A SERVER-LEVEL API FOR EPICS

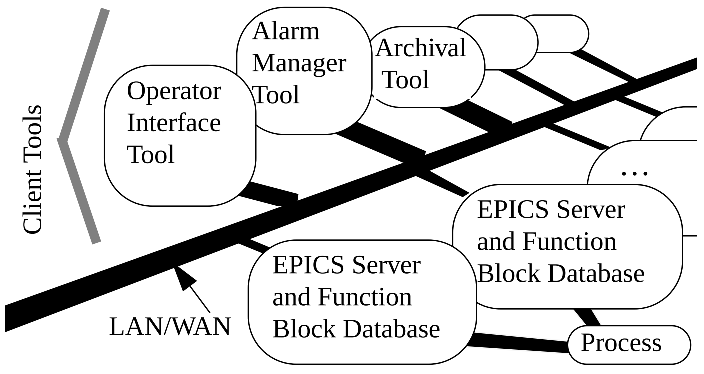
J. O. Hill
LOS ALAMOS NATIONAL LABORATORY GROUP AOT-8
MS H820, LOS ALAMOS, NM 87545, USA

The existing Experimental Physics and Industrial Control System (EPICS) applications programmers' interface - Channel Access - has in our experience been a catalyst for efficient collaborative software development. Having seen real cost and quality benefits resulting from the adoption of modular system design techniques in a control system context, we propose a new applications programmers interface (API) for EPICS to be installed just beneath the existing channel access server. This new API will encapsulate the EPICS IO system making it another modular, replaceable software component. We will thereby eliminate several existing EPICS limitations including: only one choice for the front end operating system; only one EPICS front end architecture; difficulties exporting process variables from client-side applications and difficulties creating transient process variables. Potential applications are gateways between existing control systems and the expanding EPICS tool set, gateways between non-essential users and the live control system, access to alternative data stores such as commercial databases and light-weight IO controller implementations. We believe that this API will result in greater freedom to pick and choose components of EPICS and ultimately a wider application of EPICS.

1.0 INTRODUCTION

The Experimental Physics and Industrial Control System (EPICS) is a process control and data acquisition software toolkit in use at a number of sites world wide. The software was designed for general utility and has been successfully installed into a wide range of applications including particle accelerators, experimental physics detectors, astronomical observatories, municipal infrastructures, petroleum refineries, and manufacturing. A scalable, fault-tolerant system that follows the "standard model"[1] can be created with the toolkit. Compilers and filters are used to instantiate control algorithms in front-end computers from function block and state-machine formalism-based input. EPICS communication occurs within a software layer called channel access (CA) that follows the client-server model and employs the internet protocols (Figure 1). A mature set of client-side tools provide operator interface, alarm handling, archival tasks, backup, restore, state sequencing, and other capabilities. Client-side interfaces are provided to commercial packages such as IDL, TCL/TK, MATLAB, and Mathematica. There is also an expanding library of hardware device drivers that have been written for use with EPICS. Recently we have seen a number of sites working on generic physics and control theory applications that will interface directly with EPICS. All of these components taken together form a toolkit that allows control system installation while writing a minimum of low level code.The details can be obtained on the world-wide web[2] and from previous papers[3][4][5][6]. EPICS is very unusual among control system software packages in that it has been developed by a collaborative effort of several laboratory and industrial partners[7].

FIGURE 1. EPICS



2.0 PRESENT LIMITATIONS

There are at present several limitations that we would like to remove from EPICS. Currently, the EPICS client-side tool set only interfaces with the EPICS IO function block database. We believe that the generic nature of the EPICS client-side tools and the channel access protocol should permit their use in a wide range of applications. For example, there is no compelling reason why the client-side tools could not be used unmodified with the many present and future control systems developed at other labs. When designing the software for a control system there are trade-offs required between maintaining stability for installed systems and allowing evolution for new projects. For instance, we can foresee many potential improvements to the EPICS IO function block database but are reluctant to make abrupt changes because we must maintain stability for the many existing EPICS installations. Modest architectural changes to EPICS would make it possible for present and evolved versions of the IO function block database to coexist in the same control system, thereby allowing rapid evolution of new capabilities while maintaining stability.

Another limitation results when components from the client-side tool kit must compute an intermediate result. For instance an operator screen might command another tool to initiate an emittance scan and then display the result when it completes. Where should this result be stored? Presently within EPICS all communication of this nature inevitably occurs through “soft records” in the IO function block database. There are several problems with this solution. One of these is that if two operator screens initiate an emittance calculation at almost the same time but with different input parameters then one operator will be required to wait until the other operator’s emittance calculation is completed. Worse still, if mutual exclusion isn’t built into the emittance calculation program the operators risk a collision where their answer may not result from the input parameters specified. Another negative is that resources are consumed in a front end IO controller for storing and updating emittance calculation parameters when this may not have been that processor’s original purpose. Our conclusion is that these intermediate parameters are not now stored in a natural location within the client server hierarchy. If, when it was appropriate, the parameters were stored within the tools themselves, we would see improved interconnection within the EPICS tool set and less load on critical front end machines.

Note that some of the above “limitations” are not limitations unless you are committed to a tool-based approach. We could interface the client-side tools to other control systems by making large source code changes in them at each site that they are used. We could rapidly evolve the software without regard for the stability required by installed systems. We could also combine existing tools together by making application-specific source code changes each time that they are used in a different combination. However we have not chosen this path because we would not benefit in the long term from the labor of individuals at other sites unless software developed at one site can be used at another without the need for site-specific source code changes.

We believe that a distributed system design employing direct “single-hop” connections between individual hosts has advantages related to improved latency, dispensation of load, and fault tolerance. However there are low priority clients of the control system for which the above factors are less important. For these low-priority clients a more important concern may be to minimize load on critical servers in the front end IO controllers. Each client connecting to a particular server drains resources from that server and the network. Modest architectural changes would allow multiple low-priority clients to share one connection to a server in a critical machine. These changes would guarantee that low priority clients do not overload a critical server or network, and therefore compromise critical functions in the control system. In this way unrestricted use of the system by low priority clients could be safely implemented.

3.0 COST AND QUALITY

When developing software there are many metrics to maximize in order to obtain the best design. However the dominant factor related to the cost and quality of a software product is the size of the software distribution. Increased distribution always lowers the per site costs. Increased distribution also decreases the probability of becoming the site that discovers (and therefore isolates) an esoteric bug. This leads to the conclusion that increased distribution also improves quality. Our overriding challenge as software designers is to identify common patterns of usage and adopt a common infrastructure so that we maximize the number of users of each and every line of code. Therefore to minimize cost and maximize quality it is necessary for us to evolve the architecture of our systems as we see new opportunities for common infrastructure.

4.0 A NEW SERVER-LEVEL API FOR EPICS

A new server-level application programmer's interface (API) has been added to EPICS. Previously the EPICS server was only able to communicate with the EPICS IO function block data base and this was the only source and destination of data in the system. A diagram depicting the evolution of the software architecture is shown in Figure 2. With a server API it is now possible for EPICS to communicate with alternative data stores in addition to the function block database. Note that on the left side of the figure the server source code and the IO function block source code are tightly coupled. This results in a lower utility of both components. On the right in the figure the server is packaged as a library with a carefully designed API. This allows the server library to be used not just with the function block database but also with many other software components. We will refer to these new components as server-side tools. This small modification to the system architecture opens up a number of new server-side possibilities and therefore proportionately increases the utility of the existing EPICS client-side tool set (Figure 3).

FIGURE 2. SOFTWARE LAYERING WITHIN EPICS BEFORE AND AFTER A SERVER-LEVEL API

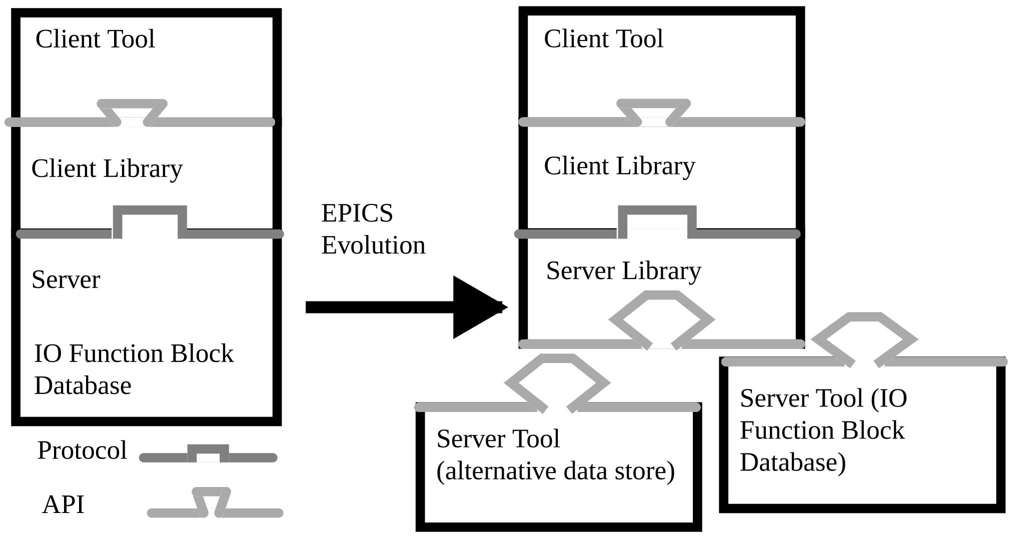
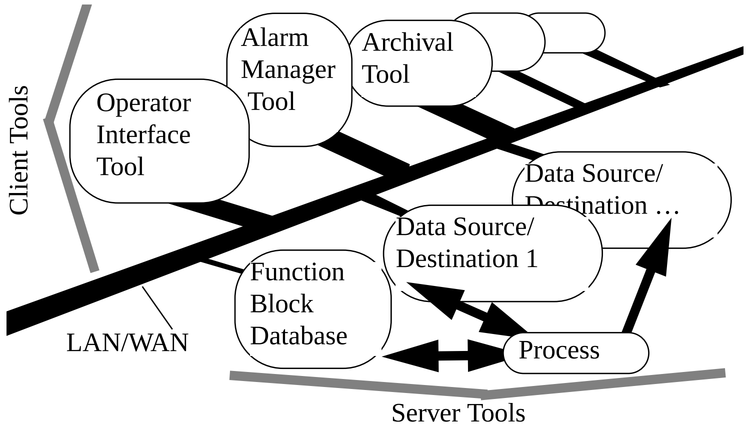

Client Tool
Client Library
Server
IO Function Block Database
Protocol
API
EPICS Evolution
Client Tool
Client Library
Server Library
Server Tool (alternative data store)
Server Tool (IO Function Block Database)

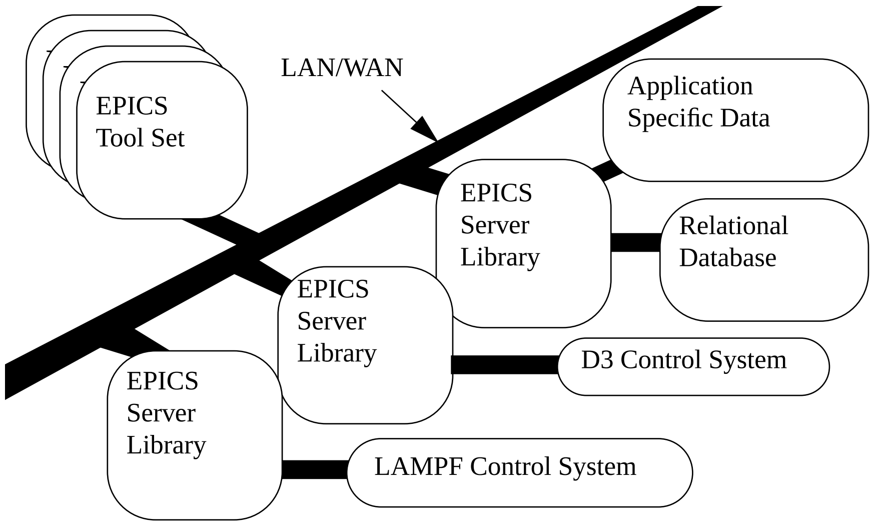
FIGURE 3. SERVER-SIDE API EXPANDS THE UTILITY OF THE EPICS TOOL SET



5.0 POTENTIAL APPLICATIONS

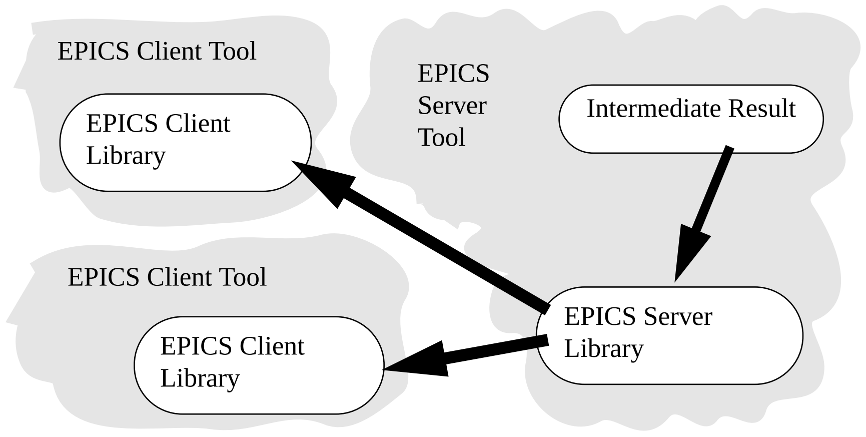
It is difficult to anticipate the full range of applications that might take advantage of the new interface. We list a few of the more important ones here in order to illustrate the practicality of this change. Perhaps the most important application will be to create EPICS server-side tools that will translate requests made by EPICS clients into actions made by foreign control systems (Figure 4). A prototype of this type of EPICS server (based on modifications to the original EPICS server source code) was demonstrated by Gabor Csuka at DESY and was used to interface between EPICS client-side tools and the D3 control system[8]. Next, the DESY-modified source code was further enhanced by Stuart Schaller at LANL for use as a gateway between EPICS client-side tools and the LAMPF/PSR control system[9]. We anticipate that the new server-level API for EPICS will become an integral part of similar projects. We predict a significant reduction in the labor required to implement and maintain gateways of this type because the new API is designed for this use and therefore cleanly packages server library functionality while carefully minimizing the source code that must be provided in-between the server library and the alternative data store (in the server-side tool).

FIGURE 4. FOREIGN AND/OR ALTERNATIVE DATA STORES



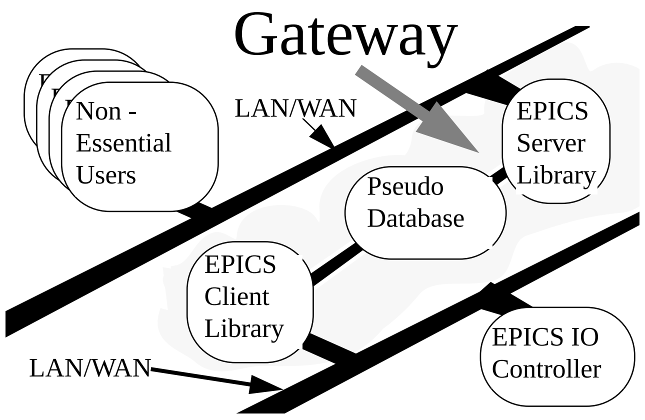
Another important application of the server-side API will occur when there is a tool that will receive certain parameters, perform some requested function, and then provide a result in the form of another parameter. In the past EPICS required that these intermediate parameters (between an EPICS client-side tool and some other tool) be stored in the IO function block database in the form of “soft” records. As stated above this isn’t a natural place to store these parameters. The new server API allows *any* EPICS tool to export incoming and outgoing parameters (Figure 5). This approach allows new and existing applications to take advantage of the EPICS client-side tool set, with minimal effort, while avoiding the limitations described above. For example a modeling program might wish to take certain input from an operator interface tool and provide output back to the operator interface tool, an archival tool, and an IO controller. The new server-level API and associated libraries will allow the various elements of EPICS to be combined easily and then recombined in response to the changing requirements of a particular situation. The interconnection between tools will be determined by configuration and not by source code changes.

FIGURE 5. INTERMEDIATE RESULTS EXPORTED FROM A NATURAL LOCATION



Another important application of the server-side API is required when a large number of low-priority clients need access to an operational EPICS system. In this situation each client consumes system resources in one or more of the servers. If the number of low priority clients becomes too large there is the possibility that excessive loading might compromise critical functions performed by the server's processor. The new server-level API will make it possible to combine into one program the server and client libraries in order to create a gateway between low priority clients and critical machines. This gateway would serve as a proxy and guarantee that N clients consume no more resources on the critical machine than one client would consume by itself (Figure 6). Of course the penalty paid will be increased latency for low priority clients that are required to pass through a gateway. In return, clients that use the gateway receive less restricted access to the system because there is no longer any concern that they might overload it.

FIGURE 6. GATEWAY OFF-LOADS LOW PRIORITY CLIENTS



As a final example, the new server API will provide the necessary freedom to develop new front-end architectures for EPICS. Since it will be easy to install the server library into many different applications, there is the possibility of light-weight IO controller implementations which are dedicated to a particular task such as closed loop control. Since the design and development of these new architectures can proceed independently of the function block IO database source code maintenance we will be confident that we can add new features without risk of breaking existing installations. Since the new server library is being written to run on multiple platforms including UNIX, VxWorks, MS windows and VMS, application designers will have the freedom to choose the operating system that best meets their needs.

6.0 INTERFACING DATA WITH THE CLIENT-SIDE TOOL SET

It is straightforward to interface data with the EPICS client-side tool set using the new server-level API. The application surrounding the data will become an EPICS server-level tool and the data will be exported as an EPICS process variable. A server-level tool must create a server instance and post change-of-state events when the data is modified. The server tool must also supply functions to be called when the process variable is located (existence test), attached to, detached from, read or written. A function that returns a valid range for the data must also be supplied. A server tool supplied function is also called when the client initiates or terminates monitoring the state of the data. The functions above define the minimum interface. Additional interfaces are required only if you wish to access a greater range of EPICS functionality. For instance, additional functions must be provided by a server-level tool if it needs finer control over a client’s access rights when a client is modifying or reading the data.

7.0 CONCLUSION

With the new server-level API, projects will have greater freedom to choose and combine components of EPICS. The new interface will allow for modular development of new subsystems with minimal duplication of effort. The new interface is designed to encourage the development of tools for one project that may be used effectively on other projects and at other sites. A wide range of existing and future software systems will be able to efficiently take advantage of the EPICS tool set. The new API allows this to occur even when these system share little else architecturally with EPICS. Exporting data to the EPICS client-side tool set is straightforward when the new API is used.

[1] B. Kuiper, “Issues in Accelerator Controls”, Proc. ICALEPCS, Tsukuba, Japan, 1991, pp 602-611.

[2] W. McDowell et al., “EPICS Home Page”, “http://epics.aps.anl.gov/asd/controls/epics_home.html”.

[3] L. Dalesio et al. “The Experimental Physics and Industrial Control System Architecture: Past, Present, and Future”, Proc. ICALEPCS, Berlin, Germany, 1993, pp 179-184.

[4] L. Dalesio et al, “The Los Alamos Accelerator Control System Database: A Generic Instrumentation Interface”, Proc. ICALEPCS, Vancouver, Canada, 1989, pp 405-407.

[5] J. Hill, “Channel Access: A Software Bus for the LAACS”, Proc. ICALEPCS, Vancouver, Canada, 1989, pp 352-355.

[6] J. Hill, “EPICS Communication Loss Management”, Proc. ICALEPCS, Berlin, Germany, 1993, pp 218-220.

[7] M. Knott et al, “EPICS: A Control System Software Co-development Success Story”, Proc. ICALEPCS, Berlin, Germany, 1993, pp 486-491.

[8] M. Clausen, G. Csuka, Internal Communication.

[9] S. Schaller et al, “Generalized Control And Data Access At The LANSCE Accelerator Complex —Gateways, Migrators, And Other Servers”, these proceedings.