A MODULAR ON-LINE SIMULATOR FOR MODEL REFERENCE CONTROL OF CHARGED PARTICLE BEAMS*

C.K. Allen#, C.A. McChesney, LANL, Los Alamos, NM, USA
N.D. Pattengale, Sandia National Laboratory, Albuquerque, NM, USA
C.P. Chu, J.D. Galambos, W.-D. Klotz, T.A. Pelaia, A. Shishlo, ORNL, Oak Ridge, TN, USA

Abstract

We have implemented a particle beam simulation engine based on modern software engineering principles with intent that it be a convenient model reference for high-level control applications. The simulator is an autonomous subsystem of the high-level application framework XAL currently under development for the Spallation Neutron Source (SNS). It supports multiple simulation techniques (i.e., single particle, multi-particle, envelope, etc.), automatically synchronizes with operating accelerator hardware, and also supports off-line design studies. Moreover, since it is implemented using modern techniques in the Java language, it is portable across operating platforms, is maintainable, and upgradeable.

INTRODUCTION

To support the operation of the SNS accelerator we have built a development framework for high-level control applications called XAL (for a description of XAL see [2]). The framework includes a simulation engine that we call the XAL model subsystem. This subsystem works in conjunction with the application framework, or as a stand-alone particle beam simulator. XAL contains a utility for automatically generating modeling lattices and synchronizing them with the operating machine. Here we outline the architecture of the model subsystem and the mathematical models upon which it is based.

DYNAMICS

In all simulations we parameterize phase space using homogeneous coordinates in $\Re^6\times\{1\}$. Letting $\mathbf{z}$ denote a point in phase space it has the representation

$$\mathbf{z} \equiv \begin{pmatrix} x & x' & y & y' & z & z' & 1 \end{pmatrix}^T \tag{1}$$

where the prime indicates differentiation with respect to the design path length parameter s. Note that we use (z,z') as the longitudinal phase coordinates rather than $(z,\Delta p/p)$. Mathematicians typically use homogeneous coordinates to parameterize the real projective spaces $\Re P^n$; they are also widely used in computer graphics for three-dimensional rendering because translation, rotation, and scaling can all be performed by matrix multiplication.

Single Particle Simulation

In single-particle simulations we propagate the phase space coordinates $\mathbf{z}$ of the particle. Each beamline element n is obligated to provide a transfer map $\mathcal{M}_\mathbf{n}$ (embodied by the `PhaseMap` class) that represents the action of the element. Note that the characteristics of this map typically depend upon the parameters of the beam being propagated. In any case, the particle coordinates are propagated according to the transfer equation

$$\mathbf{z}_{n+1} = \mathcal{M}_\mathbf{n}(\mathbf{z}_n). \tag{2}$$

Although provisions for high-order dynamics are included in the `PhaseMap` class, currently most of the beamline elements simply provide a transfer matrix to represent its dynamics. This transfer matrix $\mathbf{M}_\mathbf{n}\in\Re^{7\times7}$ is just the linear part of $\mathcal{M}_\mathbf{n}$, or $\mathbf{M}_\mathbf{n}=\partial\mathcal{M}_\mathbf{n}/\partial\mathbf{z}$ and we have

$$\mathbf{z}_{\mathbf{n+1}} = \mathbf{M}_\mathbf{n}\mathbf{z}_\mathbf{n}. \tag{3}$$

Note that because we employ homogeneous coordinates it is still possible to perform translations, such as those produced by steering magnets, using a transfer matrix. Such matrices have the form

$$\mathbf{M}_\mathbf{n} = \begin{pmatrix} m_{xx} & m_{xx'} & \cdots & \Delta x \\ m_{x'x} & m_{x'x'} & & \Delta x' \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}, \tag{4}$$

where Δx, $\Delta x'$, Δy, … are the translations along the respective coordinate axes. This fact is especially useful when performing rms envelope simulations where only the linear part of the transfer map is used.

RMS Envelope Simulation

For rms envelope simulations we propagate the (homogeneous) correlation matrix $\boldsymbol{\sigma}\in\Re^{7\times7}$ defined by

$$\boldsymbol{\sigma} \equiv \left\langle \mathbf{z}\mathbf{z}^T \right\rangle = \begin{pmatrix} \langle x^2 \rangle & \langle xx' \rangle & \cdots & \langle x \rangle \\ \langle xx' \rangle & \langle x'^2 \rangle & & \langle x' \rangle \\ \vdots & & \ddots & \vdots \\ \langle x \rangle & \langle x' \rangle & \cdots & 1 \end{pmatrix} \tag{5}$$

where $\langle\cdot\rangle$ is the phase space moment operator with respect to the beam distribution. Substituting Eq. (3) into the above then unwinding the definitions forms the state transfer equation for rms envelopes

$$\boldsymbol{\sigma}_{\mathbf{n+1}} = \left\langle (\mathbf{M}_\mathbf{n}\mathbf{z}_\mathbf{n})(\mathbf{M}_\mathbf{n}\mathbf{z}_\mathbf{n})^T \right\rangle = \mathbf{M}_\mathbf{n}\boldsymbol{\sigma}_\mathbf{n}\mathbf{M}_\mathbf{n}^T. \tag{6}$$

Note that these operations do not commute for the full transfer map $\mathcal{M}_\mathbf{n}$. Thus, to account for space charge we must linearize the self electric fields of the beam. To this end we employ a weighted linear regression of the true fields. Considering the x-plane, we proceed by assuming a self electric field E_x of the form

$$E_x \approx a_0 + a_1 x. \tag{7}$$

* Work supported by the US Department of Energy
# ckallen@lanl.gov

Weighting the regression with the beam distribution itself yields the following approximation:

$$E_x \approx \frac{\langle xE_x \rangle}{\langle x^2 \rangle}\left(x - \langle x \rangle\right), \tag{8}$$

(the moment $\langle E_x \rangle$ is zero by Newton's third law). For beams with ellipsoidal symmetry, that is, having distributions $f(\mathbf{z})$ of the form

$$f(\mathbf{z}) = f\left(\mathbf{z}^T \boldsymbol{\sigma}^{-1} \mathbf{z}\right), \tag{9}$$

the moment $\langle xE_x \rangle$ can be computed analytically in terms of elliptic integrals whose arguments are elements of the matrix $\boldsymbol{\sigma}$. These expressions can be found in the literature [1]. Note that Eq. (8) will yield a transfer matrix form for space charge effects that may be applied in the same fashion as Eq. (6).

Ensemble Simulations

The state $\mathbf{E}$ of a particle ensemble is represented as a collection of phase space coordinates

$$\mathbf{E} = \left\{ \mathbf{z}_\alpha \in \Re^6 \times \{1\} \,|\, \alpha \in I_\mathbf{E} \right\}, \tag{10}$$

where $I_\mathbf{E}$ is an index set. The transfer equation for this state object is

$$\mathbf{E}_{n+1} = \mathcal{M}_\mathbf{n}(\mathbf{E}_n) \equiv \{ \mathcal{M}_\mathbf{n}(\mathbf{z}_\alpha) | \alpha \in I_{\mathbf{En}} \}. \tag{11}$$

Although this state can be represented as a point in the Cartesian product of phase spaces $(\Re^6 \times \{1\})^N$ where $N=|\mathbf{E}|$, it is more constructive to think of it as a set with collective properties. This process is most easily captured as a software object, which we have done with a class `Ensemble`. Ensemble objects are responsible for calculating self-fields and other collective properties. There are many ways to compute these fields, including grids, finite elements, and direct summation. These computation techniques are currently under development. We point out that the software is so designed such that specific self-field calculations may be swapped at run time. Once the self electric fields are computed, the ensemble state can be advanced according to the equations of motions. Considering the x phase plane, we have the approximation

$$\Delta p_x \approx \frac{q}{\gamma^2} E_x \Delta t\,, \tag{12}$$

where the relativistic factor γ accounts for the collective magnetic fields and Δt is the time step. From the above we infer

$$\Delta x' = \frac{\Delta p_x}{p} \approx \frac{1}{\beta\gamma mc} \frac{q}{\gamma^2} E_x \frac{1}{\beta c} \Delta s\,, \tag{13}$$

where m is the particle mass and β is the synchronous velocity normalized to the speed of light c, and Δs is the distance traveled along the design trajectory during Δt.

SOFTWARE ARCHITECTURE

We are able to support the various particle beam simulation techniques due to a novel approach in software architecture. By employing a variant of the Element/Algorithm/Probe design pattern introduced by Malitsky and Talman [3], we separate the machine representation from the beam representation and the dynamics calculations. In this scheme, systems for representing the accelerator, the beam, and the beam dynamics are decomposed into separate software components that communicate through the well-defined

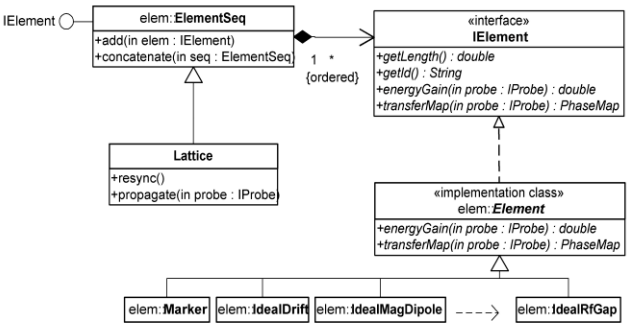

Figure 1: machine representation component

software interfaces `IElement`, `IProbe`, and `IAlgorithm`, respectively.

Machine Representation

A major effort in accelerator simulation is simply representing the machine. By decoupling the machine representation from the machine's action on the beam, the representation then can be used to support any number of simulation techniques. Figure 1 is a UML structure diagram outlining the machine representation component of the simulator. At the heart of this component is the `IElement` interface, which is exposed by any object representing a modeling element of the machine. Note that we provide the (abstract) implementation class, `Element`, which provides a variety of common functions that modeling element must accommodate in support of the `IElement` interface. Most objects representing beamline elements are derived from this convenience base class. In the figure we see derived classes must provide energy gains and transfer maps specific to the modeling element, done by implementing the abstract methods `energyGain()` and `transferMap()`.

Shown in Figure 1 is the aggregation `ElementSeq`, which is an ordered sequence of `IElement` objects. It, too, exposes the `IElement` interface, since it may be considered a composite modeling element. The values obtained here, however, would be the aggregate results of all members in the sequence. We also see that the `Lattice` object is just a specialized sequence. Much of the `Lattice` class function is conceptual, however, it also provides access to the important mechanisms of probe propagation and online synchronization. Through the method `propagate()` the `Lattice` object coordinates the operation of the machine representation, beam representation, and beam dynamics. The online synchronization mechanism, which automatically synchronizes the Lattice object to the parameters of the operating hardware, is accessed via the method `resync()`.

Beam Representation

Figure 2 depicts the basic architecture of the beam representation component. The interface to this component is called `IProbe`, as seen in the figure. Note that the interface for the dynamics subsystem, `IAlgorithm`, is associated with `IProbe`. Thus, each probe object, representing some aspect of a charged particle beam, also specifies its own dynamics. There may be several types of dynamics calculations available for any particular probe (e.g., linear, third-order, etc.).

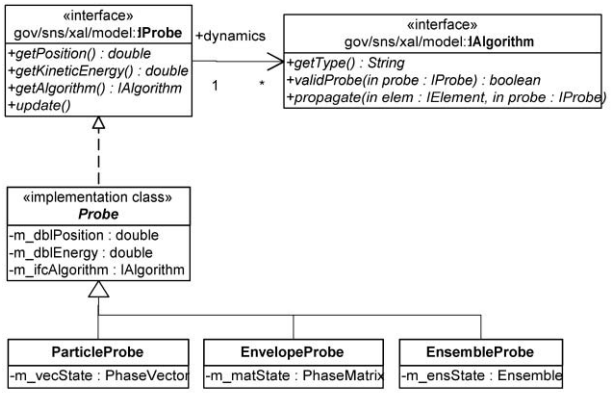

Figure 2: architecture of beam representation component

In Figure 2 we see that the (abstract) implementation class `Probe` is provided to assist in the implementation of particular probes. It provides necessary bookkeeping as well as access to trajectory objects (not shown), which store probe histories along the lattice. The maintenance of actual probe states is left to the particular probe implementation. In the figure we see that the state of a `ParticleProbe` is the vector of particle phase space coordinates, the state of an `EnvelopeProbe` is the correlation matrix of moments up to second order, and the state of an `EnsembleProbe` is an ensemble object.

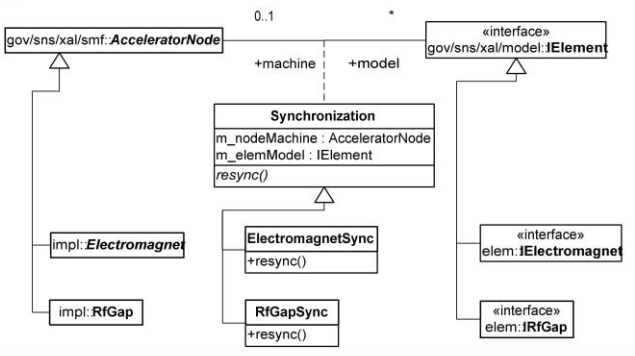

Figure 3: synchronization mechanism

Machine Synchronization

Synchronization with the operating hardware is accomplished through a subsystem based on the (abstract) association class `Synchronization`. It supports communication between the XAL model system and the XAL SMF (Standard Machine Format) system, which otherwise have no knowledge of one another.

Shown in Figure 3, these synchronization objects understand the vernacular of both XAL subsystems. The children of the `AcceleratorNode` class, belonging to SMF, represent actual accelerator components and supports communication with these devices. The interfaces derived from `IElement` represent modeling elements. Note that is a one/none-to-many association between accelerator devices and their modeling counterparts. This condition is necessary because, for example, "drifts" are not controllable devices of the accelerator and actual devices may require several modeling elements to represent (e.g., quadrupoles with trim windings for steering, etc.). Referring to the figure we see that each type of accelerator device requires the implementation of a particular synchronization class that understands how to communicate with both the device and the modeling element. Once implemented, any synchronization request is carried out by invoking the abstract method `resync()` in the base class. Thus, to remain synchronized with the operating machine the `Lattice` object maintains a set of synchronization objects, invoking `resync()` on each whenever required.

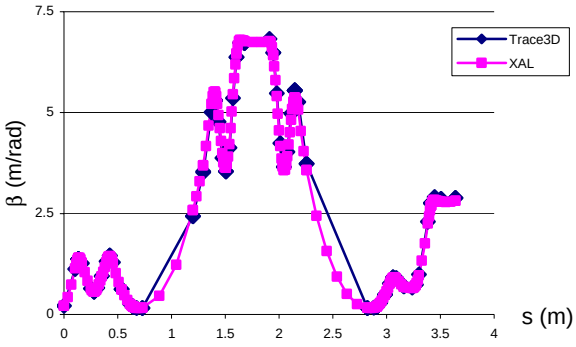

Figure 4: transverse beta simulations of the SNS MEBT

VERIFICATION

To verify the operation of the XAL modeling subsystem we have compared the results for single particle and envelope simulations against those of Trace3D. Figure 4 shows such a comparison for a simulation of the SNS Medium Energy Transport (MEBT) system. The figure plots the Twiss parameter beta in the (horizontal) transverse plane computed by each code. There we can see that the results are essentially equivalent.

REFERENCES

[1] C.K. Allen and N.D. Pattengale, LANL Internal Report LA-UR-02-4979.

[2] C.M. Chu, *et. al.*, "Applications Programming Structure and Physics Applications", these proceedings.

[3] N. Malitsky and R. Talman, "The Framework of Unified Accelerator Libraries", ICAP 1998.