CBN 98-9

Integration of Unified Accelerator Libraries with CESR.

Nikolay Malitsky and Tom Pelaia,
Laboratory of Nuclear Studies, Cornell University, Ithaca, NY 14853

March 20, 1998

1 Introduction

The performance of an accelerator depends to a large extent on the quality of the theoretical algorithms and the level of their integration with the control software. Modern accelerator facilities are complex industrial-scale systems that are characterized by tightly bound, diverse scientific and technical problems. New severe requirements for accelerator parameters result in the strong specialization of scientists on particular physical effects, algorithms, or technologies. On the other side, the accelerator overall performance is determined by a variety of combined heterogeneous effects and requires steering theoretical and experimental activities of several laboratories in a common direction. There is a present need for a software environment that will facilitate reuse and integration of diverse accelerator algorithms, provide compatible and independent implementation of critical applications, and promote standardization of the best accelerator solutions and approaches. The Unified Accelerator Libraries (UAL[1]) has been addressed to solving this task.

The UAL is an object-oriented and modular software environment for accelerator physics which comprises an *accelerator object model* for the description of the machine (SMF, for Standard Machine Format), a collection of *Physics Libraries*, and a *Perl interface* that provides a homogeneous shell for integrating and managing these components. At this time, the UAL joins several libraries: Platform for Accelerator Codes (PAC[2]), a collection of Accelerator Objects that can be shared, exchanged, or converted by other codes and processes; TEAPOT++, a collection of C++ physics modules conceptually derived from TEAPOT[3]; ZLIB++[4], a differential algebra package for map generation; and Accelerator Libraries' Extensions (ALE), a collection of the UAL extensions, such as adaptors, user-friendly interfaces, DA Runge-Kutta and Lie integrators, and others. This software environment has been used to build LHC and RHIC models and to simulate their performance [5][6]. The reported work presents the design and implementation of the UAL-based CESR off-line simulation module.

2 CESR Database

The CESR control system[7] enables independent control of most powered electric and magnetic elements in the storage ring. Elements are numbered individually within groups called nodes and labelled with appropriate twelve character mnemonics. For example, sextupoles are controlled and monitored by referring to the sextupole node labelled "CSR SEXT CUR" and the relevant element numbers, (in this example ranging from 1 to 98), for the specific sextupoles. An optics can be regenerated simply by restoring the appropriate data to every element of every node with the exception of a few mechanical changes that are not automated such as IR quad rotation angles and for some exotic experiments, quadrupole polarity flips. The optics will also depend on the actual magnet positions and errors. Together, the magnet positions and errors along with the control system data and a few other non automated data specified in text files, completely describe an optics.

A complete description of a CESR optics is available as a collection of files which we will refer to as the CESR database. A layout file, Master6.dat, provides survey based data describing the lengths and positions of every major element in the storage ring. Also, it provides a mapping between each physical element and its corresponding node mnemonic and element control system reference. A saveset file is a snapshot of the present control system state and contains an unordered list of node mnemonics and the data for the corresponding elements in computer units. Several such saveset files may exist for one particular design optics but reflect different tuned conditions. Restoring a saveset will restore the conditions that existed when the saveset was taken. CESR has recently adopted a MAD derivative, commonly called BMAD[8], to describe the design optics. Typically, a BMAD optics description consists of two files. One file is a layout file which may be shared among several different design optics since it contains layout information likely to change only with scheduled infrequent hardware movements. A second file contains information specific to an optics such as the design strengths of magnets. These files may be used to provide information not saved or restored in savesets, but necessary to complete the description of an optics. Finally, a number of files exist which specify how to convert the computer units for elements of nodes to physical MAD style units.

3 CESR UAL Adaptor

The migration of accelerator parameters between the CESR database and UAL environments is implemented as the Perl package. The overall architecture is illustrated in Fig. 1 and Fig. 2.

The CESR::SMF::SMF class is a typical object adaptor that inherits an SMF interface and overrides two persistence service methods, *store* and *restore*, for translating SMF data to and from CESR files. The BMAD layout files are parsed by CESR::SMF::Shell. Because the BMAD input format is very similar to the MAD language, this class reuses the services previously developed for the SMF-MAD bridge. The PAC::MAD::Shell class does not only provide the connection of two environments, but also implements a user-friendly MAD-specific interface and gives MAD users convenient access to underlying SMF structures and services. The following example illustrates the similarity between a MAD directive for the definition of a quadrupole and the corresponding PAC::MAD::Shell method:

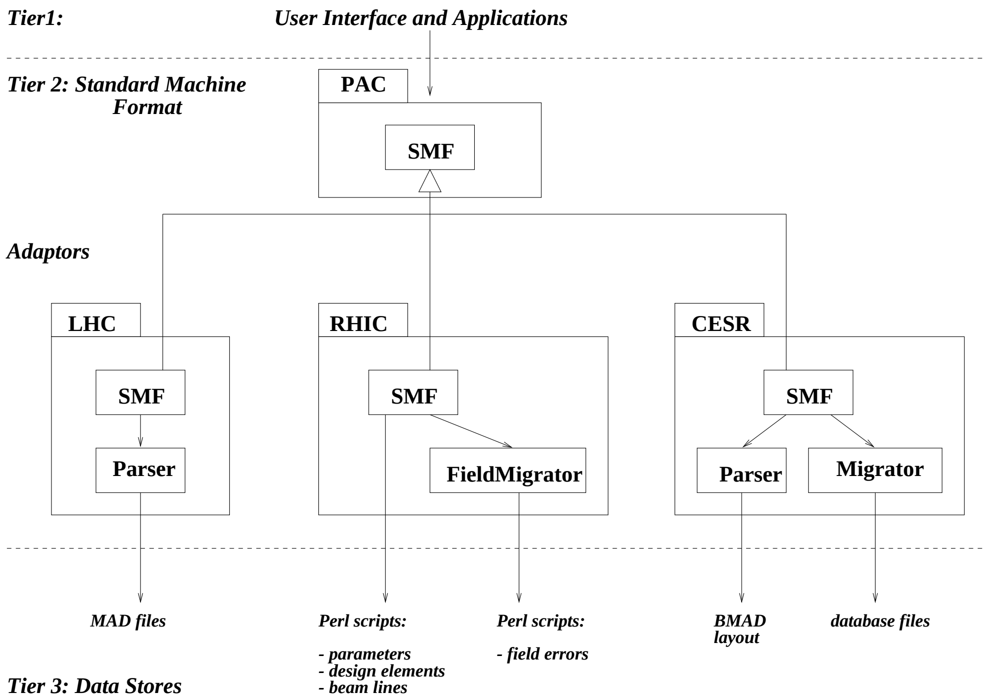

Figure 1: Three-tier UAL architecture

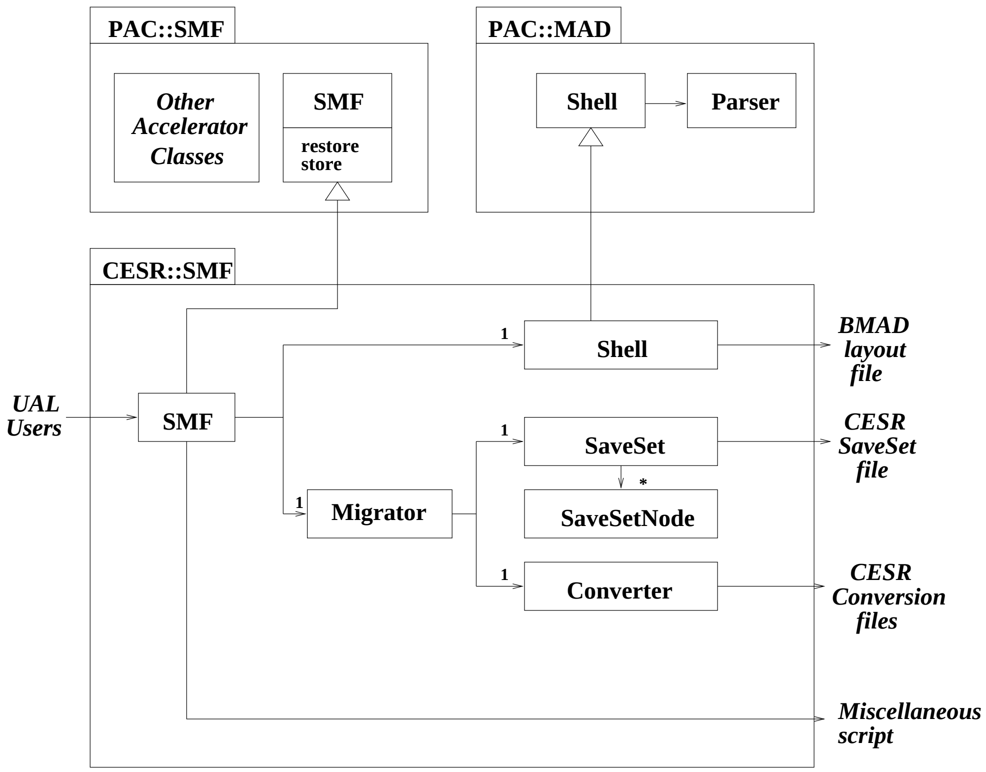

Figure 2: Structure of the CESR-SMF Adaptor.

MAD 8.	qf1 : quadrupole, l = 1.2, k1 = 2.3e-5
MAD::Shell.	element("qf1", "quadrupole", l => 1.2, k1 => 2.3e-5)

The last line is written directly in a programming language and does not require additional efforts for its translation. Moreover it can be processed inside other Perl modules and statements.

The CESR::SMF::Migrator class extracts data from CESR database files and records them in Perl objects. Each CESR file is managed by the corresponding Perl class. An instance of a SaveSet is a collection of SaveSetNode objects hashed by mnemonic, thus forming an object representation of a saveset file. It provides the following services: reading of saveset nodes, diagnostic printing of data to the terminal window, and counting of nodes and elements. Methods for producing new saveset files from SaveSet objects will be implemented in the future. A CESR::SMF::Converter class converts data between computer and physical MAD format units. Its algorithms are based on several CESR conversion files that are stored in Perl data structures after initialization. The class implements two methods, CU_To_MAD and MAD_To_CU that transform data in both directions.

The present CESR database does not include all required data (such as quadrupole rotation, solenoid field, *etc.*) for simulating accelerator parameters. Eventually, they may be organized as additional files or replaced by the BMAD persistent representation. To complete the bridge between the CESR and UAL environments we describe all miscellaneous data directly in a Perl script. This script works as an open universal interface to the SMF data structures and permits one to insert measured fields, distribute position or field errors, represent composite elements by Taylor maps, and make other project-specific extensions.

Most of CESR elements are described by the standard MAD attributes. But there are two element types, wiggler and element with the superimposed quadrupole and solenoid fields, that require special consideration.

There are two permanent magnet wigglers installed in CESR which are typical devices of most electron facilities for generating synchrotron radiation. They affect the beam dynamic aperture and should be included in the simulation experiment to predict the machine luminosity and other integral characteristics. The wiggler field is very ununiform and 3-dimensional, therefore it cannot be correctly approximated by the standard 2-dimensional elements. The UAL environment provides the mechanism for representing such "unconventional" elements by Taylor maps of arbitrary order. The wiggler is implemented as the CESR::DA::Wiggler Perl class (See Fig. 3), a specialization of the DA Runge-Kutta integrator. The Perl language supports operator overloading and creates ideal conditions for automatic differentiation. The DA::RK::Integrator class generates a Taylor map for elements with no magnetic field, and the CESR::DA::Wiggler reuses its algorithms for a wiggler described in the Halbach's approximation.

In the CESR interaction region there are areas where the CLEO detector solenoid covers two CESR permanent magnet quadrupoles. It is convenient to represent such areas by a new element type with superimposed characteristics. The superposition of element attributes is a distinguished feature of the SMF design [9]. At this time, the corresponding tracking algorithm has been implemented directly in the the TEAPOT integrator. The new TEAPOT version will provide a more powerful solution that allows accelerator scientists to extend and

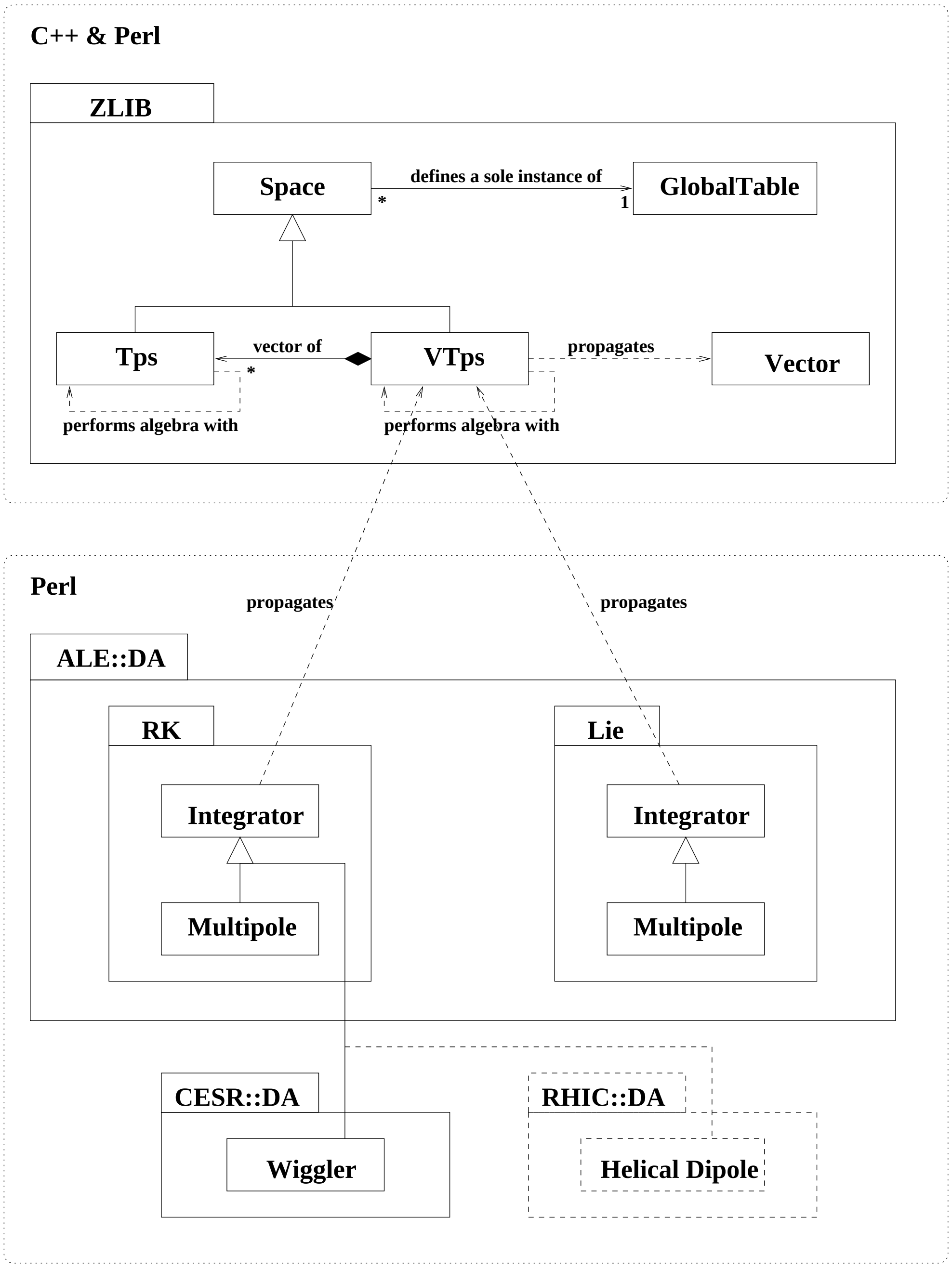

Figure 3: UAL differential algebra packages and their application to a CESR wiggler.

augment existing algorithms by their own independently developed approaches.

4 Application Programming Interface

The Application Programming Interface (API) to the UAL environment is implemented in the Perl language. The development of software in two languages, interpreter and compiler, has many benefits ([1]) and is becoming a commercial standard (Java/JavaScript, Visual C++/Visual Basic, etc.). On Unix platforms the Perl environment provides the homogeneous shell to control and integrate together diverse C++ libraries and Perl services. One can reuse Perl modules to extend existing interfaces in various ways: to make them more specific or convenient; to add new capabilities; or to integrate with other reusable components.

However, such useful and powerful features as modularity, reusability, and extensibility necessary in the period of software development and customization may disrupt the interface or make it too complex and error-prone, increase the learning curve, and eventually diminish its effectiveness. This problem can be solved by introducing an additional layer, an end-user shell. In computer terminology, it is called a facade whose main purpose is to hide complexity of underlying components and its organization and to simplify their control. Each team and collaboration may define their own interface better suited to their tasks, background, vision, or other requirements (See Fig. 4). For our project we have reused the ALE::UI::Shell class provided by the ALE package. Its interface and the sample Perl script are documented in the Appendix.

5 Scenario

The integration of CESR and UAL environments is designed to provide a consistent approach for CESR off-line simulation with the primary purpose of facilitating machine study and operation control. To verify robustness and correctness of future results, the facility has to be tested against an independent analysis scheme. It is desirable if this alternative variant utilizes a different architecture and different theoretical algorithms, and intersects with the UAL components only in two places: input data and final results. The CESR-BMAD-MAD chain satisfies these criteria and has been chosen for comparing primary accelerator parameters: tracking results and Taylor maps. Fig. 5 illustrates the structure diagram of this scenario.

The control system along with a lattice layout file serves as a common data space for the two alternative approaches. For a fixed lattice layout, the control system can reproduce an optics from a CESR saveset and a collection of various conversion tables. Lattice layout information is provided in the BMAD layout description file and is typically stable over several months, and only changing to accommodate new hardware installation and update survey data and unique lattice configurations.

At the time that we made our comparison, CESR was in the process of converting its optics design and description from the "Z" format to the new "BMAD" format. A Z-file lattice description has been used to load an optics into the CESR control system and produced a saveset, a persistent representation of the CESR element parameters. This data is brought into the UAL formalism via the CESR::SMF package. This package initializes in-memory

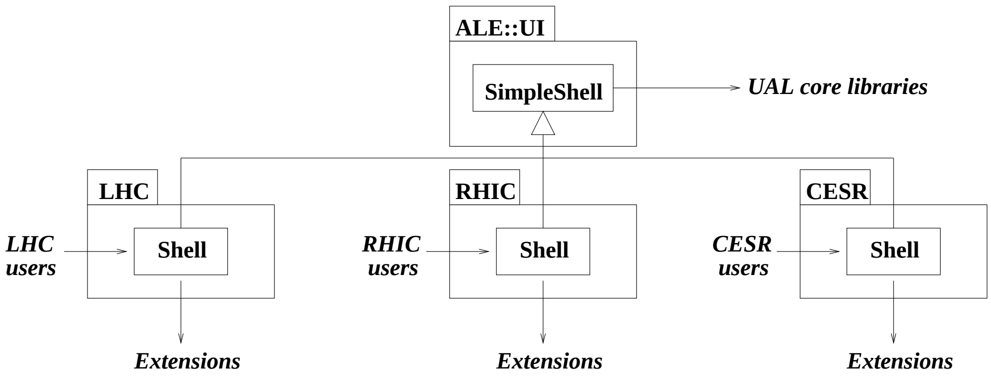

Figure 4: User shells: application-specific user interfaces.

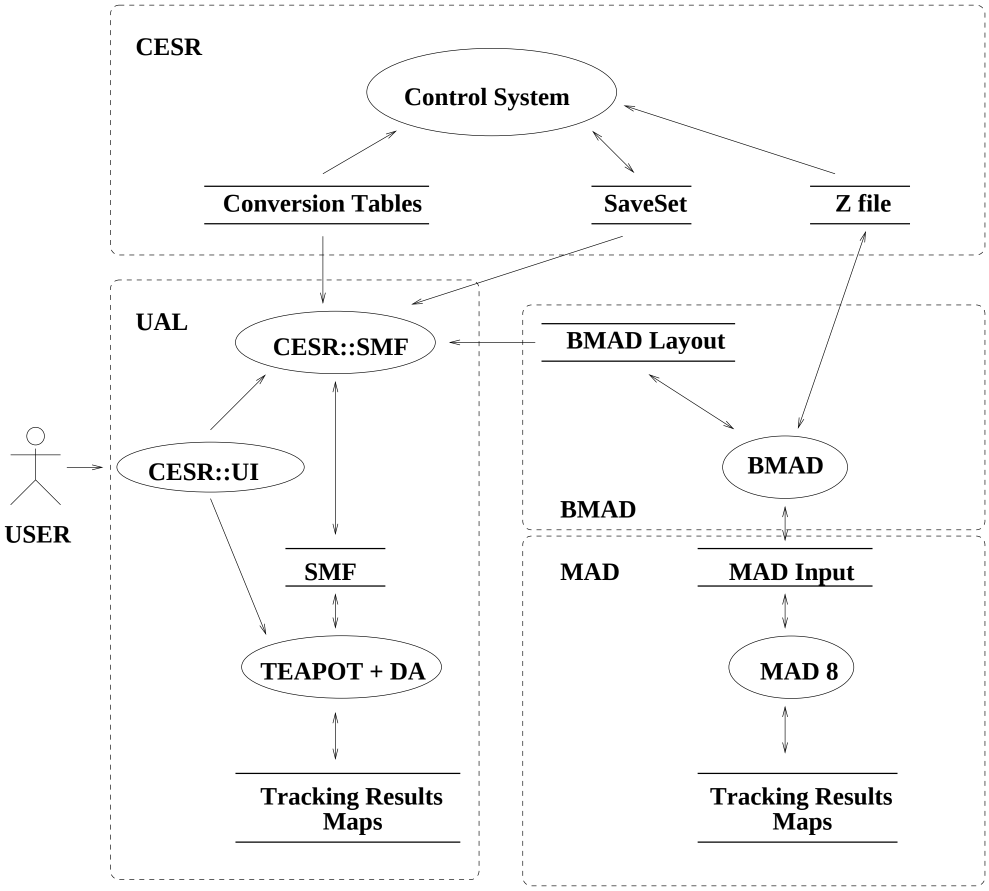

Figure 5: Structure of the CESR off-line simulation module.

SMF data structures from three different sources: the saveset, conversion tables and the BMAD layout file. All components are glued together by the CESR::UI package that provides the uniform CESR-specific user interface to SMF data and UAL tracking, analysis, and fitting services. The tracking results and Taylor maps are produced by the same TEAPOT algorithms using the automatic differentiation technique[1].

The BMAD-MAD scheme is initialized by the same Z and BMAD layout description files. These sources include only linear element parameters, such as dipole, quadrupole, and solenoid fields and can be described directly in the MAD input format. However, there is one element type that does not have prototypes in the MAD environment. We call it "solquad". This element has arisen from a superposition of the CLEO solenoid and an IR permanent magnet. The BMAD program is designed to support this superposition of attributes and to generate the standard MAD input file where the solquad is represented by a linear transform matrix. We have evaluated this file and found that MAD generates different tracking results in TRANSPORT and LIE4 modes:

position:	x	px	y	py
MAD/LIE4	-0.38204e-04	0.87381e-04	0.50631e-04	-0.88505e-07
MAD/TRANSPORT	-0.11604e-04	0.17941e-05	-0.92072e-05	0.31670e-03
UAL/TEAPOT	-0.11605e-04	0.17841e-05	-0.91972e-05	0.31692e-03

The TEAPOT one-turn map coefficients are consistent with TEAPOT and TRANSPORT tracking results. The same procedure has been repeated for the case when the solquad is represented by the standard MAD skew quadrupole (without a linear matrix). These tracking results and Talor maps agree with reliable accuracy:

position:	x	px	y	py
MAD/LIE4	-0.37256e-04	-0.18726e-05	-0.85675e-05	0.32130e-03
MAD/TRANSPORT	-0.37256e-04	-0.18726e-05	-0.85675e-05	0.32130e-03
UAL/TEAPOT	-0.37271e-04	-0.18852e-05	-0.85645e-05	0.32151e-03

The discrepancy between UAL and BMAD-MAD schemes is not significant and has well-controlled sources: convenrsion mechanism and TEAPOT thin element representation.

6 Conclusions

A CESR SMF package has been developed to provide UAL access to the CESR database. Using the Shell object to abstract the details from the user, a user may import CESR database information to the Standard Machine Format (SMF), for various operations including particle tracking, data manipulation and output. Studies may be performed on both the design optics and the actual optics run for high energy physics. It is hoped that off-line studies will provide insight on machine errors and account for discrepancies. In addition to studying machine

[1]This tracking engine is implemented as a C++ class template and instantiated for floating-point and truncated power series (Tps) variables

errors, we have interest in studying electron injection from the Synchrotron into CESR in the presence of stored electrons and positrons. During electron injection, one must accomodate three beam apertures simultaneously, that of the injected electrons, and the stored positrons and electrons. Additionally, a pulsed three element closed bump is used to bring the stored electron beam closer to the injection septum to reduce the amplitude of the injected electron excursions. The bump is a half wave sine pulse lasting for six turns. The filling rate and stored beam lifetimes are sensitive to the pulsed bump closure. However, the bump cannot be closed simultaneously for both the stored electrons and positrons. It would be useful to study the injected electrons in the presence of the two stored beams in order to develop a clearer understanding of the limiting apertures.

References

[1] N.Malitsky and R.Talman. *Unified Accelerator Libraries*, AIP 391, Williamsburg, 1996

[2] N.Malitsky, A.Reshetov, G.Bourianoff. *PAC++: Object-Oriented Platform for Accelerator Codes*, SSCL-675, June 1994.

[3] L.Schachinger and R.Talman. *Teapot: A Thin-Element Accelerator Program for Optics and Tracking*, Particle Accelerators, **22**, 35(1987).

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

[5] N.Malitsky and R.Talman. *Study of LHC Aperture Dependence on Tune Separation Using Thin Lenses, Phase Trombones, and "Unified Accelerator Libraries"*, LHC Project Note, 1998.

[6] F.Pilat, S.Tepikian, C.G.Trahern, N.Malitsky. *A model of RHIC using the Unified Accelerator Libraries*, RHIC/AP/146, 1998

[7] C.R.Strohman and S.B.Peck. *Architecture and Performance of the New CESR Control System*, PAC 89, Chicago, IL, pp. 1687-1689.

[8] D.Sagan and D.Rubin. *Private Comminications.*

[9] N.Malitsky, R.Talman, *et.al. A Proposed Flat Yet Hierarchichal Accelerator Lattice Object Model*, Particle Accelerators, **55**, 67(1996).

Appendix.

Class ALE::UI::SimpleShell

Extends:

The SimpleShell class provides a simple user-friendly (MAD+TEAPOT-specific) interface to UAL environment.

Sample Script: SimpleShell.pl

Public Methods

Constructor

- **new()**
  Constructor.

Accelerator description, selection, etc.

- **split(%parameters)**
  Splits selected elements into several thin multipoles.
    parameters{$pattern} - an associative array of pattern for selecting elements and TEAPOT split number (1 - IR, 2 - IR2, etc.).
  Example :
    split("^(i_qx|qx).*" => 1)
- **use($lattice)**
  Selects an accelerator lattice for operations (Builds a lattice from a MAD line or MAD sequence with the same name).
    lattice - a lattice name.
  Example :
    use("lhc")

Analysis

- **analysis(%parameters)**
  Finds the closed orbit and performs twiss analysis of the machine.
    $parameters{print} - an output file name (default: "./analysis").
    $parameters{delta} - a momentum offset.
  Example :
    analysis("print" => "./data/analysis.out", delta => 1.0e-3)

Figure 6: The ALE::UI::SimpleShell interface.

Fitting

- **hsteer(%parameters)**
  Flattens the orbit horizontally (Delegates the request to the corresponding TEAPOT command).
  $parameters{adjusters} - a regular expression for selecting adjusters.
  $parameters{detectors} - a regular expression for selecting detectors.
  Example :
  hsteer(adjusters => "^kickh\$", detectors => "^bpmh\$");
- **vsteer(%parameters)**
  Flattens the orbit verically (Delegates the request to the corresponding TEAPOT command).
  $parameters{adjusters} - a regular expression for selecting adjusters.
  $parameters{detectors} - a regular expression for selecting detectors.
  Example :
  vsteer(adjusters => "^kickv\$", detectors => "^bpmv\$");
- **tunethin(%parameters)**
  Fits the tunes of the machine (Delegates the request to the corresponding TEAPOT command).
  $parameters{bf} - a regular expression for selecting focusing elements.
  $parameters{bd} - a regular expression for selecting defocusing elements.
  $parameters{mux} - a requested horizontal tune value.
  $parameters{muy} - a requested vertical tune value.
  $parameters{method} - a method to alter correction elements, multiplicative ('*') or additive ('+') (default: '*').
  $parameters{numtries} - the maximum number of iterations for the fitting (default: 100).
  $parameters{tolerance} - the maximum absolute value of the difference between requested values and the fitted values at convergence (default: 1.0e-6).
  Example :
  tunethin(bf => "^qf\$", bd => "^qd\$", mux => 28.19, muy => 29.18);
- **chromfit(%parameters)**
  Fits the chromaticity of the machine (Delegates the request to the corresponding TEAPOT command).
  $parameters{bf} - a regular expression for selecting focusing elements.
  $parameters{bd} - a regular expression for selecting defocusing elements.
  $parameters{chromx} - a requested horizontal chromaticity value.
  $parameters{chromy} - a requested vertical chromaticity value.
  $parameters{method} - a method to alter correction elements, multiplicative ('*') or additive ('+') (default: '*').
  $parameters{numtries} - the maximum number of iterations for the fitting (default: 10).
  $parameters{tolerance} - the maximum absolute value of the difference between requested values and the fitted values at convergence (default: 1.0e-4).
  Example :
  chromfit(bf => "^sf\$", bd => "^sd\$", chromx => -3.0, chromy => -3.0)

Figure 7: The ALE::UI::SimpleShell interface (cont.)

- **decouple(%parameters)**
  Zeros two elements, E12 and E22, of the matrix E = B + bar(C), as well as adjusting the tunes (Delegates the request to the corresponding TEAPOT command).
  $parameters{a11} - a regular expression for selecting the 1st sextupole family.
  $parameters{a12} - a regular expression for selecting the 2nd sextupole family.
  $parameters{a13} - a regular expression for selecting the 3rd sextupole family.
  $parameters{a14} - a regular expression for selecting the 4th sextupole family.
  $parameters{bf} - a regular expression for selecting focusing quadrupoles.
  $parameters{bd} - a regular expression for selecting defocusing quadrupoles.
  $parameters{mux} - a requested horizontal tune value.
  $parameters{muy} - a requested vertical tune value.
  Example :
  decouple(a11 => "^sqsk6\$", a12 => "^sqsk8\$", a13 => "^sqsk12\$", a14 => "^sqsk2\$", bf => "^qf\$", bd => "^qd\$", mux => 28.19, muy => 29.18)

Survey (Accelerator Geometry)

- **survey(%parameters)**
  Calculates an accelerator geometry (survey) and prints results for selected elements.
  $parameters{print} - an output file name (default: "./survey").
  $parameters{observe} - a regular expression for selecting elements (default: " ").
  Example :
  survey(print => "./data/survey.out", observe => "ip")

Tracking

- **beam(%parameters)**
  Defines beam parameters (Initializes the Pac::BeamAttributes object).
  $parameters{energy} - beam energy (GeV, default: infinity).
  $parameters{mass} - particle mass (GeV, default: 0.93828).
  $parameters{charge} - particle charge (default: 1).
  Example:
  beam(energy => 42.e+2, mass => 0.93828, charge => 1)
- **start(@positions)**
  Defines particles' initial coordinates, displacements from the reference orbit (Initializes the Pac::Bunch object)
  positions - an array of MAD particle coordinates, [x, px/p0, y, py/p0, dt, de/p0].
  Example:
  start([1.e-5, 0.0, 1.e-5, 0.0, 0.0, 1.e-5], [2.e-5, 0.0, 2.e-5, 0.0, 0.0, 1.e-5],)
- **firstturn(%parameters)**
  Tracks a particle for an one turn and prints results at selected elements.
  $parameters{print} - an output file name (default: "./firstturn").
  $parameters{observe} - a regular expression for selecting elements (default: " ").
  Example :
  firstturn(print => "./data/firstturn.out", observe => "ip")

Figure 8: The ALE::UI::SimpleShell interface (cont.)

Mapping

- **space($shell, $order)**
  Defines global parameters of 6D maps.
  order - the maximum order of Taylor maps.
- **map(%parameters)**
  Makes a one-turn 6D map.
  $parameters{order} - the map order.
  $parameters{print} - an output file name.
  Example:
  map(order => 2, print => "./data/map.out")
- **matrix(%parameters)**
  Makes a one-turn 6D linear matrix (FTPOT approach).
  $parameters{print} - an output file name.
  Example:
  matrix(print => "./data/matrix.out")

I/O methods

- **read(%parameters)**
  Reads accelerator data from local sources (Delegates *parameters* directly to the PAC::FTPOT::Shell::restore method).
  $parameters{files} - a pointer to array of MAD file names.
  Example :
  read(files => ["./data/lhc.Kinj", "./data/lhc.seq"])
- **write(%parameters)**
  Writes SMF data to a MAD file (Delegates *parameters* directly to the PAC::FTPOT::Shell::store method).
  $parameters{file} - a MAD file name.
  Example :
  write(file => "./data/shell.out")

Figure 9: The ALE::UI::SimpleShell interface (cont.)

Class CESR::UI::Shell

Extends: ALE::UI::SimpleShell

The Shell class provides a user-friendly (MAD+TEAPOT-specific) interface to UAL environment.

Sample Script: Shell.pl

Overridden Methods

- **new()**
  Constructor.

Figure 10: The CESR::UI::Shell interface.

```
use lib ("$ENV{UAL_CESR}/api");
use CESR::UI::Shell;

# Make the shell
my $shell = new CESR::UI::Shell();

# Define DA Space
my $maxOrder = 2;
$shell->space($maxOrder);

# Intialize SMF structures from CESR data sources

$shell->read(layout => "../data/cesr_layout.bmad");
$shell->read(misc   => "../data/misc.pl");

$shell->split("^(q|b|qadd\_)[0-9][0-9].*" =>  4);
$shell->split("^q01.*" => 12);
$shell->split("^q00.*" => 12);

$shell->read(line => "cesr",
             saveset => "../data/history.dat",
             energy => 5.289);

# Select an accelerator  for operations
$shell->use("cesr");

# Calculate and print out an accelerator geometry (survey)
$shell->survey(print => "./out/survey", observe => " ");

# Define beam parameters
$shell->beam(energy => 5.289, mass => 0.511e-3);

# Make Analysis
$shell->analysis(print => ">./out/analysis");

# Make matrix
$shell->map(order => 1, print => "./out/map");

# Define initial particle coordinates
$shell->start([1.0e-5, 0.0, 1.0e-5, 0.0, 0.0, 0.0]);

# Make and print a first turn track
$shell->firstturn(print => "./out/firstturn", observe => "");

# Write FTPOT/MAD input files
$shell->write(layout  => "./out/layout",
              strengths => "./out/strengths");
```

Figure 11: The sample Perl script.