

Evergreen in Context

by Robert E. Molyneux

Open Source Software in Libraries

The plan to develop Evergreen, the open-source consortial library software, was announced in June 2004 by Lamar Veatch, state librarian and head of the Georgia Public Library Service (GPLS). Evergreen went live in 2006. That two-year period was filled with activity.

It was a major decision by GPLS to develop new software of a scope never before attempted, but it was undertaken only after it had exhausted all other alternatives to run PINES, the resource-sharing public library consortium in Georgia. Evergreen was to be designed from the ground up as a new thing: a consortial library system, that is, an integrated library system for a large, geographically dispersed, resource-sharing consortium. Why did GPLS take this path and how did it go about doing it?

A Bit of History

The decision came about after a long chain of events made over many years in Georgia as it, like all states, wrestled with how best to provide public library services to its citizens. Policy decisions over many years followed a consistent direction.

Local library organizational patterns commonly follow a model of treating each library separately resulting in what today are commonly called information silos, resulting in largely separate collections with weak communications among them. Resource sharing among the libraries in most state systems is through an interlibrary loan (ILL) process.

Georgia, on the other hand, has consistently chosen another direction:

Bob Molyneux is vice president of business development with Equinox Software, Inc. He can be reached by phone: 877- 673-6457, ext. 710 or 678-269-6112; by email at bob@esilibrary.com; or by Skype: ESI-Bob. The company website is at www.esilibrary.com

joining the many small public libraries into larger resource-sharing regional libraries. Other states have worked on the idea, abandoned it, taken it up again and so on. It was an idea that the library community liked in theory, but before recent developments in computer technology, Georgia's path was unusual.

The fundamental problem that all states must wrestle with in providing public library service – indeed for all entities providing any library service – is the great disparity of size, wealth and resources of the many libraries that exist. There are a few very rich libraries; however, most have only modest resources.

Consider these figures from the latest national-level data we have for U.S. public libraries (FY 2005). Table 1 shows that the top quartile of these libraries by their total expenditures – that is, the 25% of public libraries that spent the most – spent \$7.9 billion that year while the bottom three quartiles (the 75% that spent the least) spent \$1 billion in total. Indeed, to press the point, the top 10% spent \$6.5 billion. Think about that: Ten percent of the public libraries in the United States spent about 6.5 times as much money as the smallest 75% of the libraries.

TABLE 1. Total expenditures by U.S. public libraries in FY 2005

	\$ Millions	Percent
Top Quartile: (Largest 25% of Total Expenditures)	7.9	89
Bottom Three Quartiles: (Smallest 75% of Total Expenditures)	1.0	11

The big libraries are very big, and the small libraries are very small, and the disparities in expenditures that these figures demonstrate are consistent across other variables such as collections, resources and staff. Moreover, they are consistent with the pattern for other types of libraries. However, our story is centered on public libraries in the United States because of decisions made by GPLS.

It is useful to note that the two largest open source library systems running in the United States, Evergreen and Koha, were developed for small public library settings. If you look at the expenditure figures above, it is not hard to imagine why – the large proprietary vendors prefer to concentrate their efforts where the money is. Within the open source framework, however, Evergreen and Koha have different approaches, with Evergreen being consortial for reasons that are discussed here.

The disparities in resources outlined by the data present a profound information policy question for a consensual democracy: In a system of government that relies on an informed citizenry, are such disparities acceptable? Is it acceptable for the “haves” to have access to so much more information than the “have nots”? If not, how do we ameliorate the results of this disparity?

Public library service as it is configured in the United States today developed before Google and other modern software developments that have relentlessly broken down information silos in other fields. Their configuration cannot adequately provide the vast pools of information and materials that are now available to the citizenry. One step in the right direction is consortial resource sharing, and another is to use these consortia to pool money to buy access to enhanced content for all the libraries in the consortium – especially those that could not otherwise have afforded this content – the “everything everywhere” library consortium.

In its history (at least by the 1940s), Georgia decided to follow its subsequently consistent policy – with the normal stops and starts involved with politics – to bind the small public libraries in the state into larger, regional entities for resource sharing. That strategy was Georgia’s way to get library resources to the “have nots.” The state provided both funding and direction toward that end. Although Georgia’s largest libraries are typically independent, the smaller ones today are mostly in regional systems.

PINES

In 1998, David Singleton, then at GPLS and now at Charlotte-Mecklenburg Public Library in Charlotte, North Carolina, suggested a universal borrower card for Georgia’s public library users. Such a card

would allow all citizens of Georgia to borrow an item from any library. Given both the amount of state funding and state strategic practices to group libraries into regions, such an idea was possible. Of course, the idea was not new to Georgia, but it fell on receptive Georgia soil since the infrastructure and the general acceptance of the idea of sharing resources were in place. Because of years of preparation, state guidance and vision, the idea would work.

TABLE 2. PINES timeline

1999	PINES goes live with 26 library systems and 96 outlets
June 2004	Work begins on Evergreen
September 2006	Evergreen goes live over the Labor Day weekend. 44 Georgia public libraries with 252 outlets are now running an open source consortial library system.
September 2008	On Evergreen’s second birthday, it is running in 62 public library systems with 294 outlets in five U.S. states and one Canadian province. It is also running on its first academic library. PINES has 50 systems with 275 outlets and regularly circulates 100,000 a day and will reach about 16 million circulations for the year. There are nearly 10 million items in the PINES online catalog.

Y2K – the computer problem that existed because of the way computers handled dates – in the end was what led to the creation of PINES (the Georgia public library resource sharing network – the network of Georgia’s regional library systems). Y2K money was allocated by the state, and an RFP for a statewide system was issued. One vendor’s product stood out, and PINES was born in 1999 with Phase 1 linking 26 library systems with 90 outlets (central libraries, branches and bookmobiles).

PINES was an immediate success. PINES users liked being able to borrow books from a larger set of libraries with a richer set of collections than one local library. They liked access to the longer tail, and PINES added more libraries and more users. Library users altered their behavior, and many bypassed their local libraries to have access to the larger, virtual collection. PINES grew until it started having problems with the underlying software that, famously, resulted in the system’s production servers having to be rebooted during the middle of the day in order to be able to handle control of the transactions load. The software had reached a previously unknown limit.

Limits

Given that the software that ran PINES is running other systems that have higher circulations and larger bibliographic databases than PINES, what limit had been reached?

There were several, but the central one was the system could not keep up with the update load. The software was not designed for a consortium spread over a large area with updates such as checkouts, check-ins and cataloging changes to the central database coming from so many different places. In such a system, there are many people logged into the system changing the database at any one time, and the database, to be useful, has to be current. It should reflect what is checked in or out (transactions) and what the system owns (inventory) so that users searching will know if an item is available and librarians will know the state of items in the system. These functions are basic to a modern inventory control system.

In PINES, there are thousands of terminals across the state logged into the database making changes, and each change has to be reflected in the database by “reindexing” it. While rebooting in this old system allowed the system to catch up with the changes, it could not be done frequently enough. The limit reached was a hard one, and there was no solution using that software. Meanwhile, more libraries wanted to join PINES. All the while, the circulation load was large and increasing.

Design Architecture of Evergreen

Any large piece of software has a design architecture: how it organizes its elements, what tools it uses, that is, how it does what it does. The design architecture, in turn, is a function of various factors, the two most important being system architecture and computer costs.

Many old versions of old software still litter the ILS ecosystem. The software then running PINES began development in the era when mainframes ruled the earth. This legacy software has been adapted and changed and added to over the years. Elements had been bolted on and kludged, but, it is a fact that updating deployed software has proven to be an almost impossible task almost every time it has been tried. There are a thousand problems, and what most vendors of library software have done is start over using the more modern updated practices of the time.

When the original PINES software was developed, computers were immensely expensive and software comparatively primitive. Better practices and more capable small computers existed when Evergreen was envisioned, and the developers used more modern software development processes to run on more capable computers that no longer cost a million dollars each to buy. An examination of alternatives in the market found none that had the capabilities necessary to run PINES then or in the future. What was Georgia to do? Limit PINES? Give up and abandon a policy direction it had followed so consistently and that had produced such benefits? Over software?

After much discussion, GPLS embarked on a revolutionary path – an open source, consortial library system to run PINES.

Development Begins

Work began on Evergreen mid-2004. The later a software system is developed, the more modern practices can be used in that development. Evergreen, for instance, was web-aware and used Unicode for text storage and representation from the beginning. To implement these capabilities, Evergreen did not require the costly and difficult rewrites to the foundational code necessary for ILSs that had their basic code written before Unicode had been developed. Unicode is important because it allows a system to handle non-Roman scripts, including characters in Chinese and Arabic.

Technology – The Underlying Software Architecture

Evergreen’s development began with practices then in place in the business world. These practices today result in software that is modular, scalable and relatively easy to update – at least compared to legacy software that is even as little as five years old.

Evergreen uses a service-oriented architecture (SOA) with representational state transfer (REST) and “n-tier” architectural design concepts. Let us explore these concepts a bit.

The development philosophy behind these elements is that software is based on distributed services that communicate and collaborate through a set of standards, that is, it should be “service oriented.” In Evergreen, the open scalable request framework (OpenSRF, pronounced “open surf”) is such an SOA framework.

OpenSRF is, in many ways, the key to Evergreen because it does so many things. OpenSRF provides load balancing and is robust and scalable. It also allows the development of software without requiring a detailed knowledge of Evergreen's structure. In effect, the Evergreen layer consists of a number of applications each of which rides on top of OpenSRF. The result is that the developer need only know the APIs and what information the program elements require and what they will reply to write new capabilities into Evergreen. OpenSRF handles the details of implementing a stateful, decentralized service architecture while the Evergreen code supplies the interface and business logic. OpenSRF is now a separate open source project.

Networking follows a design philosophy that is often introduced to students with the OSI Reference Model. The model describes seven layers that pass information among each other to provide network services in a manner similar to what one finds with SOA. "N-tier" application architecture is a related idea where the tiers communicate and collaborate with each other by established standards. We see these operations in our daily lives. If, for example, a new version of Firefox comes out, we can change the old version for the new one without changing the operating system or reformatting our hard drives. We usually do not even have to reboot. Why? Because of the underlying architecture where each program communicates with the other parts of the network software by established rules – application programming interfaces (APIs) – that specify how the various elements involved exchange needed information. So, to change a part or one tier does not require a massive rewrite of the entire code, just what needs to be adjusted. As mentioned, the Evergreen middle layer rides on top of OpenSRF, and the user interfaces (the OPAC and staff client) ride on top of Evergreen.

In short, REST is a set of principles that establishes an architectural style for large, scalable applications.

Evergreen also uses other open source projects – there are no proprietary pieces in Evergreen – to supply needed functionality. One advantage of open source applications is that a developer working on one project can use what is already tested and proven by other projects. PostgreSQL ("Postgres") is an enterprise-grade relational database system with over 15 years of development.

It was chosen for Evergreen because it was virtually the only open source database system that had the capabilities to support a system with the database structure and the transactions load the size of PINES. Another key application is XUL (pronounced "zool") which stands for XML user interface language. Developed by the Mozilla project, XUL allows fully featured cross-platform web applications. Perhaps the best known XUL-driven application is the Firefox web browser.

There are, as you see, a number of pieces to the Evergreen puzzle. These modules work together, and each can be changed when necessary without affecting the others as long as the changes do not affect the communications among them. Using these practices, coupled with the open source framework, provided a flexible method for very rapid development of Evergreen in the early days and continues to this day.

The Interface

Focus groups were formed to ask library staff and patrons what features they would like to see and how they would like the OPAC to perform. How well they did that job is indicated by the increased usage that followed the introduction of Evergreen. For instance, year-over-year, holds increased by 30% – same libraries, same network, but a new interface that users found easier to use, and they did.

Development Process

The general process for development was "release early, release often," a process followed to this day. Small, iterative changes were made and released for comment and testing – not rare mega updates. If there were a problem, the software could be fixed relatively quickly. In addition, the software was open source so that it was developed in the open, not secretly by a small group. Many eyes looking at software makes better code.

Another aspect of Evergreen's development is "scratching an itch." That is, development tends to focus on the problems that people want fixed, not necessarily theoretical problems or capabilities dreamt up by product managers in response to real or imagined competitive pressures. Any kind of software development will have its strengths and weaknesses. Scratching

MOLYNEUX, continued

an itch will solve problems people really have, but the community does need a mechanism to guide development strategically to make sure that while solving this or that problem the software stays a coherent whole.

Development Philosophy

As a result of what was learned about the first iteration of PINES, there were several key results expected from Evergreen. The original charge was to build something “scalable, robust and fault-tolerant.” And, of course, open source. What do those three new terms mean?

Scalable means that the software can be deployed in different-sized libraries and that adding capacity is relatively easy. At the point where the decision was being made to move from the proprietary system that ran PINES, the server upgrade for the old system would have cost \$1.5 million for a large server. Evergreen development cost the GPLS much less than that.

Learning that lesson, Evergreen can be upgraded by adding “commodity” servers – that is, servers from any manufacturer or any do-it-yourself builder, and they can be added as needed. OpenSRF, clearly, allows Evergreen to scale easily.

Because Evergreen is easy to use – remember the focus groups – as a library’s users learn the software, they use it more, and growth can be done by adding relatively cheap, redundant servers, not million dollar boxes running proprietary software. Add a server or two, load OpenSRF, and you have upgraded. Evergreen runs on laptops, is currently running on a home library, and runs PINES with 16 million circulation transactions.

Robust. Evergreen was designed to be able to keep functioning in dire circumstances. When a backhoe severed the PINES network, the Evergreen staff clients allowed libraries that were disconnected from the central databases to continue checking out books. When the network was restored, the database was updated by an established procedure. There was no need to

write down transactions on paper. Consider: PINES has had a number of days recently with 100,000 circulations. If half the network were down for a half a day, that could be 25,000 sheets of paper to be managed. Not with Evergreen.

Fault-tolerant. Servers can fail, and the system will keep on functioning. The use of redundant servers allows one to fail and the others to keep on functioning. Evergreen was designed as much as possible to never go down.

The Future

Reflecting its history, Evergreen does not have acquisitions nor serials modules. Work on these two services is ongoing with releases expected by Spring 2009. It is a development project, but one coming from the library world to solve library problems.

As was observed above, it has been quite difficult in the past to update software to reflect new demands on it. It did prove relatively easy to add web awareness to most existing library software but rather more difficult to add Web 2.0 capabilities and Unicode. Consortial capabilities have proved daunting. And as a result of Evergreen such capabilities are now touted by software in the market that may soon have those capabilities, too.

How does Evergreen avoid these kinds of pitfalls of obsolescence? Maybe it doesn’t but Evergreen’s practices reflect the current best guess about how to engineer software so updating it is never as painful as it has been in the past with code bases even a few years old.

Welcome to the future, version 1. ■

Resources Mentioned in the Article

- [1] NCES/NCLIS. Enhanced Longitudinal Public Library Data File (PLDF3). Retrieved October 20, 2008 from <http://68.163.78.28/statsurv/NCES/pldf3/index.html>