Particle Accelerators, 1996, Vol. 55, pp. [345–351] / 99–105
Reprints available directly from the publisher
Photocopying permitted by license only

THE CLASSIC PROJECT

F. CHRISTOPH ISELIN

CERN Division SL, 1211 Geneva 23, Switzerland

(Received 22 January 1996; in final form 22 January 1996)

The CLASSIC project was started with the aim of providing a uniform interface to accelerator codes, including input language, data structures, and interface to control systems. This should allow data and algorithms to be interchanged easily between programs. It should also allow the use of the same algorithms to talk to a simulation model or the real machine.

Keywords: Lattices; maps.

1 INTRODUCTION

The complete solution of accelerator physics problems usually requires several programs to be run. Exchange of data and algorithms among programs is still very difficult due to unnecessary differences in input format and internal data structure. To alleviate the problems the development of a C^{++} class library called CLASSIC (Class Library for Accelerator System Simulation and Control) has been started by John Irwin at SLAC. This library shall provide services for building portable accelerator structures and doing analysis on these structures.

Section 2 summarises the historical background which lead to this project, and Section 3 outlines its goals and anticipated time schedule.

2 HISTORY

2.1 Input Language

In order to solve all aspects of a problem in accelerator physics it is usually required to run several computer programs. Different programs use a wide variety of input formats to describe an accelerator lattice. Some well known examples are:

TRANSPORT:[1] In the original version both element definitions and commands occur in-line within a list of the structure. Element and command codes are numeric. Newer versions also understand mnemonic codes and allow more flexibility in command placement.

PETROS:[2] The fixed-format input first defines all different elements, and then lists their names in the order of their occurrence in the machine.

SYNCH:[3] Uses a fixed input format and mnemonic command codes. Commands and definitions can be intermixed freely.

Many other formats have been used, all of which have their own weakness and/or strength. This makes it difficult to exchange input data between programs, as the risk is important that the data used for different programs are not consistent.

These problem led to the definition of a format-free "standard input language" with mnemonic type codes for elements during a workshop at SLAC.[7] However, the standard language lacks definitions for command codes. The language has been implemented in many programs, e.g. in MARYLIE,[4] MAD,[5] TRANSPORT[1] and DIMAD.[6]

2.2 Machine Structure in Storage

Like the input language, the data structure describing the accelerator in computer memory differs widely from one program to the other. The first-generation of programs used a static structure based on fixed FORTRAN arrays (example: SYNCH[3]).

The second-generation of programs have implemented some more or less sophisticated memory management in FORTRAN (examples: MARYLIE,[4] MAD[5]). Very sophisticated systems for general memory management exist, like the CERN-written package called ZEBRA.[8] The use of this package in MAD reduced the memory overhead significantly, but made it difficult to a non-initiated programmer to add new features.

In third-generation of programs the trend goes to object-oriented techniques with true dynamic memory allocation. The most promising language to do this is C^{++}. The first attempt at such a program is BEAMLINE.[9] More recently, a new version of MAD, MAD^{++}, has been started at CERN.

The differences in internal formats make it impossible to exchange program code between different programs. Therefore, when switching from one program to the other, users have no choice except translating the data files, with the obvious risk of errors described in the previous subsection. A portable data structure for general use in optics programs would be of inestimable help.

2.3 Map Representation and Dynamic Analysis

Many different methods have been used to represent transfer maps and for analysis of optical behaviour of the machine. Most first-generation programs used "TRANSPORT maps", i.e. transfer matrices, possibly augmented with second-order, sometimes third-order terms (example: TRANSPORT[1]). The main problem with these maps is that truncation makes the maps non-symplectic. Some programs even ignored linear coupling completely.

Some second-generation and third-generation programs use Lie-algebraic maps (examples: MARYLIE,[4] MAD[5]). Coupling is fully handled in these programs, and map analysis uses normal forms in N dimensions. Other programs use Taylor maps (examples: TEAPOT,[10] TRACY/DESPOT,[11] Turchetti's program[12]). These programs expand the transfer maps into a Taylor series and apply normal form algorithms to the result. Treatment of single resonances is also feasible with both methods. However, more than 2 degrees of freedom are difficult to handle without Lie algebra. Both methods can be based on maps for finite-length elements (examples: MARYLIE,[4] MAD[5]), or on thin lens approximations (example: TEAPOT[10]).

Linear normal form analysis gives the linear lattice functions.[13] Their original form ignores coupling, and they are only applicable for nearly linear lattices. Later coupled lattice functions were introduced.[14] The transfer matrices are first decoupled by a similarity transformation, and the resulting block diagonal matrices are analysed. Again, this method is hardly applicable to strongly non-linear machines.

An uniform portable representation of maps would facilitate the transfer of maps from one program to another. One program might then be optimised for map generation and the other for map analysis. Whatever transfer map representation is chosen, it should allow a complete analysis of the motion. It should enable a large range of analysis methods to be applied. The most flexible representation seems to be based on polynomial ("Taylor") maps. Early approaches include the packages MXYZPTLK[16] and ZLIB.[17] A good polynomial algebra package will allow analysis by Lie algebra, by truncated power series, by generating functions, and by many other algorithms. Of course all the linear analysis methods are also available.

Some problems, like dispersion and some non-linear problems, could be tackled by synchrotron integrals. These are mainly useful for quasi-linear lattices with very weak coupling. It turns out that it is sometimes very cumbersome to evaluate such integrals correctly, especially for fringe field effects.[15] Note that synchrotron integrals are not readily available from transfer maps; but their values can be deduced in some cases via normal form analysis.

2.4 Interaction of Optics Programs and Control Systems

Often optics programs run in control computers so as to permit studies on machine parameter variation without disturbing machine operation. The machine parameters are fetched from the machine and/or from a data base, and the results may or may not be fed back to the control system. A flexible interface is required to connect the optics program with the machine. *Ad hoc* methods have been developed, (example: lm[18]), but a lot remains to be done for a portable interface.

A reasonably accepted interface seems to be CDEV.[19] This package defines a standard interface for control systems. If optics programs could interface to CDEV or to a similar system, they could be written in a portable way and could easily exchange data with the control system.

3 PLANS FOR CLASSIC

3.1 Goals

The following goals have been defined in a workshop, held in August 1995 at SLAC:

- Provide a C^{++} class library for accelerator design, simulation, and operation.
- Provide a mechanism for C^{++} code sharing in the accelerator community.
- Provide a platform to exchange new ideas in code development.

A long-term goal would be to allow interfacing with control systems. Optimised routines will be written to manipulate truncated power series.[20] The algorithms will also include methods for evaluating Poisson brackets and operations on truncate power series maps.

3.2 Possible Uses of the CLASSIC Library

Using the CLASSIC library, the data structure within a program can be built in at least three ways.

1. The program user can provide a program module which generates the accelerator structure by using a set C^{++} constructor calls.
2. The CLASSIC library is embedded in a complete program like MAD^{++}. The accelerator structure is generated from an input file read with a standard language parser.
3. The machine structure is extracted from a data base in the form of a C^{++} module.

The analysis part can be pre-written and linked with the data structure, or it may be provided by the user. For flexibility the class library should have the following features:

- Different language decoders can be plugged in easily.
- New algorithms for map generation and analysis can be added without problems.
- Integration methods can be replaced at execution time.

3.3 Organisation of the CLASSIC Project

The CLASSIC library provides an open development environment. The code is stored in a repository residing in the `cvs` repository

```
/afs/slac.stanford.edu/g/atsp/classic/prototype
```

at SLAC. Users having an `afs` account at SLAC can access it using the UNIX “cvs” program. A WWW home page is planned, and two mailing lists exist:

classic general A list of people who want to be informed about the CLASSIC project.

classic workers A list of potential contributors to the CLASSIC project. At the time of writing the following are known:

John Irwin, SLAC	Chairman
Scott Berg, SLAC	System manager
Yunhai Cai, SLAC	Technical coordinator
Tong Chen, SLAC	Beam-beam
Alex Dragt, Univ. Maryland	TPSA algorithms
James Holt, FNAL	GUI, correction schemes
Chris Iselin, CERN	Beam-line and Element classes
Roger Jones, SLAC	Matrix and Vector classes
Hamid Shoaee, CEBAF	Controls interface
Kathy Thompson, SLAC	Linac and wake-fields
Weishi Wan, Univ. Colorado	Integrators
Chip Watson, CEBAF	CDEV interface
Yiton Yan, SLAC	Map analysis

3.4 Time Table

A tentative structure was defined during the first CLASSIC workshop at SLAC in August 1995. This structure will probably be revised in a second workshop on CLASSIC in early 1996. Implementation has been postponed until after that workshop. The time estimate to implement a prototype is of the order of some months, once a suitable design has been decided.

References

[1] K.L. Brown, D.C. Carey, F.Ch. Iselin and F. Rothacker, *TRANSPORT, A Computer Program for Designing Charged Particle Beam Transport Systems.* Simultaneously published as CERN 80–4, FNAL-91, SLAC-91.
[2] H. Wiedemann, *User Guide for the Computer Code PETROS.* PEP-PTM-146, 1978.
[3] A.A. Garren *et al.*, *SYNCH, A Program for Design and Analysis of Synchrotrons and Beamlines: User's Guide.* SSCL-MAN 0030 rev, LBL 34668, BNL 49925, FNAL-PUB 94–013, 1994.
[4] D.R. Douglas and A. Dragt, *MARYLIE: The Maryland Lie Algebraic Transport and Tracking Code.* IEEE Trans. Nucl. Sci., 30 (1983) 2442–2444.
[5] H. Grote and F. Ch. Iselin, *The MAD Program (Methodical Accelerator Design), User's Reference Manual.* CERN/SL/90–13 (AP) (Rev. 4). Also available under the URL "`http://hpariel.cern.ch/fci/mad/mad.html`".
[6] R. Servranckx, *User's Guide to the Program DIMAD.* SLAC Report 270 UC-28, 1984.
[7] D.C. Carey and F.Ch. Iselin, *A Standard Input Language for Particle Beam and Accelerator Computer Programs.* CERN LEP-TH/84-10 (1984). Proc.

[8] R. Brun and J. Zoll, *ZEBRA Data Structure Management System.* CERN Program library code Q100.

[9] L. Micelotti, *C^{++} Objects for Beam Physics.* Proc. IEEE, 1991.

[10] L. Schachinger, *TEAPOT: A Thin Element Accelerator Program for Optics and Tracking.* SSC 52, 1985.

[11] H. Nishimura, *TRACY: A Tool for Accelerator Design and Analysis.* Proc. EPAC, Rome 1988, p. 803.

[12] A. Bazzani, E. Todesco, G. Turchetti and G. Servizi, *A Normal Form Approach to the Theory of Nonlinear Betatronic Motion.* CERN 94–2, 1994.

[13] E.D. Courant and H.S. Snyder, *Theory of the Alternating Gradient Synchrotron.* Annals of Physics, 3, 1–48, 1958.

[14] D.A. Edwards and L.C. Teng, Parameterisation of Linear Coupled Motion in Periodic Systems. IEEE Trans. on Nucl. Sci., 20, 885, 1973.
L. C. Teng *Concerning n-Dimensional Coupled Motion.* FN 229, FNAL, 1971.

[15] J. Jäger and D. Möhl, *Comparison of Methods to Evaluate the Chromaticity in LEAR.* CERN PS/DL/LEAR/ Note 81–7.

[16] L. Michelotti, *MXYZPTLK Version 3.1 User's Guide: A C^{++} library for Automatic Differentiation and Differential Algebra.* FERMILAB-FN-535 Revised, 1995.

[17] M. Malitsky, A. Reshetov and Y. Yan, *ZLIB++: Object-Oriented Numerical Library for Differential Algebra.* SSCL-659, 1994.

[18] F.Ch. Iselin, *The LEP Model Interface for MAD.* Proc. ICALEPCS, KEK, Tsukuba, Japan, 1991, p. 546.

[19] Documentation available via the URL "`http://www.cebaf.gov/cdev`".

[20] A.J. Dragt and M. Venturini, *Design of Optimal Truncated Power Series Algebra Routines.* Parts I and II, to be published.