XAL Workshop

A Primer on the XAL Online Model

Outline

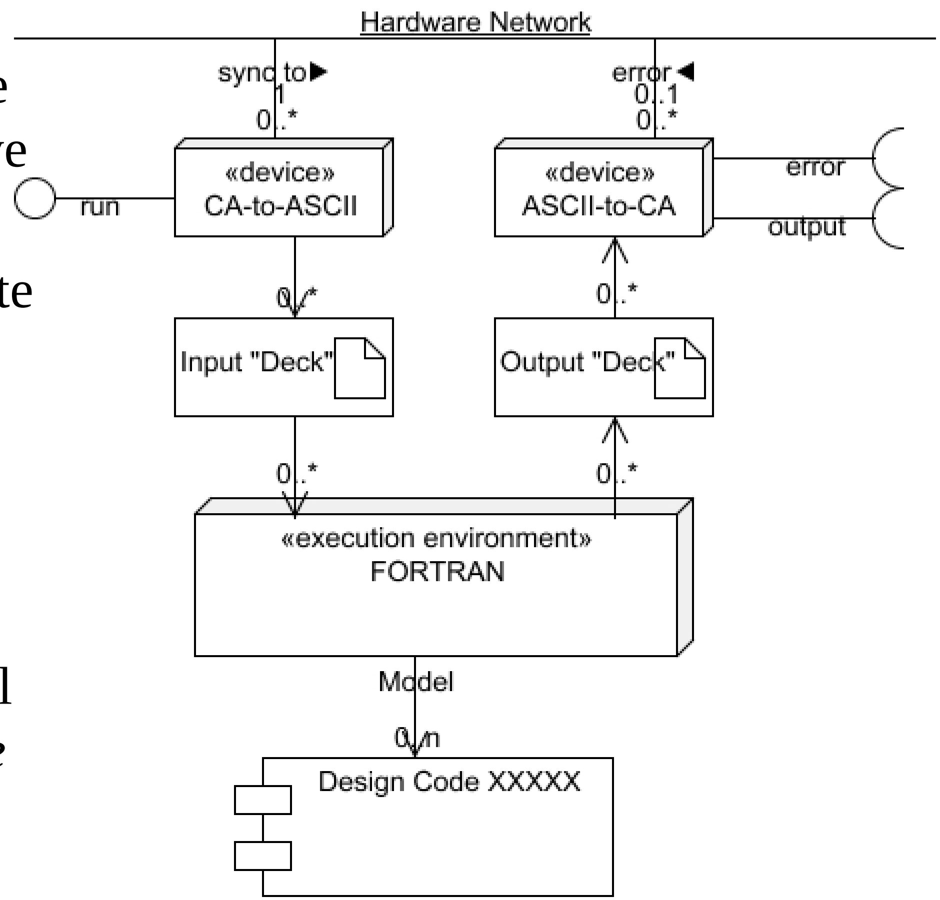
1. Overview and Architecture
2. Selecting the Hardware you want to Model
3. Beam Aspects – Instantiating a Beam Probe
4. Running the Model
5. Retrieving Simulation Data
6. Synchronization to Hardware

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 (e.i., 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 SMF hierarchy and creates an appropriate model of it
  - Aspects of the beam may then be simulated – and we know the simulated beam state completely

What the Online Model is **Not**

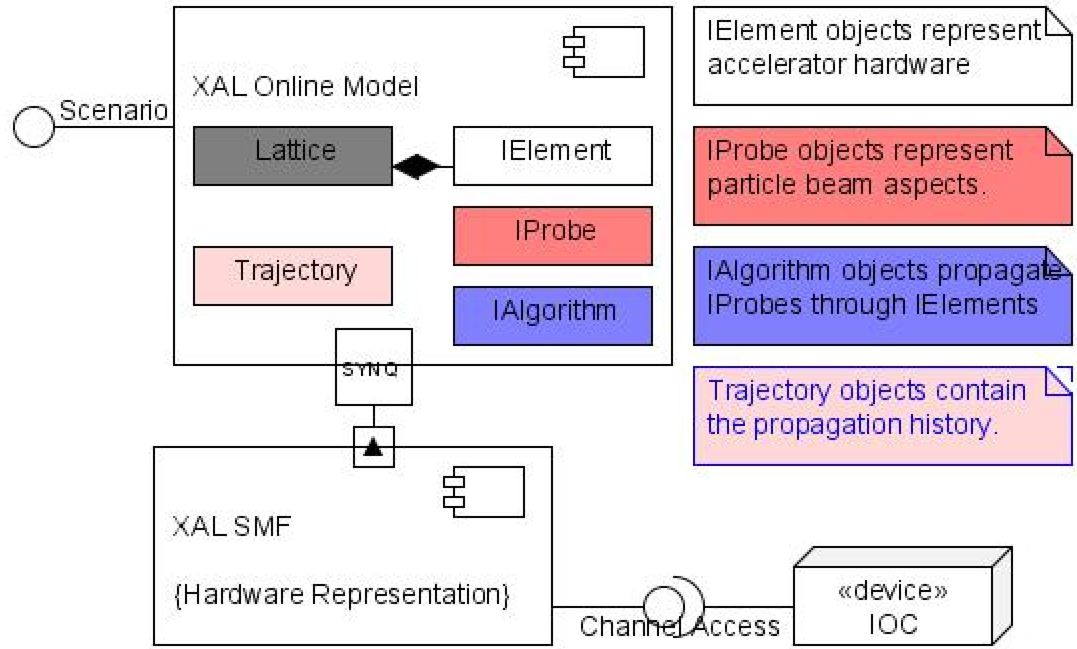
- The online model is **not** a design code connected to the control system through native interfaces and drop files
- I does **not** maintain a separate “lattice file” defining the machine layout and beam parameters
- It **does** uses fast simulation techniques (not multi-particle) in order to be useful as an *online model reference*



Online Model Architecture

Element/Algorithm/Probe

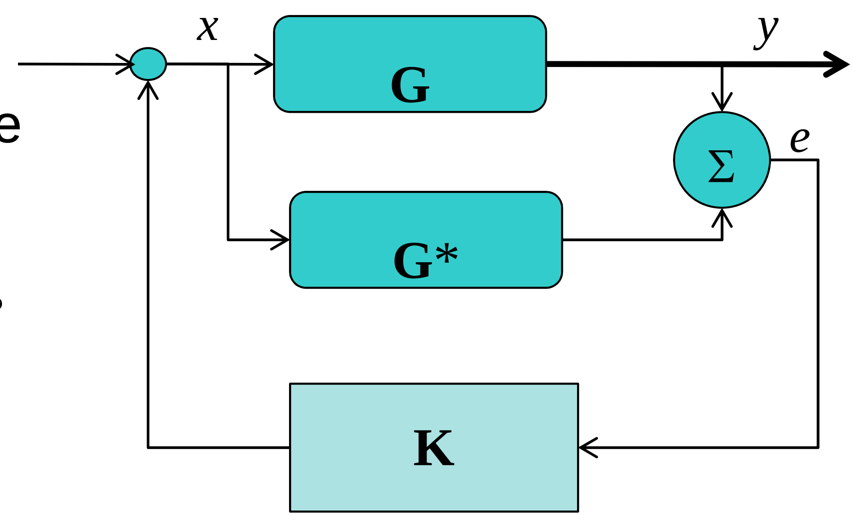
- Architecture based upon the *Element/Algorithm/Probe* design pattern
  - N. Malitsky
- All online model aspects encapsulated as a *Scenario*
  - Aspects are input, output, hardware, beam, etc.
- Scenario synchronizes with hardware through SMF accelerator object
  - The Scenario object is generated from the SMF object



Objective
Model Reference Control

The XAL online model is designed to be used programmatically within applications, as an online model reference.

- "Prediction"
  - What is the beam shape downstream?
- "Parameter estimation"
  - What is the emittance
  - What is the beam energy
- "Regulation"
  - Orbit correction
  - Phase settings



G – Accelerator system
G*- Accelerator system model
y – output
x – input
e – error signal
K – feedback control law

Basic use of the Online Model

- To use the online model we instantiate a `Scenario` object which is attached to the AcceleratorSeq
  - The basic unit of hardware modeling is the `AcceleratorSeq` object
  - The online model is encapsulated by a `Scenario` object which contains
    - The simulation input (aspects of the beam we are modeling, magnet settings, etc.),
    - The hardware,
    - The output (the beam 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 gov.sns.xal.smf import Accelerator
from gov.sns.xal.smf import AcceleratorSeq
from gov.sns.xal.smf. import AcceleratorNode
from gov.sns.xal.smf.data import XMLDataManager

from gov.sns.xal.smf.proxy import ElectromagnetPropertyAccessor

from gov.sns.tools.xml import XmlDataAdaptor

#   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

- Must also create an algorithm ("Tracker") object for the probe

```
# Import tools from XAL
from gov.sns.xal.model.probe import Probe
from gov.sns.xal.model.probe import EnvelopeProbe
from gov.sns.xal.model.alg import EnvTrackerAdapt

# create and initialize a probe algorithm
etracker = EnvTrackerAdapt();

etracker.initializeFromEditContext(gblSeqTarget);
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)

```
# Import tools from XAL
from gov.sns.xal.model.probe import Probe
from gov.sns.xal.model.xml import ProbeXmlParser

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


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

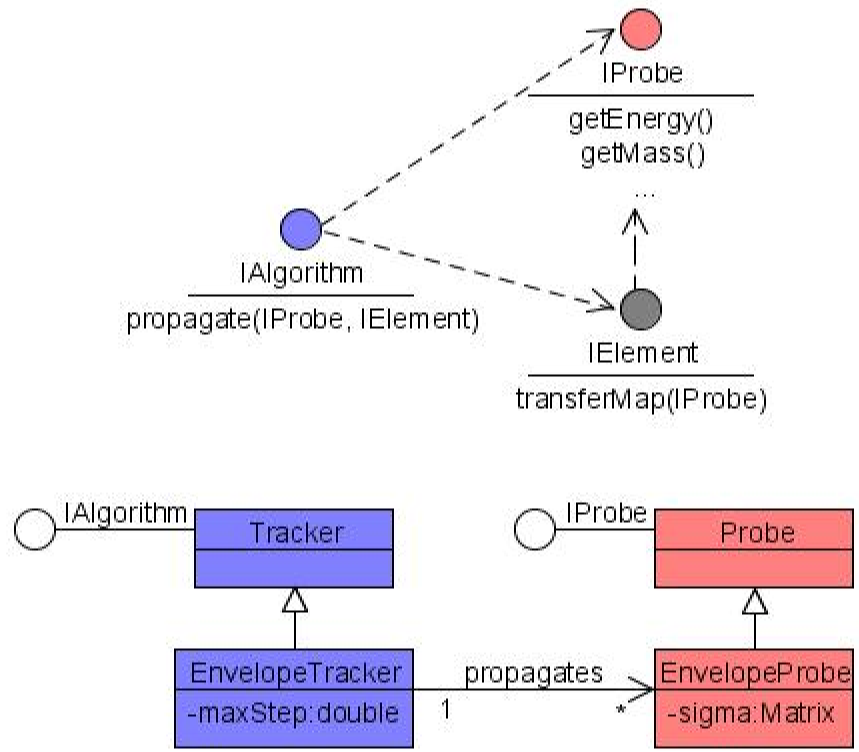
Using the Online Model

Manually Setting Probe Parameters

In addition, many probe parameters may be set programmatically

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

    probe.setBeamCurrent(0.);
    probe.setBeamCharge(0.);
    probe.setKineticEnergy(885.e6);
```



Using the Online Model

Instantiating the Scenario Object (to the Design Mode)

```
# Import XAL tools
from gov.sns.xal.model.scenario import Scenario
from gov.sns.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

Initializing a Scenario Object to a Previous Machine Config.

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

```
# Import XAL tools
from gov.sns.xal.model.scenario import Scenario
from gov.sns.tools.pvlogger 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);

 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.

```
# 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 Twiss parameters of the final state
twiss Final =  dataFile.getTwiss()
```

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 (e.g., does it look more like the actual machine?)

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


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

# Change the field of a quadrupole magnet
model.setModelInput(nodQuad, strNodeParam, dblNewVal);
```

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

Summary

- The online model is represented by a `Scenario` object
- A Scenario is instantiated for a particular `AcceleratorSeq`
  - Can be synchronized to the design values, current machine state, or a past machine state saved by the PV logger
- The beam is represented by a `Probe` object which can be created with the `ProbeFactory`
- The output of the online model is contained in a `Trajectory` object which is retrieved from the `Scenario`
- Various parameters of the online model can be changed by the user to simulate a “what if” scenario