TRACY#*

H. Nishimura
Lawrence Berkeley National Laboratory, Berkeley, CA 94720, U.S.A.

Abstract

Tracy# is a C# class library for single particle beam dynamics in full 6-dim canonical phase space. The code is based on Goemon that is a C++ version of the Tracy2 library. This paper describes the new features in Tracy# from a software engineering aspect.

INTRODUCTION

During the ALS[1] design phase, Tracy[2] was developed in Pascal for modeling and tracking studies by using embedded Pascal compiler/interpreter to parse the user logic. It evolved to a 6-dimensional version called Tracy2[3]. Subsequently its accelerator physics library was separated from Tracy and ported to C/C++ independently at multiple light sources, including SLS[4], Diamond[5], Soleil[6] and NSLS-2[7], mostly for model-based accelerator controls. At the ALS, the library was re-written in C++ as Goemon[8], which has now been re-written in C# [9] and called Tracy#.

FEATURES

Library Layers

Tracy# is a library that implements single particle beam dynamics for modeling, simulation and analysis studies.

An application built with Tracy# has 3 layers:

- Physics layer, which is Tracy# itself.
- Accelerator layer to model particular accelerators.
- Application layer.

The Physics layer is for beam dynamics and uses the following integrators:

- The 4x5 linear matrix formalism.
- The 2nd and the 4th-order symplectic integrators[10] in 6-dim.
- K-pot Hamiltonian[11] that models small rings properly.

In Tracy# these integrators were implemented in C# taking advantage of some of the important language features. For example, operator overloading is used for:

- Vector and matrix calculations.
- Differential algebra (DA) [12].
- Lattice definitions.

The second layer is to model particular accelerator structures. This is the place where virtual accelerators are built in forms of C# classes. Starting with an ideal lattice, a virtual machine is enhanced to be a practical one that provides realistic error emulations, various control knobs, and customized physics routines to calculate a range of quantities including dynamic apertures.

The application layer is for client application programs that access the virtual accelerators. These programs can use any features of the .NET libraries that cover database access, XML, networks and graphics. Tracy# is designed to be compatible with these standard .NET libraries. Actually, Tracy# needs only one external routine that is for the singular value decomposition to invert the large sensitivity matrices. We chose an open–source math library in C#[13].

Environment

Tracy# is built on the .NET Framework 3.5, and works on Windows XP SP2 as well as Vista. The development environment is Visual Studio 2008. The programming language for the physics layer is C# 3.0. The client programs are also developed in the same development environment and usually in C# 3.0. However, they can be in other .NET languages, such as Visual Basic.NET or IronPython[14] that we mention later.

Tracy# can also work non-Windows platform by using MONO[15] that is an independent, open-source implementation of the .NET Framework that covers various platforms, including Linux, Solaris and Macintosh. As Tracy# itself is a plain C# code, it is compatible with MONO. It is also possible to make its application program portable by carefully choosing the graphics components for GUI programming.

Implementation

As mentioned, Tracy# uses advanced features of the .NET Framework. In particular, .NET generics proved invaluable especially List< >. In case of Goemon in C++, we did not use C++ generics called template to keep the compile-and-link time reasonably short.

Porting the routines for vector/matrix and DA was not trivial. Goemon used local variables allocated on the memory stack to carry out calculations rapidly. This trick in C/C++ does not work with C#. Therefore the C# routines needed fine tuning to restore execution speed[9]. Currently, we are rewriting routines further to enable parallel computing as mentioned later.

Graphics is not a core part of Tracy#. Instead, the accelerator layer uses it extensively. The graphics programming was in WinForm of .NET 2.0, and is migrating to Windows Presentation Foundation (WPF) of .NET 3.0.

Tracy# also uses relational databases and XML as described in the following sections.

*Work supported by the U.S. Department of Energy under Contract No. DE-AC02-05CH11231

DATABASE

There has been an increasing need for data management to track lattice configurations, simulation conditions, and the results of calculations with proper context. Tracy# makes use of standard SQL and XML for these management tasks. Both technologies are well supported on .NET.

Currently, Tracy# uses ADO.NET 2.0 for database access. Its DataSet component is a kind of on-memory database embedded in a client program. Its external data source can be almost any major SQL database including MySQL.

At the ALS MySQL is the most common database system. It is used by controls applications including various Matlab programs dealing with automated machine control[16]. Tracy# reads tables produced by these programs to retrieve the current operational reference values. For example, there is a table for the measured sensitivity matrix produced by a Matlab program. This is the table most referred by Tracy#.

Tracy# usually restores the data of a database table to an ADO.NET DataTable object in memory, and then they can be stored in other database, or to XML. This part is not limited to any particular database system.

XML

Tracy# uses XML extensively to manage its own data through a new API called LINQ to XML[17]. The merit of using XML rather than SQL is the flexibility to support complex data structures. One XML file can replace multiple SQL data tables and relations among them. Although XML may look complicated, it simplifies the data management significantly. This is especially true with LINQ to XML as it provides an API that is much simpler than other XML APIs like DOM or SAX. For example, an XML node can be created independently from the XML document, and later it can be inserted to another XML node that may already be in the XML document. Therefore, we can even use an XElement object like an ordinary variable. Here XElement is a new class of LINQ to XML that supports XML nodes.

Currently, Tracy# uses XML for the following functions:

- Lattice description
- Algorithm description
- Data storage

Lattice Definition in XML

In constructing a virtual accelerator using Tracy#, a lattice structure is typically defined in the creator of its custom ring class by using operator overloading like: SEC=SYM+L1+QF+L2+BL2 where SEC is a new beam line object and the right-hand side is a series of either single elements, such as magnets, or lines of them. So the lattice is defined in C# which allows the lattice to be defined in a very detailed fashion.

There is also a complementary way of defining a simple lattice in a descriptive manner. For example:

```
<Lattice Name="BR0">
  <Element Class="Quad" Name="QF" L="0.15"
           K="2.62488449385119" />
  <Element Class="Quad" Name="QD" L="0.1"
           K="-2.44458454557616" />
  <Element Class="Bend" Name="B" L="1.05"
           K="0" T="15" T1="7.5" T2="7.5" />
  <Element Class="Drift" Name="L1" L="0.5469" />
  <Element Class="Drift" Name="L2" L="0.4969" />
  <Element Class="Drift" Name="L3" L="2.0938" />
  <Eline Name="S1" Line="QD, L1,B,L2,QF"/>
  <Eline Name="S2" Line="QF,L3, QD"/>
  <Eline Name="S3" Line="QD,L1,B,L2,QF "/>
  <Eline Name="SGA"  Line="S1,S2,S3,-S1"/>
  <Element Class="Marker" Name="SYM"/>
  <Eline Name="SEC" Line="SYM,SGA,-SGA" />
</Lattice>
```

Tracy# accepts such a lattice defined in XML that follows the same convention of defining a lattice in Tracy2 and Goemon. Without writing a complex lattice parser, or defining a special grammer, we can use XML to define a lattice as a data input to Tracy# at runtime.

XML can also be used to define a very complex lattice. We had such scenario with the radiation safety studies for top-off injection at the ALS[18]. This was a tracking study far off the reference orbit. The horizontal offset can be over 50 cm which is enough to bypass the bending magnet (the vacuum chamber has an ante-chamber that allows this path). Therefore magnetic field profiles and the chamber apertures must be included in the lattice. We created a special version of Tracy# for this study. Transferring the lattice in XML, extra nodes were added to assign the magnet field profiles and chamber geometries. XML was flexible enough to accommodate complexities of the lattice definitions.

Simulation Logic in XML

Similar to the case of a simple lattice definition in XML, simple simulation logic can be described in XML. Here is an example of initializing the ideal lattice of the ALS Booster Ring that uses the lattice just defined above.

```
<Ring Name="BR0" Lattice="BR0" ELine="SEC"
                 Symmetry="4">
  <Energy Value="1.9E9" />
  <GetTwiss Value="1" />
  <SetQforTune QF="QF" QD="QD" />
  <FitTune Nux="5.25" Nuy="2.75" />
</Ring>
```

This example shows that XML can describe simple logic in a reasonable manner.

Note that XML is used for both lattice definitions and algorithms, which means that they can be in the same XML document. Currently, we have very simple parsers

that use LINQ to XML, and add to their functionality whenever reuired.

XML also served an important role for providing very complex logic as in the case of the top-off radiation safety studies. There was no established methodology from other accelerators that could be applied to the ALS. Therefore the specification of the safery calculation evolved as our understanding of the problem grew. The flexibility of XML was crucial to accommodate the evolving nature of the simulation.

XML for Data Storage

Tracy# uses XML as its primary data format of files. The most recent example is for our sextupole upgrade project[19] to reduce the storage ring emittance down to 1/3 of the current value. We are in a process of optimizing the lattice with two new families of sextupole magnets. There is a need to store various lattice configurations, the result of simulations for each one of them, and retrieve the stored content. The resulting XML file contains over 1 million lines. We wrote several client programs of Tracy# to read and update the XML file.

An interesting observation is that there is no real distinction between input and output data in the case of XML. This is similar to the case of using a relational database. A program reads what it needs from the XML files, avoiding redundant calculations and keeping the previous context, then carries out certain tasks and updates XML by changing the attribute values, and also adding new nodes. This means that it grows as the simulation evolves. If we use a relational database, the process of adding nodes corresponds to modifying a table design (schema) or creating a new table with new relations which will be far more complex than a case of XML.

Potential problems with XML can be its file size and processing speed. Fortunately, we have not yet experienced any significant limitation.

The XML effort happened first with the control system upgrade in C#[20]. Tracy# is benefitting from our previous and continuing work on the control system. Routines for database access using ADO.NET, and graphics programming in WPF that were first used for the machine controls have also been migrated to Tracy#. Concerning resource sharing, our C# development is catching up with Matlab that has been used for both physics[21] and controls[16] at the ALS.

INTERACTIVE SCIPTING

Interactive scripting offers a convenient option to the traditional software development cycle which includes editing, compiling before execution. Our early attempts were TracyM and TracyML[22] that have been superseded by AT[16] in Matlab. An inconvenience is that an interactive programming language isolates itself and requires specially compiled modules to access external routines that are written in compilers. However, on the .NET Framework, it can access the .NET libraries, including Tracy#, normally and directly without any extra layer.

IronPython Example

Below is an example of using Tracy# interactively from IronPython[14]. Invoking IronPython, establishing the link to the Tracy# library, and importing its name spaces, we can use all the routines directly without any special layer. This example calculates the emittance of the nominal and the new low-emittance lattices of the ALS storage ring. (>>> is a prompt.)

```
>>> import sys
>>> sys.path.append('T:/Tracy/IronPython')
>>> from ipTracy import *
>>> SR=ALSSRW()
>>> SR.getTwiss(1)
True
>>> SR.fitNuxNuyEta(14.25, 9.20, 0.06)
True
>>> SR.getTwiss(1)
True
>>> SR.calcIntegral()
>>> SR.calcEmittance(1.9E9)
>>> print 'Emittance=', SR.NtlEmittance
Emittance= 6.8094734722e-009
>>> SR.fitNuxNuyEta(16.25, 9.20, 0.15)
True
>>> SR.getTwiss(1)
True
>>> SR.calcIntegral()
>>> SR.calcEmittance(1.9E9)
>>> print 'Emittance=', SR.NtlEmittance
Emittance= 2.16587971373e-009
>>>
```

Here T:/Tracy/IronPython is the location of the directory of the Tracy# library and the module ipTracy.py contains lines to import the several .NET assemblies of Tracy#. This kind of capability of interactive scripting is one of the major merits of the .NET Framework.

ONGOING EFFORT

There are multiple efforts in progress to upgrade Tracy# by taking newer functions of the .NET Framework. The most important one is the parallelism to make use of modern multi-core CPUs. We have just added new routines to track particles simultaneously and testing it[23]. A parallelized for statement has tripled the execution speed in case of tracking for dynamic aperture calculations on a PC with a quad-core CPU.

AKNOWLEDGEMENTS

The author thanks C. Steier for his encouragement and useful advices. He also appreciates C. Timossi for technical discussions on computational issues and useful comments.

REFERENCES

[1] LBL PUB-5172 Rev. LBL, 1986
A. Jackson, PAC'93, Washington, D.C, USA, p.1432, (1993)
[2] H. Nishimura, EPAC'88, Rome, Italy, p.803(1989)
[3] J. Bengtsson, E. Forest and H. Nishimura, 'Tracy User Manual", unpublished, ALS, LBNL
[4] M. Böge, J. Chrin, ICALEPCS'01, San Jose, USA, p.430 (2001)
[5] M. Heron, et al, EPAC'06, Edinburgh, UK, p.3068 (2006)
[6] P. Brunelle, A. Loulergue, A. Nadji, L.S. Nadolski, EPAC'04, Lucerne, Switzerland, p.2032 (2004).
[7] L. B. Dalesio, ICALEPCS'07, Knoxville, USA, p.253(2007)
[8] H. Nishimura, PAC'01, Chicago, USA, p.3006, (2001)
[9] H. Nishimura and T. Scarvie, EPAC'06, Edinburgh, UK, p.2263 (2006)
[10] E. Forest and R. Ruth, Physica D, vol. 43(1) p.105(1990)
[11] E. Forest, “Beam Dynamics, A New Attitude and Framework”, Chap.12, 1998, Harwood Academic Publications
[12] M.Berz, SSC-152, 1988
Leo Michelotti, IEEE PAC89, CH2669-0(1989)839.
N. Malitskey, A. Reshetov and Y. Yan, SSCL-659, 1994.
[13] P. Selormey, "DotNetMatrix: Simple Matrix Library for .NET", http://www.codeproject.com/KB/recipes/psdotnetmatrix.aspx
[14] http://ironpython.codeplex.com
[15] http://www.mono-project.com
[16] J. Corbett, G. Portmann, A. Terebilo, PAC'03, Portland, USA, p.2369(2003)
[17] http://msdn.microsoft.com/en-us/library/bb387098.aspx
[18] H. Nishimura, R. J. Donahue, R. M. Duarte, D. Robin, F. Sannibale, C. Steier, W. Wan, PAC'07, Albuquerque, USA, p.1173(2007)
[19] H. Nishimura, S. Marks, D. S. Robin, R. D. Schlueter, C. A. Steier, W. Wan, PAC'07, Albuquerque, USA, p.1170(2007)
[20] H. Nishimura, C. Timossi, G. Portmann, M. Urashka, C. Ikami and M. Beaudrow, PCaPAC'08, Ljubljana, Slovenia, p.122(2008)
[21] G. Portmann, J. Corbett and A. Terebilo, PAC'05, Knoxville, USA, p.4009(2005)
http://www-ssrl.slac.stanford.edu/at/
[22] H. Nishimura, ICAP'98, Monterey, USA(1998).
http://www.slac.stanford.edu/xorg/icap98/papers/F-Tu06.pdf
H. Nishimura and W. Decking, EPAC'98, Stockholm, Sweden, p.1224(1998)
[23] http://msdn.microsoft.com/en-us/concurrency