Rapid Application Development Using the Tcl/Tk Language*

Johannes van Zeijts, CEBAF, 12000 Jefferson Avenue, Newport News, VA 23606

Abstract

During the last year high level applications at CEBAF were written using the Tcl/Tk scripting language [1]. This language is rapidly gaining in popularity, in part due to the ease of constructing programs with X11 graphical user interfaces, and in part due to the ease of adding compiled user code for specialized purposes. Extensions to the language provide object oriented programming [2], [3], which was used to develop a hierarchy of classes relevant for high level accelerator control.

We describe the basic language features, some 3rd party add-on packages, and local additions to the toolbox. Next we describe the features of the accelerator object hierarchy, and finally describe applications written using this toolbox such as the ModelServer prototype, Slow Orbit and Energy Lock, the Linac Energy Management System, and other applications.

I. Introduction

Tcl is an interpreted scripting language with only one data type: strings. The language is not intended for number crunching but rather to provide scripting control to user contributed compiled packages. Tk is one such package written by the author of Tcl. Tk provides a way to create spectacular X-11 graphic user interfaces in a minimal amount of time without any knowledge of underlying GUI library packages [4], [5]. Interface elements are simply addressed by a string name and can be placed on the screen and configured by scripts or interactively. A rich set of commands is available for event driven programming.

Locally developed code provides access to control system information, lattice simulation codes, and matrix calculations [6]. Additionally we use the 'expect' extension [7] to provide a programmed interface to existing 3rd party programs like Mathematica and Matlab, and the 'blt' extension [8] to provide line and bar graph support. Finally the Tcl-DP extension [9] was used for the network connections between server and client programs.

The goal of the object oriented approach in this context is to provide a set of classes with a well defined interface to allow non-expert users to create high level applications without having knowledge about the implementation of any of the lower level functionality. We provide two sets of class hierarchies, one for beam line elements like correctors and beam position monitors, and one for applications like orbit and energy lock, and autosteering. Additionally, the object oriented extensions provide a clean way to produce and maintain large amounts of code.

II. Accelerator Objects

Accelerator beam line elements are well described in a hierarchical way. An 'Element' class has placeholders for properties common to all beam line elements like position and lattice function information. 'Magnet' and 'Bpm' classes specialize the Element class and add element specific information. 'Dipole', 'Corrector', and 'Quad' classes are again a specialization of the 'Magnet' class. Tables with live information about beam line elements can be generated with just a few lines of code. Here we show the beam position panel (1) and the corrector panel (2). User defined columns can be easily created and filled.

BPM		Actions		Selections
Name	on	pos	beam status	gold
IPM1E02	x	0	-666	0.0066
0	y	0.00	No Beam	0.0415
IPM1A01	x	0	-666	-0.2253
0.00	y	0.00	No Beam	0.1482

Figure. 1. Beam Position Monitor Panel

Corrector		Actions		Selections
Name	on	bdl	kmol	angle
MBT1L03H	h	0.000		0.0
MBT1L03V	v	82.243		0.394585
MBT1L04H	h	20.836		0.0789287
MBT1L04V	v	0.000		0.0
MBT1L05H	h	0.000		0.0
MBT1L05V	v	87.455		0.255759
MBT1L06H	h	-8.469		-0.0206492
MBT1L06V	v	0.000		0.0

Figure. 2. Corrector Magnet Panel

III. Application Classes

Application classes are less obvious. In this case we have decided to provide a basic 'BpmLock' class which handles all basic actions of high level application which concern bpms, like selecting, reading gold values, beam loss checks, etc. The 'CorrectorLock' class inherits all this from the bpmlock class and adds all actions concerning correctors. At this level virtual functions are introduced for building response matrices, solving for new correctors, checking solutions, setting correctors, and communication with the Matlab based compute server. These methods

*This work was supported by the U.S. Department of Energy, under contract No. DE-AC05-84ER40150.

provide the basic functionality needed for an orbit lock application but are redefined in more specific classes like 'autosteering'. At the same level, inheriting from the bpmlock class, we provide a class for controlling cavities. This one provides the basic functionality needed for an energy lock application. New high level applications like Quad Centering simply define a new class deriving from the correctorlock class and add their specific methods [10]. A new autosteer program for the CEBAF arcs was brought on line in a very short time by inheriting from the existing autosteer class and only redefining a very limited number of methods responsible for the correction algorithm [11].

IV. Linac Energy Management

This application is required to calculate gradients for the RF Cavities in the CEBAF linacs given a requested energy increase. After successfully setting the gradients, the program is to set the quadrupoles in the linac to provide either a 60 or 120 degree fodo lattice. The main extra input to the program is a 'fudge factor' which accounts for non-crested cavities and gradient calibration errors. The program has been operational for more than a year, with only slight adjustments to the exception handling over this period. Figure (3) gives the main control panel.

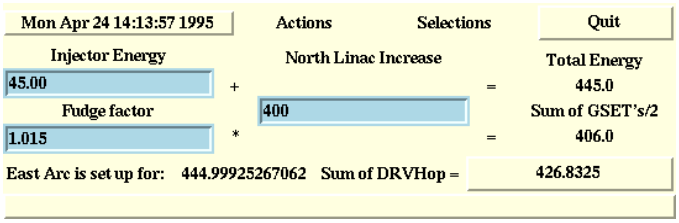

Figure. 3. Linac Energy Management Control Panel

V. Model Server

All optics related information at CEBAF is concentrated in a 'Model Server' application. This application is to store all relevant beam line element information and is to serve up transfer and response matrices between two arbitrary points in the machine to requesting client applications on the control network. The optics calculations are usually performed by Dimad [12] but hooks are available to include space charge and polarization codes.

Applications connect to the model server by declaring a 'Model Client' object. This object supports query calls to retrieve beam line elements and provides methods to retrieve response matrices. It is also the conduit through which beam line elements get element specific information from the model server like layout and lattice functions. As an example we provide the code used to produce figure (1):

```
ModelClient new
Bpm :: addlist [new elements bpms arc1]
Bpm :: on pos
Bpm :: Window .b
.b on pos status gold
```

The model server does not have a graphic user interface, but a 'model sniffer' application is available in the control room to update the information after lattice changes, see figure (4). Producing the model server in Tcl with lattice information for

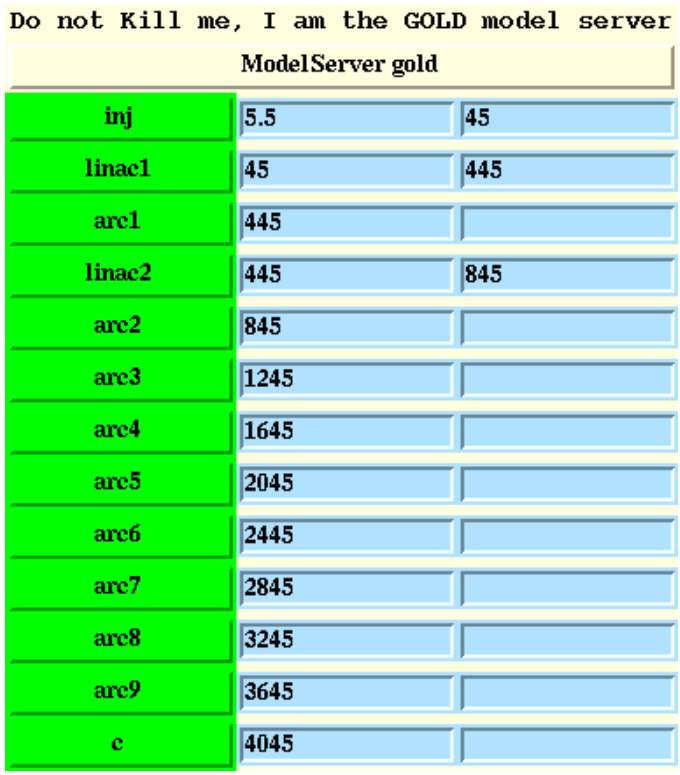

Figure. 4. Model Server Status Panel

the whole machine turned out to be a bit of a challenge due to the memory overhead of the '[incr tcl]' object package. Process sizes routinely reached the machine limit. For this and other reasons the model server application was the first chosen to be converted to C++ [13]. That application will maintain the existing model server in functionality and interface, and is expected to improve memory usage and data throughput by a significant amount.

VI. Slow Orbit and Energy Lock

One of the most important high level applications produced were the slow feedback loops. Their task is to provide a means to obtain reproducible results for the setup of linacs and arcs. They maintain a 'golden' orbit at certain beam position monitors and maintain the design energy by obtaining an energy offset value from the beam position monitors throughout the arcs. On start up, response matrices are collected from the model server and dispatched to the matrix package. Singular value decomposition is used to obtain correction values for orbit and energy while at the same time protecting against singular equations. A 'memory' factor is introduced to facilitate the running of several locks simultaneously without introducing spurious oscillations. Figure (5) shows the control panel for the energy lock.

VII. Auto Steering

Linac autosteering is required to on demand steer the beam to golden values while maintaining small corrector values. The algorithm is identical to the orbit lock application and the corrector

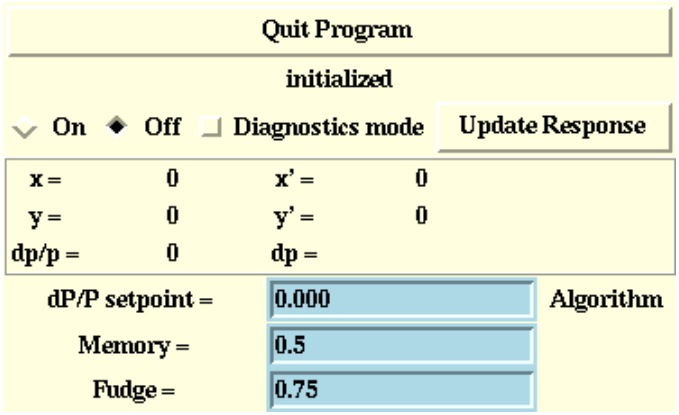

Figure. 5. Control Panel for slow Energy Lock

goal has been achieved by only using correctors at positions with large beta functions. As usual the main exceptions are in detecting and handling of malfunctioning beam position monitors. A typical resulting corrector pattern is given in figure (6). The Arc Auto Steering algorithm is based on the linac steering but uses a much more complex algorithm [11], executed by Mathematica. A typical corrector change chart is shown in figure (7).

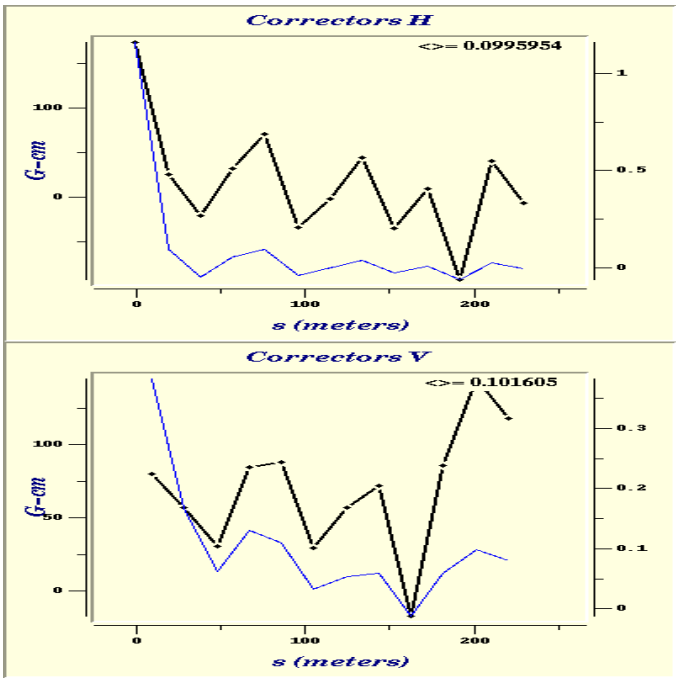

Figure. 6. Corrector Display, in Gauss-cm and mrad

VIII. Conclusion

We have found the Tcl/Tk environment to be an ideal tool for rapidly producing fully functional prototypes of high level applications. Emphasis of application development can be put on the physics of the problem and on the exception handling, since long learning curves for windows programming are cut out of the process.

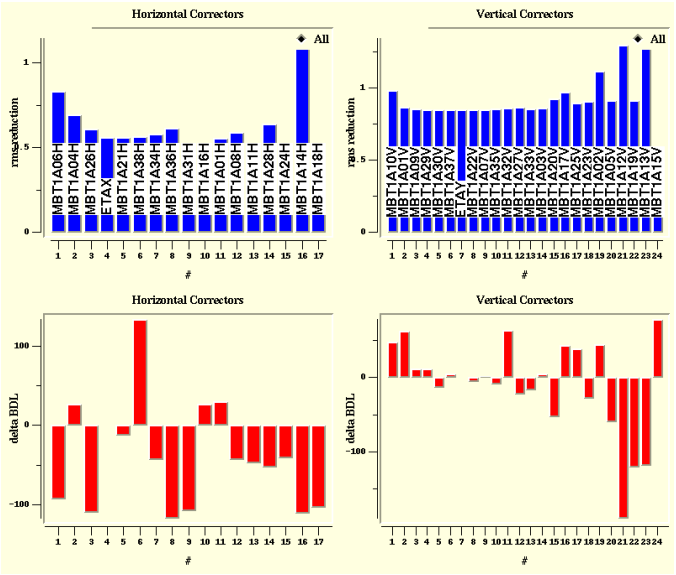

Figure. 7. Arc AutoSteering corrector change barcharts

References

[1] J. K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley Professional Computing Series (1994)

[2] M. J. McLennan, *[incr Tcl]: Object-Oriented Programming in Tcl*, Proceedings of the Tcl/Tk Workshop, UCB, June 10-11, 1993.

[3] IXI Limited, *Object Tcl*, http://www.x.co.uk

[4] B. Welch, *Practical Programming in Tcl and Tk*, Prentice Hall (1995)

[5] Tcl/Tk is available from ftp.aud.alcatel.com:/tcl

[6] J. van Zeijts, *High Level Application Prototyping*, CEBAF-TN-94-038

[7] D. Libes, *Exploring Expect: A Tcl-based Toolkit for Automating Interactive Programs*, O'Reilly and Associates, Inc. (1995)

[8] G. Howlett, *Bacon, Lattice and Tomato graphs for Tcl*

[9] L. A. Rowe et.al., *Tcl Distributed Programming*

[10] R. Li, *Quad Centering*, CEBAF MCC procedure

[11] Y. C. Chao, *An Orbit Correction Algorithm for General Beam Lines*, submitted to N.I.M.

[12] R. Servranckx et. al., *DIMAD manual*

[13] B. Bowling et. al., *Integrated On-Line Modeling at CEBAF*, these proceedings.