THE UNIVERSAL ACCELERATOR PARSER *

D. A. Bates, LBNL, Berkeley, CA 94720, USA
D. Sagan, Cornell University, Ithaca, NY, 14850, USA
A. Wolski, University of Liverpool, UK, and the Cockcroft Institute, Daresbury, UK.

Abstract

To promote the sharing of lattice description files between modeling codes, a new lattice description standard named the Accelerator Markup Language (AML) has been developed. Based upon the standard eXtensible Markup Language (XML) format, AML has extensive capabilities for modeling accelerators. Additionally, since it is based upon XML, AML has the flexibility to be easily extended to satisfy changing requirements. This flexibility allows for AML files to be used beyond lattice descriptions to act as a database of accelerator information.

In conjunction with AML, an open source software library called the Universal Accelerator Parser (UAP) is being developed. This library, when integrated into a program, enables the reading of AML lattice files. Included in the UAP software are routines for expression evaluation and beam line expansion so that a "flat" lattice that represents each lattice element is easily constructed. Furthermore, the UAP library is structured so that it can accommodate language modules that read in lattice files using formats other than AML. Specifically, modules for reading MAD-8 and MAD-X format lattices are being developed. As an added benefit, the UAP can act as a translator between the languages it has modules for.

INTRODUCTION

The complexity of modeling high energy particle beams has resulted, world wide, in the development of a number of modeling programs. Since each program has its own strengths and weaknesses, it is often desirable to analyze a given accelerator using a variety of such programs. Additionally, cross-checking results between different codes is often essential for validating these programs. Different programs generally require input files in different formats and this is a significant obstacle to applying a variety of such programs to a single accelerator project. At present, there is no generally accepted lattice format that is sufficiently comprehensive to meet the needs of the accelerator community.

To address this problem, a lattice description format called Accelerator Markup Language (AML) has been created [1]. AML is based upon the standard eXtensible Markup Language (XML) [2] which provides the necessary flexibility for AML to be easily extended to meet changing requirements. Moreover, the extensibility of XML enables AML files to be used beyond lattice descriptions to include such information as the control system configuration, blueprint and other documentation, magnet history information, etc. In short, AML could be used as the basis for a complete database of an accelerator laboratory complex.

A major impediment to the adoption of a lattice description format like AML is the complexity of the software that is needed to read an input file. This complexity is compounded with AML for two reasons. First, the attributes of machine elements can be expressed using arithmetical expressions. Furthermore, the ordered list of lattice elements that comprise the accelerator may be expressed using suitable subsequences (called "sectors" in AML) of machine elements which are then used as components in defining other sectors. Thus, any software has to be able to deal not only with expression evaluation, but also with "lattice expansion" in which the hierarchy of sectors is combined to give a single "flat" list of lattice elements that can be used for tracking and other types of analysis.

To alleviate the need to independently create the necessary software for each and every program, a software library, known as the Universal Accelerator Parser (UAP), is being developed to perform the necessary parsing, expression evaluation, and lattice expansion for AML lattice files. A centralized source for the parsing software has the added benefit that if AML is modified or extended to meet changing requirements, these changes can be quickly propagated to the programs that use the UAP library.

The UAP library is being designed in a modular fashion so that, with the inclusion of appropriate software modules, the UAP library can be extended to read in additional lattice formats. Such a module needs only to be able to parse a lattice file of the given format and then translate the information to AML form. Once this is done, the UAP's expression evaluation and lattice expansion routines can be used to produce a flat lattice. Since expression evaluation and lattice expansion are, by far, the most complicated part of the process for any reasonably sophisticated language, adding a language module to the UAP library is a far simpler task than coding from scratch. Currently, modules for MAD-8 and MAD-X are being developed. Furthermore, the UAP can then act as a translator between the languages it has modules for.

With AML, one can set up element dependencies where the settings of one element affect attribute values of other elements. In conjunction with this, the UAP library provides bookkeeping routines for managing these dependencies. This creates a powerful tool for simulating the control

* Work supported by the National Science Foundation and by the US Department of Energy under contract numbers DE-AC02-05CH11231.

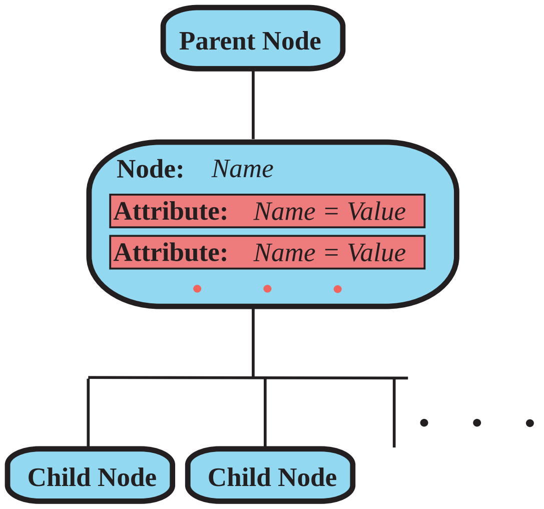

Figure 1: Schematic illustration of the basic UAP Node. The basic node has a name, a set of attributes, a parent node and a set of children.

system. For example, a power supply controlling a set of quadrupoles may be simulated with AML. An analysis program can vary the simulated power supply voltage and the lattice model set up by the UAP is kept up-to-date, without the need for additional code within the analysis program to maintain the correct dependencies.

ACCELERATOR MARKUP LANGUAGE

In order to understand the UAP code, it is important to understand something of the structure of AML. Since AML is based upon XML, AML represents data in a hierarchical fashion. The root (top level) node is `<laboratory>`. Among other things, the `<laboratory>` node can have `<element>` subnodes (children) describing the machine elements of an accelerator and `<machine>` subnodes describing the sequence of elements that make up the lattice. Multiple `<machine>` nodes can be present defining different machines and these machines may be connected together to form a complete accelerator system. For example, one machine can define a LINAC which is then connected to another machine representing a ring which, in turn, is connected to multiple transfer lines, etc. A `<machine>` is made of `<sector>`s which are lists of elements. A `<sector>` is essentially equivalent to a MAD “list” or “sequence”.

AML has four concepts that complicate the bookkeeping that must be done to create and maintain a flat lattice: “superposition”, “multipass”, “controllers”, and “girder”. These concepts are explained below.

The position of an element along a beamline may be determined by its position in a list of elements comprising the lattice. Alternatively, the position may be given explicitly with respect to some reference point. This latter case is known as superposition:

```
<sector name = "this_sect">
  <element ref = "sol" />
  <element ref = "drft" />
  <element ref = "q2"
           superimpose_at = "0.8"
           ref_element = "sol" />
</sector>
```

This example shows a `<sector>` comprised of three `<element>`s. The center of the third `<element>` q2 is positioned 0.8 meters from the center of `<element>` sol. Superposition can complicate matters since, as in real machines, elements are allowed to overlap spatially.

The multipass construction is used where the beam goes through parts of the machine multiple times. For example, in an Energy Recovery Linac, the beam may go through a LINAC section first to accelerate the beam and then to decelerate it to retrieve its energy. To model this, `<sector>`s in AML may be designated as `multipass`:

```
<sector name = "ERL">
  <sector ref = "linac"
          multipass = "true" />
  <sector ref = "turn_around" />
  <sector ref = "linac" multipass = "true"
          reflection = "true" />
</sector>
```

In this example, the beam first goes through the LINAC section, is turned around, and then goes back through the LINAC in the opposite direction. The utility of using multipass will be discussed below.

With AML, `<controller>` nodes may be defined that simulate the effects of anything which affects machine parameters, such as klystrons, power supplies, or control room knobs. For example:

```
<controller name = "ps1"
            variation = "ABSOLUTE" >
  <control element = "q1"
      attribute = "multipole:k1"
      coef = "2.3 * sin(ps1)" />
  <control element = "sol"
      attribute = "multipole:ks"
      coef = "-5.7 * ps1" />
</controller>
```

In this example, the `<controller>` controls the k1 attribute of a quadrupole and the ks solenoid strength attribute of a solenoid.

A `<girder>` is a support structure that supports a set of elements. The spatial orientation of an element on the girder is a combination of the orientation of the girder and the orientation of the element with respect to the girder. Thus a girder may be thought of as a special form of a controller.

UNIVERSAL ACCELERATOR PARSER

The UAP software stores data using a model whereby data is represented as a tree of nodes. The basic node structure is schematically shown in Figure 1. The basic

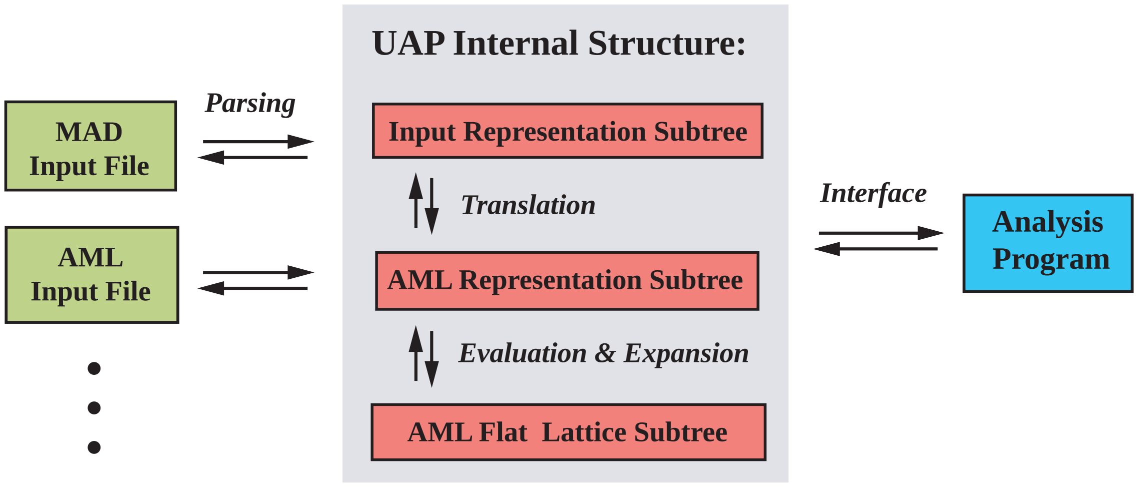

Figure 2: Schematic diagram of the memory structure set up by the Universal Accelerator Parser. The UAP root node has three children. The Input Representation Subtree (IRS) is a faithful representation of the information in the input file. The AML Representation Subtree (ARS) is the equivalent AML representation. Lattice expansion and expression evaluation lead from the ARS to the AML Flat Lattice Subtree (AFLS).

node structure has a name and a set of attributes, with each attribute having an attribute name and an attribute value (which is just a text string). This has a one-to-one correspondence with XML syntax:

```
<node_name attrib1_name = "value1"
           attrib2_name = "value2">
```

Additionally, a node has associated with it a set of child nodes and, except for the root node, a parent node.

When a lattice file is read in, the UAP software sets up three children of the root node as shown in figure 2. The information in a lattice file is stored in the "Input Representation Subtree" (IRS). The IRS has a one-to-one correspondence with the information in the input file (including comments), so the reverse process of generating a lattice file from an IRS is straightforward. If the original file was not an AML file, the UAP software translates from the IRS to the "AML Representation Subtree" (ARS). The ARS represents the lattice in AML format. If the input file is an AML file, then the IRS and ARS are identical and no conversion is needed. Lattice expansion and evaluation of any mathematical expressions of the ARS leads to the "AML Flat Lattice Subtree" (AFLS).

An example will make clear the roles of, and the relationships between, the three subtrees. Consider a MAD file:

```
! MAD input file
q1: quad, l = 2*3
s2: sextupole
l1: line = (q1, 2*s2)
use, l1
beam, energy = 5.2
```

Using XML-like notation with indentation indicating tree layout and the ending tags suppressed, the IRS structure in memory that is created for this file would look like

```
<input_representation>
  <doc>
    "! MAD input file"
  <element name = "q1"
           key = "quadrupole" l = "2*3" />
  <element name = "s2"
           key = "sextupole" />
  <line name = "l1">
    <element name = "q1" />
    <element name = "s2" repeat = "2" />
  <use line = "l1" />
  <beam energy = "5.2" />
```

Converting to the ARS involves such things as converting from the units used by MAD to the units used by AML (for example, converting GeV to eV), and replacing lines by sectors, etc. The equivalent ARS for the above example is:

```
<AML_representation>
  <laboratory>
    <doc>
      "MAD input file"
    <element name = "q1" />
      <quadrupole>
        <length design = "2*3" />
    <element name = "s2" />
      <sextupole>
    <machine>
      <sector name = "l1">
        <element ref = "q1" />
        <sector repeat = "2">
          <element ref = "s2" />
      <root_sector ref = "l1" />
      <beam>
        <energy design = "(5.2) * 1e9" />
```

Conversion from the ARS to the AFLS involves expression evaluation and lattice expansion:

```
<AML_flat_lattice>
  <machine>
    <tracking_lattice>
      <element name = "beginning">
      <element name = "q1">
        <quadrupole>
          <length design = "6" />
      <element name = "s2">
        <sextupole>
      <element name = "s2">
        <sextupole>
    <beam>
      <energy design = "5.2e9" />
```

An analysis program would convert the information from the AFLS to whatever memory structure the program uses for its calculations.

Adding a new language module involves creating code for parsing a lattice and creating an IRS along with IRS to ARS translation code. Back-translation and back-parsing code to translate an ARS to an IRS and an IRS to a lattice file would also be needed if translation from a different language into the target language is desired. Since expression evaluation and lattice expansion are considerably more complicated than parsing or IRS to ARS translation, the UAP architecture allows for fairly simple incorporation of different lattice languages. Conversion between different languages involves reading in a file to create first an IRS and then an ARS, and then back conversion from the ARS to the appropriate IRS and finally to a lattice file.

When the ARS is built from the IRS and the AFLS is built from the ARS, the UAP software maintains "twin" pointers in the nodes which connect equivalent nodes in the IRS, ARS, and AFLS. This provides support for lattice design work where a possible procedure could be as follows:

1. Read in a lattice file and construct an IRS, ARS, and finally an AFLS.

2. Taking the AFLS as a starting point, use a modeling code to vary a given set of parameters to optimize specific lattice attributes.

3. Using the twin pointers, find the nodes in the IRS corresponding to the parameters in the AFLS varied in the optimization.

4. Modify the IRS nodes using the optimized values of the parameters.

5. Construct a lattice file with the optimized values from the IRS.

The result is a file that has the same layout as the original file, but with optimized parameter values.

SUPERPOSITION AND MULTIPASS

As outlined in a previous section, the AML syntax allows elements to be superimposed on top of other elements as outlined above. When this is done, the `<tracking_lattice>` that is created in the AFLS will contain hybrid elements that represent the regions that are created by the intersections of the elements. Using the superposition example above, if elements `sol` and `drft` are 2 meters long and element `q2` is 1 meter long, then the `<tracking_lattice>` would contain the elements

```
Name      Length  Element Type
-------   ------  ------------
sol|1     1.3     Solenoid
q1|sol    0.7     Hybrid quad/solenoid
q1|drft   0.3     Hybrid quad/drift
drft|1    1.7     drift
```

The `q1|sol` element represents the space where the solenoid and quadruple overlap

while the `sol|1` element represents the part of the solenoid where there is no overlap. Similarly, the `q1|drft` element represents the part of the quadrupole outside of the solenoid and the `drft|1` element represents the portion of the drift not inhabited by the quadrupole. To be useful, an accelerator modeling program would have to be able to handle the hybrid elements that are generated. A major advantage of this approach is that the UAP software performs all the necessary bookkeeping, making creation of flat lattices easier and less error prone. This represents a significant saving of time and effort in maintaining any analysis program that uses the UAP software.

As an added benefit, as the `<tracking_lattice>` elements are constructed, UAP keeps a list of the original elements (in this example, `sol`, `q1`, and `drft`) in another part of the AFLS. Appropriate pointers are maintained between these "master" elements and the "slave" hybrid elements they control. The UAP provides bookkeeping software so that programs can vary the attribute values in the master elements and have these changes mirrored in the attribute values of the hybrid elements. For example, a change to the solenoid strength in the `sol` element would be propagated to the `sol|1` and `q1|sol` hybrids.

A similar situation occurs with multipass sectors. Using the multipass example above, if the `linac` sector looked like:

```
<sector name = "linac">
  <element ref = "A" />
  <element ref = "B" />
  ...
</sector>
```

then the expanded lattice would have elements:

```
A|1, B|1, ..., B|2, A|2
```

The `|1` and `|2` suffixes indicates which pass through the `linac` section it is. As in the case with superposition, `A` and `B` master elements are set up with pointers between these elements and the corresponding elements that appear in the flat lattice. Again, the UAP's bookkeeping routine will make sure that changes in the attributes of the master elements are reflected in their slave elements. Thus, for example, if element `A` is given a positional offset, the UAP bookkeeping routine will transfer the offset value to the `A|1` and `A|2` elements.

Controller and girder elements work in a similar fashion. The appropriate master elements are set up in the AFLS and the UAP bookkeeping routine will update the appropriate attribute values with changes in the master elements.

STATUS

The AML/UAP development project is a collaborative effort among a number of laboratories and anyone who is interested is invited to join the effort. The project home page is at

```
http://www.lns.cornell.edu/~dcs/aml
```

The UAP software is written in C++. There is a Fortran90 interface planned to allow easy interoperability with Fortran programs. Additionally, a Java port will be maintained for platform independence. The UAP code is open source and is released under the GNU Lesser General Public License [4]. The UAP source code is maintained in a CVS repository and is available at the SourceForge.com [3] web site.

CONCLUSION

The Universal Accelerator Parser software is currently under development for use as a common library among accelerator codes for lattice parsing. Its use holds the promise of greatly improving the interoperability between different programs. As an added benefit, the UAP code can translate lattice files between the languages it has modules for. In particular, MAD-8 and MAD-X modules are being developed.

Additionally, the UAP library contains bookkeeping routines to simplify the task of simulating the control system, and defining and manipulating complex beamline features such as physically overlapping elements, and element support structures.

THANKS

Thanks must go to the people who have contributed to the AML/UAP project. In particular, we would like to thank Yves Roblin for developing the AML Schema. Additionally, Mike Forster, Theo Larrieu, Tom Pelaia, Frank Schmidt, Peter Tenenbaum, Nick Walker, Mark Woodley as well as Nikolay Malitsky and Richard Talman are to be thanked for useful discussions.

REFERENCES

[1] D. Sagan et al., The Accelerator Markup Language and the Universal Accelerator Parser", 2006 Europ. Part. Acc. Conf., Edinburgh (2006).

[2] XML was developed by the World Wide Web Consortium: `http://www.w3.org/XML`

[3] See: `http://sourceforge.net/projects/accelerator-ml`

[4] See: `http://www.gnu.org/licenses/lgpl.html`