Sharing Control System Software

Peter Clout
Vista Control Systems, Inc.
134 B Eastgate Drive, Los Alamos, NM 87544 USA

Abstract

Building a custom accelerator control system requires effort in the range of 30–100 person–years. This represents a significant investment of time, effort, and risk, as well as challenges for management. Even when the system is successful, the software has not yet been applied to the particular project; no custom control algorithms, either engineering or physics–based, have been implemented; and the system has not been documented for long–term maintenance and use. This paper reviews the requirements for sharing software between accelerator control system projects. It also reviews the three mechanisms by which control system software has been shared in the past and is being shared now, as well as some of the experiences. After reviewing the mechanisms and experiences, one can conclude there is no one best solution. The right software sharing mechanism depends upon the needs of the client site, the client resources available, and the services the provider can give.

I. PROBLEMS WITH DEVELOPING CONTROL SYSTEM SOFTWARE

Sharing software is a solution; the problem is the risk, cost and time taken to develop the control system software. Before any application to the accelerator in question, the control system software represents an investment of between 30 and 100 or more person–years [ref. 1]. Multiplying this figure by any developed country's average programmer's salary with overheads easily turns this into multi–million dollar investments for just one part, albeit important, of the overall control system. If one then adds in the maintenance, support and improvement of the software over the life of the accelerator, the number can easily be multiplied by factors of between two and five to obtain the lifetime cost.

Clearly, such expenditures should not be entered into without examining the alternatives.

Apart from cost, the risk of the software being incomplete or insufficient at the time it is needed is also a serious issue.

For the accelerator field as a whole, this problem is getting bigger simply because the number of accelerators is growing as new accelerator applications are developed. As evidence for this one only has to plot the growth in the attendance of the conferences such as this one. One recent estimate [ref. 2, 3] for providing desirable and reasonable control system user facilities at the major accelerators world–wide adds up to in excess of one billion dollars over ten years. The inference is that unless changes are made, only a fraction of the requirements will be met.

II. THE VALUE OF SOFTWARE

Software is intellectual property and each piece of software represents a solution to a problem or a component of a solution to a problem. Like other forms of intellectual property, the value of a piece of software is derived from

1. Understanding of the problem.
2. Analyzing the problem and its requirements towards a solution.
3. Literacy with the techniques and skill with the associated tools to be used.
4. Effort expended to provide the particular solution.

The first three could be thought of as setting the hourly rate and the last item being the multiplier that sets the cost of a solution.

Software also represents a value to the user. This value is measured by the overall satisfaction with the system. Basically, what is the cost of not having that piece of the solution? There is another component of the cost of software—risk. This is the risk that the software solution to the problem will either simply not be working when needed or will not fulfill one or more of the requirements. Custom–developed software represents the highest risk with the level of risk being determined by the size and track record of the team working on the problem.

III. DIFFICULTIES IN ACCELERATOR CONTROL SYSTEMS DEVELOPMENT

Accelerator control systems are large in the number of channels, complex in the engineering and physics of the process and uncertain in the sudden appearance of new requirements. All these factors are greater than they are in industrial systems. In developing a control system, or any other system, one cannot reduce the complexity of the system to a level below the basic complexity of the application.

From the first operation of the accelerator, one can confidently expect that the requirements on the control system will grow rapidly and in unpredictable ways as the understanding of the physics of the accelerator is developed and confirmed and the needs of operations understood. There will be the need to incorporate into the control system some of the basic physics codes that were written for accelerator design rather than accelerator operations. There will also be the need to incorporate control algorithms developed during the R&D phase and also during operations.

If the architecture and design of the control system hardware and software are not structured, open and flexible, then expanding requirements will be harder and harder to meet and the result will be a nightmare to maintain. Very often, small control systems are required at the R&D stage of a project. The ability to make a version of the final system available early on will save work and simplify project integration.

IV. REDUCING THE COST AND RISK OF ACCELERATOR CONTROL SYSTEM DEVELOPMENT

Table 1 lists some of the accelerator control system software sharing. It will be seen that there has been an increasing pace of software sharing in recent years as the advantages come to be appreciated and the complexity of the requirements and the basic tools of computing increased rapidly (compare the Plot–10 graphics library with Xlib!). The industry has developed software development methodologies and tools to support these methodologies. These help to combat the complexity of the requirements and reduce the development time for the solution.

Tools are also being developed to assist with the implementation of the design. Some of these are general–purpose tools (Visual Basic, Object Vision, OBLOG, OBLOG CASE, Dataviews, SL–GMS, IDL, etc.) which reduce the writing of code, and others are specific application shells or toolboxes (Factory Link, LabView, VXL, Wonderware, Vsystem, Basestar, RTAP and so on) Each application shell or toolbox is targeted at a particular class of applications and allows the user to start developing the specific application as soon as the initial design is complete. One can expect this picture to change rapidly in the future.

V. ISSUES IN SHARING SOFTWARE

A. Architecture

The general hardware and software architecture of the system will determine how easily the shared software will fit. This section discusses some of the issues of the operating system interface and the application interface.

1. Computer, Operating Systems, Graphics and Networking Choices

It is unfortunately still true that it is a substantial job to port a system developed using the full facilities of one choice of hardware and software to a different family of choices. Improvements in this area are slow but they are occurring (UNIX, POSIX), although other market forces like Macintosh, MSDOS and Windows/NT confuse this improvement.

2. Application Interfaces

a. I/O Hardware

In many of the more recent software architectures, this interface is standardized so that the applications do not need to know about the exact details of the hardware connection. Some form of standardized hardware access is a requirement if software is to be shared. This can be by using protocols such as are being developed and used by the European Physical Society Group on Experimental Physics Control Systems, or by using a real–time database or, indeed, both.

b. Software Data Bus

This is a software data interface for communication between the software components of the system. This can include a real–time database and embedded features such as alarming, data conversion and the storing of secondary information about each item of data. Mechanisms such as pipes and local and remote procedure calls are very simple mechanisms that know nothing about the application. Many systems have built specific control system functions and communications on top of these primitives.

c. User Interface

Different user interface environments can make the use of software from another institute difficult. Clearly, the ASCII terminal interface is common to nearly all systems (remember EBCDIC?) but there are a number of graphics interfaces in use, although the two rather different low–level interfaces, X–windows with Motif and MS–Windows, are presently the primary software interfaces.

d. Operating System Services

Programs directly use many services of the operating system for which they are written. In this case there is often considerable re–engineering to be done to port the program to another operating system. Here, even UNIX does not help as each UNIX supplier has modified UNIX for their particular view of their users' needs. The POSIX set of standards will be a great improvement once they are all finally agreed upon and commercially available. It is unfortunate that the POSIX standard most needed for accelerator controls, the real–time extensions, is the one that is yet to be agreed.

The other operating system interface issue is the file system. If one considers the three primary operating systems as MS–DOS (and Windows), UNIX (in its many different flavors) and OpenVMS (on the VAX and the Alpha/AXP) then one has three different naming conventions and restrictions and three very different sets of file structure capability.

B. Support and Maintenance

All software needs support and maintenance. This is either provided in–house for the personnel costs involved, or it is provided by the supplier of the software for a fee. If no support is available, it will still be a cost to the user because of the effort to get systems working and working effectively. Either way it will be a cost to the user. For this reason, no software is free.

C. Control

One of the reasons for the call for "open" systems is so that users can feel in control of the system. Control means that regardless of the unexpected requirements that arise during the life of the system, the system can be adapted and grown to meet those requirements. This is, of course, vital in research.

D. Documentation

No software is complete until the documentation is written. Experience here has been that this job is often not started until the need is more than pressing. For software to be shared and successfully used at another site, good documentation is required.

VI. STAGES OF SOFTWARE SHARING

Using software engineering techniques, the results of any stage of control system implementation can be shared in order to reduce effort and improve quality. Clearly, the more stages that are shared, the more cost and risk are reduced. These stages are

1. The concepts of the system and the understanding of the problem
2. The analysis of the problem
3. The design of the solution
4. The implementation of the solution, the basic system without the specific application
5. The complete implementation including the application

The ability to share stage one and stage two are only constrained by the type of accelerator and its operation requirements (such as the need for super–cycles).

Sharing the design of the solution will require accepting some constraining technical choices, such as networks, computer and operating system and so on. However, this is not such a strong constraint at the design phase as it is in the last two phases when actual executable code is shared.

Depending on the software engineering tools used, non–executable code can be shared in the analysis and design phases.

VII. METHODS AND EXPERIENCES IN SHARING COMPLETE ACCELERATOR CONTROL SYSTEM SOFTWARE

To date there have been three methods of sharing software. These are "As–is," Collaboration and Commercial. Each is described below, with some of the advantages and disadvantages listed. Table 1 lists some of the known software sharing experiences in the accelerator control system field.

Table 1: Software Sharing

Originator	First Distribution Date	Receiving Institutes	Method
CERN/SPS	1970	DESY,KEK	"As–Is"
HMI [ref. 4]	1978	CRL, KFA	"As–Is"
FNAL[ref. 5]	1984	NSCC, Loma Linda	"As–Is"
HMI [ref. 4]	1985	CRL*	Collaboration
LANL/PSR	1985	KFA	"As–Is"
SLAC [ref. 6]	1986	BEPC, IHEP* Duke U., Oxford Instruments	"As–Is"
LANL/TCS	1987	TRIUMF, BNL, CRPP, CERN/LEAR, GSI, PSI, HMI, GA	"As–Is"
CEBAF	1989	Bates, SSC, LLNL, etc	"As–Is"
VCS	1990	Various	Commercial
LANL [ref. 7]	1990	ANL/APS, Duke U., LBL, SSC	Collaboration

**Application Programs Also Used*

Also included for comparison is the case of an institute developing their own system, the "roll–your–own" method. The list is ordered in decreasing cost, development time and risk. Where costs and effort are mentioned, they are for the basic system to the point that it is being implemented for a particular project and they do not include any application effort.

A. Roll–Your–Own Method

Here the institute develops their own system using basic computing tools.

Advantages

1. Complete control of the software function.
2. Free choice of computers, displays and I/O system.

Disadvantages

1. Considerable initial development cost, $2.2–20M.*
2. Considerable risk.
3. Highest support and maintenance cost, 4–20 people, $300K–$4M/yr.*

* Programmers are assumed to cost between $75K/yr at salary plus overhead and $200K/yr. at salary, overhead and burden costs.

B. "As–Is" Method

Here the complete software, including sources, and any documentation is provided from another institute on an "as–is" basis. Recipients then have to develop and support the software on their own, thus the software provides a substantial initial start to a project. Of course, the recipient can call for either free or paid help from the source institute of the software and, in practice, this has often been given. However, the people who wrote the software initially usually have to respect the schedules and demands of their home institute first; therefore, the external requests usually create added pressure with little recognition or reward.

Advantages

1. Significant design and implementation work are saved. This is probably valued at about 60% or more of the "roll–your–own" cost.
2. Experience from another project is used initially.
3. Receiving institute has full control of further development.
4. Considerable risk reduction, the amount depending on the further development required.

Disadvantages

1. Local continuing support and development costs incurred. This can easily add 4–20 people to the staffing requirements at a cost of $300K–$4M/yr*
2. Local variations of the "as–is" software are usually developed, inhibiting further sharing between the institutes.
3. Software developed for in–house use is not usually engineered to be easily installed by other sites. Thus, there will be a steep and costly learning curve.
4. Distribution of the software adds a load on the writers of the software in the form of preparing and making distributions and answering support requests. This is not often offset by recognition or reward from their home institute.
5. Restricted, if any, choice of computers and I/O system.

C. Collaborations

In this form of software sharing, one institute takes the lead and provides the initial software that is then further developed by a group of institutes under some form of common management. The key requirement of a collaboration is that the control system software remains one system with no local variants. If local variants develop then the relationship is likely to move from a collaboration to an "as–is" relationship.

Advantages

1. Sharing of development costs.
2. Broader experience feeds into the requirements for new releases.
3. Considerable risk reduction.

Disadvantages

1. Cost of local support and development team, 3–8 people at a cost of between $225K–$1.6M/yr.*
2. Complexity and expense of management between different institutes to keep the software common. Frankly, this is an achievement in a single group or institute! This management issue also results in a loss of local control for development decisions and relies on goodwill between the members.
3. Restricted, if any, choice of computers and I/O system.

D. Commercial Systems Developed for Accelerator Controls

In this software sharing model, a company, through normal commercial arrangements, effectively becomes the control system software group for the customers. Experiences with institutes using commercial control systems developed initially for industrial applications has been poor because of the additional requirements of physics research applications that are uncommon in industry. My company is the only example that has started with the physics market for developing, selling and supporting a control system toolbox. As far as I am aware, no other company has more than a single control system sale active in this market.

Advantages

1. Minimal support and development costs, a fraction of a person locally and $4.5–50K/yr support and maintenance charges to the supplier, total, 23–150K/yr.*
2. Considerable risk reduction that is essentially complete if the products meet the requirements as demonstrated before the sale.
3. Support is available and of good quality because of broad support experience of the company personnel and the direct reward to the company and the employees of the company for good product and support.
4. The company normally controls the key sources, ensuring compatibility and the ability for customers to share code between themselves.
5. Company can and is motivated to provide application help at critical times to customers.
6. Product is engineered for distribution and installation and documentation is provided.

Disadvantages

1. Initial license cost, in the range of $30K to $350K.
2. Control depends on the documented "openness" of the product.
3. Issues of the company failing have to be addressed.
4. Restricted choice of computers and I/O system.

Figure 1 illustrates the minimum and maximum costs likely to be incurred by each method over a ten year period.

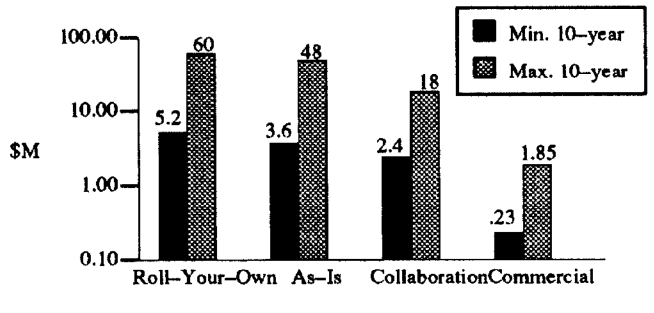

Figure 1: 10–Year Control System Software Costs

It should be noted that the commercial solution has an order–of–magnitude cost advantage. This is because of the efficiencies of commercial operations and the economies of scale, as well as engineered, tested and documented software.

VIII. SHARING SOFTWARE COMPONENTS OF A SYSTEM

For this to be successful, the interfaces as listed above have to be the same or the differences must be manageable. If the data bus is common, the problem is almost completely solved and some agreement here would greatly facilitate software sharing. The other interfaces will be as influenced by the market forces as by our community.

Experiences here have been to successfully use some of the physics beam analysis codes within a control system and to use commercial products for a part of a control system. Past results here have been mixed, usually because the commercial products chosen were developed for small industrial applications.

IX. PLANNING FOR THE FUTURE

The accelerator controls community has a choice. It can let the commercial products develop and use them as it can or it can be proactive in influencing the commercial developments. In parallel with the first option of commercial laissez–faire, the community can continue to develop its own systems and share them as before. The basic problem with this approach is that it will greatly slow the development of commercial solutions that are focused on the class of applications represented by accelerator control systems. There are two ways to influence commercial developments. One is for the community to be a significant customer of one or more software companies and the second is for the community to develop broadly applicable standards and to purchase products based on those standards.

The developments one might look for in a commercial package that is specifically designed for accelerator control are interface packages for incorporating some of the standard physics codes, control programs for accelerator specific tasks and so on.

X. SUMMARY

Methods of sharing software, either in analysis and design stages or in the complete system, have been defined with the advantages and disadvantages explored. Is there one right solution? Currently, I think that the answer is no. It depends on the number and skills of the programmers available to the project. If resources are scarce, then a commercial solution is the only solution. If ample resources are available and the institute wants complete control and will accept the risks, then an "as–is" solution or "roll–your–own" solution is indicated. Between these two extremes sits a collaboration such as the EPICS collaboration.

The important aspect of the choice is to understand that a choice is being made and there are advantages and disadvantages to each choice. Equally important is to defer making any component or personnel decisions until the overall strategy is decided. If one starts by hiring systems programmers, one has already eliminated some choices. Equally, computer, operating system and I/O subsystem decisions will restrict the choices for the most expensive and risk–prone component of the control system, the software.

XI. ACKNOWLEDGEMENTS

I would like to acknowledge the review and constructive criticism of the draft of this paper by Peter Lucas of Fermilab, and Axel Daneels and Berend Kuiper of CERN.

XII. REFERENCES

[1] A. Daneels, "Current Trends in Accelerator Controls: The Issue of Accelerator Software, Particle Accelerators, 1990," *Proc XIV Intl. Conf. on High Energy Accelerators*, Vol. 29, p. 875, Tsukuba, Japan, Gordon and Breach Publ.

[2] B. Kuiper, Private Communication

[3] B. Kuiper, "Issues in Accelerator Controls." *Proc XIV Intl. Conf. on High Energy Accelerators*, Vol. 29, pp. 602–611, Tsukuba, Japan, Gordon and Breach Publ.

[4] Winfried Busse, Private Communication

[5] Peter Lucas, Private Communication

[6] Sam Howry, Private Communication

[7] L.R. Dalesio, M.R. Kraimer, and A.J. Kozubal, "EPICS Architecture," *Proc. Intl. Conf. on Accl. and Expt. Phys. Contl. Sys.*, Nov. 1991, KEK, Tsukuba, Japan, KEK Proceedings 92–15