INTERFACING OF THIRD-PARTY ACCELERATOR CODE WITH THE LUCRETIA FLIGHT SIMULATOR*

Stephen Molloy†‡, Royal Holloway, University of London,
Mauro Torino Francesco Pivi, Glen White, SLAC National Accelerator Laboratory,
Yves Renier, LAL, Orsay

Abstract

The Flight Simulator is a tool used for international collaboration in the writing and deployment of online beam dynamics algorithms. Written as an add-on to the Lucretia tracking software, it allows simulation of a beamline in a control system environment identical to that in the control room. This allows the testing and development of monitoring and correction tools by an international collaboration by making the control system transparent to the user. The native beamline representation are those adopted by Lucretia, so, in order to allow third party software, to interface with this system, it was necessary to develop functionality to convert the lattice to a universal representation. Accelerator Markup Language (AML), and its associated Universal Accelerator Parser (UAP), were used for this purpose. This paper describes the use of the UAP to convert the internal beamline representation to AML, and the testing of this conversion routine using the lattice description of the ATF2 final focus experiment at KEK, Japan. Also described are the inclusion of PLACET and Strategic Accelerator Design (SAD)[9] based algorithms using appropriate converters, and tests of these on the ATF2 extraction line.

INTRODUCTION

The Flight Simulator (FS)[1] is a middleware software package, designed to facilitate the development and deployment of various tuning algorithms currently under development (e.g [2][3][4]) for the ATF2 accelerator facility[5]. It has been written as part of the Lucretia[6] beamline tracking package, and it allows, through the use of an EPICS Channel Access connection to the machine, or Lucretia's tracking simulation engine, code to be developed in an identical environment to that found in the control room. Thus, software may be thoroughly debugged in *simulation mode* prior to deployment on the actual machine, thus increasing the efficiency of beamline operations.

One of the principle goals of the FS was to allow users to develop their algorithms in whatever software package they felt was appropriate (e.g. *SAD* is in widespread use at KEK), so a method had to be found to circumvent the fact that Lucretia's data structures are kept in its native, Matlab, format. For this reason, it was decided to implement a utility to allow conversion of this format to Accelerator Markup Language (AML).

ACCELERATOR MARKUP LANGUAGE

Accelerator Markup Language (AML)[7][8], is based on eXtensible Markup Language (XML), and is a well developed format for describing accelerator lattices. It has been designed to be as flexible as possible, e.g. making few assumptions about the species of particle to be accelerated, and, due to its basis in XML, is capable of being extended to allow new elements, etc., to be represented.

AML was thought to be an excellent choice for the default representation language for the FS for several reasons.

Firstly, AML has been designed to be as complete a representation of a beamline as possible; including the ability to store engineering details (e.g. drawing numbers for beamline components) that may not be useful in other languages. In addition, it is structured in such a way that means that having to, for example, split a magnet in order to include a centrally located BPM, is unnecessary. In such a case, AML would have a single representation of the magnet, and include a BPM attribute on its XML tree.

In addition, many groups, including the ILC, are migrating their lattices to AML format, which means that AML may be considered to be an international standard for beamline representation, and is an acceptable format for use by a wide variety of users.

UNIVERSAL ACCELERATOR PARSER

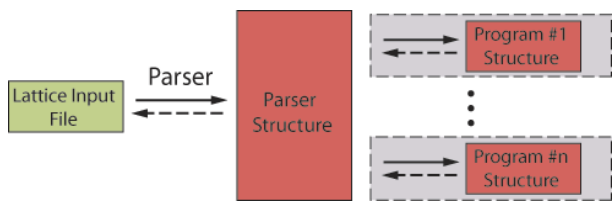

Figure 1: Schematic of the operation of the Universal Accelerator Parser.

In conjunction with AML, an additional C++ library known as the Universal Accelerator Parser (UAP) has been developed[7][8] in order to easily integrate AML with software.

* Work supported by ...
† smolloy@pp.rhul.ac.uk
‡ Formally with SLAC National Accelerator Laboratory

UAP provides a toolkit to allow AML files to be read into the working memory of a C/C++ program in such a way that allows easy dissemination into other lattice description formats, and also simplifies much of the effort of translating from that format into AML.

The operation of the UAP is shown in schematic form in figure 1. The '*Lattice Input File*' on the left of the diagram is the AML representation of the machine lattice, and the '*Parser*' is the UAP, which takes the AML description, and converts it into a C++ object, '*Parser Structure*'. Once this operation has completed, the remaining task is to convert this structure into the format required by the user — a task which is considerably facilitated with the use of several C++ methods included in the UAP.

The converse operation — that of converting some alternative lattice format into AML — is also greatly simplified by the UAP. In this case, the user would build software using the methods availble in the native format of the lattice representation code and from the UAP, to convert the original lattice to the C++ parser structure object, which would then be simply converted to AML using the UAP.

MATLAB C API

Lucretia, and, therefore, the FS, are Matlab toolkits, which means that the beamline description will be stored in memory as Matlab data structures. Due to the binary nature of this format (as compared to the ascii-text nature of AML), more work is required to allow this to be converted to the UAP parser structure.

Fortunately Matlab ships with an extensive C API that allows easy access and control of all necessary Matlab data structures.

Thus the process required to convert the Lucretia representation to AML, is to use the matlab array manipulation functions to extract the relevent data from each data structure, convert this to the UAP parser object using UAP methods, and then call the UAP functions to write this out as a text-based AML file.

This will be coded into a C++ function, which will then be wrapped, using the Matlab C API, in such a way that it is callable as a standard Matlab executable (such functions are known as *mex-files*).

LUCRETIA2AML

Lucretia2AML is the name of the executable written to perform the task of conversion between the FS, Matlab-based, format, and AML. As explained in previous sections, it is written in C/C++, and includes functions and methods included from the UAP and Matlab C/C++ libraries.

The following code snippet shows the Lucretia represenation of a sector-bend as indicated at the Matlab command line, and it can be seen that the requirement to have a centrally located marker makes the splitting of this magnet necessary. While this is entirely accurate from the point of view of tracking, it is considered a little untidy from the point of view of being a precise representation of the beamline since the physical magnet is one device, not two.

```
ans =
            Name: 'KEX1A'
               S: 0
               P: 1.300
           Class: 'SBEN'
               L: 0.2500
               B: [0.108 0]
              dB: 0
           Angle: 0.0025
       EdgeAngle: [0 0]
            HGAP: [0.0063 0.0063]
            FINT: [0.5000 0]
   EdgeCurvature: [0 0]
            Tilt: 0
              PS: 66
          Offset: [0 0 0 0 0 0]
          Girder: 0
       TrackFlag: [1x1 struct]
          Slices: [1 3]
           Block: [1 3]

ans =
     Name: 'IP01'
    Class: 'MARK'
        S: 0.2500
        P: 1.3000
    Block: [1 3]

ans =
            Name: 'KEX1B'
               S: 0.2500
               P: 1.300
           Class: 'SBEN'
               L: 0.2500
               B: [0.108 0]
              dB: 0
           Angle: 0.0025
       EdgeAngle: [0 0.0050]
            HGAP: [0.0063 0.0063]
            FINT: [0 0.5000]
   EdgeCurvature: [0 0]
            Tilt: 0
              PS: 66
          Offset: [0 0 0 0 0 0]
          Girder: 0
       TrackFlag: [1x1 struct]
          Slices: [1 3]
           Block: [1 3]
```

Thus we see that one element needs three Lucretia objects to be fully represented. The following shows the same sector-bend in AML after conversion using Lucretia2AML, and it can be seen to be much more compact, despite containing additional information (in the form of its orientation

with respect to the design orbit).

```
<element name = "KEX1A">
  <bend>
    <g_u design = "0.433633" err = "0" />
    <e1 design = "0" />
    <e2 design = "0.005" />
    <h_gap1 design = "0.00635" />
    <h_gap2 design = "0.00635" />
    <f_int1 design = "0.5" />
    <f_int2 design = "0.5" />
    <h1 design = "0" />
    <h2 design = "0" />
    <orientation origin = "CENTER">
      <x_offset design = "0" />
      <x_pitch design = "0" />
      <y_offset design = "0" />
      <y_pitch design = "0" />
      <s_offset design = "0" />
      <tilt design = "0" />
    </orientation>
  </bend>
  <length design = "0.5" />
  <marker name = "IP01" />
</element>
```

With the completion of Lucretia2AML, it was necessary to test the lattice, however, without a reliable method with which to convert the lattice back to Lucretia, it was not possible to do that at this stage. Successful tests were completed with the addition of a conversion routine between AML and SAD.

AML TO SAD

One of the initial uses of the AML lattice was to convert this to SAD format for use in a beamline tuning algorithm.

A pre-existing AML to SAD converter did not exist, so it was necessary to implement one using the UAP. This has now been released as part of the UAP.

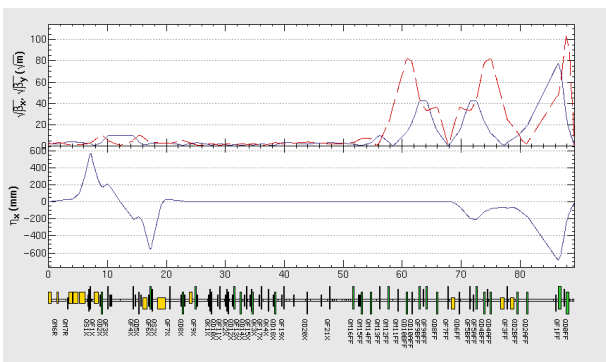

Figure 2: Twiss parameters as calculated using SAD after conversion from Lucretia.

Via a SAD-Parser written in C++, we translated the AML lattice to a SAD lattice format representation, (i.e. a SAD input file), of the beam line. The Lucretia model of the machine is then translated to SAD in two steps. This makes the flight simulator tools available to the SAD users. By running SAD with the translated representation of the ATF2 extraction line as input, we reproduced correctly the twiss parameters at the IP and expected beam sizes, in particular betax*=4 mm, betay*=0.1 mm as shown in figure 2 and vertical beam size of sigmay*=34.8 nm. This is a good test of the correctness for the codes translation chain: Lucretia-AML-SAD. We are planning the converse operation to convert a SAD lattice format into AML.

RESULTS

With the implementation of a full chain of conversion utilities between Lucretia, AML, and SAD, it is possible to test the final result by comparing the predicted twiss parameters as calculated by Lucretia and SAD, and it was found that there was excellent agreement.

REFERENCES

[1] G. White, et al, “A Flight Simulator for ATF2: A Mechanism for International Collaboration in the Writing and Deployment of Online Beam Dynamics Algorithms”, EPAC ’08, Magazzini del Cotone, Genoa, Italy, Jun 2008.

[2] A. Scarfe, R. Appleby, J. Jones, D.A. Kalinin, “ATF2 Spot Size Tuning Using the Rotation Matrix Method”, this conference.

[3] Y. Renier, P. Bambade, J. Resta-Lpez, K. Kubo, G. White, A. Scarfe, “Orbit Reconstruction, Correction, Stabilization and Monitoring in the ATF2 Extraction Line”, this conference

[4] G. White, R. Tomas, K. Kubo, S. Kuroda, Y. Renier, J. Jones, A. Scarfe, “Plans and Progress towards Tuning the ATF2 Final Focus System to Obtain a 35nm IP Waist”, this conference

[5] A. Seryi, et al., “ATF2 Commissioning”, this conference.

[6] P. Tenenbaum, “Lucretia: A Matlab-based toolbox for the modelling and simulation of single-pass electron beam transport systems”, PAC 05, Knoxville, Tennessee, 16-20 May 2005

[7] D. Sagan, et al., “The Accelerator Markup Language and the Universal Accelerator Parser”, EPAC 2006, Edinburgh, Scotland (2006).

[8] D. Bates, D. Sagan, A. Wolski, “The Universal Accelerator Parser”, Proceedings Int. Comp. Accel. Conf. 2006, p 303, (2006).

[9] K.Hirata, “An introduction to SAD”, Second Advanced ICFA Beam Dynamics Workshop, CERN 88-04 (1988).