Electric Upgrade of Unified Accelerator Libraries (UAL) Status Report and Plan

John Talman

January 8, 2012

Abstract

In a nutshell, simulating the design and performance of "electric element" lattices is called for. A code UAL/ETEAPOT is being developed for this purpose. This new code is required because of the implicit assumption in existing codes that particle kinetic energies change only in RF cavities (kinetic energy conserved, typically). Of course, this is associated with the more common almost exclusively magnetic element lattices (magnetic field does no work). ETEAPOT stands for Electric TEAPOT. UAL/TEAPOT is the existing code for magnetic elements. In an electric ring it is the *total* energy, mechanical plus potential that is conserved.

Due to the changes in potential energy associated with electric lattice elements, the mechanical energy, $\gamma m_p c^2$, changes on the same fast "betatron" time scale as do horizontal and vertical displacements. This breaks the customary paradigm (on which TEAPOT is based) in which energy oscillations are slow while betatron oscillations are fast.

For long term symplectic tracking a closed form, analytic, fully relativistic, orbit formalism is likely required. The inverse square law, "Coulomb's law", central force field is the only electric field we know of for which an analytic solution is available. The most convenient form, for our purposes, is due to Muñoz and Pavic[2]. Thin quadrupole-like "kicks" are used to correct for the fact that the optimal radial dependence of the electric field for the EDM experiment is *not* inverse square law.

Preliminary results/benchmarking of the prototype ETEAPOT code for an idealized lattice encourage the notion that there is agreement to better than one percent level among the code, analytic formulas and an entirely independent, linearized transfer matrix code.

Thus this document is a status report on the upgrade of the prototype ETEAPOT code to full UAL status.

Contents

1 Introduction and Document Overview

This is an offshoot/continuation of what started out as Richard Talmans UAL/ETEAPOT Lecture 4 and its associated document ([1], now slightly modified by John Talman). It is a "working document" intended to facilitate communication between me and others. Extensive references are enumerated in ([1]).

UAL/ETEAPOT is intended to eventually be compiled into software libraries/binaries ("server side") that a typical user ("client") uses/accesses without having to know its details. This allows him/her to concentrate on the properties of specific accelerators

("sxf file" e.g. $UAL/examples/ETEAPOT/data/pre-E_pEDm.sxf)

This includes orbital propagation, lattice functions, and eventually spin behavior. This code typically resides in "$UAL/codes/...".

The client typically writes his own C++ program that accesses and sequences this library functionality. This code typically resides in "$UAL/examples/...".

This development effort encompasses both sides (server, and client). Thus both physics, and computer code and organization must be addressed.

This report focuses on ("client side") $UAL/examples/ETEAPOT/evolver.cc and its "server side" calls.

The next section ("Main Code Flow") depicts/annotates its main interaction with the server side code, whose main sequencing class is

$UAL/codes/UAL/src/UAL/APF/AcceleratorPropagator.cc

This is the first relevant server side class (for my current purposes) called from, or used by, evolver. I write the name of the file with current programmatic control in brackets. I usually write the approximate line number of control. These numbers correspond closely to those in the google codes repository

http://code.google.com/p/ual/source/browse/trunk

but may differ slightly due to my debugging info. A key construct in this interaction is the Accelerator Propagator Description Format ("apdf"). A relevant example is shown in that (following) section.

The subsequent sections (Plan, ...) expand on this portrayal. A lot of here deferred complexity (e.g. "Lattice Parsing and Populating") is touched on in Appendices.

Thus this document is also a tentative plan on how to proceed.

2 Main Code Flow

```
[$UAL/examples/ETEAPOT/evolver.cc]
148 ap->propagate(map2);
----------------------------------------------
 [$UAL/codes/UAL/src/UAL/APF/AcceleratorPropagator.cc]
 61 void UAL::AcceleratorPropagator::propagate(UAL::Probe& probe)
 64 m_rootnode.propagate(probe);
----------------------------------------------
  [$UAL/codes/UAL/src/UAL/APF/PropagatorSequence.cc]
```

61 void UAL::PropagatorSequence::propagate(UAL::Probe& bunch)
70 (*i)->propagate(bunch);
((*i) is a pointer to a PropagatorNode
$UAL/codes/UAL/src/UAL/APF/PropagatorNode.hh
43 virtual void propagate(UAL::Probe& probe) = 0;)

[$UAL/codes/TEAPOT/src/TEAPOT/Integrator/DipoleDaIntegrator.cc]
(class DipoleDaIntegrator : public BasicDaIntegrator
class BasicDaIntegrator : public TEAPOT::BasicPropagator
class BasicPropagator : public UAL::PropagatorNode)
56 void TEAPOT::DipoleDaIntegrator::propagate(UAL::Probe& probe)
59 PacTMap& map = static_cast<PacTMap&>(probe);
...
70 ZLIB::VTps& p = map.operator*();
"Inline Print" p In"
"Propagation Code"
"Inline Print p Out"

I'll be looking to modify/supplant the "Propagation Code".

3 Accelerator Propagator Description Format

Similarly, the other propagators, e.g.
$UAL/codes/TEAPOT/src/TEAPOT/Integrator/MltDaIntegrator.hh
are "PropagatorNodes"). Thus,
class MltDaIntegrator : public BasicDaIntegrator ...

This is all encapsulated in "the apdf file" e.g.

```
<apdf>
  <propagator id="Evolver" accelerator="ring">
    <create>
      <link algorithm="TEAPOT::DriftDaIntegrator" types="Default"/>
      <link algorithm="TEAPOT::DriftDaIntegrator" types="Drift"/>
      <link algorithm="TEAPOT::DipoleDaIntegrator" types="Sbend"/>
      <link algorithm="TEAPOT::MltDaIntegrator" types="Quadrupole"/>
      <link algorithm="TEAPOT::MltDaIntegrator" types="Sextupole"/>
    </create>
  </propagator>
</apdf>
```

I concentrate on, or start from, TEAPOT::DipoleDaIntegrator here.

4 Plan

The prior two sections are intended as reference material for dialog, and my subsequent work. They are not intended to be self explanatory.

With this machinery in place, it is hoped that formula transcription will be expedited. This is anticipated as a main point of discussion.

Thus [1] will tentatively become authoritative, and $UAL/codes/ETEAPOT will tentatively become reliable.

The next section, ”Results”, shows a typical debugging output, enabled by the inline print.

5 ”Results”

($UAL/examples/ETEAPOT):
./evolver ./data/pre-E_pEDm.sxf | less

```
JDT file DipoleDaIntegrator.cc line 58 void TEAPOT::DipoleDaIntegrator::propagate(UAL::Probe& probe)
map.order() 1
map.size() 6
ZLIB::VTps : size = 6 (dimension = 6  order = 1 )

    0  0.0000000000000e+00  0.0000000000000e+00  0.0000000000000e+00  0.0000000000000e+00 -3.3362201889986e-16  0.0000000000000e+00   0  0  0  0  0  0
    1  1.0023594742931e+00  1.2558315418738e-03  0.0000000000000e+00  0.0000000000000e+00  0.0000000000000e+00  0.0000000000000e+00   1  0  0  0  0  0
    2  1.5025003539743e+00  9.9952852547515e-01  0.0000000000000e+00  0.0000000000000e+00  0.0000000000000e+00  0.0000000000000e+00   0  1  0  0  0  0
    3  0.0000000000000e+00  0.0000000000000e+00  9.9764052496588e-01 -1.2573136481542e-03  0.0000000000000e+00  0.0000000000000e+00   0  0  1  0  0  0
    4  0.0000000000000e+00  0.0000000000000e+00  1.5024996460255e+00  1.0004714741543e+00  0.0000000000000e+00  0.0000000000000e+00   0  0  0  1  0  0
    5  0.0000000000000e+00  0.0000000000000e+00  0.0000000000000e+00  0.0000000000000e+00  1.0000000000000e+00  0.0000000000000e+00   0  0  0  0  1  0
    6  0.0000000000000e+00  0.0000000000000e+00  0.0000000000000e+00  0.0000000000000e+00  2.6937532221436e+00  1.0000000000000e+00   0  0  0  0  0  1
JDT file /home/ualusr2011/JDT-UAL691/ual1/codes/TEAPOT/src/TEAPOT/Integrator/DipoleAlgorithm.icc line 29 template<class Coordinate, class Coordinates>
   void TEAPOT::DipoleAlgorithm<Coordinate, Coordinates>::passBend( ...) data.m_ir 0
JDT file /home/ualusr2011/JDT-UAL691/ual1/codes/TEAPOT/src/TEAPOT/Integrator/DipoleAlgorithm.icc line 31 template<class Coordinate, class Coordinates>
   void TEAPOT::DipoleAlgorithm<Coordinate, Coordinates>::passBend( ...) data.m_ir 0 \\

JDT file DipoleDaIntegrator.cc line 58 void TEAPOT::DipoleDaIntegrator::propagate(UAL::Probe& probe)
map.order() 1
map.size() 6
ZLIB::VTps : size = 6 (dimension = 6  order = 1 )

    0  2.7670014411292e-17 -2.7670014435503e-23  0.0000000000000e+00  0.0000000000000e+00 -1.1123749968804e-15  0.0000000000000e+00   0  0  0  0  0  0
    1  9.9430344370693e-01 -5.8487245077428e-03  0.0000000000000e+00  0.0000000000000e+00 -2.6424904019837e-01  0.0000000000000e+00   1  0  0  0  0  0
    2  4.9662042217019e+00  9.7651682270978e-01  0.0000000000000e+00  0.0000000000000e+00 -8.5583944148937e-01  0.0000000000000e+00   0  1  0  0  0  0
    3  0.0000000000000e+00  0.0000000000000e+00  9.9323275248348e-01 -1.2563204141453e-03  0.0000000000000e+00  0.0000000000000e+00   0  0  1  0  0  0
    4  0.0000000000000e+00  0.0000000000000e+00  5.0098596748253e+00  1.0004764830135e+00  0.0000000000000e+00  0.0000000000000e+00   0  0  0  1  0  0
    5  0.0000000000000e+00  0.0000000000000e+00  0.0000000000000e+00  0.0000000000000e+00  1.0000000000000e+00  0.0000000000000e+00   0  0  0  0  1  0
    6  4.6135059508075e-01  2.6304920225475e-01  0.0000000000000e+00  0.0000000000000e+00  8.9789512217552e+00  1.0000000000000e+00   0  0  0  0  0  1
```

A Lattice Parsing and Populating

For the record, the actual accelerator lattice is ”made available” via lines

```
116 std::string xmlFile = "./data/evolver.apdf";
117
118 UAL::APDF_Builder apBuilder;
119 apBuilder.setBeamAttributes(ba);
120
121 UAL::AcceleratorPropagator* ap = apBuilder.parse(xmlFile);
```

in $UAL/examples/ETEAPOT/evolver.cc.

This allows the lattice to be traversed via e.g.

```
131 UAL::PropagatorSequence& apSeq = ap->getRootNode();
132
133 int counter = 0;
134 std::list<UAL::PropagatorNodePtr>::iterator it;
135 for(it = apSeq.begin(); it != apSeq.end(); it++)
```

```
std::cout << counter++ << ” ” << (*it)->getType() << std::endl;

```

This displays/echoes the sequential elements of the current sxf file, although it adds in any implicit ”drifts”.

This functionality can be used server side e.g.

```
void UAL::PropagatorSequence::propagate(UAL::Probe& bunch)

std::cout << ”JDT file ” << __FILE__ << ” line ” << __LINE__ << ” void UAL::PropagatorSequence::propagate(UAL::Probe& bunch) n”;
if(m_nodes.empty()) return;

std::list<UAL::PropagatorNodePtr>::iterator i;
int counter = 0;
for(i = m_nodes.begin(); i != m_nodes.end(); i++)
// std::cout << ”PropagatorSequence node id = ” << counter++ << std::endl;
std::cout << counter++ << ” JDTgetType ” << (*i)->getType() << std::endl;
(*i)->propagate(bunch);

```

which is glossed over above.

B Code Trickery

Modifying files

$UAL/codes/ZLIB/src/ZLIB/Tps/Space.hh

(public) static GlobalTable* _table;

and

$UAL/codes/ZLIB/src/ZLIB/Tps/VTps.hh

(public) Tps* _vtps;

allows the ”inline” print mentioned above

```
int dim   = p.dimension();
int order = p.order();
int size  = p.size();

char s[80];

std::cout << "ZLIB::VTps : size = " << size ;
std::cout << " (dimension = " << dim ;
std::cout << "  order = " << order << " )\n\n";

int nm1 = 0, nm2;
```

```
for(int io = -1; io < order; io++){
  if(io > -1) nm1 = p.nmo(io);
  nm2 = p.nmo(io + 1);
  for(int i = nm1; i < nm2; i++){
      sprintf(s, "%5d ", i );
      std::cout << s ;
      int j;
      for(j=0; j < size; j++)
      {
        if((int) p._vtps[j].order() > io) sprintf(s, "% 19.13e ", p.vtps(j
        else sprintf(s, "% 19.13e ", 0.0);
        std::cout << s ;
      }
      for(j=1; j <= dim; j++)
      {
         sprintf(s, "%3d", p._table->jv[j][i+1]);
         std::cout << s ;
      }
      std::cout << "\n";
  }
}
```

(lifted from $UAL/codes/ZLIB/src/ZLIB/Tps/VTps.cc
275 void ZLIB::VTps::write(const char* f)
288 ostream& ZLIB::operator<<(ostream& out, const ZLIB::VTps& zvs)).

C SXF File

$UAL/examples/ETEAPOT/data/pre-E_pEDm.sxf

```
// SXF version 2.0
ring sequence
 {
     marker {
     marker {
     quadrupole {
     sextupole {
     sextupole {
     quadrupole {
     sbend {
     sextupole {
     quadrupole {
     quadrupole {
     sextupole {
     sbend {
     quadrupole {
      sextupole {
      ...
 endsequence at = 200
}
// SXF end
```

D Miscellany

`$UAL/examples/ETEAPOT/evolver.cc`

```
"split" = 0 (simple element, treated as two half elements)
  shell.addSplit(UAL::Args() << UAL::Arg("lattice", "ring") << UAL::Arg("types", "Sbend")      << UAL::Arg("ir", split));
  shell.addSplit(UAL::Args() << UAL::Arg("lattice", "ring") << UAL::Arg("types", "Quadrupole") << UAL::Arg("ir", split));
  shell.addSplit(UAL::Args() << UAL::Arg("lattice", "ring") << UAL::Arg("types", "Sextupole")  << UAL::Arg("ir", split));

"size" = 6
"order" = 1
  PacTMap map2(6);
  map2.setEnergy(ba.getEnergy());
  map2.mltOrder(1);
  map2.write("evolver0.map");

//map2.propagate();
  ap->propagate(map2);

  map2.write("evolver.map");
```

References

[1] R. Talman and J. Talman, *Juelich Lecture Notes: 4. UAL, ETEAPOT and Long Term Symplectic Tracking for Electric Lattices,* Personal Communication

[2] G. Muñoz and I. Pavic, *A Hamilton-like vector for the special-relativistic Coulomb problem,* Eur. J. Phys. **27**, 1007-1018, 2006

[3] N. Malitsky and R. Talman, *Unified Accelerator Libraries,* Proceedings of Conference on Accelerator Software, Williamsburg, VA, 1996, *The Framework of Unified Accelerator Libraries,* ICAP98, Monterey, 1998