depending on their use.

2 Control Room Applications

This section outlines the plan for applications which will be needed in a control room environment, or are otherwise characterized as high-availability, or highly interactive (see Figure 2).

2.1 Classical Applications

New application development will be object oriented in C++ or java (not C or Fortran, though of course exiting or external software in these will be used)[1]. The C++ compiler of choice will be GNU (version should be loosely controlled through the makefile system we now have in place), specifically not Solaris CC (to help any move to Linux). Java will be kept in step with the SCS default on tailored machines.

2.1.1 Presentation Layer for Classical Applications in the Control Room

The normative Java display technology is Swing. We could buy a graphics display package to help, and quite possibly the Java Analysis System (JAS) toolkit developed at SLAC by Tony Johnson may be useful. C++ applications could use the Qt multi-platform GUI building library, which is a commercial tool from Trolltech but for which there is an Open Source distribution.

2.1.2 SVGPlot

Additionally, we could develop a system tentatively called SVGPlot. Functionally similar to Handypak, SVGPlot would be a device independent plotting and rendering engine whose API understands plot primitives at the functional level of "scatter plot", "histogram", "z-plot" and so on. Its API would be an XML based language we write. The rendering would use the W3C protocol Scalar Vector Graphics (SVG), which is a device independent way of describing graphical objects in XML. SVGplot would render using a commercial SVG "viewer" such as the Adobe SVG Viewer.

Additionally, SVGPlot could be wrapped as a server, like X11, and implemented as a Web Service (see below), using either pure XML, or if it needed to be more method oriented, XML-RPC or SOAP.

If developed, SVGPlot would probably be a very attractive tool outside SLAC. W3C has already shown interest.

Note that SVGPlot doesn't, in the configuration described above, allow for interactive input – it doesn't do buttons, queries or dialogs, as X does. Some other mechanism would have to be found for that (or we extend SVG itself!)

2.2 XAL Applications and Web Services

Unlike the SCP on VMS, it seems inevitable that future physics applications will exist in a multi-platform, multi-process control system. "Web Services" is a W3C standard framework and toolkit for rapid application development in wide-area distributed environments, implemented by Microsoft, Sun and IBM among others. A "Web Service" is basically any program which communicates through XML, describes its interface in Web Services Description Language (WSDL), and makes its availability known through the Universal Description Discovery and Integration system (UDDI). Web Services will take a similar role to "shareables" in the SCP, although each will more rigorously implement some specific function (see Figure 2). It's possible that the Grid middleware will be useful for distributed high-level controls applications.

XAL, from LANL, is an XML based accelerator modeling environment, and it is planned to be extended to a model based application framework based to some extent on the KGB's Abeans and Databush. We will investigate whether they have a framework, or applications, we can adapt. If we use it, Figure 2 would also show an Enterprise Java Beans container in an App Server running Abeans, which accesses EPICS. XAL, XAL applications, and applications we write, would be glued together as Web Services.

---

[1] It's important to make the distinction between procedural and OO, since to grow a code library over the years we have to stick to one or the other because calling OO code from procedural is hard.

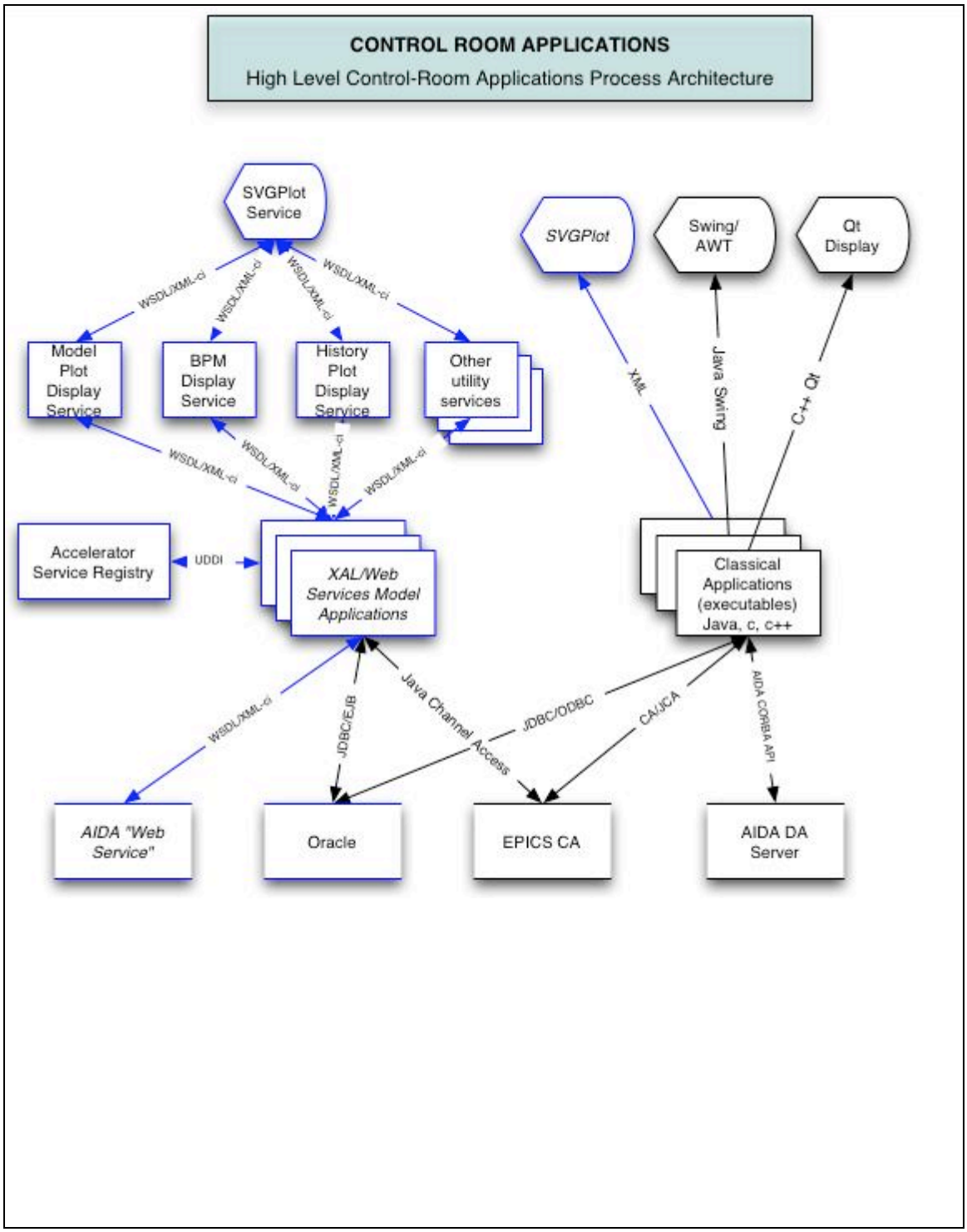

Figure 2: Control Room Applications Architecture (*AIDA data layer, Application Server, and Abeans/Databush not show*).

3 Sandbox

Ad-hoc physics application development, and algorithm validation, will be promoted by increasing the support for Matlab (see Figure 3).

3.1 EPICS Channel Access

There presently exists an EPICS data interface to Matlab sessions running on Solaris, through "Java Channel Access".

3.2 AIDA

The data available to Matlab sessions running on Solaris will be greatly increased through the development of the Accelerator Independent Data Interface (AIDA). AIDA will provide BPM data, History Data, and the model data of the PEPII Optics machine. The mechanism is that AIDA will offer a Java interface, through which Matlab will call it through its existing Java interface.

AIDA has been designed to make its client interface API available on any platform (for which there is a CORBA binding). Thereby, we could offer AIDA accessible data to Matlab sessions on Windows machines too[2].

We will offer some limited accelerator control to Matlab through this interface too, such as for magnets. However, we should probably concentrate on the read-only interfaces first, respecting that magnet control is necessary in some, to be decided, timeframe.

3.3 Accelerator toolbox

Andre Terebilo's Matlab Accelerator Toolbox is a tracking code implemented in Matlab (I think it's comprised of wrapped Tracy elements) being used by SPEAR-3 and ANL. I believe there is a plan to use that as a Matlab model engine for PEPII.

[2] The AIDA development stream on Windows was stopped last year because investigating and solving all the research programming problems entailed by developing AIDA on all platforms in parallel, was spreading our resources too thin. But, we could easily reinstitute the Windows client-side interface once a core AIDA system has been developed.

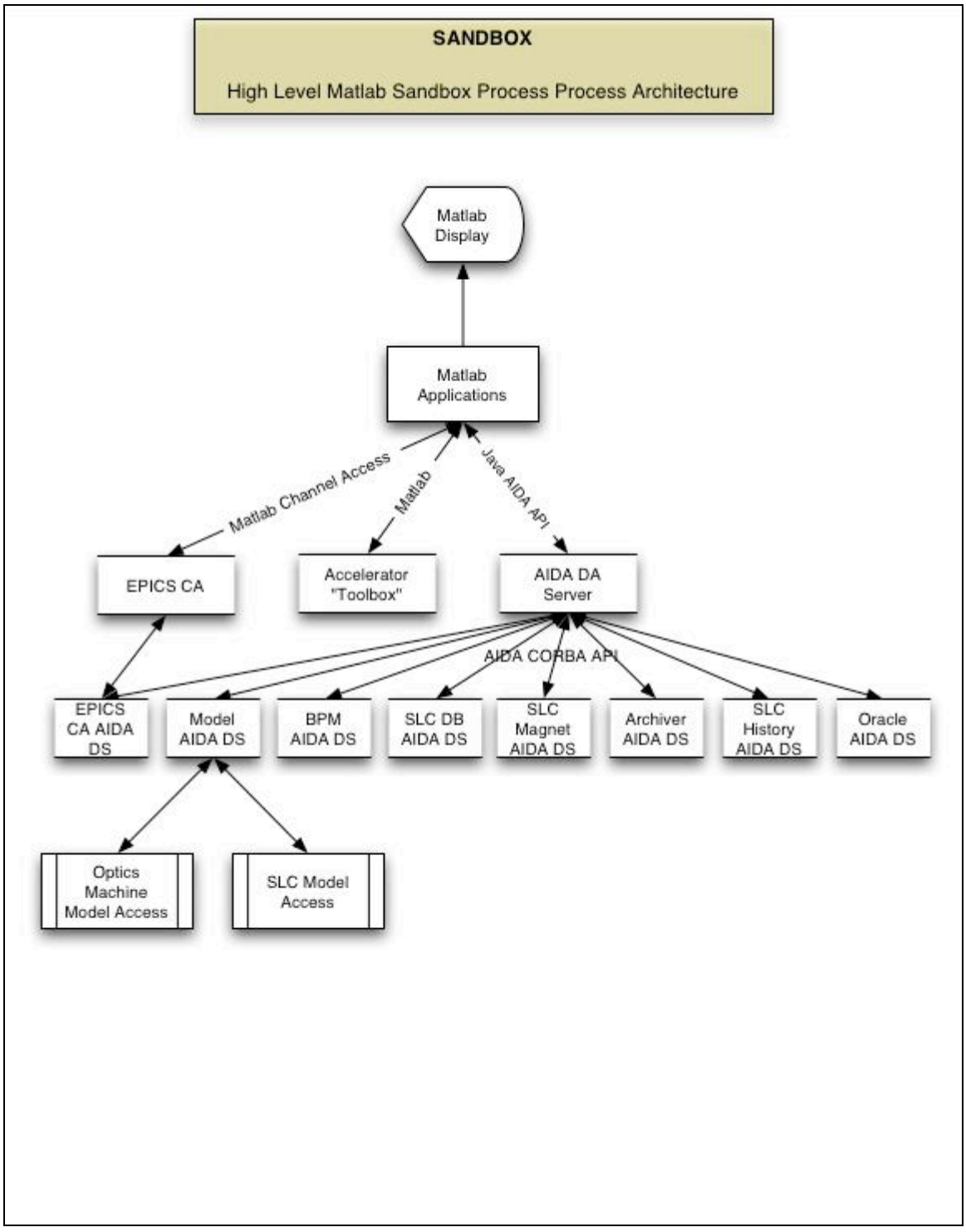

Figure 3: Matlab Sandbox Support Architecture.

4 Summary and ad hoc Displays

4.1 Summary displays

By "Summary" displays, is meant those which play a role similar to that of DCXP and SIP displays now in the VMS control system. EPICS displays will mostly fulfill this role, but there are two important cases of canned summary displays, which aren't easily accommodated by EPICS:

1. where the source data is not from the EPICS control system
2. where we want to put the display on the Web.

We propose to create such displays using rapid web page authoring technology (see below).

4.2 ad-hoc and weakly-configured displays

Ad-hoc displays are read-only displays, like summary displays above, which a user such as an operator or accelerator physicist, wants to create quickly, without the need for the software Group's support. The displays may be temporary or long-term, but are not part of the normal control room operation. These are things like a personalized plot of some archived device, or a table of all Artemis entries in the last 24 hours. It's true, you can get that from the Artemis web, but if that data is in Oracle, then a user can very quickly create their own PHP or EJB application to access it; and if it's an EJB it can be dynamically updating.

"Weakly-configured" displays are those like the history-plots and the z-plots applications; they are simple off-line data acquisition and display engines, which have access to a large number of things all of the same type, so the formatting and presentation of the data is always basically the same but the user specifies at run-time the actual data to acquire.

4.3 Technologies for these kinds of displays

We propose all of these kinds of displays should be done through the web, using the standard industry Web Server/Application Server[3] combination. Specifically we would use an Apache web server. The Application Server may be JBOSS since it has a very good reputation, and is relatively cheap, or Oracle AS. Since SLAC has a site license for Oracle and the db to which the Application Server connects will be Oracle, it's reasonable to assume the Oracle AS is optimized for Oracle. These two would be hosted on some Unix node (as opposed to Windows), probably in the same network location as mccora.

The Application Server will host the usual Enterprise Java Beans (EJB) container to process "persistency"[4] for Enterprise Java Beans applications. EJB is a standard technology for heavily interactive or complicated database applications.

A second mechanism for interfacing web pages with databases (and other data acquisition mechanisms) is through "servlets". This is another "three-tier architecture for persistency". Servlets are the programs which get data for applets (the tier typically running in a browser). A servlet engine is required to host the servlets, JBOSS has a good one, Tomcat is the Open Source one that comes with Apache/Jakarta. Its job is to just to wrap the program doing the work in such a way as it's understood by the web server, since the communication from applet – to web server – to servlet is standardized.

Support for hosting Web Services are also now being implemented by Application Server providers, so we will probably use those down the line.

The simplest db interaction is through PHP (Hypertext Preprocessor), which contains simple methods for accessing databases (through the ODBC interface I think), and creating HTML text from the result.

---

[3] An "Application Server" is a server which implements the Java 2 Enterprise Edition (J2EE) Standard for three-tier (Web browser – web server – database) applications. Its basic job is to host an Enterprise Java Beans "container", and to manage the integrity of database interactions of the applications it hosts.

[4] "Persistency" is just the industry term for interactions with databases. The default EJB container knows how to do "persistency" for EJB programs, which can be implemented as applets in a browser.

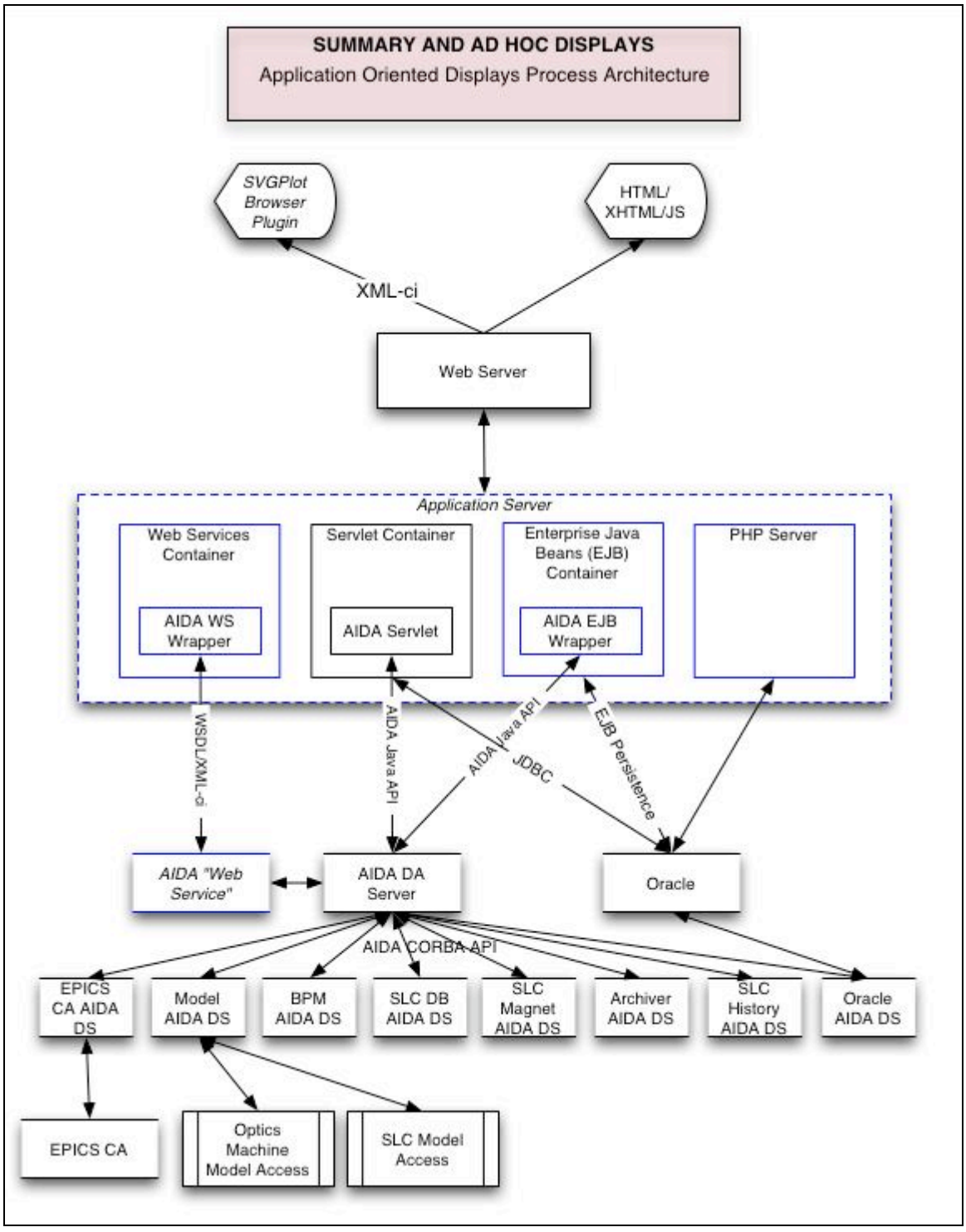

Figure 4: Architecture for simple Oracle db displays.

5 Global Considerations

5.1 Platforms

Development work will be carried out on the AFS file system and SCS supported AFS nodes. Production process will be hosted mainly on Solaris machines on the lavc network (with some exceptions). We presume that the lone "optics" machine, and compiled matlab machine, will be superceded by a system of such machines, and become more closely standardized and integrated as part of the new Unix Development Environment. Lattice file version control will be moved the Unix Development Environment's CVS system.

It may be that we will move to Linux for production side processes in future, but not before we have used Linux for development, and that depends on SCS' support of a Linux login environment. Windows variants will not be used for production, and its use for development will be monitored to avoid the case that Windows creeps into the control system and thereby requires significantly extended support.

5.2 Risks

A large number of protocols and APIs are used in the system interconnects. Each arrow represents an interface API, and these are of various types and protocols. This represents a risk because of the number of technologies we need to learn and support. The worst cases of new technology are shown in blue in the diagrams. However, other than a few in-house protocols, these new interfaces are industry norms, and moving into the use of Application Servers (eg Oracle AS), which is a well-established industry framework, will allow us to learn and employ the protocols in a well –structured environment rich in examples. We can develop a skill-matrix to drive a skill-matching schedule; and we are completing a new web-site structure that will promote effective programmers' documentation.

Web Services specialists are presently in heavy demand in the job-market. Fortunately, we need only one or two real specialists.

A number of display technologies are employed (Qt, Swing, Matlab, SVG), which is bad for developing core competence and code reuse. Also, the large number of display technologies makes inventing a single user-interface driver like button-macros very unlikely. If we can find a way to use Matlab plotting from C, C++, and Java, we should standardize on Matlab for all displays except "Summary and Ad-hoc", which should continue to use SVGPlot or some applet. Maybe JoiMint from DESY now includes such an applet?

Employing commercial web browsers as the thin-client display medium, means upgrading hardware and personnel practices for remote displays – no more old NCDs.

Using client-side Java, and web-browsers as the thin-client display medium, creates a dependency on individual users' choice of desktop web-browser. We can't control how those are configured, and there are a large number of combinations of platforms and implementations of browsers. Unfortunately, it may well be that their configuration has to be quite specific to support each of the middleware technologies we intended to use. We may have to limit ourselves to some supported configurations – we at least have a much more limited problem than a commercial web site in this respect.

Deliberately putting off the decision about the level at which to migrate VMS services means that if the day does come when we have to port everything from VMS, there will be that much more to do. However, we should have learnt a lot by the time that day comes, and have a framework in which to make the move.

Using a SLAC-wide application development environment based on Application Server technology takes us significantly closer to SCS. However, that very proximity may be what we need to tip us into a co-operative resource-sharing environment. In the early days, we will work with SCS to create what we need together. Then, we will develop our own instances of these technologies for production.