A NOVEL ONLINE SIMULATOR FOR APPLICATIONS REQUIRING A MODEL REFERENCE*

C.K. Allen†, C.A. McChesney, LANL, Los Alamos, NM, 87545, USA
C.M. Chu, J. Galambos, W.-D. Klotz, T. Pelaia, A. Shislo, ORNL, Oak Ridge, TN, 37831

Abstract

In developing the control system for the Spallation Neutron Source (SNS) we have built an online particle-beam simulator. It is an independent subsystem of the high-level applications programming framework called XAL. The simulator architecture is based upon the Element-Algorithm-Probe design pattern where the same simulation engine can support various simulation strategies (e.g. single-particle and envelope simulations). Moreover, the system automatically synchronizes to the machine configuration and operating conditions; thus, no support for external off-line modeling is necessary. As is XAL, the simulator is implemented in Java and stores persistent data (lattices, trajectories, etc.) in XML format.

INTRODUCTION

The Spallation Neutron Source (SNS) is using an application framework for high-level control application development called XAL [1]. XAL is a Java-based, device-oriented framework designed to expedite application development and decouple the developer from any low-level machine interaction. Indeed, the underlying connection mechanism, EPICS in the case of SNS, is hidden from the developer. The XAL framework also includes an online model for simulating various aspects of machine behavior; this modeling subsystem of XAL is the concern of this paper.

The design objective of the XAL online model is to provide a clean interface for high-level physics and control applications requiring a model reference. This aim includes automatic configuration and synchronization to live hardware as well as presenting the same interface to differing simulation strategies (e.g., single-particle, multi-particle, envelope, response matrix, etc.). It is also valuable to build a modeling system that is easily maintained and upgraded to support the varying requirements of commissioning and operating a particle accelerator. We are able to achieve these various goals due to a novel approach in software architecture called the Element-Algorithm-Probe design pattern introduced by Malitsky and Talman[4]. This design strategy decouples the machine model from beam representation and dynamics calculations.

The numerical simulation techniques and underlying physics have been outlined in a previous paper [2]. The techniques for envelope simulation are covered in detail by Allen and Pattengale [3]. Here we concentrate on the architecture and use of the online simulator. We also present the results of validation and verification studies, which demonstrate that the simulator is, indeed, correctly modeling the accelerator system.

*Work supported by the US Department of Energy under the auspices of SNS
†Corresponding author ckallen@lanl.gov

SOFTWARE ARCHITECTURE

The Element-Algorithm-Probe design pattern separates the machine representation from the beam representation and the dynamics calculations. In accordance with this scheme, subsystems for representing the accelerator, the beam, and the beam dynamics are decomposed into separate software components that communicate through the well-defined software interfaces `IElement`, `IProbe`, and `IAlgorithm`, respectively. In addition we have another subsystem for dynamically synchronizing the model parameters to live hardware configuration, design parameter values, and/or user-specified values.

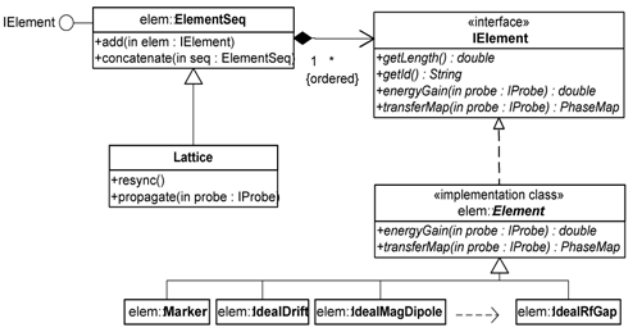


Figure 1: machine representation component

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 may be accessed via the method `resync()`.

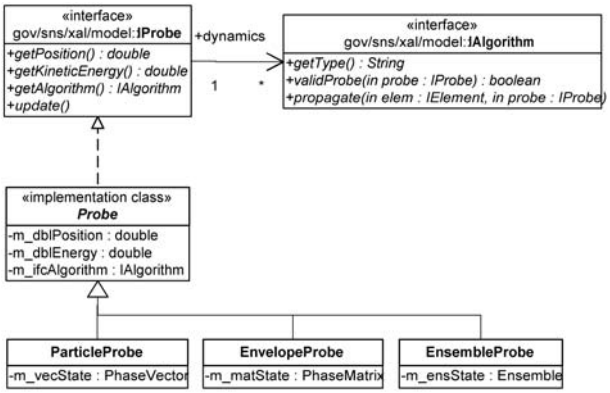


Figure 2: architecture of beam representation component

Beam Representation

Figure 2 depicts the basic architecture of the beam representation component. The interface to this component is called `IProbe`. 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 can be several types of dynamics calculations available for any particular probe (e.g., linear, third-order, etc.).

In Figure 2 we see that the (abstract) 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.

Machine Synchronization

Synchronization with the operating hardware is accomplished through a subsystem based on the (abstract) class `Synchronization`. It supports communication between the XAL model system and arbitrary data sources, which otherwise have no awareness of one another. One data source in particular is the XAL standard machine format (SMF) subsystem, which represents the actual machine hardware.

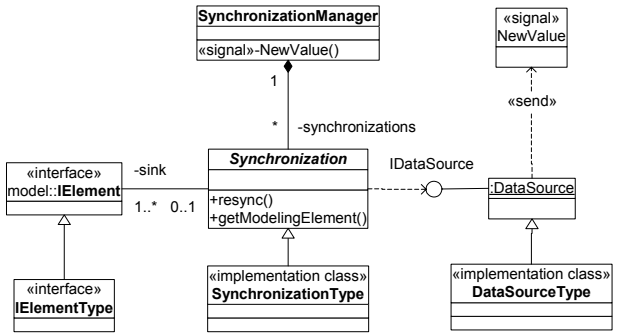


Figure 3: synchronization mechanism

Shown in Figure 3, the synchronization mechanism is governed by a `SynchronizationManager`, which is composed of many `Synchronization` objects. Each modeling element, represented by the IElement interface, may have, at most, one synchronization object associated to it. (This condition is necessary because, for example, "drifts" are not controllable devices of the accelerator.) Also indicated in the figure is that each type of modeling element requires the implementation of a particular synchronization subclass that understands the communication vernacular of the element. This polymorphism is not as difficult to support as one may suspect, since many elements are controllable through the same interface (e.g., dipole correctors and quadrupole lenses are both `IElectromagnet`'s). We also subclass the base class `DataSource` for each data source we support. Once implemented, any synchronization request is carried out by invoking the abstract method `resync()` in the base class. Thus, to remain synchronized with the data source the `SynchronizationManager` invokes `resync()` on each synchronization object whenever required.

VERIFICATION AND VALIDATION

In these contexts, *code verification* asks the question, "Does the code compute what it is supposed to?" whereas *code validation* asks "does the code compute the answer?" We have verified that the online model against the simulation code Trace3D [5] for both the single-particle and envelope simulations. Moreover, the envelope simulation has been verified, to a certain extent, using actual SNS commissioning data. The response matrix calculation has been validated by hand but remains untested in a working control application.

Verification

Single particle simulation results for Trace3D and XAL are essentially identical. For rms envelope simulations, comparison is more involved due to an anomaly of Trace3D. For space charge dynamics one must evaluate the elliptic integral

$$R_D(x,y,z) \equiv \frac{3}{2}\int \frac{dt}{(x+t)^{1/2}(y+t)^{1/2}(z+t)^{3/2}}. \qquad (1)$$

The XAL online model evaluates this integral numerically according to an algorithm by Carlson, which is accurate to arbitrary precision [6]. Trace3D approximates the integral as

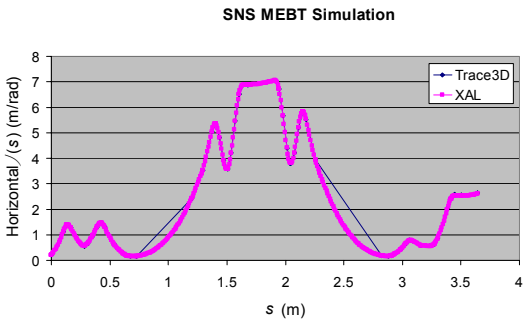
$$R_D(Z^2,Y^2,X^2) \approx \frac{3}{XZ}\frac{1}{X+Y}\left[1-\xi\left(\frac{Z}{\sqrt{XY}}\right)\right] \qquad (2)$$

where $\xi(s)$ is the "form factor"

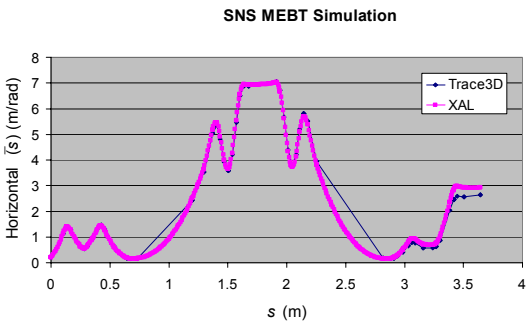
$$\xi(s) \equiv \frac{s}{2}\int_0^\infty \frac{dt}{(t+1)\left(t+s^2\right)^{3/2}} = \frac{1}{1-s^2}\begin{cases} 1-\dfrac{s\cos^{-1}s}{\sqrt{1-s^2}} & s<1, \\ 1-\dfrac{s\cosh^{-1}s}{\sqrt{s^2-1}} & s>1. \end{cases} \qquad (3)$$

Actual evaluation of $\xi(s)$ is done through interpolation of a lookup table. The above approximation is accurate only to first order, the accuracy decreasing with beam eccentricity.

By employing the same approximation for R_D as Trace3D, the XAL online model duplicates the predictions of Trace3D; this result is shown in Figure 4a where we see the simulation results for the SNS medium energy beam transport (MEBT) section. The values for the Twiss parameter $\beta(s)$ are shown in the figure. In Figure 4b the simulation results are shown when XAL implements the more accurate numerical evaluation of R_D. Note that there is discrepancy here.



a) XAL using approximation of R_D



b) XAL using numerical evaluation of R_D

Figure 4: online model verification for SNS MEBT

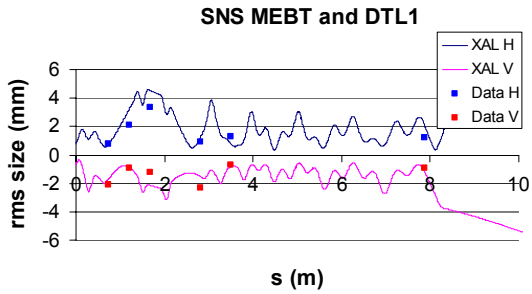


Figure 5: online model validation for SNS MEBT

Validation

In order to verify the predictions of the XAL online model, we compared the results of the rms envelope simulation against wire scanner data from the SNS accelerator. Figure 5 shows data taken from six different beam locations along with the predictions of the XAL online model for the SNS MEBT and first drift tube linac (DTL). Although not exact, the results are clearly correlated. At the time of this writing it is suspected that the initial conditions of the beam into the MEBT section are not accurately characterized.

STATUS AND SUMMARY

The XAL online model currently supports single-particle simulation, envelope simulation, and response-matrix calculation. Multi-particle simulation is still in development; the architecture is in place but the space-charge calculation has not been implemented. Also, XAL has no current capacity for modeling storage rings, although this should be a straightforward extension. All the currently operational simulation capabilities have been verified and the envelope simulation has been validated against SNS commissioning data.

REFERENCES

[1] J. Galambos et. al., "XAL Application Programming Framework", these proceedings.

[2] C.K. Allen, et. al., "A Modular Online Simulator for Model Reference Control of Charged Particle Beams", PAC03, Portland, OR, May 2003.

[3] C.K. Allen and N.D. Pattengale, LA-UR-02-4979 http://lib-www.lanl.gov/cgi-in/getfile?00796950.pdf.

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

[5] K.R. Crandall and D.P. Rusthoi, "TRACE 3D Documentation", LANL Report LA-UR-97-886.

[6] B.C. Carlson, "Computing Elliptic Integrals by Duplication", *Numer. Math.*, Vol. 33, 1-16 (1979).