Tracy and Thor to thor-scsi-lib: Lessons learned

Pierre Schnizer, Waheedullah Sulaiman Khail, Teresia Olsson

Helmholtz-Zentrum Berlin (HZB), Germany

2. Oktober 2023

Overview

Acknowledgement

Johan Bengtsson for preparing his code base, the updated documentation of the physics and maths involved [1], many tests and reviews of the developed code, reimplementing linear optics optimisation code in python, teaching proper dynamics. . . , kayaking

Markus Ries practical machine steering knowledge . . . good nerves

Guabao Shen for NSLS II virtual accelerator code share

Thomas Birke introduction to EPICS control system

BESSY II and MLS all people that make it all actual work

all that I am not even aware that they make my work possible

Refactoring: Overview

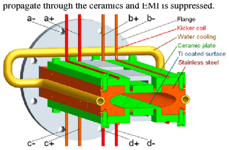
Towards thor-scsi-lib

- TRACY II code basis: split up
- lattice parser $\leftarrow$ FLAME [2]
- TPSA $\rightarrow$ gtpsa [3] $\leftarrow$ gtpsa-cpp
- modernised language "std::" containers, "arma::mat" for matrices (interface)
- autotools $\rightarrow$ cmake
- split up: multipole evaluation $\rightarrow$ field kick
  - delegates:
    - field interpolation
    - radiation calculation (only if there)
  - lets observe: phase space

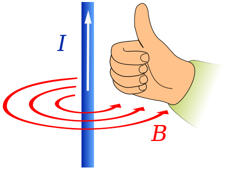
  thus fine grained control if required or not
- python interface $\leftarrow$ pybind11 [4] $\rightarrow$ elements in pyton
- many parameters: double or truncated power series objects
- worked on user interface simplification

Machine elements in python

Example: non linear kicker



[5]

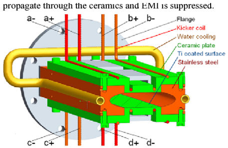


Source: Wikipedia
by Jfmelero
Element in Python
Called from C++ code

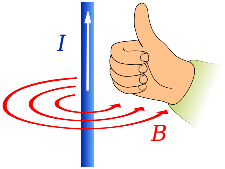
```python
class AirCoilMagneticField(tslib.Field2DInterpolation):
    """Field of an air coil"""

    def __init__(self, *, positions, currents):
        tslib.Field2DInterpolation.__init__(self)

    def field_py(self, pos, field):
        x, y = pos
        dz = x + y * 1j - self.positions  # offset from wire
        r = np.absolute(dz),  phi = np.angle(dz)
        B = (self.precomp * 1 / r * np.exp((phi + np.pi / 2) * 1j)).sum()
        field[0], field[1] = B.imag, B.real


class NonlinearKickerField(AirCoilMagneticField):
    """Field created by a classical telephone transmission cable"""

    def __init__(self, *, pos, current):
        p = np.array([pos, pos.conjugate(), -pos.conjugate(), -pos])
        currents = np.array([current] * len(pos)) * [1, -1, -1, 1]
        AirCoilMagneticField.__init__(self, positions=pos, currents=currents)
```

Machine elements in python

Example: non linear kicker



[5]




```
struct aircoil_filament {
    double x, y, current;
};

template<class C>
AirCoilMagneticFieldKnobbed(
    const std::vector<aircoil_filament_t> filaments,
    const double scale=1e0);
template<typename T>
inline void _field(const T& x, const T& y, T *Bx, T *By) const {
    const double precomp = mu0 / (2 * M_PI) * this->m_scale;
    *Bx = *By = 0e0;
    for(const auto& f: this->m_filaments){
        const T dx=x-f.x, dy=y-f.y, r2=dx*dx + dy*dy; // offset from wire
        *By += precomp * f.current / r2 * dx;
        *Bx += precomp * f.current / r2 * dy;
    }
}
```

Source: Wikipedia
by Jfmelero
Element in Python
Called from C++ code

Data models

Simplify processing

Definition

- intuitive schema of used data
- uses:
  - sub data models
  - primitive types

Example: BBA

measurements for magnet → measurement point → bpm's → bpm planes

Recommandations I

Start: definitions

- target
- basis
- Cross check with original author

Very useful: documentation of physics model [1]

Start: preparations

- code parts: standard libraries → replacement
- version control system
- automatic documentation tool (sphinx, doxygen,)

Recommandations II

Refactoring preparation

- work plan → "identify rip apart and reassemble"
- build and test system (run frequently)
- Build up of test system
  - total function test
  - "safety warnings"

Refactoring: Step I

- upgrade code base → modern standard
- as long as checkable with test base

End: Hold point: upgraded code base

Recommandations III

Refactoring: Step II

- Start with largest intervention
- Run full function test (e.g. with compatibility layer)

Refactoring: cont.

similar to above

Don't forget

- distribute early
- distribute often

Detailed in [6]

Outside view: where we are

- Start: tracy, thor scsi, single particle dynamics
- Target: implementation of a digital twin
- On boarding: software engineer $\rightarrow$ review of architecture
  - Data models
  - Interacting components: but as independent as possible
  - μ-service architecture
  - review of existing solutions

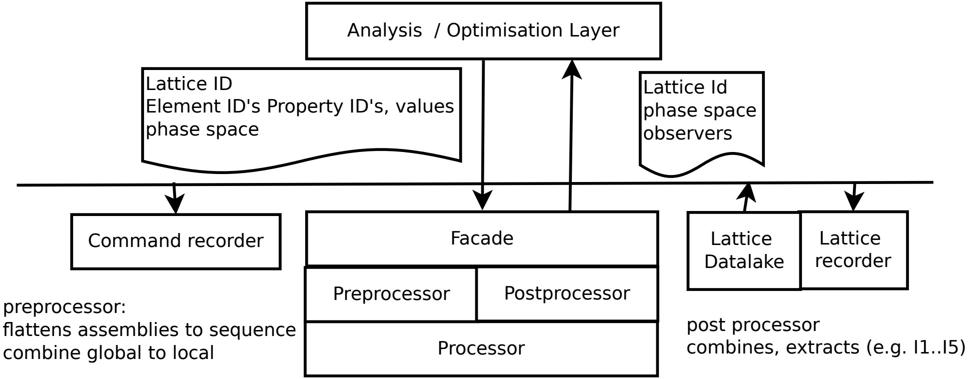
On calculating single particle dynamics

Outsiders view

- Apply kicks to particles described canonical variables
- at the right place
- in the correct coordinate system
- inspect result: at end or in between
- draw conclusions

Single particle dynamics: an architecture

Proposal: overview



Details explained below, influenced by python architecture patterns [7]

Basic building blocks

bricks, mortar, sand

Canonical variables

- phase space variables x, px, y, py, $delta$, ct...
- operations on these: arithmetic, trigonometric, exponent

Knobs

- properties of elements e.g. K, K_2,
- properties of coordinate transformation e.g. Δ_x, Δ_y
- operations on these: arithmetic, trigonometric, exponent

Variables depend on knobs, knobs depend on variables [3]

Identities

used for

- element locations: e.g. `q1m1d1r`
- element identities: e.g. `q1m1:#4`
- property identities: e.g. `K`

sole demand

- unique within its context
- for debugging: values meaningful for humans

Implementation

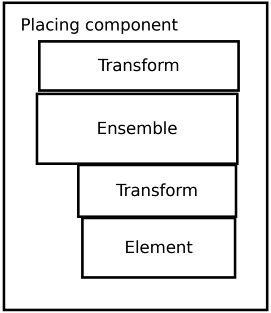
Knobs, variables

- double, complex
- interval, numerical stabilised
- truncated power series

Core of calculation

The processor

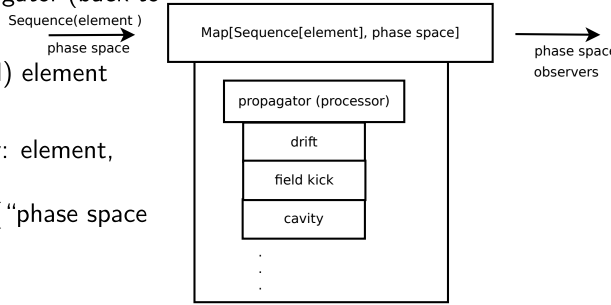
- propagate phase space through elements “linac like”
  - accelerator: global coordinate system (Frenet Serret, canonical variables)
  - elements: local coordinate system ← from machine, to assembly, to element
- separable: element properties and propagator (back to Tracy II or (py)AT)
- “linac” accelerator: sequence of (placed) element descriptions:
  - dedicated propagators: selected by: element, phase space, (calc config)
  - observers: for inspection, storage (“phase space monitor”, “watch point”)



Core of calculation

The processor

- propagate phase space through elements "linac like"
  - accelerator: global coordinate system (Frenet Serret, canonical variables)
  - elements: local coordinate system $\leftarrow$ from machine, to assembly, to element
- separable: element properties and propagator (back to Tracy II or (py)AT)
- "linac" accelerator: sequence of (placed) element descriptions:
  - dedicated propagators: selected by: element, phase space, (calc config)
  - observers: for inspection, storage ("phase space monitor", "watch point")



Phase space, Element

On variables and knobs

- phase space: variables $(x, px \ldots)$
- element: knobs (e.g. K)

variables depend on knobs, but knobs not on variables

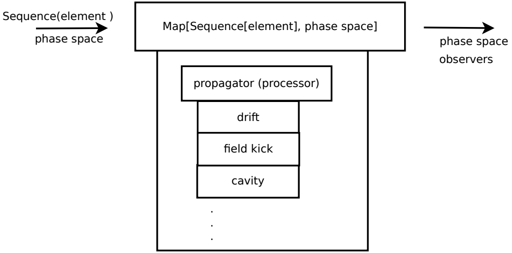
- variables knobs implementation: floating point, truncated power series, stabilised floating point calculation, interval calculation
- depending on use case

Calculation Engine: implementation

- Define abstract base classes
  - transform / element
  - phase space
  - "kick" propagator
- implement propagators: split up
  - multipole: field kick, interpolation, integrator,
  - radiation: as delegate
  - **NB**: integration integrals, diffusion matrix → post processing
- implement dispatcher: (element, phase space) → propagator

  Dynamically typed languages: run time

  Static typed language: templates, polymorphism



Calculation engine ↔ scientific work bench

A slim interface

- Motivation: studies modify some selected parameter of lattice independent of propagation engine: modify parameters, inspect
- Abstraction
  - on specific lattice (lattice id)
  - subset of its elements: change set value (property id)
  - propagate phase space and inspect

  (lattice id, element id, property id, value)
- nlattice = lattice.update(element id, property id, value) implementation: copy only as required (father figure: pandas [8], xarray [9]) handled in Facade

Analysis and optimisation

- Preconditions:
  - calculation / propagation engine
  - stored lattices, elements
- interaction with propagation engine: separation: more updates than required new_handle: e.g. orbit response matrix: change steerer setting: calculate closed orbit. next steerer: just start with handle again advantage: propagation of exceptions: no undefined state

ORM: dangerous

```
def measure_orbit_response(steerers, dI):
    for steerer in steerers:
        lattice[steerer].K += dI
        calculate_closed_orbit()
        lattice[steerer].K -= dI
```

ORM: handle exceptions

```
def measure_orbit_response(steerers, dI):
    for steerer in steerers:
        try:
            lattice[steerer].K += dI
            calculate_closed_orbit()
        finally:
            lattice[steerer].K -= dI
```

Analysis and optimisation

- Preconditions:
  - calculation / propagation engine
  - stored lattices, elements
- interaction with propagation engine: separation: more updates than required new_handle: e.g. orbit response matrix: change steerer setting: calculate closed orbit. next steerer: just start with handle again advantage: propagation of exceptions: no undefined state

Facade: update

```
def measure_orbit_response(
        lattice, steerers, dI):
    for steerer in steerers:
        t_lat = lattice.update(
            steerer, "K", dI)
        calculate_closed_orbit(t_lat)
```

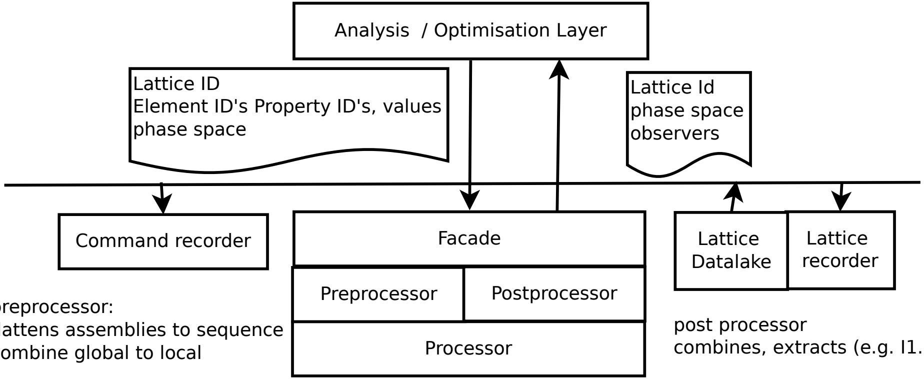
- supports: message bus, command recording, results ↔ machine setting
- multiprocessing: Sequence[commands] → partitioning[10] → jobs distribution

Design & Analysis: handling (magnet) families

- families: subset of magnets
- layer: analysis and optimisation
- implementation: separate
  - selecting subset $\rightarrow$ generator
  - apply change $\rightarrow$ lambda function

Single particle dynamics: an architecture

Proposal: overview



preprocessor:
flattens assemblies to sequence
combine global to local

post processor
combines, extracts (e.g. I1..I5)

Details explained below, influenced by python architecture patterns [7]

Thor scsi and (py)AT

Status

- Element description: (abstract base type?)
- processor: maps strings → propagator
- analysis scripts: tied to processor implementation

Modernised architecture

- processor: propagator implementation ← from abstract base type
- AT legacy processors: provide proxies to make them callable

Refactoring recommendations

- Split up of code base:
  - C code integrator: used by AT and pyAT e.g. "at_integrators"
  - AT matlab code base: e.g. "AT"
  - python code base: e.g. "py(AT)"

  development: git submodules?

Software architects and engineers: supervise and steer process

What's missing

From steady state to transient

Or the concept of time (compare Functional mockup interface standard) or open simulation platform [11, 12].

Steady state

- make change
- wait
- inspect result

Transient

- split up of calculation
- different speed
- exchange of progress
- $t_i \leftarrow$ change of "machine characteristic": e.g. kicker fired:
  - advance all integration until t_i
  - "restart" integration at t_i

Processor implementation

Language of choice

Boundary conditions

- CPU intensive task
- core of calculation $\rightarrow$ defines execution time

Compiled language: C++

- implement as templates:
  - `template<typename knob> struct element;`
  - `template<typename var> struct phase_space;`
- processor: dispatch to sub-processor: std variant, polymorphism

Dynamically typed language: JIT

- python: fast JIT?
- LuaJIT: demonstration by mad-ng

py(AT) recommendation: passenger view

- Currently: spin up of code base
- Consider:
  - define architecture
    - data models
    - interfaces: abstract base classes
    - layers
    - components
  - split up
    - shared code base
    - legacy code
    - language used
  - adhere: self set standards
  - gain:
    - components: simply development separation
    - layers: separate tasks, separate development
    - XXX

Target: simplify your life down the road

Conclusion

- Thor scsi: refactored code base, python interface, UI experience
- pyAT: active vibrant community, review of legacy code
- Proposal:
  - architecture review, split up of repository → more managble code functionality increasing
  - layers / components:
    - upcoming needs → changes → simpler implementation
    - work on subparts
    - roll your on: build on higher level products
- thor-scsi-lib next step: refactoring to processor

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.

Z. He, J. Bengtsson, M. Davidsaver, K. Fukushima, G. Shen, and M. Ikegami.
The fast linear accelerator modeling engine for FRIB online model service, 2016.

L. Deniau and C. I. Tomoiaga.
Generalised Truncated Power Series Algebra for Fast Particle Accelerator Transport Maps.
In *6th International Particle Accelerator Conference IPAC2015, Richmond, VA, USA*, pages 374–377, 2015.

Wenzel Jakob, Jason Rhinelander, and Dean Moldovan.
pybind11 – seamless operability between c++11 and python, 2017.
https://github.com/pybind/pybind11.

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, San Sebastián, Spain*, 2011.

P. Schnizer, W. Sulaiman Khail, J. Bengtsson, and M. Ries.
Progress on thor scsi development.
In *14th International Particle Converence Venezia*, pages 3366–3369, 2023.

Harry Percival and Bob Gregory.
Architecture Patterns with Python.
O'Reilly Media, Inc., 2020.

Wes McKinney.
Data Structures for Statistical Computing in Python.
In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.

S. Hoyer and J. Hamman.
xarray: N-D labeled arrays and datasets in Python.
Journal of Open Research Software, 5(1), 2017.

Ian Foster.
Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering.
Addison-Wesley Longman Publishing Co., Inc., USA, 1995.

OSP Interface Specification OSP-IS, Febrary 2022.

Florian Perabo, Daeseong Park, Mehdi Karbalaye Zadeh, Øyvind Smogeli, and Levi Jamt.
Digital twin modelling of ship power and propulsion systems: Application of the open simulation platform (OSP).
In *2020 IEEE 29th International Symposium on Industrial Electronics (ISIE)*, pages 1265–1270, 2020.