A model of RHIC using the Unified Accelerator Libraries

F.Pilat, S.Tepikian, C.G.Trahern, BNL
N.Malitsky, Cornell

1. Introduction.

The Unified Accelerator Library (UAL) [1] 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. Currently available physics libraries include TEAPOT++, a collection of C++ physics modules conceptually derived from TEAPOT [2], and DA/ZLIB [3], a differential algebra package for map generation.

This software environment has been used to build a flat model of RHIC which retains the hierarchical lattice description while assigning specific characteristics to individual elements, such as measured field harmonics. A first application of the model and of the simulation capabilities of UAL has been the study of RHIC stability in the presence of siberian snakes and spin rotators. The building blocks of RHIC snakes and rotators are helical dipoles, unconventional devices that can not be modeled by traditional accelerator physics codes and have been implemented in UAL as Taylor maps.

Section 2 describes the RHIC data stores, Section 3 the RHIC SMF format and Section 4 the RHIC specific Perl interface (RHIC Shell). Section 5 explains how the RHIC SMF and UAL have been used to study the RHIC dynamic behavior and presents detuning and dynamic aperture results. If the reader is not familiar with the motivation and characteristics of UAL, we include in the Appendix an useful overview paper.
An example of a complete set of Perl Scripts for RHIC simulation can also be found in the Appendix.

2. The RHIC Data Stores

The RHIC model has been adapted to the UAL Standard Machine Format (SMF) via a set of C++ programs. Referring Figure 1, the following general features are apparent. At the top are located various data stores: the optics design, NameLookup, magnetic field measurement and alignment databases. In the middle of the figure are described various processes and intermediate data stores such as HookLat, the HookedLattice model, magstat, the FieldModel and at the bottom of the figure the *rhicSMF* program and the entire set of UAL libraries and interfaces. We discuss each of these general areas.

The optics, NameLookup, magnetic field and alignment databases are all structured in terms of various tables. In this form the data is actually not convenient for many of the RHIC applications (not necessarily involved with UAL) that need the data stored therein. For example, the optics design is structured as a set of hierarchical beamlines in the database, and we need to generate a flat sequential list description to provide the hooks for the field and alignment data to individual magnetic components. A set of processes, *dbsf* and *HookLat* are available to do this. First, *dbsf* create a flat lattice structure from the optics database, and then *HookLat* adds to this the Hook structure which associates information from the magnetic field and alignment data. *HookLat* needs in addition to the optics structure the data from NameLookup. NameLookup contains the cross-reference of names from the lattice to both the magnet serial and survey names which provide the keys to the field measurement and alignment databases. When *HookLat* has finished its process-

ing, a temporary hooked lattice structure (HookedLattice) with all installed component names is available for construction of the RHIC SMF model.

The magnetic field database is analyzed by a process called *magstat.* Individual and average multipole coefficients are computed for each magnet and magnet type and the output is stored in a structure called the FieldModel. The FieldModel is indexed by the magnet serial name. The alignment data is analyzed analogously by a program called *survstat.* Although we have not incorporated the alignment data as yet, the output of *survstat* will similarly be structured to provide input to the SMF generation process. Each installed magnet in this SurveyModel structure will be indexed by its survey name.

The final stage in the generation of the RHIC SMF is the C++ program *rhicSMF.* This program reads both the HookedLattice and FieldModel data to produce a set of Perl scripts. These Perl scripts are written according to the RHIC Ascii SMF format and incorporate the lattice definition and assignment of multipole coefficients to each of the installed magnets. Currently, the program generates an SMF description only for the RHIC Blue ring because the algorithm which assigns the front and end multipoles as a function of magnet orientation of each component in the ring is accurate only for the blue ring. In principle, an SMF model for both rings can be constructed by the *rhicSMF* program.

3. The RHIC Ascii SMF Format

Access to the UAL suite is provided via a Perl-based shell environment. Each UAL project (RHIC, CESR, LHC, etc.) has developed a set of Perl modules which provide a user friendly means of calling the suite of UAL programs, e.g., particle tracking via TEAPOT++ or differential algebra and mapping techniques. As mentioned earlier the *rhicSMF* program loads the state of RHIC data sources to a set of Perl scripts that provide access to the SMF data structures. One can consider these scripts as the *Ascii SMF representation.* The SMF for RHIC has four levels.

The *first level* comprises the definition of various parameters such as magnet or vacuum pipe lengths and magnet strengths. UAL supports the algebraic manipulation of these parameters to the same extent as the Perl language permits. So a group of parameters can be used to calculate other parameters in as complicated an expression as the user would like. This is one advantage of using the Perl language as an interface/parser compared to building one's own (e.g., as in MAD).

The *second level* defines all the elements used in the RHIC lattice. This includes all the standard element types from MAD with their associated attributes as well as generic types with additional complexities that may require, for example, the use of a Taylor map expansion to properly model the element's magnetic behavior.

The *third* and *fourth levels* define the beam line hierarchy and the flat lattice, respectively. A beam line in the level three description or beam line hierarchy, similar to that used by MAD, comprises a set of level two elements defined as a sequential collection of element names. The total collection of beam lines has a tree structure wherein the topmost beam line contains a sequential list of all the elements of the lattice upon unraveling its embedded components.

The level four or lattice description is a special structure corresponding to the flat version of the topmost beam line where all the individual magnetic field and misalignment errors can be associated.

For example, the set of Perl scripts for the RHIC SMF description ultimately appears as the following set of files:

```
script.pl

rhicSMF_level_1.pl
rhicSMF_level_2.pl
rhicSMF_level_3.pl

rhicSMF_DRG_DR8_deviations.pl
rhicSMF_D5_deviations.pl
rhicSMF_D96_deviations.pl
rhicSMF_DRX_deviations.pl
rhicSMF_DRZ_deviations.pl
rhicSMF_QR4_deviations.pl
rhicSMF_QR7_deviations.pl
rhicSMF_QRG_deviations.pl
rhicSMF_QRI_deviations.pl
```

```
rhicSMF_QRJ_deviations.pl
rhicSMF_QRK_deviations.pl
rhicSMF_SRE_deviations.pl
```

script.pl is the master script. It sets up the UAL shell environment and loads the other Perl scripts into memory upon execution via Perl. The second set of three files defines the SMF levels 1, 2 and 3 as defined above. The third set of files defines the magnetic field errors for each individual magnet in the blue ring. These error sets are organized by magnet type following a RHIC naming convention, e.g., DRG is the generic designation for arc dipole magnets.

The definition of the methods used for the error sets requires some explanation. A *FieldMigrator* module was created especially for the RHIC shell to handle the incorporation of these error sets. It includes two components. The first component is the definition of a regular expression or symbolic pattern based on the generic name of the lattice element. This pattern is used to define an index into the lattice (i.e., at level four of the SMF) to locate all the elements of a given type according to the specified field set pattern. The second component of the FieldMigrator is the definition of the array (actually a pointer to array in the Perl sense) which contains the errors to be assigned to each element according to the index defined by the pattern. One word of caution: at this point in the development of the RHIC shell, one has to insure that the sequential ordering of the array is exact with respect to the order of the index. Otherwise elements will be assigned error sets incorrectly. In future this restriction will be removed, i.e., other methods will be developed to provide for the proper matching of an individual element and its error set.

An example of this kind of code for the DRG magnets:

```
$field_set_pattern = "^d\$" ;

$field_set = [
["DR8506",[0,  -1.009289243597379e05,  -0.01896240397061743,    7.232627888663886,
               -39.93115261941632,      17945.52976543179,      1879.11306444312,
                1302851.724680563,      66645876.68558266, 0,    0.0002611918224339885,
               -0.001828954447488584,  -0.08172184420885445,   -1.032533480722652,
                19.57409442128251,     -555.9042815644231,      8769.194300734562,
               -551206.4989033153,      52114068.98722253],
           [0,  1.124734270399047e-05,  0.004343142356164382,   0.004586389422275162,
               -0.1616896609886837,     4.125159299583085,     -173.2981494699225,
               -2321.697696247766,     -367491.578206075,       14328191.49684335,  0,
                6.403454723049433e05,   0.0001308546134603,     0.004754816152471708,
                0.03627652650387134,   -0.9328249672424059,     31.50875444907683,
               -364.8382094103632,      30513.74115068492,      2281897.164312091],
           [0, -1.645399594997022e-05,   0.01587065644788565,  -0.0009457808695652173,
               -0.5482937863013698,    -1.077931073257892,      1477.594748111971,
               -895.5119685527098,     -94194.59224776652,     -2494166.667969028, 0,
               -5.446337438951757e-05,  0.006012186472900535,   0.003161240166765932,
                2.441928469803454,      0.06218833114949365,   -1396.335328743299,
               -1525.687057534246,      680589.0961000593,      424539.00731387721, ],
```

The regular expression "^d\$" will locate and define an index to all the arc dipole elements in the lattice. The $field_set array defined afterward contains the magnet serial name, if known, followed by three sets of multipole coefficients (b0-b9, a0-a9 in that order) for the body, front and end sections of the magnet. The coefficients are defined in such a way so that by adding their values to the ideal strength parameters of the magnet defined in level two of the SMF, the actual field specification for each magnet is recovered.

The above set of Perl scripts provides the complete specification of the RHIC model in the UAL environment. In order to do some calculations one has to invoke the teapot tracking engine with its associated methods. Their description will be the subject of the next section.

4. The RHIC Shell Interface

The following Perl script (reproduced without intervening comments in the Appendix) is a sample script for reading the RHIC SMF into program memory followed by the invocation of methods that are commonly used in tracking and analysis. This script also includes the insertion of Taylor maps as a replacement of certain beamlines. In particular, the snake and helix sections of the blue ring are replaced by maps in this example script. We will explain each of the sections of the script in turn.

```
use lib ("$ENV{UAL_RHIC}/api");

use RHIC::UI::Shell;
```

These statements set up the UAL environment. The environment variable UAL_RHIC (along with others) is set by the script /usr/public/ENV/setup-ual. It must be sourced before this (or any other UAL Perl script) will work.

```
if ($ARGV[0] eq "" or $ARGV[1] eq "") {
    print "Usage: perl Shell.pl 'mapType' 'nray_max'\n";
    exit;}
```

These statements process the two required command line arguments of the script. The two arguments in this example are the subdirectory name where the Taylor map files are stored, and the number of rays (i.e. directions in phase space) for tracking, respectively.

```
my $file = "my_file";

my $dir     = "$ENV{UAL}/rhic/data/update/storage";
my $dirr    = "$ENV{UAL}/rhic/data/update/irfilter";
my $trkdir  = "/track/mac/97/storage";
my $maps    = "/track/mac/97/maps/storage/$mapType";
my $ftpot   = "$trkdir/ftpot";
```

These lines set up some local (via the Perl "my") Perl variables to define various file paths used in the methods that follow.

```
local $smf = new Pac::Smf();
```

This defines the Smf object.

```
require "$dir/rhicSMF_level_1.pl";
require "$dir/rhicSMF_level_2.pl";
require "$dir/rhicSMF_level_3.pl";
```

As discussed earlier in this note, the RHIC SMF levels 1, 2 and 3 are defined in a set of files which here are loaded into memory via the Perl "require" statement. $dir points to the location of these files. To change from storage to the injection lattice, one must modify $dir appropriately.

```
local $shell = new RHIC::UI::Shell();
```

This statement defines the RHIC Shell which points to the various methods we will use to track, analyze, etc.

```
my $maxOrder = 7;
$shell->space($maxOrder);
```

Currently the shell requires the initialization of the maximum order of the mapping space via the above two state-

ments. These statements are needed whether you use maps explicitly or not.

```
$shell->read("maps" => [
                         "$maps/rot05b3.map",
                         "$maps/rot06b3.map",
                         "$maps/rot07b3.map",
                         "$maps/rot08b3.map",
                         "$maps/snk03b7.map",
                         "$maps/snk09b7.map",],);
```

These statements read in the Taylor maps for the beamline elements specified by the file names. The read method takes the parameter "maps" as a key to a Perl hash. This key then points off to the set of map files which are to be inserted into the SMF. For example, there is a beamline in the lattice called "rot05b3" (a helix section). The read/maps method will replace this beamline with the map named rot05b3.map. In general the name of the Taylor map file is beamline.map for a given beamline.

```
$shell->read("line"   => "blue",
             "fields" => ["$dir/rhicSMF_DRG_DR8_deviations.pl",
                          "$dir/rhicSMF_QRG_deviations.pl",
                          "$dir/rhicSMF_DRX_deviations.pl",
                          "$dir/rhicSMF_D96_deviations.pl",
                          "$dir/rhicSMF_D5_deviations.pl",
                          "$dir/rhicSMF_QR4_deviations.pl",
                          "$dir/rhicSMF_QR7_deviations.pl",
                          "$dir/rhicSMF_SRE_deviations.pl",
                          "$dir/rhicSMF_QRI_deviations.pl",
                          "$dir/rhicSMF_QRJ_deviations.pl",
                          "$dir/rhicSMF_QRK_deviations.pl",
                          "$dir/rhicSMF_DRZ_deviations.pl",
                          "$dirr/irAtw.pl",
                          "$dirr/irBtw.pl",],);
```

The next statements read in the set of field errors for each of the major magnet types measured in RHIC. The error sets (and their structure) were discussed in Section 4, and this is the shell syntax needed to load the level 4 deviations into the SMF. The order in which the error sets is read is not relevant. For convenience of presentation in this note we have loaded all errors at the same time. In practice one will find that the lattice will not track properly the first time if all of these error sets are imposed at once, so one breaks the addition of the error sets into groups followed by a set of corrections. The correction methods will be discussed further below.

At this point all four levels of the SMF are defined and can be used.

```
$shell->use("blue");
$shell->beam("energy" => 250.);

$shell->start([1.0e-5, 0.0, 1.0e-5, 0.0, 0.0, 0.0]);
```

However, before one can start to invoke the various methods, one has to set up a few parameters. For example, one has to issue the "use" statement to the shell in order to identify which beamline in the lattice to analyze and track. One then sets the beam energy and the initial six dimensional phase space coordinates for first turn tracking. The particle's phase space vector (the default particle is the proton) is defined as [x, px, y, py, delta, dE/E].

```
$shell->firstturn("print"=>"./out/firstturn/$file",observe=> "");

$shell->survey("print" => "./out/survey/$file", observe => "");

$shell->twiss("print" => "./out/twiss/$file");
```

In each case a method takes a set of parameters, like "print" or "observe" whose values are specified via the "=>" symbol. Note that the directory specified by the "print" parameter must already exist. The "print" method will not create a directory.

```
$shell->map("order" => 2, "print" => "./out/maps/$file");

$shell->matrix("order" => 1, "print" => "./out/matrices/$file");
```

The methods map and matrix invoke DA mapping methods to create either a one turn 6 dimensional map or matrix. Now we take up the correction methods in the way they are normally used.

```
$shell->analysis("delta"=>0.0,"print"=> ">./out/analysis/$file");
$shell->hsteer("adjusters"=>"^kickh\$", "detectors" => "^bpmh\$");
$shell->vsteer("adjusters"=>"^kickv\$", "detectors" => "^bpmv\$");
$shell->analysis("delta"=>0.0, "print"=>">>./out/analysis/$file");
$shell->decouple("a11"  => "^sqsk6\$",  "a12"  => "^sqsk8\$",
                 "a13"  => "^sqsk12\$", "a14"  => "^sqsk2\$",
                 "bf"   => "^qf\$",     "bd"   => "^qd\$",
                 "mux"  =>  28.19,      "muy"  => 29.18);
$shell->analysis("delta"=>0.0, "print"=>">>./out/analysis/$file");
$shell->chromfit("bf" => "^sf\$", "bd" => "^sd\$",
                 "chromx" =>-3.0, "chromy" => -3.0);
$shell->analysis("delta"=>0.0,"print"=>">>./out/analysis/$file");
```

The above methods invoke the sequence of corrections that are needed to define a properly corrected lattice for tracking. These include analysis, *hsteer*, *vsteer*, *decouple*, and *chromfit*. These methods are usually invoked in the above order as well. For those who have used Fortran TEAPOT before, the parameters required by each method are probably familiar. In any case, refer to the teapot manual for detailed explanations of the methods. The specification of the corrector elements in the steering and decouple methods by a regular expression is, of course, an inherent feature of Perl that UAL has adopted. Also, the decouple method used by UAL is different than the one used in the Fortran version of teapot: four corrector families are specified rather than two.

The decouple command performs a tunethin operation in preparation for setting the skew correction in the lattice, so a separate call to the tunethin method itself was not needed above. However, the *tunethin* method is available directly via the statement:

```
$shell->tunethin("bf" => "^qf\$", "bd" => "^qd\$",
                 "mux" => 28.19, "muy" => 29.18);
```

We are now ready to do tracking. There are some parameters to set prior to invoking the teapot tracking algorithm.

```
$file8    = "./out/8/$file";
$numturns = 1024 ;
$numsteps = 1 ;
require "$trkdir/setrays_sto.pl";
```

This tracking script, setrays_sto.pl, is RHIC specific. It sets an appropriate set of particle phase space initial conditions and invokes the tracking method. This script is also included as an example in the Appendix.

Now suppose we wish to compare the output from teapot++ with the fortran version of teapot. We can write out a lattice and error set with the following Perl statements:

```
$shell->write("file"  => "$ftpot/dat/$file",
              "field" => "$ftpot/errors/$file_errors");
```

The ftpot compatible lattice file is written to the file specified by "file" and the error sets are written to the file and directory specified by "field". The error file itself refers to a series of fort.nn files which contain the actual error set data and which are written in the same directory.

5. Simulation of RHIC in the presence of helical dipoles

Two siberian snakes and 4 spin rotators per ring are planned to achieve polarized protons in RHIC. Each snake or rotator consists of 4 helical dipole magnets: the latter are unconventional devices where the magnetic field rotates by about 360^0 along the axis of the magnet. These devices cannot be modeled with conventional accelerator physics codes. We used the UAL environment to successfully model the helical dipoles and study their impact on RHIC beam dynamics. The model and results are explained in the following.
The field of a helical dipole can be represented by a Bessel function expansion as [6]:

$$B_r = -k \sum_{m=1}^{\infty} m \cdot I'_m(mkr) \cdot \left\{ \tilde{a}_{m-1} \cos[m(\phi - kz)] + \tilde{b}_{m-1} \sin[m(\phi - kz)] \right\}$$

$$B_z = k \sum_{m=1}^{\infty} m \cdot I_m(mkr) \cdot \left\{ \tilde{b}_{m-1} \cos[m(\phi - kz)] - \tilde{a}_{m-1} \sin[m(\phi - kz)] \right\}$$

$$B_\phi = -\frac{1}{kr} B_z$$

The expected field harmonics (derived by field calculations and measurements of one prototype) for the RHIC helical dipoles [7] are listed in the following tables: Table 1 and 2 show the harmonics for the body (coefficients of the above Bessel function expansion), Table 3 and 4 show the harmonics for the edges (in this case they are coefficients of the usual dipole Taylor expansion, since edges are modeled as thin elements at the entrance and exit of the helix module) Each snake or rotator consists of 4 helical dipole magnets separated by drift spaces (see Figure 2)

Figure 2: Model of a RHIC Siberian Snake or Rotator systems

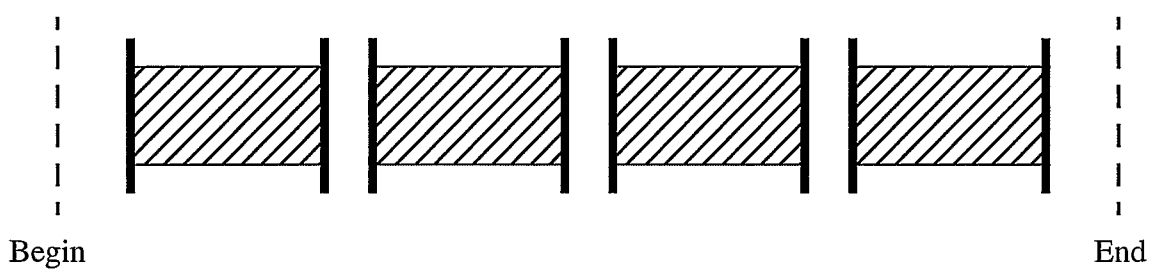

Table 1: Expected Body Harmonics, 3.1cm reference, 4.0T excitation.

n	$\langle \tilde{b}_n \rangle$	$\Delta\langle \tilde{b}_n \rangle$	$\sigma(\tilde{b}_n)$	$\langle \tilde{a}_n \rangle$	$\Delta\langle \tilde{a}_n \rangle$	$\sigma(\tilde{a}_n)$
1	0.0	2.0	2.0	0.0	2.0	2.0
2	2.0	2.0	2.0	0.0	2.0	2.0
3	0.0	2.0	2.0	0.0	2.0	2.0
4	2.0	1.0	1.0	0.0	1.0	1.0
5	0.0	1.0	1.0	0.0	1.0	1.0
6	0.5	1.0	0.5	0.0	1.0	0.5
7	0.0	1.0	0.5	0.0	1.0	0.5
8	-10.0	1.0	0.5	0.0	1.0	0.5
9	0.0	3.0	0.5	0.0	3.0	0.5
10	2.5	1.0	0.5	0.0	1.0	0.5

Table 2: Expected Body Harmonics, 3.1cm reference, 2.0T excitation.

n	$\langle \tilde{b}_n \rangle$	$\Delta\langle \tilde{b}_n \rangle$	$\sigma(\tilde{b}_n)$	$\langle \tilde{a}_n \rangle$	$\Delta\langle \tilde{a}_n \rangle$	$\sigma(\tilde{a}_n)$
1	0.0	2.0	2.0	0.0	2.0	2.0
2	-3.0	2.0	2.0	0.0	2.0	2.0
3	0.0	2.0	2.0	0.0	2.0	2.0
4	4.0	1.0	1.0	0.0	1.0	1.0
5	0.0	1.0	1.0	0.0	1.0	1.0
6	0.0	1.0	0.5	0.0	1.0	0.5
7	0.0	1.0	0.5	0.0	1.0	0.5
8	-9.0	1.0	0.5	0.0	1.0	0.5
9	0.0	3.0	0.5	0.0	3.0	0.5
10	6.5	1.0	0.5	0.0	1.0	0.5

Table 3: Expected Lead End Harmonics, 3.1cm reference.

n	$\langle b_n \rangle$	$\Delta\langle b_n \rangle$	$\sigma(b_n)$	$\langle a_n \rangle$	$\Delta\langle a_n \rangle$	$\sigma(a_n)$
1	0.0	2.0	2.0	0.0	2.0	2.0
2	-3.0	2.0	2.0	-9.0	2.0	2.0
3	0.0	1.0	1.0	0.0	1.0	1.0
4	1.0	1.0	1.0	-1.5	1.0	1.0
5	0.0	0.5	0.5	0.0	0.5	0.5
6	0.5	0.5	0.5	-1.0	0.5	0.5
7	0.0	0.5	0.5	0.0	0.5	0.5
8	0.5	0.5	0.2	-0.2	0.5	0.2
9	0.0	0.5	0.2	0.0	0.5	0.2
10	0.1	0.5	0.2	0.1	0.5	0.2

Table 4: Expected Return End Harmonics, 3.1cm reference.

n	$\langle b_n \rangle$	$\Delta\langle b_n \rangle$	$\sigma(b_n)$	$\langle a_n \rangle$	$\Delta\langle a_n \rangle$	$\sigma(a_n)$
1	0.0	2.0	2.0	0.0	2.0	2.0
2	-3.0	2.0	2.0	9.0	2.0	2.0
3	0.0	1.0	1.0	0.0	1.0	1.0
4	1.0	1.0	1.0	1.5	1.0	1.0
5	0.0	0.5	0.5	0.0	0.5	0.5
6	0.5	0.5	0.5	1.0	0.5	0.5
7	0.0	0.5	0.2	0.0	0.5	0.2
8	0.5	0.5	0.2	0.2	0.5	0.2
9	0.0	0.5	0.2	0.0	0.5	0.2
10	0.1	0.5	0.2	-0.1	0.5	0.2

Each system has been represented as a Taylor Map: the coefficients of the map have been calculated by the code *HELIX* [8] which uses a Runge-Kutta Integrator and a Differential Algebra package. The input to *HELIX* was generated by the code *MKHELIXMAP* which reads the expected harmonics listed in Tables 1-4 and writes the field error tables needed for each helical magnet. Through command line options to *MKHELIXMAP* random errors could have been included in the field error tables. The resulting map is non-symplectic and this led us to necessarily limit our

investigations to short term tracking: symplectification methods are currently under investigation. In UAL each SMF element can be represented as a map so the snakes and rotators have been included in the RHIC model quite naturally. The UAL environment gives the possibility of representing all sorts of unconventional accelerator components. It is also worth mentioning that snakes and rotators are fixed energy devices, thus different Taylor maps must be calculated at different energies to properly account for kinematic effects. The present model of the helical dipoles includes, as explained, magnet field nonlinearities, but helical field setting errors, errors in the helical wavelength and survey errors were not modeled.

The study of beam stability consisted of an investigation of linear effects and subsequently an analysis of nonlinear effects, primarily detuning and short term dynamic aperture both at injection and storage configurations for protons. The results will be summarized in the following.

The observation of the linear transfer matrices of snakes and rotators at injection evidenced significant focussing in both horizontal and vertical planes, and, in the case of the snakes, linear coupling. Compensation of focusing and coupling are within the capability of the RHIC correction system. The effect is not relevant at storage, where the linear transfer matrices of snakes and rotators are essentially drift space matrices. Beta functions and dispersion variations arising from the necessity to retune the lattice because of snakes and rotators were also analyzed. The results, listed in Table 5 are acceptable for operations, and the variation can always be reduced, if deemed necessary, by rematching the optics with snakes and rotators.

Table 5: The maximum variations of beta functions and dispersion from the ideal RHIC lattice and with combinations with ideal snakes and rotators.

Configuration	$\frac{(\Delta\beta_x)_{max}}{\beta_x}$	$\frac{(\Delta\beta_y)_{max}}{\beta_y}$	$\frac{(\Delta\eta_x)_{max}}{\eta_x}$	$(\eta_y)_{max}$ [m]
Ideal lattice, from QF	0.95%	0.39%	0.17%	0.0
Ideal lattice, from QD	0.92%	0.32%	0.26%	0.0
with Snakes, from QF	7.67%	19.99%	6.50%	0.038
with Snakes, from QD	7.51%	21.46%	5.97%	0.083
with Snakes + Rotators, QF	12.46%	19.37%	7.88%	0.047
with Snakes + Rotators, QD	12.28%	21.40%	7.23%	0.104

For the detuning analysis, tune footprints are calculated for amplitudes varying from 1 to 7 σ at injection $\sigma = 1.1115mm$ and from 1 to 5 σ at storage $\sigma = 0.1365mm$. As usually done in RHIC studies, the analysis is done on momentum and for momentum deviations of $\pm 2.5\sigma_p$.

The tune footprint for RHIC with errors and no helical dipoles is shown in Figure 3. The ideal helical dipoles (that is, without harmonics) do not contribute significantly to the detuning as seen in Figure 4. When we add errors to the snakes and rotators (Figure 5) the tune shift with amplitude is increased but at a level that does not threaten machine stability. The helical dipoles do not measurably affect the tunes at storage, as expected.

For the dynamic aperture analysis, as already remarked, studies are limited to short term tracking (1000 turns), and the results are very preliminary, since the RF cavities were off during tracking. Future studies will need to address synchrotron oscillations, but the goal of the analysis was to compare results with and without helical dipoles rather than obtain absolute dynamic aperture values. Disclaimers notwithstanding, aperture results are summarized in Table 6 for injection and Table 7 for storage.

Table 6: Proton survival at injection energies, $\sigma = 1.1115$ mm at the IP.

Comments	DA (x = y) in σ	DA (x >> y) in σ	DA (x << y) in σ
Machine & errors, without maps	11.507	11.507	12.170
Machine & errors, ideal helical maps	12.832	12.832	10.181
Ideal machine, helical + systematic	11.507	12.170	10.844
Ideal machine, helical + randoms	11.507	12.170	10.181
Ideal machine, helical + Δb_n	11.507	11.507	10.181
Ideal machine, helical + randoms + Δb_n	10.844	11.507	10.844
Machine & errors, helical + systematics	11.507	10.844	10.181
Machine & errors, helical + randoms	10.844	10.844	10.844
Machine & errors, helical + Δb_n	10.844	11.507	11.507
Machine & errors, helical + randoms + Δb_n	10.844	10.181	10.181

Table 7: Proton survival at storage energies, $\sigma = 0.1365$ mm at the IP

Comments	DA (x = y) in σ	DA (x >> y) in σ	DA (x << y) in σ
With maps	9.37	10.47	8.58
Without maps (a1 = 0 in DRZ)	10.45	10.47	8.58
Without maps (IR corr a1 not 0 in DRZ)	9.91		

Both at injection and storage the benchmark is the machine with errors and no helical dipoles. From the different configurations studied one can observe that the degradation in dynamic aperture is at worst 2 σ at injection and 1 σ at storage. Both values do not pose a challenge as far as machine performance is concerned, but, as already observed, such results need to be verified when other effect are taken in consideration, such as synchrotron oscillation and alignment errors.

6. Conclusions

RHIC have been modeled using the Unified Accelerator Library software environment. As an application we have studied the beam stability of RHIC in the presence of snakes and rotators. The impact of helical dipoles on beam dynamics is measurable, but does not threaten significantly machine performance.

The UAL environment is ready to be used and is being actively developed.

References

[1] N.Malitsky and R.Talman, AIP 391, 1996
[2] L.Schachinger and R.Talman, Particle Accelerators, 22, 35(1987)
[3] Y.Yan, SSCL-300, 1990
[4] N. Malitsky, R. Talman,"A Proposed Flat Yet Hierarchical Accelerator Lattice Object Model",BNL RHIC/AP/74
[5] D.C. Carey, F.C. Iselin, "Standard Input Language for Particle Beam and Accelerator Computer Programs", Snowmass, Colorado, 1984
[6] V.Ptitsin, RHIC/AP/41
[7] A.Jain, Private Communication
[8] N.Malitsky, RHIC/AP/51

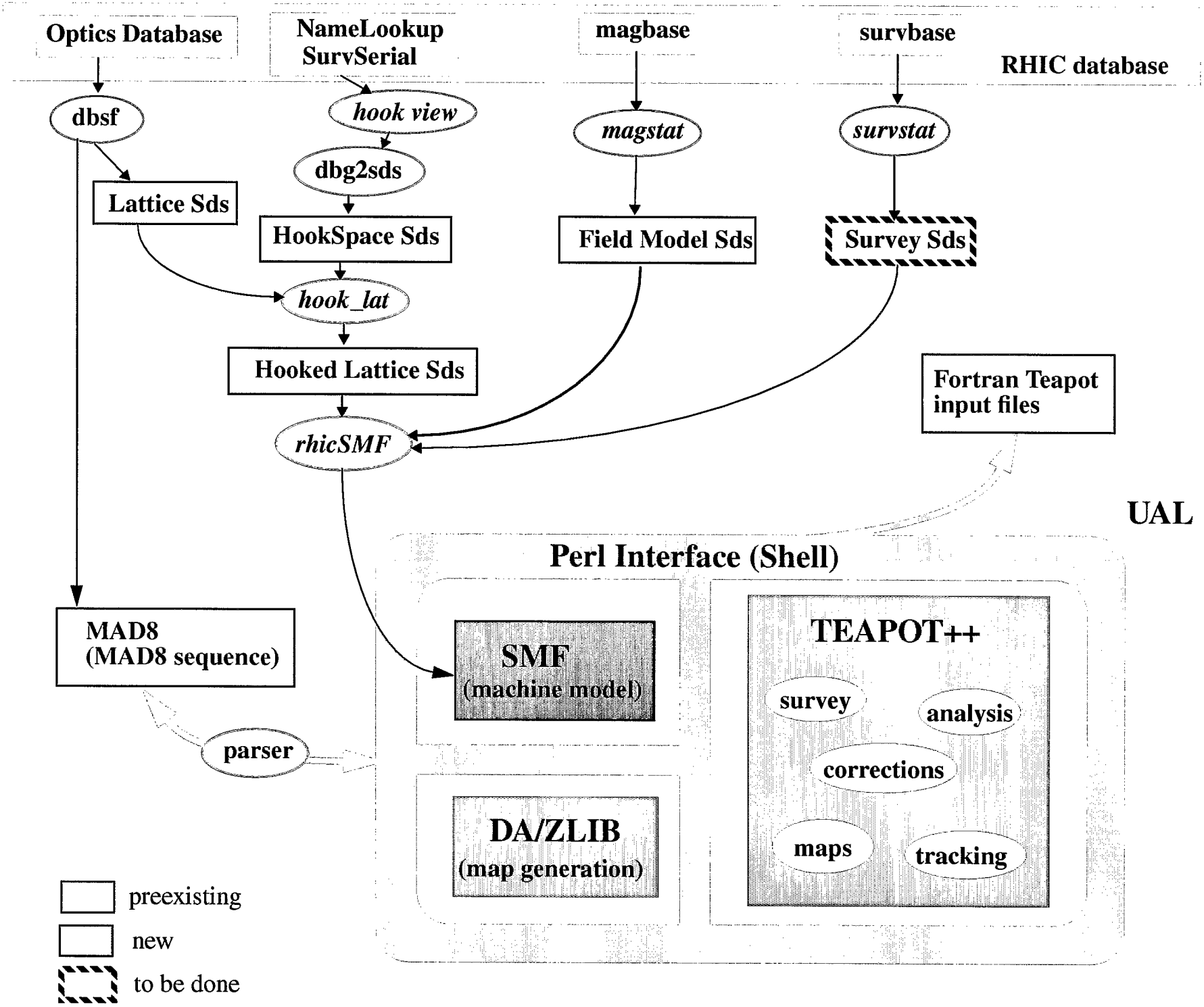

Figure 1: Overview of the UAL RHIC Model

Appendix

A) Example of shell script for RHIC simulations (Shell.pl)

```
#============== begin SMF description ================================

use lib ("$ENV{UAL_RHIC}/api");

use RHIC::UI::Shell;

if ($ARGV[0] eq "" or $ARGV[1] eq "") {
    print "Usage: perl Shell.pl 'mapType' 'nray_max'\n";
    exit;
}

my $mapType = $ARGV[0];
$nray_max   = $ARGV[1];

my $file = "my_file";

my $dir     = "$ENV{UAL}/rhic/data/update/storage";
my $dirr    = "$ENV{UAL}/rhic/data/update/irfilter";
my $trkdir  = "/track/mac/97/storage";
my $maps    = "/track/mac/97/maps/storage/$mapType";
my $ftpot   = "$trkdir/out/ftpot";

local $smf = new Pac::Smf();

require "$dir/rhicSMF_level_1.pl";
require "$dir/rhicSMF_level_2.pl";
require "$dir/rhicSMF_level_3.pl";

#============== Make the shell ======================================

local $shell = new RHIC::UI::Shell();

#============ Define DA Space ======================================
my $maxOrder = 7;
$shell->space($maxOrder);

# ==================== maps ========================================
$shell->read("maps" => [
"$maps/rot05b3.map",
"$maps/rot06b3.map",
"$maps/rot07b3.map",
"$maps/rot08b3.map",
"$maps/snk03b7.map",
"$maps/snk09b7.map",],);

#============== Read input files ====================================

$shell->read("line"    => "blue",
```

```
                "fields" => ["$dir/rhicSMF_DRG_DR8_deviations.pl",
                             "$dir/rhicSMF_QRG_deviations.pl",
                             "$dir/rhicSMF_DRX_deviations.pl",
                             "$dir/rhicSMF_D96_deviations.pl",
                             "$dir/rhicSMF_D5_deviations.pl",
                             "$dir/rhicSMF_QR4_deviations.pl",
                             "$dir/rhicSMF_QR7_deviations.pl",
                             "$dir/rhicSMF_SRE_deviations.pl",
                             "$dir/rhicSMF_QRI_deviations.pl",
                             "$dir/rhicSMF_QRJ_deviations.pl",
                             "$dir/rhicSMF_QRK_deviations.pl",
                             "$dir/rhicSMF_DRZ_deviations.pl",
                             "$dirr/irAtw.pl",
                             "$dirr/irBtw.pl",
                              ],);

#============ pick lattice ============================================================
$shell->use("blue");

#========== define beam parameters ===================================================
$shell->beam("energy" => 250.);     # injection: 25 collision: 250

#=========== first turn ==============================================================
$shell->start([1.0e-5, 0.0, 1.0e-5, 0.0, 0.0, 0.0]);
$shell->firstturn("print" => "./out/firstturn/$file", observe => "");

#============ survey ==================================================================
$shell->survey("print" => "./out/survey/$file", observe => "");

#============ Print Twiss =============================
$shell->twiss("print" => "./out/twiss/$file");

#======== Make and print an one-turn 6D map ===========================
$shell->map("order" => 2, "print" => "./out/maps/$file");

#======== Make and print an one-turn 6D matrix ========================
$shell->matrix("order" => 1, "print" => "./out/matrices/$file");

#======== corrections ================================================================
$shell->analysis("delta" => 0.0,"print" => ">./out/analysis/$file");
$shell->hsteer("adjusters" => "^kickh\$", "detectors" => "^bpmh\$");
$shell->vsteer("adjusters" => "^kickv\$", "detectors" => "^bpmv\$");
$shell->analysis("delta" => 0.0,"print" => ">>./out/analysis/$file");
$shell->decouple("a11"  => "^sqsk6\$",  "a12"  => "^sqsk8\$",
                 "a13"  => "^sqsk12\$", "a14"  => "^sqsk2\$",
                 "bf"   => "^qf\$",     "bd"   => "^qd\$",
                 "mux"  =>  28.19,      "muy"  => 29.18);
$shell->analysis("delta" => 0.0,"print" => ">>./out/analysis/$file");
$shell->tunethin("bf" =>"^qf\$","bd"=>"^qd\$", "mux"=> 28.19,"muy"=> 29.18);
$shell->chromfit("bf" =>"^sf\$","bd" =>"^sd\$","chromx"=>-3.0,"chromy"=>-3.0);
$shell->analysis("delta" => 0.0,"print" => ">>./out/analysis/$file");

#======= build header for file8 tracking output file ========
$file8     = "./out/8/$file";
```

```
$numturns = 16 ;
$numsteps = 1 ;

#==== prepare particles and track (rays) =======
require "$trkdir/setrays_sto.pl";

#======= Write FTPOT input files ===================================
$shell->write("file"  => "$ftpot/dat/$file",
              "field" => "$ftpot/errors/$file_errors");
```

B) Example of script for RHIC tracking (setrays_sto.pl)

```
print "prepare particles and track = ", time, "\n";

# Parameters ----------------------------------------------------------
# beam parameters
$gamma = 268.0;          # relativistic gamma
$dp    = 0.0;            # momentum deviation

# tracking parameters for rays
$nray       =  1;          # rays number
if (not defined $nray_max) {
    $nray_max = 1;         # max number of rays (15) 1=diagonal
    }
$npair      = 10;          # number of particle pairs
$lyap_flag =  0;           # if Lyapunov flag not set, particles are equally distributed
$rmin  = 0.0001;
if (not defined $rmax) { $rmax = 0.0015; }

# ray preparation -----------------------------------------------------

$npart = $npair*2;
$hpi   = atan2(1,1)*2;
$phi   = $nray*$hpi/($nray_max + 1);

my $i;
my @array = ();

if($lyap_flag == 1) {

# fill in cooordinates
   for($i = 1; $i < ($npair+1); $i++) {

  if($npair > 1) {
    $r = $rmin + ($i - 1)*($rmax - $rmin)/($npair - 1);
} else {
    $r = $rmin;
}# particle one in the pair

$ipart = 2*($i-1);
$xini =  $r*cos($phi);
$yini =  $r*sin($phi);
push @array, [$xini, 0.0, $yini, 0.0, 0.0, $dp];
```

```
# particle two in the pair

$xposp2 =  $r*cos($phi);          # p2 shifted wrt q1
$xposp2 += 1e-10;                 # small distance for Lyapunov analysis
$yposp2 =  $r*sin($phi);
push @array, [$xposp2, 0.0, $yposp2, 0.0, 0.0, $dp];
    }
} else {
# fill in cooordinates
  for($i = 1; $i < ($npart+1); $i++) {

  if($npair > 1) {
    $r = $rmin + ($i - 1)*($rmax - $rmin)/($npart - 1);
} else {
    $r = $rmin;
}

$xini =  $r*cos($phi);
$yini =  $r*sin($phi);
push @array, [$xini, 0.0, $yini, 0.0, 0.0, $dp];
    }
} # end if lyap_flag

$numparts = $npart;

$shell->start(@array);
$shell->run("print" => ">$file8", "turns" => $numturns, "step" => $numsteps);
1;
```

Unified Accelerator Libraries *

Nikolay Malitsky and Richard Talman

Laboratory of Nuclear Studies, Cornell University, Ithaca, NY 14853

Abstract

A "Universal Accelerator Libraries" (UAL) environment is described. Its purpose is to facilitate program modularity and inter-program and inter-process communication among heterogeneous programs. The goal ultimately is to facilitate model-based control of accelerators.

The performance of an accelerator depends to a large extent on the quality of the theoretical algorithms built into its control system. Evolution of new computing technologies and accumulating accelerator experience supply the necessary conditions for the next phases of integration—simulation facilities and (later) artificial-intelligence based control systems. However, several intermediate steps must be taken first, the foremost being the design and implementation of a framework for distributed accelerator programs.

There is a complementarity between the inherent heterogeneity of accelerator problems and personalities and the homogeneity that is essential for sharing codes. The Unified Accelerator Libraries (UAL) aim to create an environment in which diverse accelerator algorithms can be used together, connected to arbitrary data sources and integrated with other computer software. Each accelerator program is considered as a separate self-contained class, that may have its own internal organization and methods. Connection with the UAL is by common data objects (element attributes, beam parameters, *etc.*) On the one hand, this provides flexibility for each physicist to develop his or her own ideas and algorithms. On the other hand, one

*This manuscript has been authored under contract number DE-FG02-95ER40920 with the U.S. Department of Energy.

can consider the UAL to be a tool for comparing different approaches and selecting the most effective ones.

The *description* of accelerator structures is a key part of accelerator programs. There are many elements of different physical types, each having many attributes, all organized in a more or less hierarchical fashion. This has resulted in the creation of diverse lattice description languages and formats, each requiring a parser to perform the conversion from human readable ASCII files to the internal data structures needed to harness powerful computation algorithms (often written in FORTRAN or C in the past.) A disadvantage of such a "proprietary", "embedded" parser is that it makes the code "closed", impeding its re-use and extension. One result is that small, *ad hoc*, special-purpose, codes proliferate, often in the form of hard-to-support, "unofficial" modifications of existing programs. The development time of such codes/versions is typically short and, regrettably, so usually is their useful lifetime. In this way implementation of proprietary algorithms without an architectural focus impedes their integration with other software, leaving similar problems for subsequent developers.

The promise (some might say triumph) of object-oriented technology is to overcome this problem by enabling the replacement of the proprietary input format by one of the object-oriented programming languages. This can exploit the similarity of object-oriented methodologies to mathematical (algebraic) abstractions to make the application programming interface (API) simpler and more "physical" than present, code-specific, grammars. In principle (and in practice) the accelerator can be described using only overloaded assignment and arithmetic operators. "Addition" enables one to construct physical elements from some simple "bricks" (element attributes), and these elements can be concatenated into beam lines by "multiplication". Adoption of a standard programming language with its well-defined, highly disciplined grammar and lexical elements brings many benefits: support of major (conditional, iterative, *etc.*) statements and expressions, extensions in many different areas, interfacing to popular freeware and commercial applications (GUI, databases, communication, *etc.*), stability and portability, development and maintenance support by thousands of users and vendors, and natural connection to new tendencies and technologies in computer science.

Because of their common nature, the choice among different object-oriented programming languages is not essential; it depends on the particular task and environment. We feel the software should be built using two languages, with the core of algorithm's (developer's world) written in compiled languages while their invocation (user's world) is controlled by an interpreted language. In summary, since the latter constitutes the "input language" for accelerator programs, that language should be a standard, object-oriented, extensible,

interpreted language. At this time, of the three candidates having similar functionality, Python, Scheme, and PERL, we have chosen PERL because of its popularity and better (for us) syntax.

For accelerator modeling the API is conventionally divided into two main parts, Accelerator Description and Actions. Certainly, the Accelerator Description must not be determined by the particular interpreted language. It should be based on the same object model as the one recognized by the core algorithms and, as such, it should be considered to be a "wrapper" around this common object-oriented accelerator description. From the software developer's perspective, the accelerator is characterized by two contradictory peculiarities. On the one hand, the accelerator is built from ideal standard elements with well-defined design principles. On the other hand, the resulting model is to be used as a tool for predicting and improving performance in the presence of non-ideal, perhaps not previously classified, devices. (This complementarity has contributed in the previously mentioned proliferation of special purpose codes.) We support an appreciable range of such possibilities by dividing all accelerator elements into three conventional categories, MAD, COSY (describable by Taylor series map), and WILD (a catch-all for special-purpose elements such as internal targets, electron cooling inserts, *etc.*)

The second main API feature, Actions, have traditionally consisted of an even more heterogeneous collections of commands and directives; again this is due to the variety of physical missions. The power of a modern computer language (in our case PERL) then makes it possible to merge these diverse descriptions seamlessly into a model is which beam dynamics in different sectors of the same lattice is described by different computation algorithms. A typical application might describe most of the lattice by TEAPOT conventional algorithms and one or more sectors by truncated Taylor series maps or WILD elements. As well as supporting this modularity, PERL automatically supplies the conditions for studying "dynamic" accelerator performance in which some of the optical parameters are varied, either adiabatically or impulsively. This permits the modeling, and eventually control, of effects like energy ramping, transition crossing, tune modulation, resonant extraction, and so on.

At this time the UAL joins three object-oriented accelerator programs: Platform for Accelerator Codes (PAC), Thin Element Program for Optics and Tracking (TEAPOT), and Numerical Library for Differential Algebra (ZLIB). The PAC is a collection of common data objects that can be shared, exchanged, or converted by other codes and processes. The central place in this architecture is taken by an accelerator object model, the Standard Machine Format (SMF), that is an extension of the Standard Input Format (SIF)[1]. In comparison with the design[2] SMF has been enhanced to speed

up the access to element attributes. Its source code and a functioning example (the Cornell Electron Storage Ring, CESR) are available now from the UAL site (http://w4.lns.cornell.edu/~tpot). Two other parts, the TEAPOT tracking engine and the ZLIB library, have been used recently and successfully at Brookhaven National Laboratory, for evaluating performance with a helical magnet insert[3] [4] and in investigations of tune modulation[5].

The next stage of UAL development, integration of TEAPOT matching and correction algorithms, requires the specification of new PAC classes (Adjusters, Buses, and Detectors) as well as the choice of an underlying communication facility. Because the direct integration of high level application algorithms with a particular communication layer reduces portability and runs the risk of obsolescence, we plan to employ the Common Object Request Broker Architecture (CORBA), an industry standard for distributed objects. CORBA provides an abstract specification that can be used on top of virtually any communication layer; it isolates the application software from the communication system, hiding heterogeneous features such as low level protocols, server-host hardware, operating systems, *etc.* Also this allows developers to connect applications written in different computer languages (C++, Java, *etc.*) The similarity of our architecture to CORBA-based frameworks such as Data Interchange and Synergistic Collateral Study (DISCUS)[6] makes straightforward the connection of UAL with distributed control systems.

References

[1] D.C. Carey and F.C. Iselin, *Standard Input Language for Particle Beam and Accelerator Computer Programs*, Snowmass, Colorado, 1984

[2] N. Malitsky, R. Talman, *et.al.*, *A Proposed Flat Yet Hierarchical Accelerator Lattice Object Model*, BNL RHIC/AP/74, 1995

[3] N. Malitsky, *Application of a differential algebra approach to a RHIC helical dipole*, BNL RHIC/AP/51, 1994

[4] W. Fischer, *Preliminary Tracking Results with Helical Magnets in RHIC*, BNL RHIC/AP/110, 1996

[5] W. Fischer and T.Satogata, *A Simulation Study on Tune Modulation Effects in RHIC*, BNL RHIC/AP/109, 1996

[6] T.J. Mowbray and R. Zahavi, *The Essential CORBA*, John Wiley & Sons, Inc., 1995