Overview of xcode

Ilya Agapov
Original motivation: spontaneous radiation

- Need for diagnostics, power loads and potentially science cases
- Numerical methods well understood, single particle solver provided by O. Chubar (SRWlib)
- Issues for xfel.eu: long undulators, narrow UR bandwidth, need to account for: electron optics, emittance, energy spread

Effect of orbit distortion, flash

SASE1 (0.05nm), energy spread effect

SASE1 (0.05nm), emittance effect on rad spot after mono

SASE1, emittance effect
Along the way: need beam dynamics module

Need:
- Beam optics model to account for emittance effects
- Matching (not to rely on external optics)
- Alignment errors, beam jitter to see effect on SR
- Orbit correction and steering (bumps and the like, no DFS)

Flash-like optics, effect of misalignment

SASE3 optics
PBBA: need 'on-line tools'

- Main motivation of SR is for diagnostics
- Tuning undulator K and phase shifter based on SR properties
- Any study cases should be easily transferrable to the control room

SR properties for various phase shifter strengths
Emerging concept: software framework

E.g.: HEP, detector simulations (Geant4)

- Complex geometries
- Diverse physics
- Large collaborations
- On-line event displays

For accelerator physics some attempts are known. However, we would need something simple, with the following features:

- Geometry: simple and extensible. MAD is not extensible, XML not simple. choose python. Can easily extend e-optics model to: aperture info, x-ray optics
- Scripting: python library the best choice
- Physics modules should be plugged in using the same model and IO
- Extension to on-line tools should be easy (Karabo is in python)
- Xframework supports these features, many are implemented, some tbd.
'xframework' refers to the common framework + integrated modules, the distribution including 3rd party codes (srw, genesis) is referred to as 'xcode'

- Open source https://code.google.com/p/xfel-xcode/
- SVN, unit tests, otherwise 'agile development'
- 1Yr in development
- Automatically generating genesis input from standard xframework decks, easy controls of run parameters
- Postprocessing tools for genesis: I/O and statistical analysis
- 1D python fel model
- Optimization routines and parameter scans (python)

Radiation parameters of SASE3, with postprocessing GUI (left)
SRW solver (O.Chubar and P.Elleaume, "Accurate and Efficient Computation of Synchrotron Radiation in the Near Field Region", EPAC-98)

Based on the same e beamline model. Standard xframework components from which radiation can be calculated: undulators with arbitrary polarization (analytical models and tabulated fields), quads, dipoles, sextupoles.

Other solvers included (e.g. Monte-Carlo photon generator, bottom left)

Solvers interchangeable

Benchmarks are/have been done, as well as calculations for xfel with all effects (bottom right)

SRW also allows for x-ray optics calculations. x-ray optic components (and particularly their placement) have not been standardized on xframework level, but direct access to appropriate srw functionality is always possible
Beam optics

- Completely embeddable and extensible e optics module – not an optics code
- Standard optics calculations included in distribution as scripts
- Embeddable: call from any python code
- Extensible: user can define new elements, redefine transfer maps and attach any additional features to beam elements w/o changing the module
- Features beyond single-particle electron dynamics can be added as extensions (no collective effects so far)
Example of DA calculations @ Siberia2 (S.Tomin)

Optical functions of „standard“ structure ($\varepsilon_x=98$ nm rad)

**DA without ID**

![Graph showing Optical functions of „standard“ structure](image)

Siberia-2 and proposed layout of insertion devices

![Diagram showing Siberia-2 layout](image)

**Main parameters of IDs**

<table>
<thead>
<tr>
<th></th>
<th>Wiggler</th>
<th>Wiggler</th>
<th>Undulator</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_{\text{max}}$, T</td>
<td>7.5</td>
<td>3</td>
<td>0.75</td>
</tr>
<tr>
<td>$\lambda_{\text{period}}$, mm</td>
<td>164</td>
<td>44</td>
<td>7</td>
</tr>
<tr>
<td>Field decrease ($k_x=2\pi/\xi$), $\xi$, m</td>
<td>0.8</td>
<td>0.8</td>
<td>0.5</td>
</tr>
<tr>
<td>$N_{\text{period}}$</td>
<td>10</td>
<td>34</td>
<td>300</td>
</tr>
<tr>
<td>$\varepsilon_{\text{crit radiation}}/\varepsilon_1$ ($W=2.5 \Gamma eV$), keV</td>
<td>30.7</td>
<td>8-16.4</td>
<td>$\varepsilon_1=2$</td>
</tr>
<tr>
<td>Deflection parameters</td>
<td>115</td>
<td>12.3</td>
<td>0.5</td>
</tr>
<tr>
<td>Spectral range, keV</td>
<td>20-150</td>
<td>5-40</td>
<td>2-7</td>
</tr>
</tbody>
</table>

21.05.2013
Example of DA calculations @ Siberia2 (S.Tomin)

- DA with 7.5 T wiggler
- DA with single 3T wiggler
- DA 2.5mm gap undulator
- DA with all IDs (Runge-Kutta)
Radiation parameter database?

- Have python repos for SASE1/2/3 and FLASH undulator sections (derived from MAD), and partially undulator field maps
- Component XLS files translator prototype in place (Igor Zagorodnov)
- Simulation output dump on mass storage (dcache) + SQL index DB is foreseen (prototype in place)
- However: need to define the scope
- Motivation: communicating photon beam parameter to users
- These parameters can be corrected from operation experience
- With a bit of programming python functionality easily put on-line (e.g. https://www.djangoproject.com)
- Beyond simple parameter presentation, can potentially aim at correlation studies and data mining.
On-line tools

- OM (LHC): embedding mad-x into java control system (pic below). Successful for commissioning, however software complexity and support is an issue.
- Machine Interface module in xframework allows for a 'flight simulator mode' of operation (TCP-based): alignment and tuning tools could be easily transferred to control room after switching from 'virtual' to 'real' mode. Similar things have been implemented in several labs already.
- Flight simulator mode requires data exchange protocol. Optics and other features can be more easily 'embedded' in python directly.
- Scripting is a major advantage for scans etc. Python used at NSLSII too.

LHC OM beams at Ips

LHC OM Aperture scan
Pipelining/interleaving simulations

- LHC collimation: (SixTrack/FLUKA)
- CLIC Beam Delivery collimation: Wakefields + secondaries (BDSIM/PLACET)
- XFEL Beam Dynamics: space charge, CSR, wakefields, etc....
- Always reinventing protocols to exchange data between codes; different physics can be included only iteratively, not on small time steps
- Open python library can provide much simpler solution to the problem

From Tracking Studies of CLIC
Collimation system
Agapov et al PRST-AB (2009)
Wakefields calculated with GdFidl, beam core tracked with PLACET and the halo with BDSIM. In this case wakefields Have negligible effect.
• Integration of all functionality into user Qt interfaces easy, some prototypes for FEL and SR in place. Usable software is more of an issue.
• Would need extra software engineers to write GUIs properly
Input decks

- No parsing needed
- Current input decks are derived from official MAD decks and included in repo

```python
beam = Beam()
beam.E = 17.5
beam.sigma_E = 0.001
beam.l = 2.5e-10
beam.emit_x = 1.752e-11
beam.emit_y = 1.752e-11
beam.beta_x = 33.7
beam.beta_y = 23.218
beam.alpha_x = 1.219
beam.alpha_y = -0.842

und = Undulator(nperiods=73, lperiod=0.068, Kx=0.0, id = "und")
d2 = Drift (l=0.45, id = "d2")
# phase shifter
b1 = RBend (l=0.0575, angle=0.0, id = "b1")
b2 = RBend (l=0.0575, angle=-0.0, id = "b2")
psu=(b1,b2,b2,b1)
# quads
qf = Quadrupole (l=0.1, id = "qf")
qd = Quadrupole (l=0.1, id = "qd")

cell_ps = (und, d2, qf, d2, und, d2, qd, d2)
sase3 = (und, d2, qd, d2) + 11*cell_ps
```
Embedding python is simple

- Twiss calculation

  ```python
  exec(open("../repository/flash/flash.inp"))
  tw0 = Twiss(beam)
  lat = MagneticLattice(flash_sase, beam.E)
  tws = twiss(lat, tw0)
  ```

- Running a SR calculation and saving into hdf5

  ```python
  traj, int1 = srw.calculateSR_py(lat, beam, screen, runParameters)
  dump = Dump()
  dump.readme = 'test 1_1'
  dump.index['beam'] = beam
  dump.index['screen'] = screen
  dump.index['intensity'] = int1
  dump.index['trajectory'] = traj

  xio = XIO('data/int_test_1_1.h5')
  dump.dump(xio)
  ```
Have a python module supporting beam optics, SR, and FEL calculations
However: development stage of many components is beta/prototype. To produce high quality software, will need to sync with the needs of other XFEL groups
Some functionality is within our research needs, but many options are open:
Interface to S2E?
On-line tools?
Parameter database?
Additional beam physics?
No entities beyond necessity
Opportunity for information exchange from linac down to photon beamlines
We are willing to collaborate and embrace new ideas