THE FERMILAB LATTICE INFORMATION REPOSITORY*

J.-F. Ostiguy, L. Michelotti, M. McCusker-Whiting, M. Kriss
FNAL, Batavia, IL 60540, USA

Abstract

Over the years, it has become increasingly obvious that a centralized lattice and machine information repository with the capability of keeping track of revision information could be of great value. This is especially true in the context of a large accelerator laboratory like Fermilab with six rings and sixteen beamlines operating in various modes and configurations, constantly subject to modifications, improvements and even major redesign. While there exist a handful of potentially suitable revision systems - both freely available and commercial - our experience has shown that expecting beam physicists to become fully conversant with complex revision system software used on an occasional basis is neither realistic nor practical. In this paper, we discuss technical aspects of the FNAL lattice repository, whose fully web-based interface hides the complexity of `Subversion`, a comprehensive open source revision system. The FNAL repository has been operational since September 2004; the unique architecture of `Subversion` has been a key ingredient of the technical success of its implementation.

INTRODUCTION

When the SSC project was starting in the early 1990s, relational database technology was coming of age. At the time, researchers at the SSC and other large accelerator facilities including Fermilab, considered using database technology to track lattice configuration and revisions. Early enthusiasm was quickly tempered by high costs and a generally inadequate basic software infrastructure. While it was generally agreed that keeping track and disseminating lattice information electronically would be beneficial, accelerator scientists could not justify the time and effort involved in dealing with cumbersome and cryptic software.

In the early 2000s, the Fermilab Collider Run II was facing technical difficulties. Many groups, both internal and external were interested in acquiring authoritative information about the optics of the Tevatron, the anti-proton source and associated transfer lines. Despite the best intentions, the process of getting in touch with the right people, getting hold of the right version and tracking last minute changes was time-consuming and potentially error-prone. The Internet revolution of recent years spawned a considerable amount of high quality software; this prompted us to revisit efforts to put together a system to electronically track lattice revisions.

* Work supported by the U.S. Department of Energy under contract No. DE-AC02-76CH03000.

OBJECTIVES

The high level and most important objective can be simply stated: the system should not require any specialized knowledge on the part of its users. More specifically our objectives were the following:

- Web-based Interface: a design based on a platform independent web-based interface makes lattice information accessible from any machine equipped with a browser.
- Authentication: Read access to lattice information should be anonymous, i.e. no user authentication required. In contrast, user authentication is necessary to control uploading of new lattice files.In that case, one should take maximum advantage of the existing authentication infrastructure rather than require people to create and forget yet another password.
- Historical information tracking: the lattice repository must be able to keep track of file history e.g. creation date, submittor, author etc. It must also be possible to retrieve any revision of a given file.
- Differences: lattice information files have characteristics that make them closer to source code than ordinary text documents. While lattice files are human-readable, they are ultimately meant to be read by computer programs. Subtle differences can be difficult to spot and for this reason, the ability to track and clearly display changes is essential.
- Tagging: in some cases, lattice description files are meant to be used as a group. It should be possible to assign a common tag or label to a number specific files and a mechanism should be provided to retrieve all the tagged files as a bundle.
- Integration with optics applications: a generic mechanism to allow programs (such as our optics program CHEF [3]) to retrieve files directly from the repository would be desirable.

VERSION CONTROL SYSTEMS

The above requirements have a large intersection with the features of a software version control system. Years ago, such systems were proprietary, platform specific and expensive. The advent of the Internet enabled software projects involving developers in different physical locations and drove development of new products. Among the most successful and well-known is CVS. Its popularity and dominance arose from many factors, but the most important ones were a pragmatic approach to the problem posed by multiple developers working concurrently on the same file and openness and availability of the source code.

In the CVS model, all files under control are stored in a centralized repository. Recently, a new paradigm for revision control software has been gaining momentum: distributed revision control systems. These systems (e.g. Monotone, DARCS and Arch) do away with a single centralized repository in favor of peer-to-peer architecture. Each developer keeps a repository, and the tools allow easy manipulation of changes between systems over the network. In contrast to software files in a large project, lattice information files are generally not heavily dependent on each other and do not undergo significant changes on a daily basis. The "developers" are typically concentrated in one or perhaps a few facilities.In that context, the overhead and the complexity associated with a decentralized system would be difficult to justify.

A determinant consideration for our requirements was that revision control integrate gracefully with web-based interfaces. In many systems, including CVS, operations on the repository must be performed either through a monolitic command line client or a crude network protocol; integration with a web interface typically implies parsing text output. This approach results in a brittle design subject to failure whenever the client produces unexpected output (due to unexpected input data for example) or due to changes output format modifications following a new release.

`Subversion`

In late 2003, `Subversion`, a new centralized revision control system was about to be released. To capitalize on the the popularity of CVS, it was designed as a highly compatible replacement for CVS, from the standpoint of the its high level user interface. The internal architecture of `Subversion` however, differs substantially from that of CVS. Among other things, the core functionality is implemented in libraries. These libraries are called by the standard client but can also be called directly from custom code written in scripting languages such as Perl and Python for which bindings are provided. By design, all output generated by the libraries is structured to be easy to handle by a program.

An interesting feature of `Subversion` is its supports for the WebDAV protocol. WebDAV, often referred to simply as DAV (Distributed Authoring and Versioning), is, in a nutshell, a set of extensions to the http protocol to enable distributed web authoring tools. WebDAV is supported via an extension module for the popular Apache httpd server.

Because of its design and high quality documentation, and despite its relative immaturity `Subversion` was selected as the foundation for our lattice repository infrastructure.

THE LATTICE REPOSITORY ARCHITECTURE

As previously mentioned, our most important objective was to allow users to upload and retrieve lattice information with a minimum of fuss and without an expectation of specialized knowledge on their part. To hide the complexity of `Subversion`, our approach was as follows:

The `Subversion` libraries, repository, and a web server (Apache) are all hosted on a common server. Interactions with the repository are done through custom high level scripts(mostly written in Perl at this point) which in turn call the `Subversion` libraries. The architecture is illustrated schematically in figure 1.

In a typical software development context, each user maintains a local copy of the file repository. Using a dedicated client program, local file copies are periodically committed and/or synchronized with those held in a master repository. For our lattice repository application, *all operations are performed on behalf of actual users by a unique proxy user*. The proxy user owns,and as needed creates and destroys temporary working copies of the repository on which the `Subversion` libraries operate. The obvious question is: what happens when more than one user attempt to perform operations simultaneously? Correctly dealing with this issue is actually at the heart of the surprising complexity of revision control systems.

`Subversion` addresses the issue in an elegant manner: repositories are created and modified by BerkeleyDB, an embedded, high performance transactional datastore. Berkeley datastore transactions are what in computer science parlance are referred to as Atomic, Consistent, Isolated, and Durable (ACID) transactions.

- Atomic: either all of the changes occur or none of them do. If for any reason a transaction cannot be completed, everything this transaction changed can be restored to the state it was in prior to the start of the transaction via a rollback operation.

- Consistent: Transactions always operate on a consistent view of the data and when they end always leave the data in a consistent state. Data may be said to be consistent as long as it conforms to a set of invariants.

- Isolated: a given transaction appears to be running all by itself on the database. The effects of concurrently running transactions are invisible to this transaction, and the effects of this transaction are invisible to others until the transaction is committed.

- Durable: Once a transaction is committed, its effects are guaranteed to persist even in the event of subsequent system failures. Until the transaction commits, changes are guaranteed not to persist in the face of a system failure, as crash recovery will rollback their effects.

While reliance on the Berkeley transactional datastore provides significant advantages, there are also some disadvantages. To operate reliably, embedded databases need to be very strict about correct file access privileges. Because the data are stored in binary format, special software is needed to restore the integrity of the database. In theory, this integrity should be preserved by construction. However, in real life all software has bugs, and hardware failures also may lead to unpredictable behavior.

Another important issue we had to face was the fact that in the `Subversion` paradigm, the concept of "version" refers to entire repositories and not to individual files. This view of the world is advantageous in the context of large software projects where there is significant coupling between all files needed to build a single executable. To circumvent this apparently serious limitation, `Subversion` supports "properties" which are basically user-defined file level attributes. While it should be clear that overuse of properties can degrade performance, they are used in the lattice repository to store file level version information as well as authorship and origin.

Authentication

At Fermilab, lattice information is the responsibility of machine departments. Each Department Head (or a delegate) has the responsibility of verifying and certifying files. To guarantee the origin of official files, user authentication is required. The Laboratory uses Kerberos as a centralized authentication mechanism and it was decided that we should take maximum advantage of the existing infrastructure. Unfortunately, we discovered that Kerberos-based web authentication while possible was, surprisingly, not very well supported. One major cause of this state of affair is that the UNIX and Windows notions of Kerberos are subtly different. Furthermore, Kerberos support in commercial browsers such as Internet Explorer remains poorly documented. While we have not completely abandoned the idea of pure Kerberos authentication, we opted for the next best thing: web (X.509) certificates. Using the KX.509 program (originally from the University of Michigan) any user with an existing Kerberos ticket can acquire a X.509 certificate. This certificate can in turn be transmitted by most web browsers to a web server and used to authorize restricted operations.

CONCLUSIONS

Work on the Lattice Repository started in early February 2004; it became operational in August 2004. A sample user interface screenshot is presented in figure 2. Files have been collected for all Fermilab machines and transfer lines. From a technical standpoint, we consider our experience a success. The fact that the necessary infrastructure could be put together in a relatively limited period of time is a testimony to the quality of the free software tools developed by the internet community.

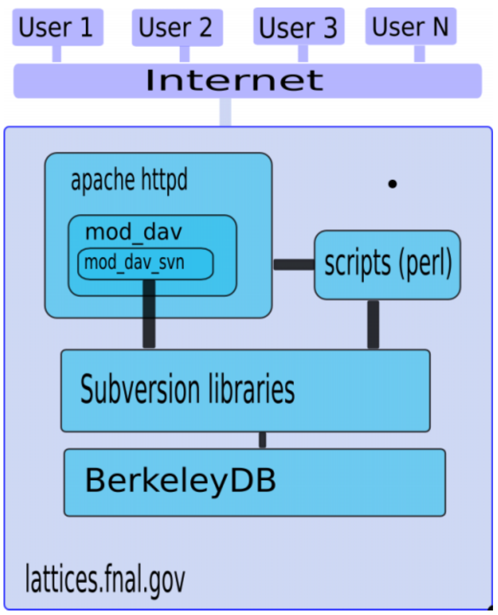
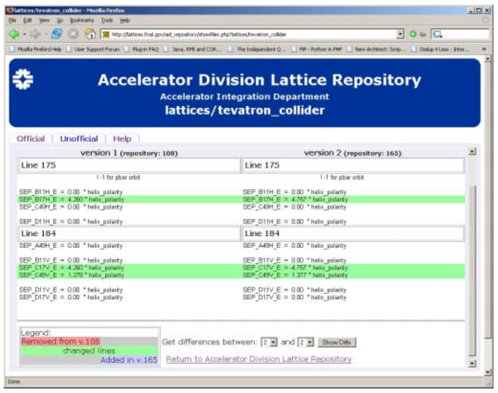


Figure 1: The Lattice Repository Architecture.



Accelerator Division Lattice Repository
Accelerator Integration Department
lattices/tevatron_collider

Official | Unofficial | Help

version 1 (repository: 108)	version 2 (repository: 165)
Line 175	Line 175
!-1 for pbar orbit	!-1 for pbar orbit
SEP_B11H_E := 0.00 * helix_polarity	SEP_B11H_E := 0.00 * helix_polarity
SEP_B17H_E := 4.260 * helix_polarity	SEP_B17H_E := 4.757 * helix_polarity
SEP_C49H_E := 0.00 * helix_polarity	SEP_C49H_E := 0.00 * helix_polarity
SEP_D11H_E := 0.00 * helix_polarity	SEP_D11H_E := 0.00 * helix_polarity
Line 184	Line 184
SEP_A49H_E := 0.00 * helix_polarity	SEP_A49H_E := 0.00 * helix_polarity
SEP_B11V_E := 0.00 * helix_polarity	SEP_B11V_E := 0.00 * helix_polarity
SEP_C17V_E := -4.260 * helix_polarity	SEP_C17V_E := -4.757 * helix_polarity
SEP_C49V_E := 1.278 * helix_polarity	SEP_C49V_E := 1.377 * helix_polarity
SEP_D11V_E := 0.00 * helix_polarity	SEP_D11V_E := 0.00 * helix_polarity
SEP_D17V_E := 0.00 * helix_polarity	SEP_D17V_E := 0.00 * helix_polarity

Legend:
Removed from v.108
changed lines
Added in v.165

Get differences between: 2 and 2 Show Diffs

Return to Accelerator Division Lattice Repository

Figure 2: Screenshot: Comparing two lattice file revisions.

REFERENCES

[1] B. Collins-Sussman, B.W. Fitzpatrick and C.M. Pilato, "Version Control with `Subversion`" O'Reilly Media, June 2004, `http://svnbook.red-bean.com`

[2] BerkeleyDB is a product of SleepyCat Software Inc. `http://www.sleepycat.com/docs/index.html`

[3] L. Michelotti, J.-F. Ostiguy, "CHEF: An Interactive Program for Accelerator Optics", PAC2005 (this conference).