ISBN: 978-3-95450-227-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2022-TUPOST029

SMALL TALK ON AT

P. Schnizer, W. S. Khail and J. Bengtsson
Helmholtz-Zentrum Berlin, BESSY, Berlin, Germany

Abstract

Tracy 3 — was implemented by the 3rd author by pursuing a first principles approach, aka Hamiltonian dynamics for an on-line model to guide the ALS and LBL comissioning in the early 1990s. With its origin as a Hamiltonian based pascal online model used 90 –- it is the core of today's accelerator tool box. These Hamiltonians have not been changed. Software design has evolved since then: C++ and in particular its standardisation C++11 and C++2xa. In this paper we outline our strategy of modernisation of tracy: reorganisation of the beam dynamics library in cleanly designed modules, using well proven open-source libraries (GSL, armadillo) and so on. Furthermore, Python and Matlab Interfaces based on modern tools are being pursued. We report on the interface design, the status of modernisation. This project has been renamed to thor-scsi-lib and is available at Github. Collaboration's welcome.

INTRODUCTION

The Helmholtz Zentrum Berlin is looking into modernising and upgrading its own synchrotron light source BESSY II next to the one it operates on behalf of PTB, the light source MLS. These will now transit to a 4th and 3rd generation light source. In parallel the existing light sources BESSY II and MLS are constantly upgraded. These upgrades require proper tools for modelling these accelerators for predictable results.

Software concepts, introduced in the sixties (e.g. at Xerox Palo Alto with the development of Small Talk) are now being readily available in commonly spread languages: C++, in particular in its form of C++11 or Python. Furthermore interfacing between these languages simplify developing flexible user interfaces.

Tracy in its different implementations has been used to guide the control of the non-linear dynamics for robust design for predictable results for [1] ans the beam dynamics model – with related controls algorithms – was also used for on-line models for e.g. [2]. The beam performance predictions of Tracy gave sufficient confidence that MAX IV was proposed and funded [3]: a machine that has revolutionised the synchrotron light source landscape by introducing several paradigm shifts [4]. The code base of Tracy, however, was always adapted to the different project needs; essentially by a solo effort. Project dead lines prevented to modernise the code base and update it.

BESSY II is providing a digital twin with a "numerical engine" based on the Hamiltonian framework already provided by Tracy [5]. Given that the modernised software architecture & code base has a significantly changed interface the project name was changed to Thor Scsi. The code is available at `https://github.com/jbengtsson/thor-scsi-lib`. In this paper we outline the main changes that the code base underwent. .

FOUNDATIONS

Thor Scsi is based on the traces of Tracy 2 and Tracy 3. It is "self consistent": i.e. based on an Hamiltonian formulation and it is symplectic. The same Hamiltonian and integration method are used for tracking or computing the global properties of the lattice: e.g. linear optics, radiation effects and driving terms; engineering tolerances for a realistic lattice are consistent: e.g. the effects of magnetic roll angles studied in particle tracking will also effect the calculated vertical emittance. The charge of the tracked particles can be defined at compile time: it defaults to electrons. Mathematical details are given in [5].

UPGRADED CODE BASE

At the start of the refactoring the code base was still a result of the procedural programming paradigm. So the classes were redesigned. Each element can be seen similar to a LEGO block (see also Figure 1). Two coordinate systems are used: a left handed global coordinate system [5] and a left handed local one. Apart from the drift type, all elements are calculating in local coordinate space. These coordinate transformations are handled by implementing the Euclidean group in delegates of the classes "LocalGalilean" or ""LocalGalileanProt". The first transforms from global to local coordinates by a general rotation and translation, while the 2nd to the magnet's natural local coordinates, which generates the horizontal edge-focusing for dipole magnets.

The later one not only translated the global to local by location and translation but furthermore allows handling a non sector bend magnet . Thick lenses are implemented here as Drift-Kick-Drift sequences. Here the kick element uses a field interpolation object to obtain the strength of the magnetic field at the particle location. Thus the magnetic field representation is not limited to solely Taylor expansion or Beth's representation [6] but any 2D field representation (expansion to higher dimensions is considered to be a straightforward extension) .

At the current stage only part of the elements existing in Tracy have been ported to Thor Scsi, in particular: drift, marker, beam position monitor, magnetic multipole (dipole, quadrupole, sextupole), cavity. Radiation effects are delegated to registered objects: thus the user has full control for which elements radiation shall be computed. Tracking or global parameter computations are implemented propagating a state space object through the beamline elements: a phase space floating point vector for tracking, or a truncated

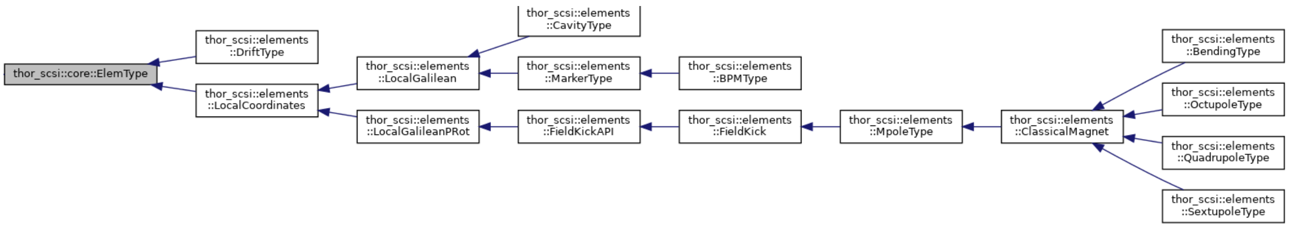
thor_scsi::core::ElemType
thor_scsi::elements ::DriftType
thor_scsi::elements ::LocalCoordinates
thor_scsi::elements ::LocalGalilean
thor_scsi::elements ::LocalGalileanPRot
thor_scsi::elements ::CavityType
thor_scsi::elements ::MarkerType
thor_scsi::elements ::FieldKickAPI
thor_scsi::elements ::BPMType
thor_scsi::elements ::FieldKick
thor_scsi::elements ::MpoleType
thor_scsi::elements ::ClassicalMagnet
thor_scsi::elements ::BendingType
thor_scsi::elements ::OctupoleType
thor_scsi::elements ::QuadrupoleType
thor_scsi::elements ::SextupoleType

Figure 1: Class hierachy used within Thor scsi.

power series object for global property computations. An observer can be registered to each element: thus the user can inspect after each pass the properties the state space properties at entrance or exit.

The lattice parser of FLAME is reused here to build the abstract syntax tree. The elements are then instantiated by a factory: here FLAME's code base was adapted to Thor scsi needs.

Python Interface

Many different scientific computational tools have been implemented in the python language: thus an interface was considered mandatory. Here Thor Scsi uses pybind11 [7] for providing a Python interface. It was found that its design turns wrapping C++ objects to nearly a "configuration task". Furthermore objects managed by C++ smart pointers can be handled transparently: thus user can now exchange the field interpolation object of the field kick. So it is quite easy for the user to swap multipole objects or implement a field provided by a nonlinear kicker [8] or a classical copper based telephone transmission cable.

Next steps will focus on making element handling more flexible and on simulations for the impact of engineering tolerances on the performance of the system. Propagation or tracking could be seen as applying a sequence of filters on a phase or state space. Thus adding an element should be as straight forward as inserting an element to a sequence. Given that the accelerator object is using a list object of C++'s standard template library (STL) this will be straight forward to implement: only some tweaking is required to keep consistently with other objects internal to the accelerator object, which provide further convenient handling to the user: e.g. simple functions to find elements with the same name in a consistent fashion.

Robust Design Studies

Any practical lattice design must be validated that it can deliver required performance parameters given the accuracy that can be obtained during manufacturing. These studies are typically performed using the distribution of different design parameters as being forecast during the design of the different elements or by the variation of these parameters obtained during manufacturing follow up.

Typically different sets are generated using a pseudo random number generator and the distribution factors. Thus the settings can be replayed at a later stage given that the pseudo random generator is called for the same sequence of parameters as before. Design studies however will first concentrate on one set of parameters (e.g. alignment tolerances) and later on others (e.g. main magnet strength).

Therefore these design studies are supported by "commands" in the following manner:

1. commands that describe a random distribution
2. commands that will apply a variation to a given property of the element

The user thus creates a sequence of command objects that describe the random distribution to be applied to the different properties. Provided machinery allows converting the distribution commands to a set of deterministic commands using a user supplied random number generator. This give the user full control on creating the distribution next to the precise set of property changes that will be applied to the accelerator. Simple functions will be provided for common tasks.

This two stage approach allows separating the distribution description from the actual sample. Furthermore as long as the sequence of commands describing distributions sequence is only extended at its end, the sequence properties can be replayed. The user still can filter out commands, that should not be applied during this part of the study.

MODERNISING CODE BASE: LESSONS LEARNED

Recommendation

Many scientific code bases of today have been used for decades. The authors assume that also other frequently used codes have not been actively maintained to follow more modern language or software engineering standards. This section is split up in a part that first describes how the this code modernisation would be conducted if done again. Then we describe how we did it. We hope this comparison is helpful for any reader facing a similar issue.

First recommendations address topics to cover before starting: definition of target and basis. The target should address why the code base is upgraded and what it should provide afterwards (e.g. conforming to some language standard, following certain design patterns). Together with this step it is worthwhile to define the start basis: i.e. the particular version, branch or similar of the different existing code bases. A cross check with the original author(s) comes in natural and can provide insights beyond the existing documentation.

During this process it is recommended to review which parts of the code base could be today provided by well written, well documented, existing libraries. This step should be rounded up with a search for an appropriate automated documentation tool: this comes in handy for the next steps as typically documentation tools provide also “bird eyes” view of the code base and its design.

Second step should address refactoring preparation: it should define a work plan which allows identifying which steps will require “a rip apart and reassembly” of the code base; this will be motivated further below. Furthermore the build and test system should be investigated: these tools will define the turn around time of any single later step. As these will be run frequently, their upgrade should be considered rather early in the refactoring process. Parallel to that test bases should be implemented: even if these are not full functional test these could be implemented as “total function test” and can be used as “safety warnings” to recognise if code refactoring broke some rather typical use case.

Then recommendations would be to start refactoring with an upgrade of the code base to a modern language standard as far as feasible in a straight forward manner (e.g. for C++ reducing name space pollution using the name-space “std::” explicitly, moving towards modern input output). Recommendations are to limit the code changes to the ones that still can be checked with the available test case. The idea is that language fixes will not increase the number of code lines the following major changes will touch. Furthermore even if the next steps are not executed, the available code base will be in a better shape than before.

After the above steps have been completed recommendations would be to start with the change that is considered the largest intervention to the code base or to its core. It is recommended that the full function test is executed after this change, even if it requires a compatibility layer. Target is to be able to run as many original programs and tests that show that the code is still producing the same results are obtained as original code. This will build confidence that the refactoring did not introduce major bugs. The other changes are advised to execute in a similar manner.

As soon as the major upgrade is finished recommendations would be to distribute it to as many friendly users as possible, even or in particular if the API has not stabilised: this will give feed back on the achieved upgrades and help discovering remaining bugs.

How We Did It

In our case the start was a code base that was originally implemented in Pascal. It was then machine translated to C and latter transformed to a C++ library. This library was then used and adapted for a considerable time span.

Upgrade started that one of the authors replaced C structs with STL objects. The build system was improved for speed up. Language was modernised, but not consistently: these updates happened then during the large interventions: thus more code lines than necessary were touched at this step, which made bug search more tedious.

First refactoring was introducing class hierarchy (see Fig. 1) followed by replacement of the hand written parser with FLAME’s one. Boost based unit tests were added in parallel. Cross checks were made to verify that the code produced the same output as the original code.

During the last step a Python interface was implemented using pybind11. This revealed requirements that Python had to the C++ code base. Furthermore one could interact and explore interface design on a much higher level.

A New Backend for AT?

AT is commonly used for evaluating the performance of synchrotron light sources; it’s core is based on Tracy 2 code base and provides a Python interface. The code base above could be used as modernisation starting point for AT’s code base. A transition to matlab’s C++ API could be worthwhile at this step too.

AT allows propagating many particles at once utilising today ubiquitous parallel processing features: similar functionality can be provided using C++’s templated implementation of the element’s propagate method: thus combining this functionality in a single object.

CONCLUSION

Thor Scsi is a refurbished modernised code base of the venerable Tracy, which has been used as design tool for various light sources or back end for online models. In this paper the main changes and concepts ere highlighted next to further steps to be implemented. Main changes are separating propagation from field interpolation and delegating radiation calculation to dedicated classes. A fully transparent Python interface is provided using pybind11. The authors welcome any collaborations.

REFERENCES

[1] J. Bengtsson, “The sextupole scheme for the swiss light source,” Paul Scherrer Institute, Tech. Rep., 1997, `https://ados.web.psi.ch/slsnotes/sls0997.pdf`

[2] J. Bengtsson and M. Meddahi, “Modeling of beam dynamics and comparison with measurements for the Advanced Light Source (ALS),” in *4th European Particle Accelerator Conference, London, UK, 27 Jun - 1 Jul 1994*, 1994, pp. 1021–1023, `https://cds.cern.ch/record/270082`

[3] S. C. Leemann *et al.*, “Beam dynamics and expected performance of sweden’s new storage-ring light source: Max iv,” *Phys. Rev. ST Accel. Beams*, vol. 12, p. 120 701, 12 2009, `doi:10.1103/PhysRevSTAB.12.120701`

[4] N. Martensson and M. Eriksson, “The saga of MAX IV, the first multi-bend achromat synchrotron light source,” *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 907, pp. 97–104, 2018, Advances in Instrumentation and Experimental Methods (Special Issue in Honour of Kai Siegbahn).

[5] J. Bengtsson, W. Rogers, and T. Nicholls, *A CAD tool for linear optics design: A controls engineer’s geometric approach to Hill’s equation*, 2021, `doi:10.48550/ARXIV.2109.15066`

[6] R. A. Beth, “Complex representation and computation of two-dimensional magnetic fields,” *Journal of Applied Physics*, vol. 37, no. 7, pp. 2568–2571, 1966, doi:10.1063/1.1782086

[7] W. Jakob, J. Rhinelander, and D. Moldovan, *Pybind11 – seamless operability between C++11 and Python*, 2017, https://github.com/pybind/pybind11

[8] T. Atkinson, M. Dirsat, O. Dressler, P. Kuske, and H. Rast, “Development of a non-linear kicker system to facilitate a new injection scheme for the BESSY II storage ring,” in *Proceedings of IPAC2011*, 2011.