CONTROL ROOM ACCELERATOR PHYSICS

Day 2

A Primer on the XAL Online Model

Outline

1. Selecting the Hardware to Model
2. Beam Aspects – Instantiating a Beam Probe
3. Running the Model
4. Retrieving Simulation Data
5. Synchronization to Hardware

A Note on the Online Model

- The online model is a fundamentally different view of the accelerator – it is a model!
- The XAL accelerator hierarchy (SMF) is concerned with hardware, and hardware only!
  - There are no "drift spaces" in hardware.
  - The hardware has no knowledge of any beam
- When the online model is instantiated the "lattice generator" inspects the XAL accelerator hierarchy and creates an appropriate model for it
  - Aspects of the beam may then be simulated – and we know the simulated beam state completely
    - Particle
    - Transfer maps
    - Beam envelopes
    - …

Basic use of the Online Model

- To use the online model we instantiate a `Scenario` object which is built from an `AcceleratorSeq` object
  - The basic unit of hardware modeling is the `AcceleratorSeq` object
  - The online model is encapsulated by a `Scenario` object which models `AcceleratorSeq`'s
  - The online model contains
    - The simulation input (aspects of the beam we are modeling, magnet settings, etc.),
    - The hardware,
    - The output (the trajectory)
  - Once the simulation scenario is run, we can recover the output according to the hardware objects that interest us
  - We can change parameters of the model and compare how the output changes with respect to the actual machine

Using the the Online Model

Defining the Hardware to Model

```
#import the XAL hardware objects
from xal.smf import Accelerator
from xal.smf import AcceleratorSeq
from xal.smf. import AcceleratorNode
from xal.smf.data import XMLDataManager

from xal.smf.proxy import ElectromagnetPropertyAccessor

from xal.smf.data import XMLDataManager

#    Global Variables
strSeqId      = "RTBT1";                    # the target sequence identifier

 # read the accelerator and retrieve the target sequence
gblAccelerator = XMLDataManager.loadDefaultAccelerator()
gblSeqTarget = gblAccelerator.getSequence(strSeqId)
```

Using the Online Model

Instantiating the Beam Probe Automatically

Beam probes of various types may be generated automatically for the beginning of a sequence object
- One must also create an Algorithm object, which specifies the dynamics of the probe

```
# Import tools from XAL
from xal.model.probe import EnvelopeProbe
from xal.model.alg import EnvTrackerAdapt
from xal.sim.scenario import AlgorithmFactory
from xal.sim.scenario import ProbeFactory

# create and initialize an algorithm
etracker = AlgorithmFactory. createEnvTrackerAdapt(gblSeqTarget)

# set some custom parameters
etracker.setMaxIterations(10000)
etracker.setAccuracyOrder(1)
etracker.setErrorTolerance(0.001)


# create and initialize a probe
 probe = ProbeFactory.getEnvelopeProbe(gblSeqTarget, etracker);
```

Using the Online Model

Instantiating the Beam Probe from a Probe File

Sometimes a user has a unique situation and requires a special beam probe which may be taken from a file
(the Tracker object is defined in this file)

This method is almost obsolete but still available

```
# Import tools from XAL
from xal.model.xml import ProbeXmlParser

# Global Data
strInitProbe = "resources/probe/Rtbt-Bpm07-Coupled-Adapt-01.probe"; # Probe file


gblProbe = ProbeXmlParser.parse(strInitProbe);
gblProbe.initialize();
```

Using the Online Model

Manually Setting Probe Parameters

In addition, many probe and algorithm parameters may be set programmatically (MKS units)

```
probe.getAlgorithm.setMaxIterations(10000)
probe.getAlgorithm.setAccuracyOrder(1)
probe.getAlgorithm.setErrorTolerance(0.001)

probe.setBeamCurrent(0.038);
probe.setKineticEnergy(885.e6);
probe.setSpeciesCharge(-1);
probe.setSpeciesRestEnergy(939.29e6)
```

Using the Online Model

Synchronizing the Online Model to the Machine Design Parameters

Recall
```
# read the accelerator and retrieve the target sequence
gblAccelerator = XMLDataManager.loadDefaultAccelerator()
gblSeqTarget = gblAccelerator.getSequence("RTBT1")
```

```
# Import XAL tools
from xal.sim.scenario import Scenario
from xal.smf import AcceleratorNode

# Global Constants
strLocStart = "RTBT_Diag:BPM07";    # simulation start location
lstLocEnd = "RTBT_Diag:BPM08“;      # simulation end location


gblNodeStart = gblSeqTarget.getNodeWithId(strLocStart)
gblPosStart = gblSeqTarget.getPosition(gblSeqTarget.getNodeWithId(strLocStart))

# Create and initialize the model to the target sequel
model = Scenario.newScenarioFor(gblSeqTarget);

# Set the probe to simulate, the synchronization model, and the starting node
model.setProbe(gblProbe);
model.setSynchronizationMode(Scenario.SYNC_MODE_DESIGN);
model.setStartNode(strLocStart);
```

Using the Online Model

Synchronizing the Online Model to an Historical Machine Configuration (PV Logger)

We can configure the model to a previous machine state which was recorded with the PV Logger tool

```
# Import XAL tools
from xal.model.scenario import Scenario
from xal.service.pvlogger.sim import PVLoggerDataSource

# Global Constants
idPvLog = 4710691;          # PV Logger Snapshot identifier

# Create and initialize the model to the target sequel
model = Scenario.newScenarioFor(gblSeqTarget);

# Set the probe to simulate, the synchronization model, and the starting node
model.setProbe(gblProbe);
model.setStartNode(strLocStart);

# Instantiate a PV logger source to the given PV snapshot ID, then initialize the model
plds = PVLoggerDataSource(idPvLog)  # retrieve from PV log ID
model = plds.setModelSource(gblSeqTarget, model);
```

Using the Online Model

Running the Online Model

```
model.run()
```

Using the Online Model

Retrieving Simulation Data

The `Trajectory` object of a probe contains all the historical state information as it passed through the beamline. The type of data the trajectory contains depends upon the simulation run.

```
# Retrieve the probe from the model then the trajectory object from the probe
probe = model.getProbe()
traj = probe.getTrajectory()

# Retrieve all the simulation data for the injection foil (hardware id “Ring_Inj:Foil”)
dataFoil = traj.statesForElement("Ring_Inj:Foil")

# Retrieve the final state of the simulation
dataFinal = traj.finalState()

# If the probe is an EnvelopeProbe, we can get the covariance matrix of the final state
matCovFin =  dataFile.getCovarianceMatrix()

# We can get all the covariance matrices of the trajectory
lstCovMat = []
for state in traj:
    matCov = state.getCovarianceMatrix()
    lstCovMat.append( matCov )
```

Using the Online Model

Changing a Hardware Parameter in the Model

Sometimes we wish to change the value of a parameter in the model to see how it affects the output

What if I did this…? Or this…?

```
#Global Data
strNodeId = "MEBT:Mag_QV01"
strNodeParam = "setField"
dblNewVal = 333.5


# Retrieve the AcceleratorNode object
smfQuad = gblSeqTarget.getNodeWithId(strNodeId)

# Change the field of the model quadrupole magnet corresponding to smfQuad hardware
model.setModelInput(smfQuad, strNodeParam, dblNewVal);
(model.setModelInput(smfQuad, "setField", 333.5)
```

The online model can then be re-run and the output collected from the `Trajectory` object as before

We call the setField() method of the modeling element with the argument 333.5

Summary

- The online model is represented as a `Scenario` object
- A Scenario is instantiated for a particular `AcceleratorSeq`
- The online model can be synchronized to
  - Machine design values
  - Current machine state
  - A past machine state saved by the PV logger
- The beam is represented by a `Probe` object which can be created with the `ProbeFactory`. An Algorithm object must also be chosen for each probe.
- The output of the online model is contained in a `Trajectory` object whose type depends upon the simulation performed by `Scenario.`
- Various parameters of the online model can be changed by the user to simulate a "what if" scenario