ACCELVIEW: A GRAPHICAL MEANS FOR DRIVING INTEGRATED NUMERICAL EXPERIMENTS*

N. Barov#, J. Grubert, J.S. Kim, FAR-TECH, Inc., San Diego, CA
S. Reiche, UCLA Dept. of Physics and Astronomy, Los Angeles, CA

Abstract

Many accelerator projects make use of integrated numerical experiments, where particle distributions are transferred between several accelerator codes. This is usually accomplished by writing custom scripts that launch the underlying programs and perform data format translation. We present a way to simplify this process by using a graphical user interface that allows one to describe the data flow in the style of the LabVIEW and Simulink environments. A module to support a new accelerator code involves writing data translators to/from a common format (SDDS), and a function to generate an input file based on a standard way of specifying an accelerator lattice (such as Accelerator Markup Language, or AML).

INTRODUCTION

Many accelerator physics problems need the use of more than one simulation code or package to treat the entire beamline and experiment. This process goes by the name of integrated numerical modelling or start-to-end simulation. A perfect example is an FEL facility, where a code such as PARMELA is used to model the photoinjector, ELEGANT is used to model the linac, and the FEL interaction is modelled with GENESIS.

Start-to-end simulations can be difficult to perform because of: 1) a lack of familiarity with all the codes involved, 2) the need to write custom scripts for translating between different particle distribution file formats, 3) the need to run each code in the chain, either by hand or by writing a custom script, and 4) the lack of graphical methods to facilitate the whole process. Although such graphical user interfaces (GUI's) do exist in other fields of science, there is no generic GUI for performing integrated numerical modelling that can easily be adapted to accelerator problems.

Our project relies on a common method for specifying an accelerator lattice that can be used with many accelerator physics codes. For this purpose, we have chosen AML (Accelerator Markup Language) [1]. In this scheme, the accelerator lattice is described using XML, which yields a number of advantages. For instance, the lattice description data can be validated against an XML schema, and can be processed using standard techniques to perform functions such as tree traversal, search, and database integration. An additional advantage is that future changes to accelerator codes should be reflected in AML, and thus easily integrated into our project.

GRAPHICAL DESCRIPTION OF A BEAMLINE

Performing start-to-end simulations with a wide range of supported codes can benefit from having a common set of tools for describing an accelerator lattice, and the ability to translate that description to the native input files of each accelerator code.

We have developed two graphical methods for this purpose, and for interacting with an accelerator project in general. One is based on LabVIEW, and the other is an in-house developed diagramming GUI that seeks to provide the flavor of LabVIEW development, and is called AccelView.

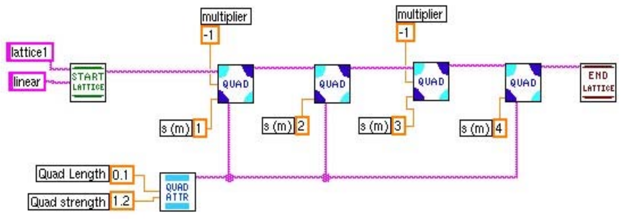

Figure 1: Defining a FODO lattice with a set of LabVIEW functions.

Figure 1 is an example of defining a beamline lattice using our LabVIEW toolkit. The line that originates at the “Start Lattice” function and goes through every quadrupole represents the beamline. Every beamline element (quad, bend, accelerator cavity, etc.) has parameter inputs for its most often used inputs, as well as an attributes input to specify less-often used parameters such as misalignments and errors, floor coordinates, and tilt. Attributes are created in a series of functions and chained together, similar to a list of elements in a beamline. Elements and attribute functions can be selected from a drop-down icon list, and further documentation is available in a help window.

After a beamline description is graphically created, the beamline data is translated to AML (Accelerator Markup Language). The data is then translated to the native input file formats of a set of accelerator codes. Additional accelerator codes can be supported by writing translator modules from AML to the native input file formats.

The LabVIEW environment allows the user to create arbitrarily complex programming projects, and this capability can effectively be used as the scripting interface to our toolkit. The AccelView environment has the same capability. In this way, the software can appeal to both non-programmers, because it is easy to learn, and to programmers, who can appreciate the power inherent in the software's scriptability.

*Work supported by US Dept. of Energy.
#barov@far-tech.com

SCRIPTING USING A GUI

In both the LabVIEW and AccelView environments, scripting ability is built-in. Typical programming constructs such as functions, loops, and conditionals are available in both environments. Inputs and outputs are linked together by wires, which control the flow of the script. Any given loop, conditional, or function will not execute until all its input wires are marked as "valid", meaning that the previous object connected to the wire has already been executed. Additionally, a sequence object is provided, which is used to control the flow of the program when wires cannot.

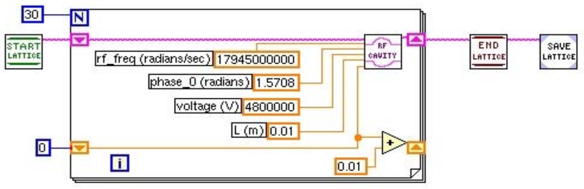

Figure 2: Using a loop to create 30 RF cavities.

This scripting capability makes it possible to easily accomplish complicated tasks. To get an idea of how this works, consider the lattice given in Figure 2. This beamline description creates a lattice consisting of 30 RF cavities, end-to-end. (The final input to the RF cavity function is the distance, *s*, given in meters.) In a similar manner, a user could create loops which run a simulation for many iterations, varying some parameter until a termination criterion is met. This powerful scripting ability provides an intuitive substitute for hand-written scripts.

RUNNING A SINGLE ACCELERATOR SIMULATION

Although our ultimate aim is to perform start-to-end simulations, it is first instructive to explain how to set up and interpret a single simulation.

Once the AML data is created, it can be translated to the format of a specific accelerator code. However, AML only describes the accelerator lattice, and not information such as the choice of algorithm used in the simulation. Every code will also tend to have some beamline elements and options that do not fit within the AML scheme. These extra parameters are given to the function used to execute a single simulation in Figure 3. (Note that the additional input values pictured here are only required if no input particle distribution is given.) The user then has the option of visualizing the results of that simulation with standard plotting functions, such as those in the SDDS toolkit.

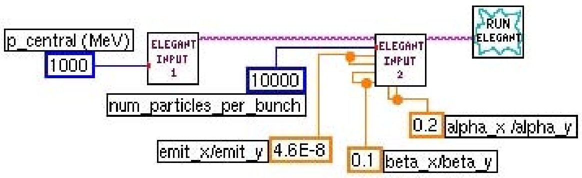

Figure 3: Additional inputs to Elegant

PERFORMING START-TO-END SIMULATIONS

A set of functions that define a start-to-end set of simulations can be chained together in a similar way as the elements of a single beamline. The output of each code is translated to a common format (SDDS), and translated once more to format expected by the next simulation in the chain.

A simple start-to-end simulation is pictured in Figure 4. The additional attributes for the "Run Parmela" and "Run Elegant" functions have been left out for clarity. In this figure, an AML lattice is loaded and then split into two subsets: one from 0 to 14.9 meters and one from 14.9 to 40 meters. The first subset provides the lattice for Parmela and the second becomes the input beamline for Elegant; the results of both runs are then plotted.

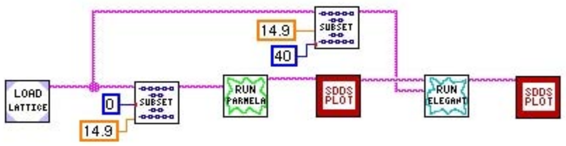

Figure 4: A simple start-to-end simulation

A start-to-end set of simulations must define a set of files and directories for running each simulation. In our scheme, a start-to-end project has a directory structure shown below:

```
project/
        project_input_file
        logfile
        common_files/
        elegant_1/
        elegant_2/
        parmela_1/
        parmela_2/
```

The file project_input_file serves as the input file to the AccelView environment. A new directory is created before running an instance of a particular accelerator code, for instance parmela_1/, parmela_2/, etc. These directories contain the phase space, the AML lattice description, and other input files needed for the simulation. The common_files directory contains input files that are imported from elsewhere into the project, such as RF cavity field maps. The master logfile contains the details of running each simulation, including any errors encountered.

Start-to-end simulations can be active for many days before the final results are available. During that time, the user can keep track of progress by using a solutions browser. This is a separate program that displays one entry per simulation, including the status, and has a set of buttons for displaying logfiles, intermediate phase-space files, and other data.

CLIENT-SERVER FUNCTIONS

Some accelerator codes have specific installation requirements in terms of the operating system version, as well as other software dependencies (compilers, utilities, etc.). This raises the possibility of having conflicting requirements between two or more of the codes in a start-to-end simulation. Sometimes, the most readily available way to perform start-to-end simulations in such a case is to manually transfer files from one system to another. For instance, a user might need to run the first part of a simulation using PARMELA on the local machine, and then the second part of the simulation using ELEGANT on a remote server. With our environment, this is taken care of automatically.

We have developed a software component that allows a code to seamlessly run remotely on a server machine. This is done by mirroring the run directories between the two machines before and after executing the simulation on the server. The method addresses security concerns by sending the user/password pair using RSA encryption. Our implementation of this is a stand-alone Python program that does not rely on some other underlying means of exchanging data such as SSH, which might not be convenient to install on all environments. This was done to facilitate installation on a diverse set of platforms.

This client-server approach is also useful for submitting jobs on a parallel computing cluster.

FUTURE DEVELOPMENT

At present, the project has support for ELEGANT and the UCLA version of PARMELA, but is in a pre-beta release stage. In the future we will improve the quality of the code, support more codes, and include several new features, as described below.

Several codes that are useful for simulating accelerator hardware need input of a 2-D geometry. This includes cavity eigenvalue codes, wakefield solvers, and magneto-static codes. Often these codes do not have a graphical front-end for specifying the problem geometry. In the spirit of being able to specify everything within an accelerator simulation through a graphical front-end, we plan to build a tool that can serve as an interface to several such codes. This approach is more efficient than having the ability to import 2-D contours from a CAD program, since an accelerator-specific 2-D geometry editor can be used to assign region properties and boundary conditions.

CONCLUSION

We have presented a way to link accelerator physics codes through a simple graphical user interface. This important tool can greatly simplify the construction of start-to-end simulations, while still providing the power inherent in its scripting ability.

The development goals have been to support multiple platforms and minimize software dependencies on those platforms in order to facilitate installation requirements.

In the future, we plan to extend the capabilities of AccelView to include the ability of specifying an entire accelerator project in a graphical interface. This includes specifying the 2-D geometry for programs like cavity RF and wakefield codes.

ACKNOWLEDGEMENTS

We wish to thank Dr. David Sagan of Cornell University for many helpful discussions regarding the AML project.

This work was supported by the Department of Energy, through SBIR grant number DE-FG02-06ER84465.

REFERENCES

[1] Information on the AML project can be found at: http://www.lns.cornell.edu/~dcs/aml/

[2] ELEGANT and the SDDS toolkit can be downloaded from: http://www.aps.anl.gov/asd/oag/SDDSInfo.shtml