Lattice Parameters Database and Operational Simulation at FNAL and SSCL

Eric Barr, Department of Computer Science, U.C. Berkeley, Berkeley, California,
Steve Peggs, Leo Michelotti, Al Russell, Selcuk Saritepe,
Fermi National Accelerator Laboratory, P.O. Box 500, Batavia, Illinois 60510,
C. G. Trahern, J. Zhou, Superconducting Super Collider Laboratory, 2550 Beckleymeade Ave., Dallas, Texas 75237*

Abstract

A collaboration has been formed at FNAL and SSCL to develop software tools for operational simulation of existing and planned accelerator systems. The toolset is organized around an accelerator lattice description maintained by a commercial database management system. This note presents a brief description of the tools presently available for public use.

I. INTRODUCTION

It is essential for the operational simulation of accelerator systems to maintain the most accurate representation of the accelerator lattice possible. This representation should also be in a form which allows the user community ease of access. Whether the user wishes to understand general problems of accelerator operation, investigate various correction schemes for error compensation or study the geometrical and mechanical layout of the accelerator, a centralized resource of lattice designs that is available over computer networks can guarantee that everyone will use the same lattice description.

The Superconducting Super Collider will be the largest and most complex accelerator system to be constructed, and it is a formidable task to describe and understand both this system and the other accelerators in the injector complex realistically. For this reason an effort to create a structure for lattice descriptions which would allow ease of access in a controlled manner was undertaken at the SSC Central Design Group. A relational database management system (RDBMS) was judged to be the appropriate technology in which to implement this strategy. The announcement of this system was first described in [1]. Since that time, this system has been successfully used at both SSCL and FNAL. In addition further development of that system has occurred which we believe can be usefully shared with the rest of the accelerator community.

II. LAMBDA

LAMBDA stands for "Loosely Associated Modules for Beamline Design and Analysis". It is a collaborative effort between FNAL and SSCL to provide lattice tools, in the form of independent code modules, in an environment centered around a RDBMS. LAMBDA provides

i) a set of data structures, database interfaces, and application software modules,

ii) detailed documentation describing the package.

The most current public versions of supported LAMBDA software are identically available on discs at the SSC and at FNAL. Version 1.0 of the documentation is nearing completion, with an expected public release date of July 1991. It will be available via surface mail by contacting either Steve Peggs (peggs@calvin.fnal.gov) or Garry Trahern (trahern@pear.ssc.gov).

Philosophies

A major problem with most conventional beamline design and analysis codes stems from the fact that the different "tools" or "functionalities" correspond to subroutines or functions in a single, large, multi-purpose code. Despite the best intentions of the author(s), it is inevitably difficult for a user to understand or develop tools in such an over-centralized environment. Breaking these packages down into several independent code modules provides for efficient maintenance of existing tools, and encourages the development of new tools. This emphasis on "Loosely Associated Modules" is roughly in accordance with the UNIX philosophy of single purpose modules for single purpose tasks.

It is only possible to write short, independent, modules if the author does not have to worry about beamline data input. LAMBDA modules are linked to the flatin lattice input module, and assume that FLAT data structures are filled and available to them. This is the glue that "loosely" holds the modules together. The flatin module is relatively short, compared to the thousands of lines of input code often found in large programs, because the input format is very simple. This is possible because the user - who needs to manipulate complex hierarchical lattice objects - edits the lattice in relational database tables, not in the beamline file. The FLAT input file is generated, in turn, by the dbsf code which reads database tables and generates ASCII files in any one of four optional formats (see section III B).

New contributors are welcome to join the collaboration, so long as they agree to maintain and document their modules. Although modules remain the intellectual property of the independent author, a reasonable level of sensitivity to the opinions of the collaboration and other users is expected. Authors are strongly encouraged to use C or C++, but other standard languages such as Fortran 77 are acceptable. Although the RDBMS in use at FNAL and SSCL is SYBASE, the proprietary dependence on this par-

*Operated by the Universities Research Association, Inc., for the U.S. Department of Energy under Contract Nos. DE-AC02-76CH03000 and DE-AC02-89ER40486

ticular commercial product has been kept to a minimum. Only one routine in dbsf uses SYBASE specific statements. It is planned to replace these references with Standard Query Language (SQL) statements, with the next release of SYBASE. When this is done, LAMBDA will perform with any RDBMS which supports embedded SQL calls.

If no RDBMS is available to a potential LAMBDA user, the user must manipulate beamlines by editing FLAT ASCII files, or by finding another way to fill FLAT structures. At present only ASCII files are used for communication between modules. Compatible binary options are planned for the two critical modules, dbsf and flatin, in a later LAMBDA release (see section IV A). The LAMBDA documentation is broken into three sections - Introduction and Overview, Database Interfaces, and Application Modules. Brief abstracts of the chapters in the second and third sections of the first release of the documentation are presented in the next two sections of this paper.

III. LAMBDA DATABASE INTERFACES

A. Lattice structures in the database

The thirty or so lattice databases currently in use at FNAL and SSCL, describing most of the accelerators in design and use at those labs, each consist of several tables of the same structure and design. These tables build up a (logically) hierarchical structure, from the "atomic" length and strength parameters, through magnet elements, to beam lines constructed from other beam lines. Optional tables allow aliasing from one set of element names to another or to define geometrical descriptions of the magnets as rectilinear boxes. Now in use for about four years, these data structures are very stable, but are still open to well deliberated modifications agreed upon by the collaboration.

B. dbsf - Data Base to Several (Lattice) Formats

Contemporary beamline design and analysis codes use a variety of ASCII formats. One format used by many codes (for example, MAD, TEAPOT, COMFORT, and DIMAD) is often referred to as "standard format"[2]. The code dbsf, which has been in use for about three years, optionally produces ASCII output in STANDARD, FLAT, MAGIC, or SYNCH formats. As such, it provides a connection to existing codes which are outside the LAMBDA package. Planned developments for later LAMBDA releases of dbsf include the use of multiple strength tables to represent dynamic optical tuning. dbsf is the only LAMBDA code that is SYBASE specific.

C. SQL procedures

About twenty Standard Query Language procedures have been written by various individuals to aid in the day-to-day manipulations of lattice databases. They perform single tasks, and are usually only a few lines long. Procedures exist, for example, to create the (empty) data base tables, to set up recommended permissions and protections, to copy between tables and ASCII files, and to build an index table of lattice objects found in the other tables. The work of many authors is collected and present in a single place.

IV. LAMBDA APPLICATION MODULES

A. FLAT structures and flatin

The FLAT structures that the flatin module produces from dbsf FLAT ASCII output have evolved, in the last three years, to a well designed and stable state. They are available to all of the application modules, and form the link between the database world and the regular computing environment. Despite the stability of FLAT structures, their crucial role as the glue that holds LAMBDA together means that they necessarily remain open to minor modifications requested by a consensus of the collaboration.

The incorporation of the FLAT structures into a binary data discipline known as the Self Describing Standard (SDS) should be available shortly. The SDS dataset is created by modules that form a part of the Integrated Scientific Tool Kit (ISTK)[3] and provides a sophisticated implementation for binary datasets in an architecture insensitive manner. The SDS dataset can then be used by other application programs which can interpret the data standard.

For example, a graphical interface to the FLAT structure is being planned which uses the SDS implementation. This CASE tool will allow the user to browse the lattice structure while maintaining the hierarchical information of the relational database tables.

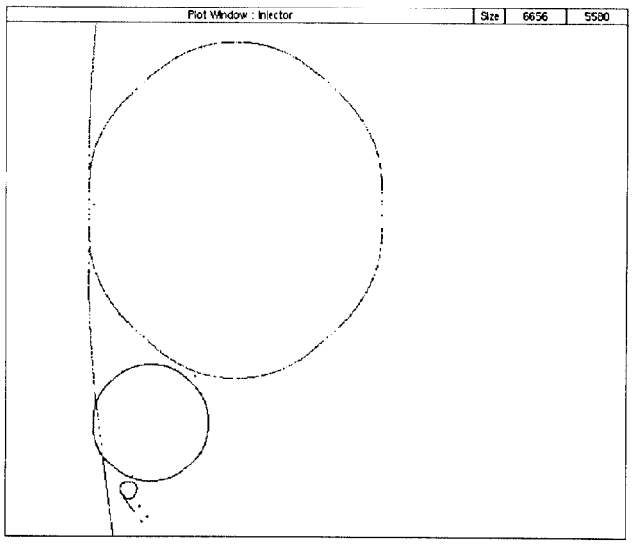

Figure 1: SSCL Injector complex as rendered by OZ.

B. survey

This module generates geometric survey information from a FLAT beamline. Coordinates are in meters or feet, with or without an initial displacement or rotation. survey also produces three projections of the beam line, in TOPDRAWER graphical format, including representations of

magnets as rectilinear boxes. The survey module has been in use in two major revisions for about three years.

At SSCL the information generated from the lattice database and the survey module has been incorporated into a graphical interface known as OZ. OZ is an object-oriented interactive graphical environment based on the X Window and ISTK tool sets. OZ provides a visualization of the entire SSCL accelerator complex in the site Cartesian coordinate system (Figure 1) and is available to any workstation on the network running X Windows. Although OZ stands outside the LAMBDA toolset as an SSCL specific code, it uses the FLAT data structures filled by flatin as well as those created by the survey module.

C. twiss

This module generates optical parameters - beta functions, dispersion, and phases - of a named beamline. Length units are meters. Output is in ASCII files as numerical tables, and as TOPDRAWER graphics (figure 2). The first version of this module has just been completed at the time of writing.

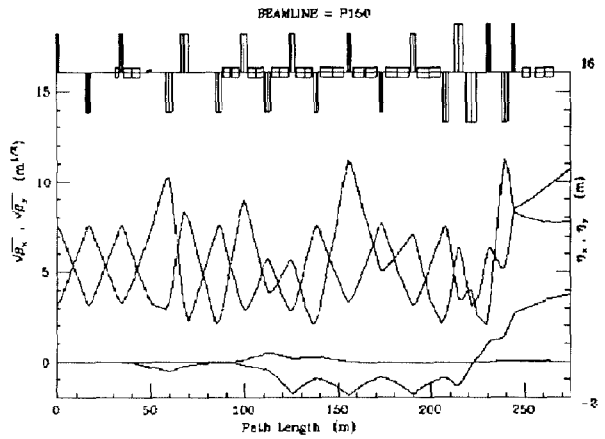

Figure 2: Example of Topdrawer output from twiss module.

D. MapWriter

MapWriter reads a FLAT format file produced by dbsf, constructs a polynomial map corresponding to a single pass in the represented beamline, and writes its coefficients, in ASCII, to a second file. Coded in C++, MapWriter uses the MXYZPTLK[4] and beamline objects described elsewhere in these proceedings[5] (see especially the discussions of (i) beamline::propagate functions and (ii) .physics files). With these, a first, prototype version of MapWriter was written - apart from preliminary boiler plate, such as #include lines and querying for options - consisting of two executable lines:

```
beamline A ( fileName[1] );
DAVector x;
  // Construct phase space map
A.propagate( x );
  // Write coefficients to a file
x.peekAt( fileName[2] );
```

At the time of writing, the library of beamline elements is being completed, and MXYZPTLK is being upgraded for greater functionality.

V. CONCLUSIONS

LAMBDA provides a flexible and structured environment in which to develop software tools for any lattice dependent calculations. The basic elements have been in use for nearly 4 years, and the tools have shown their ability both to economize the work necessary to maintain consistent lattice descriptions and to produce timely operational simulation results. We hope the accelerator community will give the LAMBDA toolset a wider distribution and begin to develop other codes within this structure.

References

[1] E. Barr, S. Peggs, and C. Saltmarsh, SSC-N-606, March 1989.

[2] D.C. Carey and F. Christopher Iselin, Proc. of the SSC Summer Study, p. 389, Snowmass, 1984.

[3] ISTK, unpublished internal documentation at SSCL, C. Saltmarsh, M. Allen and S. Acharya.

[4] L. Michelotti, "MXYZPTLK: A Practical, User Friendly C++ Implementation of Differential Algebra: User's Guide", FN-535, 1990.

[5] L. Michelotti, "C++ Objects for Beam Physics", IEEE-PAC these proceedings, 1991.