DISTRIBUTED SOFTWARE DEVELOPMENT IN THE EPICS COLLABORATION*

Leo R. Dalesio, Los Alamos National Laboratory (LANL)
Martin Kraimer, Argonne National Laboratory (ANL)
William Watson, Continuous Electron Beam Accelerator Facility (CEBAF)
Matthias Clausen, Deutches Elektronen–Synchrontron (DESY)

ABSTRACT

The Experimental Physics and Industrial Control System (EPICS) collaboration now consists of many accelerator, astronomy and particle detector projects. Many of these groups contribute code to the collaboration. The interest of the sponsoring laboratory is always the primary driver in decisions about developing new code. As a side affect of meeting the needs of a given project, the collaboration receives new developments. This paper will study the overall composition of EPICS with regards to distributed development, the sociological and technical environment in which these developments are made, the problems in distributed software development, and the benefits of developing software at multiple sites.

* Work supported under the U.S. Department of Energy, Office of Basic Energy Sciences under Contract Nos. (W–7405–ENG–36), (W–31–109–ENG–38) and (DE–AC02–89ER40486)

INTRODUCTION

The Experimental Physics and Industrial Control System (EPICS)[1] is currently in use on over 30 projects in Europe, Asia, and North America. It is used in particle accelerators, particle detectors, astronomy, commercial industry and industrial research. Every site in the collaboration has had to make some additions for their local projects. Many of these modifications have been reintegrated into subsequent releases of EPICS. Most of the sites in the collaboration upgrade to the latest releases to take advantage of the new packages and features. This environment poses some challenges and opportunity in code development, integration, and installation.

SOFTWARE DEVELOPMENT

There are three basic methods for developing code in this environment: independent development, successive development, and joint development. Independent development occurs when code is added on to existing interfaces that were specifically designed to support extendibility. Successive development is characterized by the modification of existing code. Joint development will be distinguished by the involvement of others in the various stages of development: requirements, functional specifications, design, and implementation. It is true that there is a great deal of overlap between these three methods, but they will be considered separately. In each section we will define the method through examples. Then we will look at the benefit and potential problems unique to each method.

SOFTWARE DEVELOPMENT: INDEPENDENT

When EPICS was first designed, one of the key goals was extendability. In the initial release the clearest interface for extensions was in the communications layer. This architecture is shown in figure l. The client/server model was chosen and a well defined client interface was developed[2]. Immediately, there were several clients developed for operator viewing, data archiving, and the support of sequential control. There were also some special purpose applications developed for beam physics: neural network, phase scanning and emittance measurements. As more collaborators joined, more client codes were developed. There are clients for interfaces to commercial data visualization packages such as: SL–GMS, Dataviews, and Labview. There are also connections to analysis packages such as: Matlab, Mathematica, IDL, and PV–Wave. Interfaces were also developed for tcl/TK, WINGZ, and other in–house physics applications. Commercial packages and developed packages continue to be integrated using this interface. This well defined interface allowed developers to work without interaction and coordination of the authors.

In the original release, new database functions or hardware support required modifications to existing code. To extend the database functions and I/O support coordination with the author was required. In release 3.0, the database and hardware support were provided with a clean interface for extensions. The database functions and hardware support joined the client software in the amount of activity that it experienced. The only area where there was no interface, was in the server – the EPICS database was the only data store available to the applications. In response, a portable channel access server application programmer interface (API)[3] and a Control DEVice (CDEV) API[4] are both under development. With the completion of these two APIs, complete inter–operability of EPICS clients and EPICS servers with any other clients or servers will be possible. With these two APIs, the EPICS suite will be completely extendable at every level.

Independent development produces the fastest result and gives the collaborators control over the outcome of their program. When these developments are done without consulting others, they tend to be of limited use and are frequently less than optimal. These developments produce a more complete result when some time is spent to work with others on the functional specification and design. Frequently, different sites are solving the same problem without joining forces. The ability to do independent development to extend and replace portions of the system is essential for successfully supporting research systems and results in a great deal of new functionality in this collaboration.

SOFTWARE DEVELOPMENT: SUCCESSIVE

Successive development is accomplished in several ways: iterative development by the author, hand the code to another author or cannibalize a code for a specific function. See figure 2. As this is an evolutionary approach, the results are typically beneficial. The main concern in this area is that the new code is upwardly compatible.

All general purpose applications in EPICS have gone through upgrades based on user feedback. These modifications include added features, changed behavior, improved robustness, and increased ease of use. These

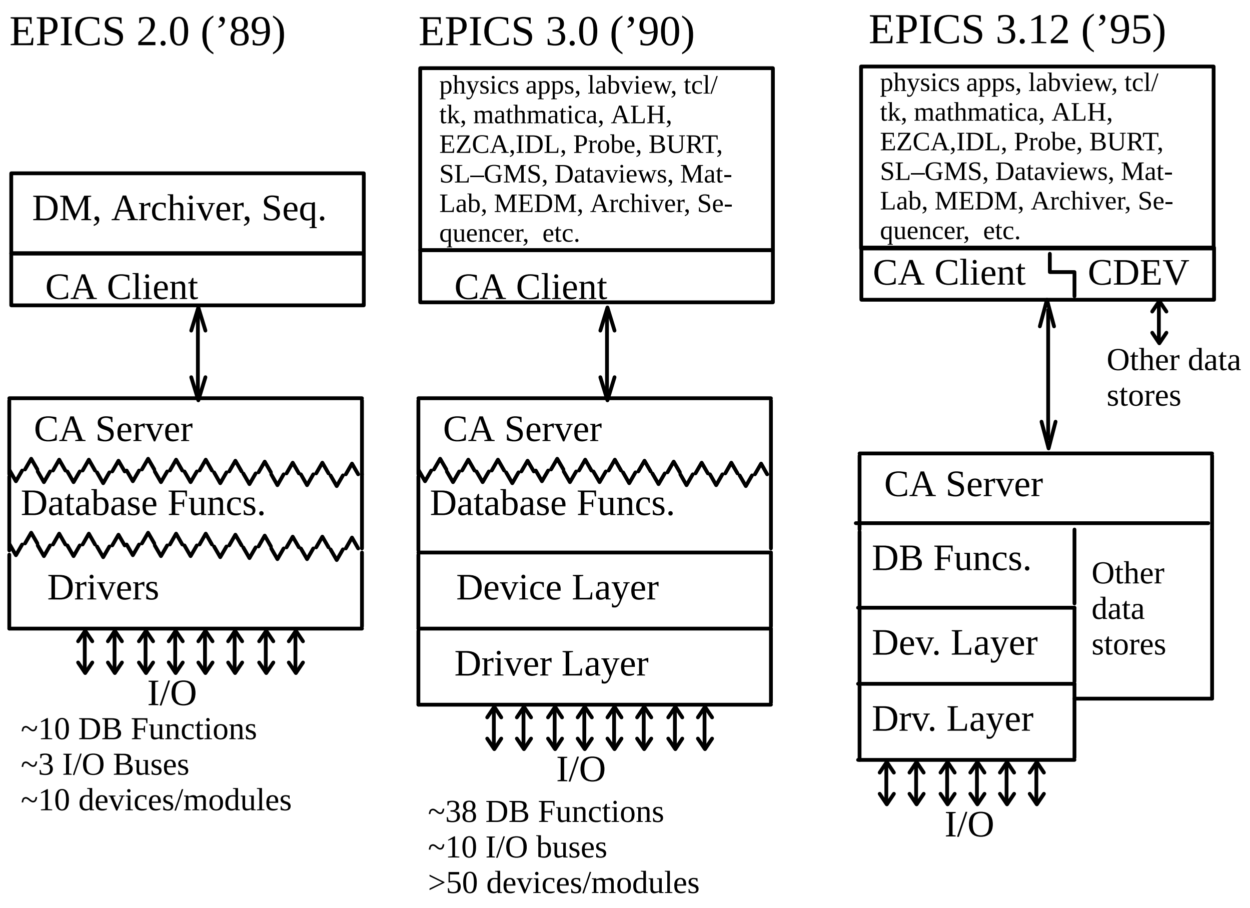

Figure 1. Program Interfaces for Extensions

modifications are made at the discretion of the author. This limitation can be overcome by passing the development of a package between authors and projects. This is particularly useful when the original work is not quite complete and the author is not able to continue. The EPICS database was the earliest example of this. The CAMAC driver was also developed through the successive efforts of developers at CEBAF, Duke FEL, and LAMPF. Successive development can also be used to take advantage of a previous code for a specific problem. As mentioned earlier, the original channel access server was tightly integrated to the EPICS database. DESY had a need for a data server to an existing system. They started with the CA server, ported it to VMS and connected it to the existing cryogenic control system. This server was subsequently acquired by the LAMPF controls group at Los Alamos and used to provide a gateway between EPICS and the LAMPF Control System (LCS)[5]. Finally, the need to integrate these existing systems came a full circle and provided one of the driving forces for developing a portable channel access server. Successive software development provides existing codes with new features, uses distributed resources for completion of codes, or provides developers with a starting point for their task.

All packages that have ever been introduced into the release have had improvements. Most of these improvements are the result of feedback from the collaboration. In every case, there is a need for upward compatibility. Within the inherent constraints of a software package, evolution is the safest road to perfection.

JOINT SOFTWARE DEVELOPMENT

Joint development is done between groups. Many of the successive and add–on developments are made after joint discussion of functional or performance improvements. Since the collaboration has started, most newly written applications have been made in this fashion. Existing applications use this method to determine which modifications will be made and how they will be incorporated. For most packages, requirements and functional specifications are

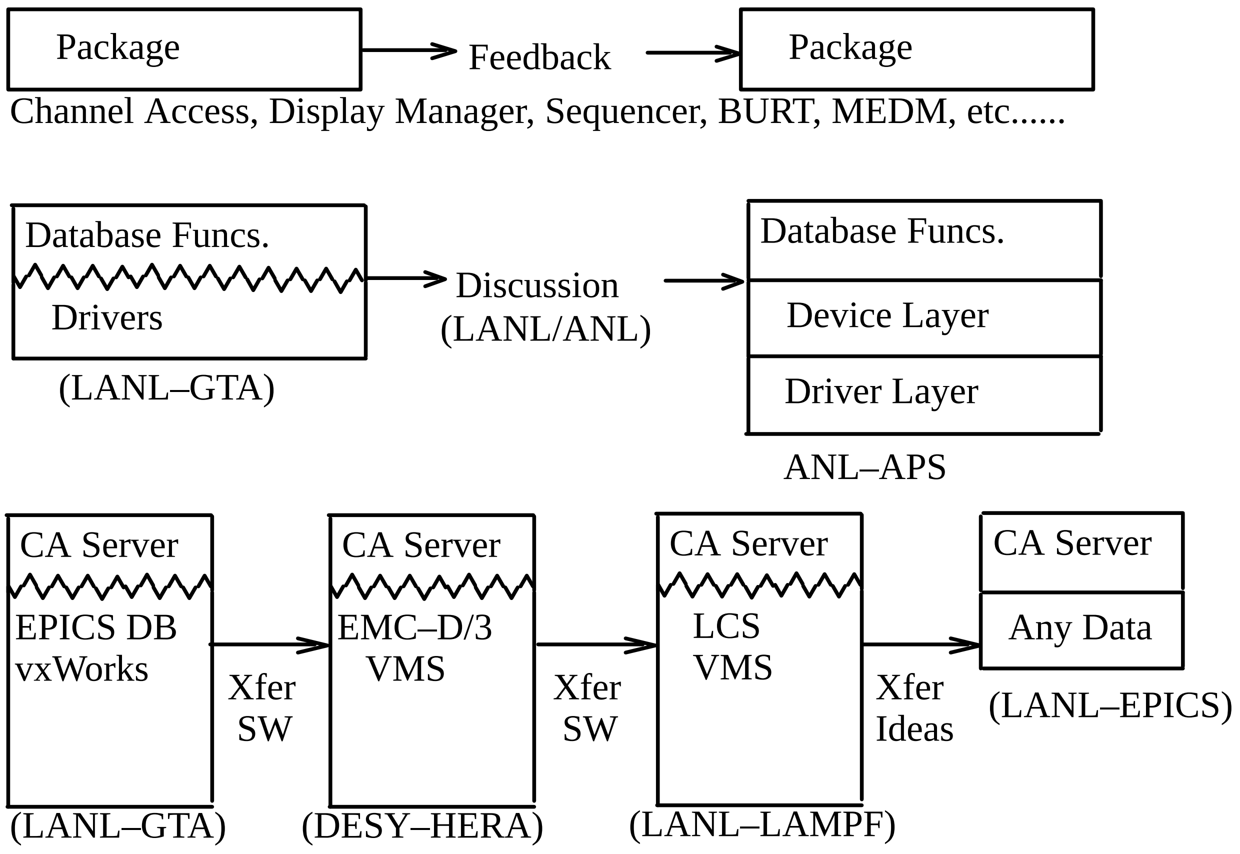

Figure 2. Successive Developments

made in a large group. See figure 3. The design is then made by a small group of people that have some experience with similar problems. The implementation is then made by a single institution. The alarm handler was the earliest joint project. The most recent have been CDEV and the portable channel access server. In some instances, the development can be distributed as well. The archiver upgrade that is currently in progress has some well defined pieces that are intended to be developed by different institutions. CEBAF, Los Alamos and Argonne have expressed interest in contributing designers and programmers to this effort. As all of these programs are in operation, it is difficult to get enough funding to provide this general solution. Hopefully everyone will benefit. Distributed effort can also be used to investigate new technology. A proposal was recently accepted for analyzing the usefulness of Distributed Computing Environment (DCE) for the EPICS community. It will be evaluated for use as a distributed name server, a communication protocol and a distributed file manager. The evaluation will be made by people at Los Alamos, Argonne, Lawrence Berkeley and CEBAF. In some cases, this method has produced some very complete (perhaps grandiose) plans. In these instances, great ideas were generated and nothing useful was delivered.

All of the general and widely used tools in EPICS are developed and modified using this method. In many cases this approach is very time consuming. Communication is more difficult. We manage using an e–mail exploder, video conferences, and bi–annual collaboration meetings[6]. For technology evaluation and new developments, it helps to combine the limited resources at several sites to produce a more complete and timely result. In every case, a more complete understanding results from the consideration of all ideas. Successful joint development requires a balance between completeness and resources: schedule and personnel.

REINTEGRATION TEST, DISTRIBUTION AND SUPPORT

There is a great deal of effort involved in the reintegration, test, distribution and support for new releases. This task is made more challenging by the absence of direct funding for EPICS. The effort required to support these

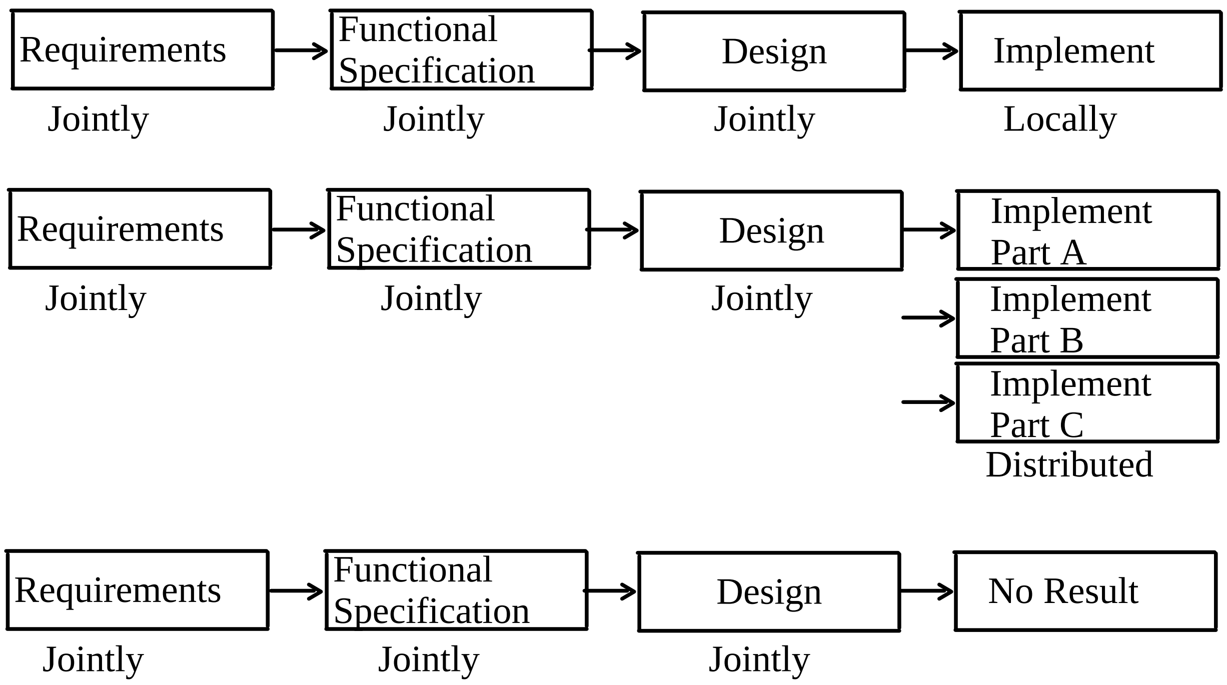

Figure 3. Joint Development of Software

tasks is too large for any operational project to fund and we no longer have any large development project team funded to handle these functions. To provide these functions we continually look for ways to reduce the cost and distribute the effort.

The reintegration of modifications is necessary to make the developments throughout the collaboration available to everyone. This effort is currently provided by the Advanced Photon Source (APS) at Argonne National Laboratory (ANL). We have reduced the scope of this effort by creating clean interfaces for extending EPICS. In the areas where this is not yet provided, code merging is frequently required. We have reduced this effort somewhat by taking advantage of source/release control tools. Under Source Code Control System (SCCS) we were required to perform source compares to merge changes from different sources. With Concurrent Version System (CVS) there is a merge function that interacts with the integrator. In the future, we may use DCE to assure that the master source is only checked out at one site at a time.

After the code is reintegrated, verification is required. Once again, it is the controls group at APS that provides the function. The largest reduction in effort was obtained by dividing EPICS into base and extensions. The base portion is handled at APS and the extensions are handled by the authors. The base consists of: channel access client/server, selected database device and driver support and the sequencer. Extensions include all of the channel access clients: alarm handler, display managers, interfaces to commercial products, archiver, etc... Another cost reduction step was to reduce the frequency of releases. Finally, there are a number of test suites that have been developed that automate some of the testing and provide a recipe to follow for the others. Verification done correctly before a release is much more cost effective than having every site find and fix errors that are introduced by upgrades.

The distribution of new releases has been distributed to different sites by platform. Lawrence Berkeley Laboratory (LBL) supports SunOS and WindowNT, Stanford Synchrotron Radiation Laboratory (SSRL) supports DEC OSF1, the Royal Greenwich Observatory (RGO) supports Solaris, the University of Chicago (UC) supports DEC VMS , SGI is supported by UC, Dupont–Northwestern–Dow beamline at APS supports Linux, and CEBAF supports the HPUX release. Only VMS, Linux, and WNT have the channel access client library supported. By distributing the build and distribution responsibility, we have provided the collaboration with expertise for a large number of operating systems while causing minimum cost to any one site.

Support for EPICS packages is also distributed. The build is supported in the same fashion as the distribution. Problems, suggestions, and extensions are frequently discussed and handled over the mail exploder. Ultimately, the author is responsible for handling these issues.

We consider reintegration, test, distribution and support crucial to the success of the members of the collaboration. To maximize the benefit of our limited resources, we identified the items that were critical and provided central support. Distributing responsibility for platforms to experts and the maintenance of code to the authors provides better, but perhaps not more timely, support. The lack of direct support for EPICS has been overcome by the willingness of APS to support the central effort, new configuration control technology and the willingness of other collaboration members to pitch in.

UPGRADING TO A NEW RELEASE

The value of a new release must exceed the cost of the upgrade. From the beginning, all configuration tools have had an ASCII format that could be reported from one release and then read into the new data structures of the new release. The major version number in EPICS has referred to the ability to communicate between nodes. Version 3.0 (1991) through version 3.12 (current) have supported communication between all versions of the clients and servers. Any combination of clients and servers works. This allows projects to upgrade on a node by node basis without re–compilation, as long as the major release number is the same. With the recent development of complete application environments and clean interfaces for extensions, local variations of an EPICS release are now supported seamlessly through an upgrade. This combination of features allows projects to make use of new releases with a minimum impact on their project.

CONCLUSION

The EPICS collaboration has been successful as exhibited by its existence since 1990 and the continued growth in the number of programs and the areas of use. Clean interfaces have supported independent and collaborative development. The continued development and added resources of the collaboration have given us the ability to integrate new technology and new methodology as additions or alternatives to our existing functionality. Backward compatibility, reintegration, and test enable these new developments to be used at all sites in the collaboration. The combined experience of the engineers and physicists in a collaborative environment provides a more complete solution to the problems we are solving.

ACKNOWLEDGEMENT

The EPICS collaboration is made up of individuals. It is the ability of these individuals to come together and share problems, ideas, criticism, and solutions that makes this work. It is a real privilege to be able to work with each other in a very concrete way and share the lessons of our combined experience. The access to these individuals is the greatest benefit of all.

REFERENCES

[1] Dalesio, L.R., et. al. "The Experiemental Physics and Industrial Control System architecture: past, present, and future, " in *Proceedings of International Conference on Accelerator and Large Experimental Physics Control Systems*, W. Busse and M.C. Crowley–Milling, Eds. (ICALEPCS, Berlin, Germany, 1993), pp. 179–184.

[2] Hill, J.O., "Channel Access: A SOftware Bus for the LAACS," in *Proceedings of International Conference on Accelerator and Large Experimental Physics Control Systems*, D. Gurd and M.C. Crowley–Milling, Eds. (ICALEPCS, Vancouver, British Columbia, Canada, 1989), pp. 352–355.

[3] Hill, J.O., "A Server Level API for EPICS," submitted to this conference.

[4] Chen, J., Akers, W., Heyes, G., Wu, D., Watson, W..”An Object–Oriented Class Library for Developing Device Control Applications," submitted to this conference.

[5] Schaller, S.C., Oothoudt, M.A., “Generalized Control and Data Access at the LANSCE Accelerator Complex — Gateways and Migrators,” submitted to this conference.

[6] Knott, M., “Communication in Support of Software Sharing and Collaborative Development,” submitted to this conference.