# ACCELERATOR MODELING ENGINE FOR CONTROL SYSTEMS*

H. Nishimura[1], C. A. Timossi[2], M. E. Urashka[3], LBNL, Berkeley, CA 94720, U.S.A.
T. Kosuge[4], K. Nigorikawa[5], KEK, Tsukuba, Ibaraki 305-0801, Japan

*Abstract*

An accelerator modeling engine is an application program interface (API) designed to be compatible with control systems and covers the field of accelerator modeling and simulation by behaving as virtual accelerators. We have created by making a DLL layer on top of the existing C++ class library called Goemon. A compatibility with COACK, STARS and EPICS is demonstrated by examples.

## SYMMETRY OF CONTROL SYSTEMS

Most of the traditional control systems distinguish console programs from their target devices (Fig.1A). Therefore, it is not trivial to introduce a physics program as a virtual device as the model must behave like a device while running as programs. On the other hand, COACK[1] and STARS[2] are symmetric in treating console programs and devices at the same level(Fig. 1B). As a result, they can easily accept physics programs as virtual devices. One of these virtual devices can be an accelerator modeling engine that emulates a target accelerator.

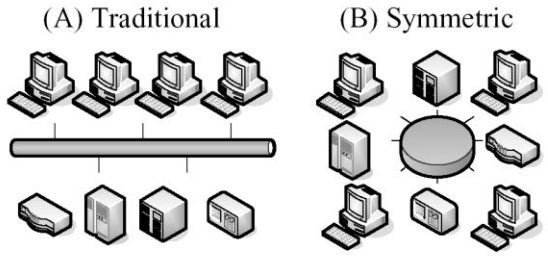


Figure 1: Symmetry of the Control System Architecture.

A symmetric architecture can be also useful to develop a virtual machine itself as it provides a hub to combine multiple programs with or without the connection to the control system.

## MODELING ENGINE

### *Roles*

Accelerator controls are becoming more and more sensitive to various parameter changes and are required to be extremely accurate. It is true that the sensitivity (response) matrices cover a wide range of the operations without using a model where linearity is assumed. However, it assumes a circulating beam behaving well enough to measure the matrices. Otherwise, we must use a model to operate the machine effectively. In such cases, the availability of virtual accelerators becomes critical for precise machine controls. An example is a commissioning of a new accelerator or a new lattice with severe boundary conditions where a circulating beam may not be expected.

Once a reasonable circulating beam is established, a virtual machine cooperates with the sensitivity matrices to characterize the accelerator optics.

### *Definition*

A modeling engine is an API (application program interface) of an accelerator physics modeling and simulation library tailored for the use in control systems. The templates of its client programs should be included to indicate its proper usage.

### *Requirements*

A modeling engine is for the real machine controls. Therefore, it would be reasonable to require the following items.

- Simple to understand, develop and maintain.
- Compatibility with the control system.
- Fast execution speed.

Most of the time, a modeling engine will be built on top of an existing program or a library. In such cases, it is important to design the API to hide the complexity of the original layer. It will also help to make the link to the control system simple. Fast execution speed is always crucial not to be a bottleneck of the control system.

Compatibility with the control system primarily means two items:

- Connectivity.
- Thread safety.

Connectivity depends on the control system. It may be a matter of simple static link, or of a modern infrastructure of a distributed system. In this paper, as we focus on Windows as the platform, connectivity means a local connection to the programs that are already a part of the control system.

Thread safety is required because the control system is always multitasking and/or multithreading. Thread safety is also required to be able to instantiate multiple objects of the model, which we discuss later.

However, these two issues may not be trivial in case of accelerator physics programs. Most of them are not libraries but stand alone programs. We cannot expect

*Work supported by the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. [1] H_Nishimura@lbl.gov.
[2] CATismossi@lbl.gov. [3] MEUrashka@lbl.gov.
[4] Takashi.kosuge@kek.jp, [5]kazuyuki.nigorikawa@kek.jp

these programs to have any multitasking capabilities. For example, if global variables are used, it will probably not be thread safe.

## *Example*

We have created a modeling engine for the ALS storage ring by using an existing C++ class customized for the ring. This class is derived from a generic ring class of the Goemon C++ library [3].

. Here are the major functions of Goemon.

- Linear optics calculations by using the standard matrix formalism.
- Advanced nonlinear optics calculations by using the full 6-dimensional routines.
- Orbit calculations and adjustments.
- Various parameter fittings.
- Simulations of realistic errors.

Flexibility and performance have been more important than simplicity in the development of Goemon. It has a range of analysis routines that are not usually required for a practical modeling. We chose a subset of member functions of the ALS storage ring class, and parameterized them to reduce the number of public routines. The result is about 50 routines that are to be sufficient in a practical modeling.

Design issues are:

- Enable multiple instances of the virtual machines.
- C API to hide the C++ hierarchy.
- Primitive types for parameters.
- Implement as dynamic link library (DLL) in which virtual machines are allocated.

The capability of multiple instances is a natural consequence of using a class for as accelerator. It makes modeling efficient and modularized. For example, you can create a virtual machine with an ideal lattice as a reference, and work on the one with realistic lattice. It is also possible to prepare virtual machines for multiple rings and beam lines and use them in parallel.

C++ classes are not easily usable from other languages. Using a C API is inevitable to make the library easily accessible from various languages. To make the parameter passing simple, we use only primitive types for API.

On Windows, we use DLL. We allocate the objects in the DLL memory space, not in the client side. Then, the DLL gives routines to operate these objects by hiding the complex details.

The result is a DLL version of Goemon for the ALS storage ring exporting C API to cover the following:

- Manipulate all the magnet settings (58 quads, 4 bends, 2 sextupoles, 24 skew quads, 164 steerings)
- Manipulate various magnet errors, including field errors, misalignments and tilts.
- Optics and orbit calculations including fittings.
- BPM emulations ( 96 BPMs )
- Various particle tracking routines.

## *Client Programs*

The DLL described above is language independent on Windows. Any language that supports DLL can be used in client programs. However, to make this process efficient, we have created import libraries and its wrapper classes to reconstruct a proxy class of the ring in several client languages.

Fig.2 shows the comparison of normal use of the C++ library Goemon directly in C++ (A) and through the DLL by various languages using the proxy class in each language. An example of Visual Basic 6.0 (VB6) is described in the next chapter.

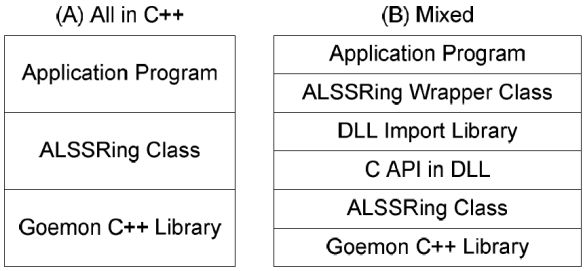


Figure 2: Goemon and Application Programs.

## *Example of Using the C API*

Here is an example of a C program using the C API in the DLL. The naming convention of the routines of Goemon is same as that of Tracy/Tracy2[4].

```
int SR1=SRCREATE(1);// (1)
GETTWISS(SR1,1);//(2)
FITTUNE(SR1, 14.25, 9.20);//(3)
SETQ(SR1, 2,7,1,GETQ(SR1,2,7,1)*1.02);//(4)
SETSTEER(SR1,0,27,0.001);//(5)
GETCOD(0.0);//(6)
double x=GETBPM(SR1,0,48);//(7)
```

This short example does the following:

(1) Create a virtual machine indexed as 1.
(2) Calculate linear optics (beta and dispersion)
(3) Fit betatron tunes to (14.25, 9.20).
(4) Increase the strength of SR07C QD2 by 2%.
(5) Set the 27th horizontal steering to kick 1 mrad.
(6) Calculate the COD for an on-energy particle.
(7) Read the horizontal orbit at the 48th BPM.

## *Example of Using the C++ Wrapper*

This example becomes with the use of a proxy class as:

```
ALSRing *SR1=new ALSRing();
SR1->getTwiss(1);
SR1->fetTune(14.25, 9.20);
SR1->setQ( 2,7,1,SR1->getQ(2,7,1)*1.02);
SR1->setSteer(0,27,0.001);
SR1->getCOD(0.0);
double x=SR1->getBPM(0.48);
```

The conversions between the engineering units and the physical units are not in the engine but stay outside as they are always required regardless of the availability of the virtual machines.

We have implemented a proxy class in various compilers including .NET languages. Java is also covered by using the Java Native Interface (JNI). An example in Visual Basic 6.0 is shown in the next chapter.

### *Example of Using the DLL in Python*

Matlab has been demonstrated the usefulness of an interactive environment[5]. In case of Goemon in DLL, something similar can be realized by using Python. There are several modules available that allow Python scripts to make function calls to C language DLLs. The ctypes module[6] makes this task extremely simple and provide for the ability to program interactively. Here is a part of the example in Python:

```
from ctypes import *
import types
gd = CDLL("gemCdll.dll")
SR1 = gd.SRCREATE(1)
gd.GETTWISS(SR1,1)
nux = c_double(14.25)
nuy = c_double.(9.20)
gd.FITTUNE(SR1, nux, nuy)
gd.GETQ.restype = c_double
c = c_double(gd.GETQ(SR1,2,7,1) * 1.02)
gd.SETQ(SR1,2,7,1, c)
```

One important note though, when using this library it is important to keep in mind one must specify the return types to expect from a function call. Also, when passing values as parameters one must be careful to match the type if it is something other than an integer or a string. So if a function expects a C double, it is necessary to cast it as a ctypes c_double. However, Python is object-oriented therefore these issues can be hidden by introducing a wrapper class, as the following example shows.

```
class SRRing:
  SR = 0
  def FitTune(self,Nux,Nuy):
    goemonDLL.FITTUNE(self.SR,
      c_double(Nux),c_double(Nuy))
  def GetQ(self,typ,sec,n):
    goemonDLL.GETQ.restype = c_double
    return goemonDLL.
                    GETQ(self.SR,typ,sec,n)
    # Lines omitted here after
```

Let us assume that a Python script file GemCDLL.py contains the class partially listed above. Then, we can access the DLL in an interactive mode. The example below shows how two instances of the virtual machines can be used in an interactive Python environment.

```
>>> import GemCDLL
>>> SR1=GemCDLL.SRRing(1)
>>> SR2=GemCDLL.SRRing(2)
>>> SR1.GetTwiss(0)
>>> SR2.GetTwiss(0)
>>> SR1.FitTune(14.25,8.20)
>>> SR2.FitTune(14.25,9.20)
>>> print SR1.GetQ(1,1,2)
2.20136629155
>>> print SR2.GetQ(1,1,2)
2.22459073967
>>>
```

### *Modeling Engine in an ActiveX control*

Once the C API is established, it is straightforward to create an ActiveX control containing it. We have created an ActiveX control by using the active template library (ATL). This will be useful when creating graphical applications in various tools. Fig.3 shows the ActiveX control on a LabView block diagram showing 15 of total 54 methods.

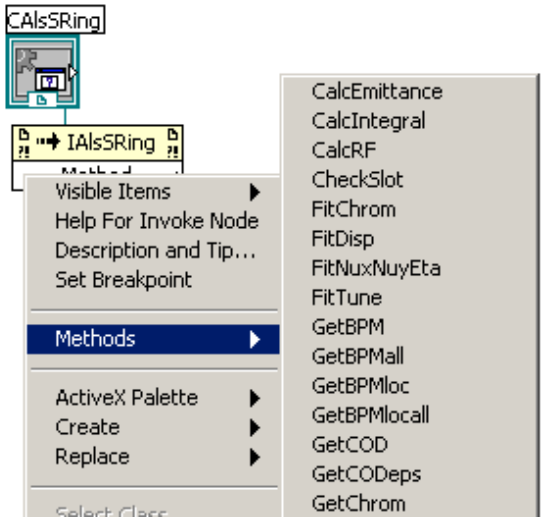


Figure 3: ActiveX Control on LabView.

## MODELING IN CONTROL SYSTEMS

### *COACK*

COACK is a framework for a control system based on the Windows architecture. It has a symmetric architecture; therefore it should be simple to integrate a modeling engine. Although COACK is a system for VB6 programs, we have two options: one to use the DLL version, or the ActiveX controls. Here we use the first option.

VB6 is not objected-oriented but object-based, therefore we can write a wrapper class in VB6. A class in VB6 corresponds to a separate class module. This is also the

place where we can import the DLL routines. Here is how the example was rewritten in VB6 by using the class.

```
Dim SR1 As New ALSSRing
Call SR1.Init(1)
Call SR1.GETTWISS(0)
Call SR1.FITTUNE(14.25, 9.2)
Call SR1.SETQ(2,7,1,SR1.GETQ(2,7,1*1.02))
Call SR1.SETSTEER(0, 27, 0.001)
Call SR1.GETCOD(0)
x=Call SR.GETBPM(0, 48)
```

A real connection to COACK requires links between the DLL routines and the event handling routines associated with the COACK XML data describing the device class of the target accelerator.

### *STARS*

STARS is a simple and lightweight system written Perl and uses TCP/IP therefore it is cross-platform by its nature. It also uses the symmetric architecture and comes with the bridge to COACK. As it does not require any special set up of the environment except the availability of Perl and TCP/IP, it works very efficiently for small systems including the development of the modeling engine itself without the real connection to the control system.

Both STARS servers and clients are in Perl. We must use the STARS C API to link to external routines. We created a C++ class (Stars) on top of the C API and used it to connect the modeling engine. In Fig.4, "interface to STARS" is this layer. It is completely hidden from the application programs.

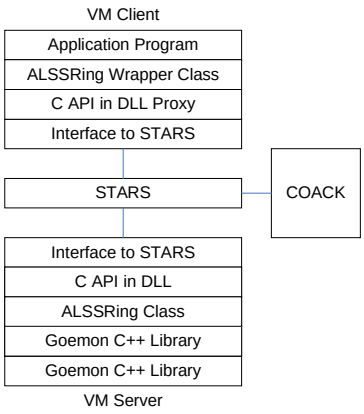


Figure 4: Virtual Machine and STARS.

### *EPICS*

EPICS[7] is the new control system being to which the ALS control system is migrating. Most of the interactive GUI programs are on the Windows consoles, while automated process control programs are in Matlab[5] running primarily on Unix/Linux workstations. Here we focus on the Windows environment.

EPICS has a traditional architecture that is not symmetric (Fig.1A). However, it is made simple to create a virtual machine on Windows by using an ActiveX component (SCAS)[8] supporting functions of a channel access server. There is also an ActiveX component (Scacom)[9] for the simple channel access clients. By adding the server component to the program using the DLL, we got a modeling engine compatible with EPICS.

## CONCLUSION

It is important to prepare a modeling engine as a C API to be compatible with control systems. In case of developments on Windows, DLL is the adequate form of the library with an option of ActiveX controls. Various programming languages can be supported very efficiently by providing import libraries. In using it in a control system, a symmetric architecture of COACK and STARS are very beneficial for modeling. In case of EPICS, two ActiveX controls SCAcom and SCAS are crucial.

## AKNOWLDEGEMENTS

We thank D. Robin and Alan Biocca at LBNL, and S. Kurokawa at KEK for their encouragements.

## REFERENCES (NOT COMPLETED)

[1] I. Abe, et al, PAC'00, Hamburg, October 2000.
[2] T. Kosuge, et al., "COACK MULTI-SERVER SYSTEM WITH STARS",
[3] H. Nishimura, PAC'01, Chicago, July 2001, p.3066. PCaPAC2002, Frascati, October 2002
[4] H. Nishimura, EPAC '88, 803,1989.
J. Bengtsson, E. Forest and H. Nishimura, "Tracy Users Manual", unpublished.
[5] G. Portmann, this conference.
J. Corbett, A. Terebilo, G. Portmann, IEEE PAC’03, 0-7803-7739-9, p2369, 2003
G. Portmann, J. Corbett, A. Terebilo, "An Accelerator Control Middle Layer Using Matlab Manual," to be published in IEEE PAC’05
[6] http://starship.python.net/crew/theller/ctypes/
[7] L.R.Dalesio, et al., ICALEPCS '93, Berlin, Germany, 1993.
http://www.aps.anl.gov/epics/
[8] C. Timossi, unpublished
[9] C. Timossi and H. Nishimura, IEEE PAC’97, 0-7803-4376-X/98, p805, 1998
http://www-controls.als.lbl.gov/epics_collaboration/sca/win32