XAL ONLINE MODEL ENHANCEMENTS FOR J-PARC COMMISSIONING AND OPERATION*

C.K. Allen[#], ORNL, Oak Ridge, TN 37830 USA
M. Ikegami, KEK, Tsukuba, Ibaraki 305-0801 Japan
H. Sako, G. Shen, H. Ikeda, T.Ohkawa, A. Ueno, JAEA, Tokai, Ibaraki 319-1195 Japan

Abstract

The XAL application development environment has been installed as a part of the control system for the Japan Proton Accelerator Research Center (J-PARC) in Tokai, Japan. XAL was initially developed at the Spallation Neutron Source (SNS) and has been described at length in previous conference proceedings [4]. Included in XAL is an online model for doing quick physics simulations [1]. We outline the upgrades and enhancements to the XAL online model necessary for accurate simulation of the J-PARC linac and transport system.

INTRODUCTION

The fundamental tenet of XAL is to provide a consistent, high-level programming interface, along with a set of high-level application tools, all of which are independent of the underlying machine hardware. Control applications can be built to run at any accelerator site where XAL is installed. Of course each site typically has specific needs not supported by XAL and the framework was designed with this in mind; each institution can make upgrades to XAL which are then available to all other users. Recently, many upgrades to the XAL online model were made to enhance operation in general and with specific regard to the J-PARC accelerator complex. This effort includes the addition of new features as well as the enhancements of existing one. Moreover, a major refactoring effort of the original architecture was undertaken in order to incorporate many previous modifications into a more robust framework. The XAL online model is built upon the Element/Algorithm/Probe design pattern and this refactoring included a significant restructuring of the Algorithm and Probe components of this architecture. Finally, in addition to this refactoring and enhancement, a significant effort was devoted toward verification of the online model. (For a comprehensive summary of this work see [3]).

RF GAP MODELING ELEMENT

Significant enhancements to the XAL RF gap modeling element, `IdealRfGap`, were performed. In addition, major refactoring efforts were devoted toward enhancing robustness and clarity (including significant commenting of the underlying simulation procedure), as well as the emittance growth mechanism described below.

Verification of the XAL RF gap simulation was made against the simulation code Trace3D [5]. Initially there was a small discrepancy between XAL and Trace3D. This discrepancy was caused by an error in the normalization procedure for longitudinal momentum.

Previously at J-PARC, the ability to model emittance growth due to phase spread through RF gap elements was added to XAL. The modeling technique implemented was the same as that used in Trace3D. The operation of this feature was verified for the transverse phase planes. However, it was discovered that the model for longitudinal phase-plane emittance growth was invalid for beam bunches with large phase spread. Since this is exactly the case for the J-PARC transport line to the RCS, such a modeling shortfall is of significant consequence. A more appropriate model for longitudinal emittance growth was developed; the details are to appear in a later publication.

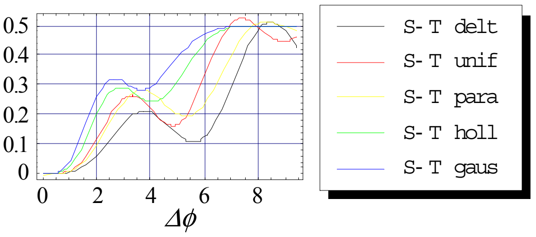

Figure 1: longitudinal emittance growth saturation.

Briefly, the form of the emittance growth function is $G(\phi,\Delta\phi) = S(\Delta\phi) - T(\Delta\phi)\sin^2\phi$ where ϕ is the RF gap phase, $\Delta\phi$ is the bunch phase spread, and S and T are bounded real functions with limiting values of ½ and 0, respectively. Thus, we can get an appreciation for the maximum emittance growth as a function of $\Delta\phi$ by looking at the difference $S(\Delta\phi) - T(\Delta\phi)$. We find that, in general $S(\Delta\phi) - T(\Delta\phi)$ is small for small $\Delta\phi$, then it increases toward a limiting value where emittance growth will saturate regardless of the value ϕ. This effect is shown in Figure 1 for several different beam distributions.

Trace3D correctly captures this effect in the transverse planes. However, it uses a two-term approximation of $G(\phi,\Delta\phi)$ in $\Delta\phi$ in the longitudinal case and, thus, this saturation effect is not captured. Consequently emittance can grow unbounded as phase spread increases. This condition can cause the beam to growth longitudinally because of the artificially high temperature.

The mechanism for simulating this emittance growth was significantly refactored in the J-PARC XAL framework. Although this capability had been added earlier, it was done more as a proof of principle and did not conform to the Element/Algorithm/Probe architecture. Therefore, significant modifications to the existing code were made to adapt the mechanism to the architecture of

* Work supported by KEK under a foreign visiting scientist grant.
[#]ckallen@ornl.gov

the online model to ensure robustness of future code operation. The primary restructuring was to put the emittance growth model into the algorithm component, its natural setting. Additionally, emittance growth was made an optional feature.

SPACE CHARGE EFFECTS MODELING

An exhaustive verification of the online model operation was performed against the simulation code Trace3D. Simulation predictions now show exact agreement, except in the presence of permanent magnet quadrupole (PMQ) elements. Because this small discrepancy exists without space charge effects, it appears to be due to differences in the way PMQs are modeled in the two cases.

Part of the challenge is the differing procedures for approximating space charge momentum "kicks." It was necessary to change the XAL space charge kick procedure in the `EnvelopeTracker` algorithm class to *exactly* that of Trace3D. There are subtleties involved: Given a step length of size h through an element n, the XAL online model now steps as $\mathbf{\Phi}_n(h/2)\mathbf{\Phi}_{sc}(h)\mathbf{\Phi}_n(h/2)$ where $\mathbf{\Phi}_n$ is the transfer matrix beamline element n and $\mathbf{\Phi}_{sc}$ is the space-charge transfer matrix. Previously, XAL stepped as $\mathbf{\Phi}_{sc}(h/2)\mathbf{\Phi}_n(h)\mathbf{\Phi}_{sc}(h/2)$. Both second-order accurate in h by the Campbell-Baker-Hausdorff theorem. Thus, the remainder term is of order $O(h^3)$, however, being a nonlinear system the errors accumulate, especially after 300 meters. To properly compare the codes you must simulate the dynamics exactly. (The differences are then indicative of the limitations in the underlying technique itself.) Another interesting fact is that Trace3D initially steps a distance $h/2$ through an element n (without space charge) then applies the space-charge momentum kick for length h, according to the scheme $\mathbf{\Phi}_n(h/2)\mathbf{\Phi}_{sc}(h)\mathbf{\Phi}_n(h/2)$. To finish the iteration procedure, the beam is again advanced a distance $h/2$ (without space charge). Of course the next iteration again steps the beam a distance $h/2$ within the element n. However, since $\mathbf{\Phi}_n(h/2)\mathbf{\Phi}_n(h/2) = \mathbf{\Phi}_n(h)$ for all n (except a PMQ), it is essentially just a leap-frog technique after that point. It is necessary to step this initial offset to obtain exact comparison with Trace3D.

The method used to compute the space charge matrix $\mathbf{\Phi}sc(h)$ within XAL is more general then that of Trace3D. This fact is due to the use of *homogeneous* phase space coordinates within XAL. However, it also complicates the space charge calculations. Several errors were discovered in the space charge mechanism during the course of this analysis. For example, a Lorentz transform was missed completely and there was an error in the treatment of off-centered beams. Moreover, the original code would work only for beams that were tilted in one phase plane (which would cover most situations). A general solution was developed involving Jacobi decomposition of the covariance matrix. Further details are described in [1].

In addition, several physical and mathematical constants differed slightly in the two codes. These values were located, coalesced, and corrected. The actual corrections were made Trace3D, since the modified values were more accurate than the original values.

PROBE HIERARCHY REFACTORING

The representation of bunched beams was completely refactored. Previously there were many questionable implementations resulting in a very brittle situation. For example, there originally were two parameters, beam current I and bunch charge Q in the `BeamProbe` hierarchy. From these you can calculate the bunch frequency $f = I/Q$ (a method existed). This quantity was *not* the machine frequency, it could be different (e.g., when not filling every RF bucket). However, *post facto* a third attribute, frequency f, had been added to the `BeamProbe` hierarchy. So, we were left with a dangerous inconsistency. Worse yet, there were many instances where the frequency was simply hard-coded into applications and, worse further, into the XAL framework itself. In retrospect the bunch frequency and beam current should have been fundamental attributes of the `BeamProbe` class (parameters most familiar to the beam physicist), from which bunch charge would be computed. The architecture was changed accordingly.

The most dangerous condition found in the Probe component was caused by the redundant state information in the `EnvelopeProbe` (a `BeamProbe` child). The primary attribute of a `EnvelopeProbe` is the *covariance matrix*, the matrix of first and second order moments of the beam distribution. However, a set of Twiss parameter attributes had also been added to the class. Not only did we have the potential for inconsistency (the covariance matrix is a Twiss parameter generalization), but we had actual inconsistencies within the implementation itself. Particularly, there was a dangerous situation relating to inheritance and the virtual method nature of Java. When calling a method to return a Twiss parameter computation from the covariance matrix you would actually get the local Twiss parameters of the probe.

All state information was moved out of the `BeamProbe` class, probably an architectural error in the original implementation. Other than bunch frequency and current, no state information belongs there. In order to deal with the redundant state information another probe class was implemented having Twiss parameters as the primary state variables (see next section). Implementing new probe classes is not as straightforward as it could be (refactoring would be appropriate here), but it is not difficult.

TWISS PARAMETER SIMULATION

Support for the direct simulation of Twiss parameters for bunched beams was added to the XAL framework. This was done to support backward capability for the `EnvelopeProbe` class, where that simulation capability was deprecated. Creation of a separate simulation mechanism for Twiss parameters required the implementation of several new classes, as well as support for these classes within the XAL persistent data mechanism. The main class for beam representation is `TwissProbe` while the simulation algorithm is `TwissTracker`. Fundamental state variables of the `TwissProbe` class are the centroid location, the response matrix, and Twiss parameters representing the beam ellipses in the three phase planes. Note that because of the nature of this state information the simulation will be inaccurate in the presences of bending magnets, misalignments, or any other elements coupling the phase planes. Space charge may be included in a `TwissProbe` simulation; however, it too is accurate only without phase plane coupling.

ALGORITHM REFACTORING

The algorithm class hierarchy of the XAL online model was refactored to add additional software capabilities and increase the robustness of the code. In addition, two classes used for simulating the RMS behavior of bunched beams were substantially refactored. These classes, `EnvelopeTracker` and `EnvTrackerAdapt`, contain algorithms for advancing `EnvelopeProbe` objects through machine elements. Also, several bugs were found in the `EnvTrackerAdapt` class, the Twiss parameters would not be computed correctly in some instances, and the phase advance also appeared to be incorrect. Finally, new documentation to the code (Javadoc) was added to explain the new architecture.

For users of the online model the following summarizes the major refactoring: 1) The `AlgorithmFactory` class is now deprecated and replaced by an implementation using Java reflection, one only needs to specify the Java class type. 2) The `EditContext` loading mechanism was moved down to the `Tracker` base class and deprecated in its child class `TrackerAdaptive`. Consequently, any algorithm and, thus, probe type can use the `model_params` automated technique for retrieving its parameters. This feature is still only implemented for the `EnvTrackerAdapt` class. 3) The `TrackerAdaptive` middle class was removed and all it functionality placed into the `Tracker` base class.

BENDING MAGNETS

The capability of simulating space charge effects within bending magnets was added to the XAL online model. Model elements require a specific architecture to support space charge calculations. It was necessary to implement a separate object for bending dipole magnets according to this architecture.

Previously there were two elements in XAL which modeled bending dipoles. `ThickDipole` modeled a bending dipole and correctly handles the dynamics when driving the dipole magnet off the design field strength. However it does not conform to the XAL architecture and, consequently, cannot handle space charge correctly. `IdealMagWedgeDipole` supported the space charge mechanism of the XAL online model, however, it did not treat the dynamics due to variations in field strength; it only handles the change in quadrupole focusing.

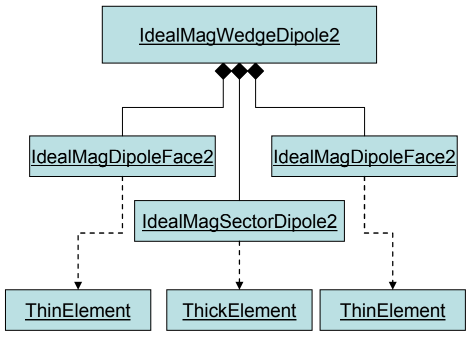

Figure 2: bending dipole architecture.

The architecture of `IdealMagWedgeDipole` is shown in the UML class diagram Figure 2. It is a composite of three separate objects, the entrance pole face, the magnet body (an `IdealMagSectorDipole` object), and the exit pole face. A new object, `IdealMagWedgeDipole2` was created which combines the aspects of the previous two classes. As shown in the figure, the magnet body was replaced with `IdealMagSectorDipole2` which contains the dynamics in the original `ThickDipole` class. In other words, the physics of `ThickDipole` was implemented into the architecture of `IdealMagWedgeDipole`.

Another refactoring effort worth noting is the removal of an SNS stripper foil exception. No stripper foils are assumed in the dipole, as was the case previously if the design curvature and the particle curvature had differing signs. A more robust design should be implemented to handle this situation if necessary. This would probably entail the creation of a new class represented a stripper foil which would change the properties of the probe object. The previous implementation may have led to some very confusing surprises in some simulations

REFERENCES

[1] C.K. Allen, K. Furukawa, M. Ikegami, and K. Oide, “Adaptive Three-Dimensional RMS Envelope Simulation in the SAD Accelerator Modeling Environment”, LINAC06 Conference Proceedings, Knoxville, TN, June, 2006.

[2] C.K. Allen, C.A. McChesney, C.P. Chu, J.D. Galambos, W.-D. Klotz, T.A. Pelaia, A. Shislo, “A Novel Online Simulator for Applications Requiring a Model Reference”, ICALEPCS 2003 Conference Proceedings, Kyongju, Korea, October 13-17, 2003.

[3] C.K. Allen, “Foreign Visiting Researcher Project Summary Report: Accelerator Controls and Simulation”, KEK internal document.

[4] C.P. Chu, S. Cousineau, J.D. Galambos, J. Holmes, T.A. Pelaia, A. Shishlo, Y. Zhang, C.K. Allen, “SNS Application Programming Infrastructure and Physics Applications”, APAC07, Indore, India, Jan 29 - Feb 2, 2007.

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