# CONTROL ROOM ACCELERATOR PHYSICS

Day 3

*High-Level Application Design*

# Outline

1. **High-Level Software**
   1. Introduction
   2. Commonality

2. **Application Frameworks**
   1. Use Cases
   2. Design

3. **Applications within the Framework**

4. **Open XAL: A Case Study**



Shallow foundation of a house versus the deep foundation of a building

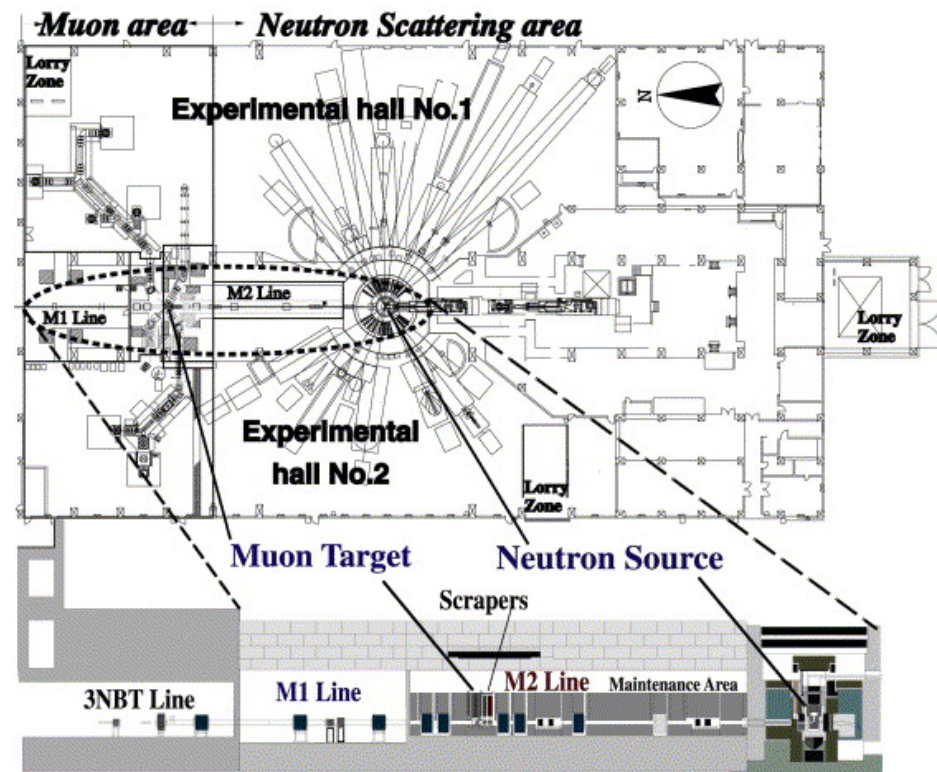# High-Level Control of Accelerators

Accelerator systems are extremely complex machines.  Moreover
- Operation requirements change
- Configuration changes
- Designs change on the fly
- Constant upgrade projects

Accelerator control systems require **a lot** of software.  Moreover they req.

- Maintenance

- Upgrades

It is worthwhile to invest significant design time for robust controls software that can adapt gracefully to this changing environment

# High-Level Control Applications
## Definition

Software applications needed for the commissioning, tune-up, diagnosis, operation, automation, and optimization of accelerator systems

For Example,

- Machine state diagnostic
  - Save/Compare/Restore
- PV scans
- Orbit Display
- Orbit Difference

- Orbit Bump
- Orbit Correction
- Transverse matching
- RF Phase and amplitude matching
- Ring closed orbit

# High-Level Control Applications
## Commonality

Notice there are many qualities/tasks/requirements common to most high-level control applications

- Data-centric operations
  - Machine configuration
  - Data acquisition and correlation
  - Data presentation, analysis, and interpretation
- Data analysis
  - Signal processing
  - Computation
    - Linear Algebra
    - Optimization
  - Modeling and simulation
- High level of user interaction (GUIs)

Rather than letting each application implement each of these features (in a potentially inconsistent manner), we provide them within a **framework**.

# High-Level Application Frameworks
## Definition and Objectives

Application Frameworks offer consistent solutions to problems and tasks common to a set of related applications

- The framework is centralized, any changes in framework are seen by every application
- Consistent interface to hardware
- Configures to hardware (on the fly)

If implemented well - framework can provide a Rapid Application Development environment (RAD)

- Subject matter experts (SME) tend to be physicists and research engineers
- If SMEs can test ideas/algorithms within the framework – they can then be easily implemented in a robust fashion by software personnel

# High-Level Application Framework
## A Control System Design Strategy

A framework for high-level control application is a sound design strategy for creating a **robust** control software system that can respond gracefully to changes in the accelerator system itself.

- Immediately recognizes changes in machine configurations
- Trivial scaling to machine upgrades
- Centralized management of control applications

The cost of this flexibility is a front-loaded design and implementation strategy – That is, there is significant initial overhead to this approach.

WARNING: Front-loaded strategies tend to scare (traditional) management since no code is being written in the initial phase of the project
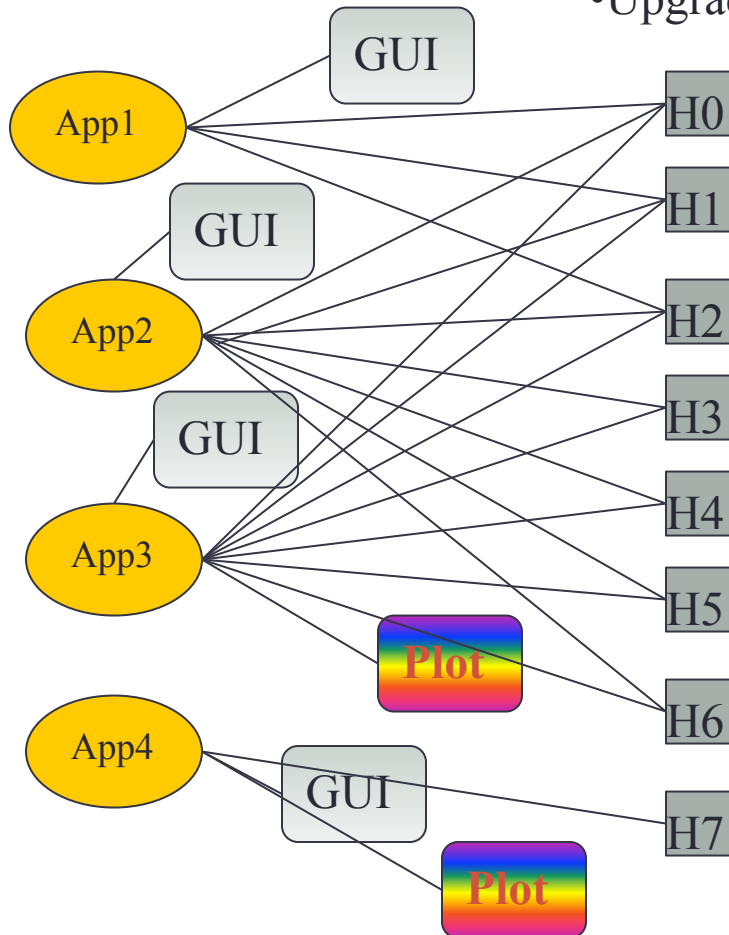
# High-Level Application Framework

## Architecture Comparison With and Without a Framework
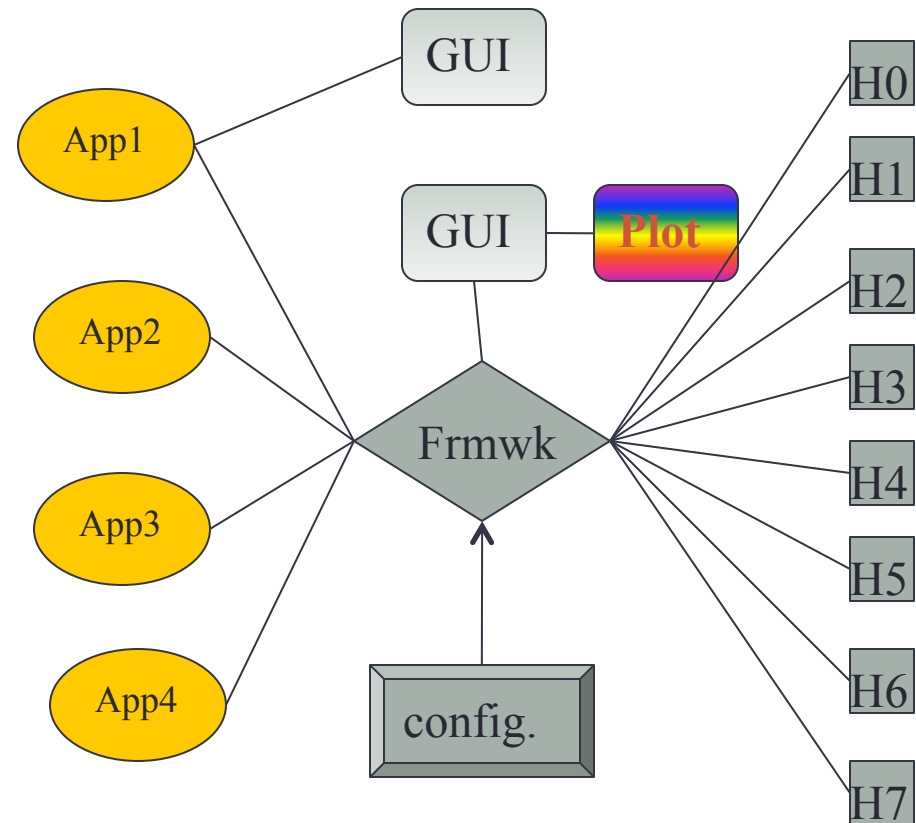
Which system would you rather
- Test?
- Maintain?
- Upgrade?

Consider a hardware failure at H2

**Back-loaded design**

**Front-loaded design**

# High-Level Applications Framework
Design

Designing the suite

- Want to provide common set of features, tools, for…
  - Machine configuration
    - Database queries
    - Machine connection
  - Data Analysis
    - Plotting
    - Linear algebra/mathematical manipulation
    - Signal processing
  - Application implementation
    - "GUI Application Framework" in the traditional software sense

But how to start? ….. Use Cases!!

# High-Level Application Framework
## Design: UML Use Case Diagram

Working out use cases provides a good method for visualize potential software structures to support your scenarios

We organize our use cases by extracting common behaviors – common behavior implies common software

The importance of a common scenario is demonstrated by the number of incoming edges – indicates critical software
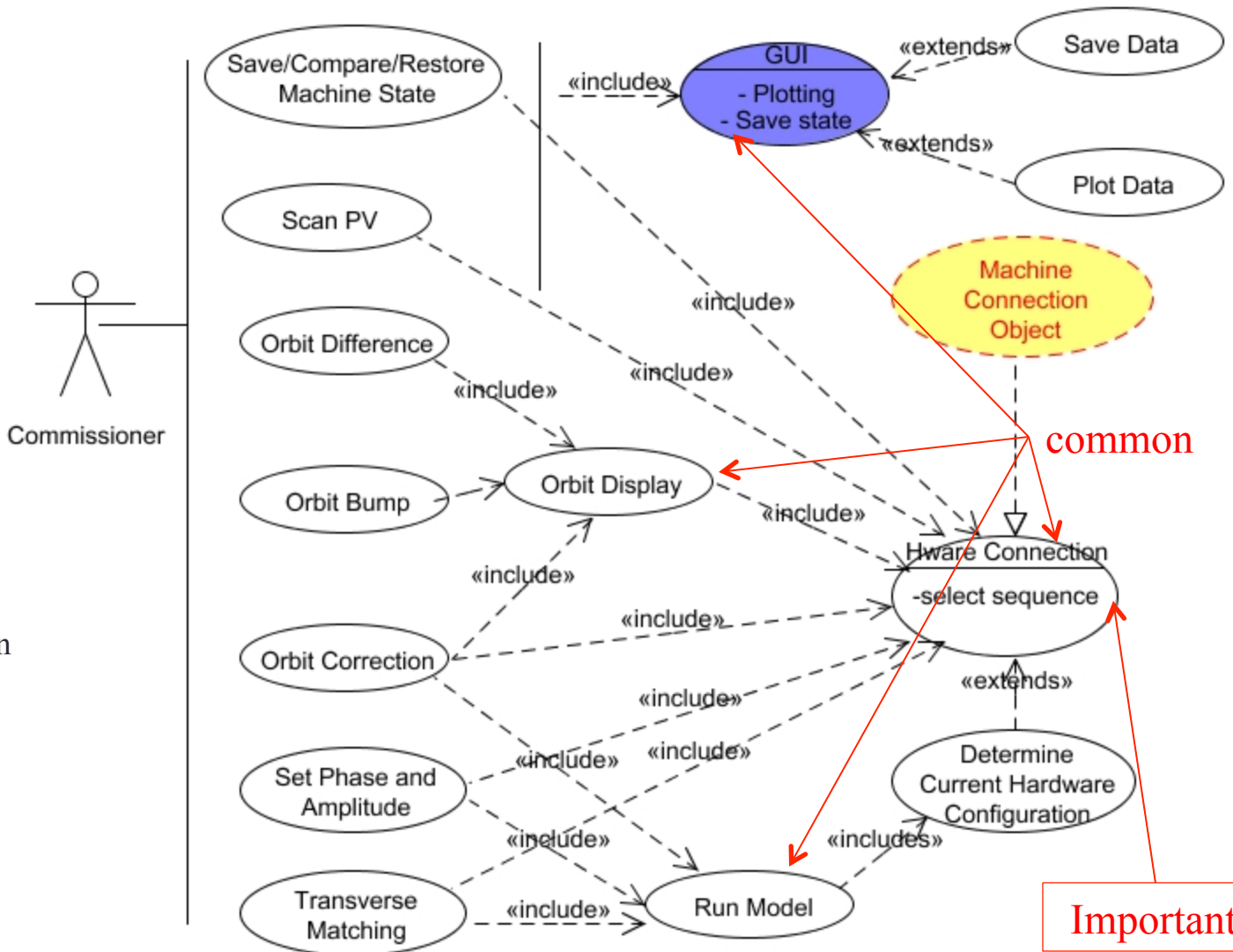
# Application Framework Use Cases

Observations

- The "Machine Connection" collaboration is a realization of the low-level hardware connection
  - We assume this exists, for example take EPICS as our subsystem

- Implementation of *high-level* hardware connection is <span style="color:red">critical</span>
  - Any shortcomings here adversely affect all applications
  - But, future improvements and upgrades benefit *every* application

- Implementation for hardware configuration is also <span style="color:red">critical</span>
  - It may be called at any time as an extension point of the hardware connection use case

# Application Framework Use Cases
Observations (Continued)

- Choosing accelerator sectors to work on is also important and common
    - Design for a robust generic technique

- There are common scenarios that can be coalesced
    - Orbit display
    - Online model (used less often - less critical - possibly deferred?)
    - Use cases suggest they may be implemented as independent components

- All applications use a GUI interface
    - We should provide a consistent framework for rapid creation

# Application Framework Use Cases
## NOTE: Sub-dividing Use Cases

Note that it is possible, and wise, to further divide up important use cases.

This action helps to elaborate the user interactions and anticipate needed software capabilities

The use cases drive the software designs …

**Save/Compare/Restore Use Case**
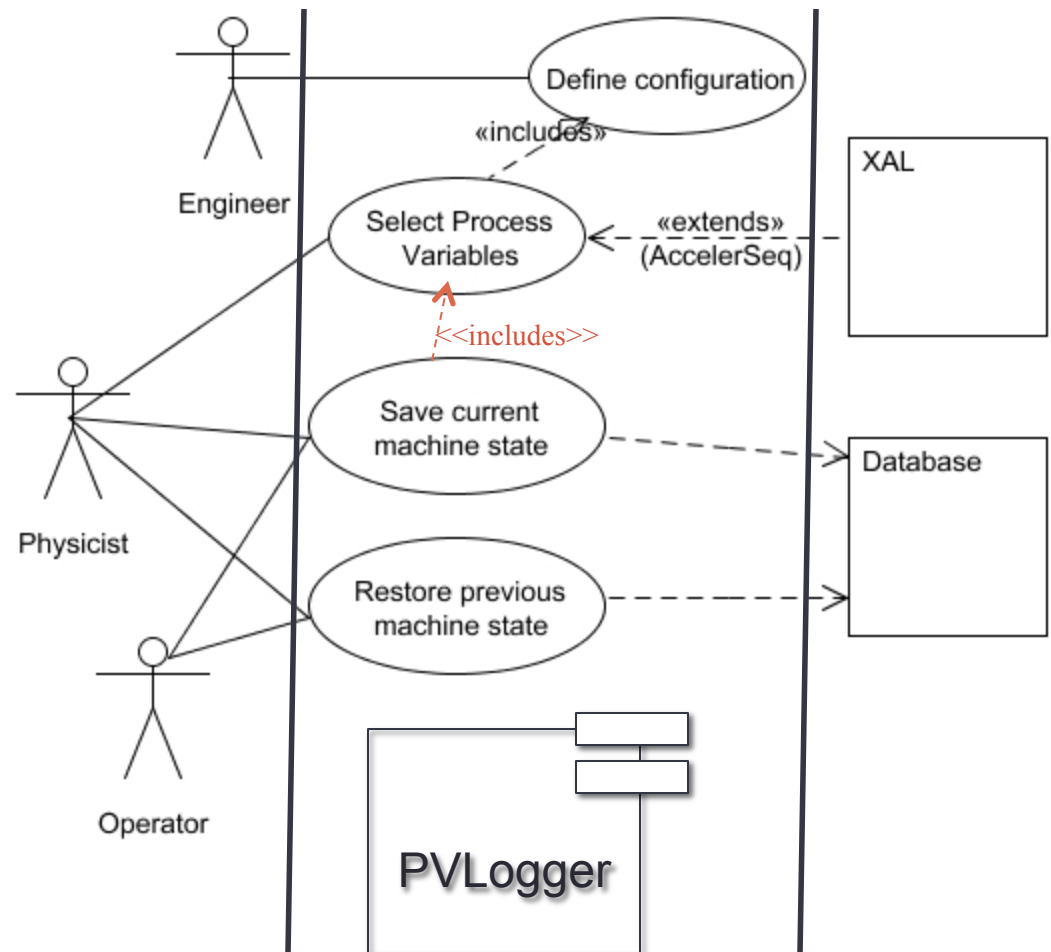
# Common Scenarios
## More Evidence for a Framework

- The use case diagram demonstrates that there are many common scenarios in our high-level control environment
  - GUI
  - Orbit Display
  - Online Model
  - Hardware configuration and connection

- Let's try to provide some example designs for software components which could realize these scenarios…

# Application Framework Design
## Hardware Connection Component

- Software components
  - Recall *software components* implement natural design solutions to shared scenarios

- Consider the hardware connection scenario
  - Common scenario
  - Extensions
  - Critical

Next page
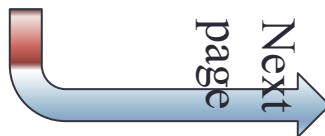
# High-Level Application Frame
## Implementing the Hardware Connection Scenario

The hardware connection component is further divided into subcomponents with simple user interface

```
Accelerator {
    getAccelerator()
}
```

**Interfaces and components**

• Allows independent development of subcomponents

• Wholesale substitutions of subcomponents

• Facilities testing of subcomponents

components

Produces objects representing hardware in current configuration

Accelerator

Standard Machine Format (SMF)

user interface

AbstractChannel

JCA

CAJ

Arb. CA Proxy

CA

EPICS

IOC «delegate»

IOC «delegate»

XML
- Hware config
- PV Map (Dev,PV)
- Model parameters

JDBC

Database

Mechn. provides machine configuration from central database but also isolation from database errors

Newtowrk

Deployment
«epics» Hardware Drivers

# Application Framework Design
## Orbit Display Component

- Consider the orbit display scenario
  - Requires callbacks
  - Requires plotting display for data
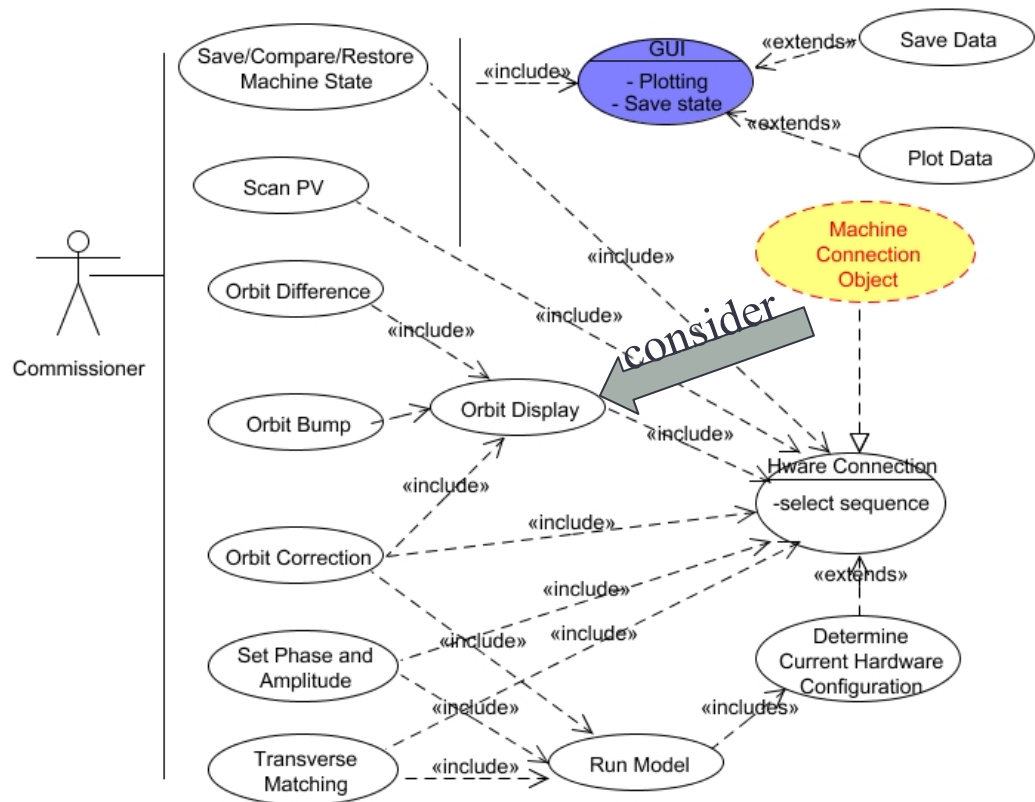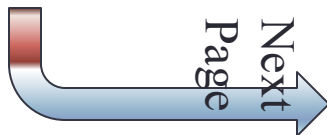
# High-Level Application Framework
## Implementing the Orbit Display Component

• Have explicit use of callbacks or "monitors" signal new BPM value then held in a buffer

• Display is updated according to user specified repetition rate rather than by when BPM change

• 2D plotting performed by "Plot2D" class, actually a façade for this capability.
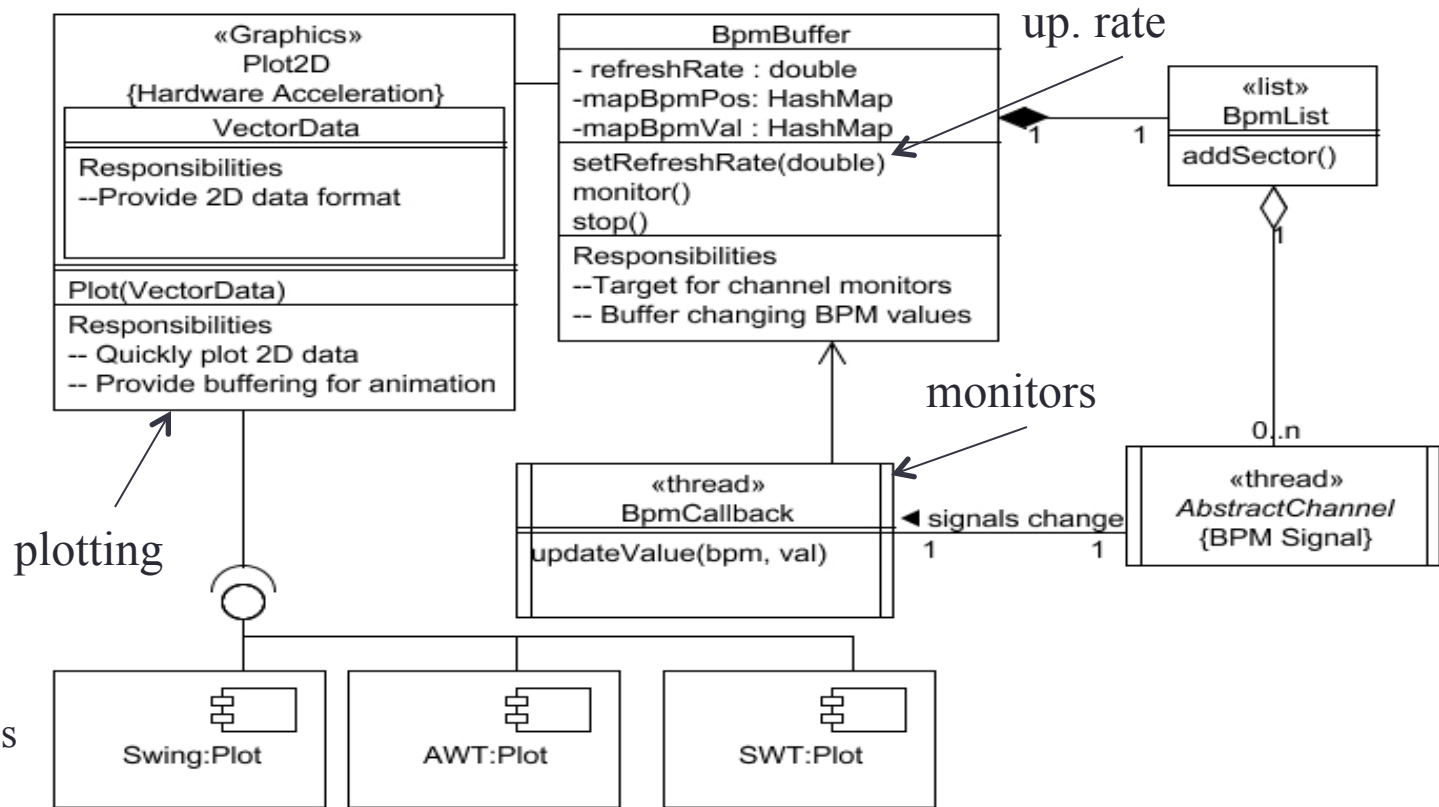


2D Plotting arises in several other scenarios, thus, it is a common "sub scenario." It may be wise to implement a robust software subcomponent for it.

# Application Framework Design
## Online Model Component

- Consider the online model
  - Must produce simulation data for the machine in its current running state, past states, and design configuration
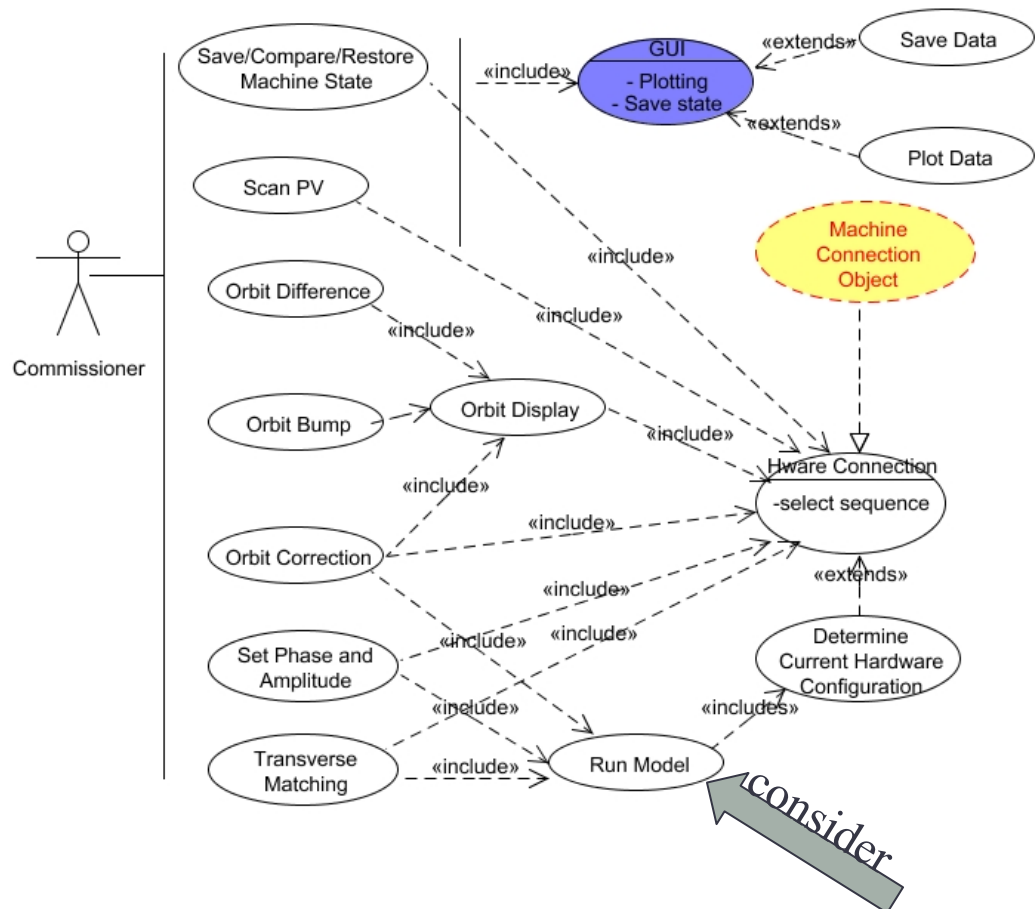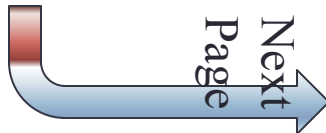
# Application Framework Design
## A Note on Implementing the Online Model

There are significant differences between accelerator design codes and the requirements for an online model which might make such use brittle

### Design Code

- Used for machine design and data analysis
- Produce design parameters
- Runs off-line
- Static input from "deck"
- Static configuration is OK ("deck")
- Must predict fine structure

### Online Model

- Used for Model Reference Control (MRC)
- Estimation of actual parameters
- Runs in real time –fast!
- Dynamic input from running machine
- Dynamic configuration a must
- Only data that sensors can detect

# Application Framework Design
## Implementing the Online Model

Here is the "Traditional" Approach

Any comments?
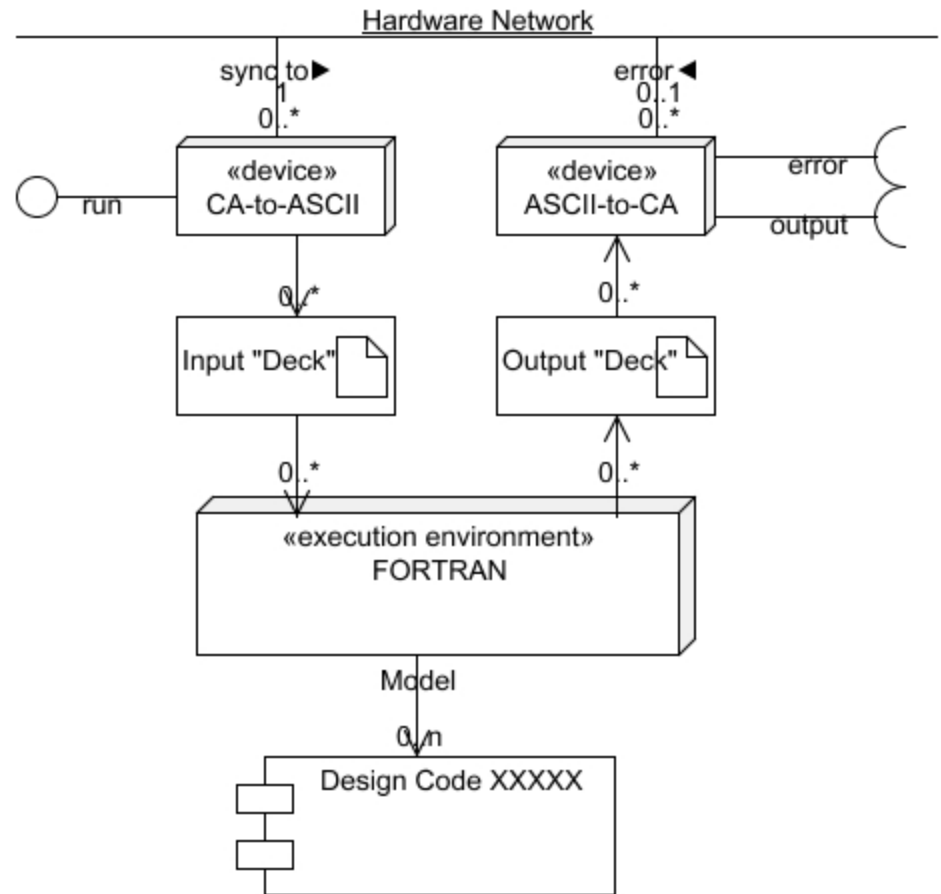
# Online Model Design
## Modern Design
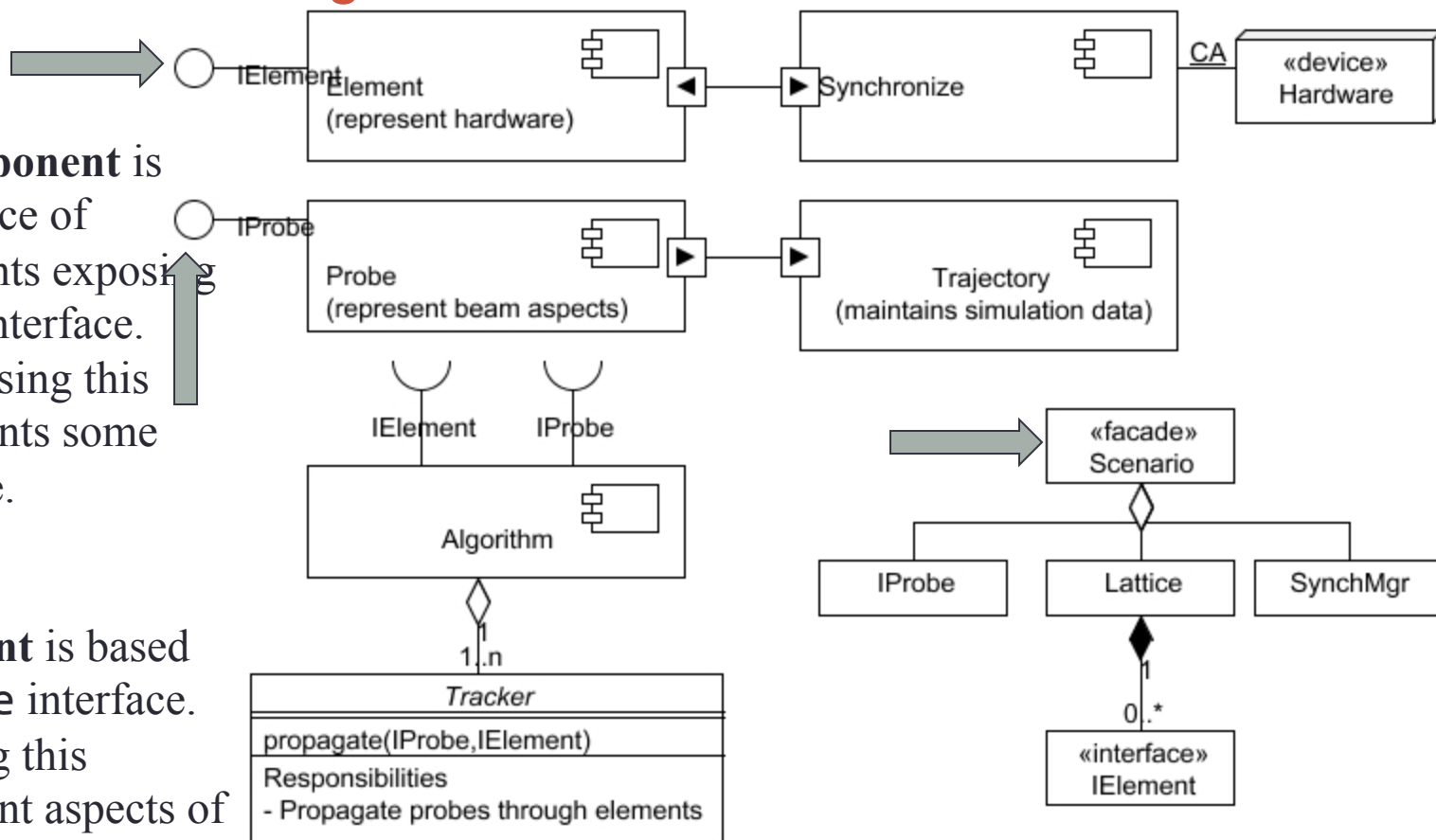
**Hardware component** is built as a sequence of modeling elements exposing the `IElement` interface. Any object exposing this interface represents some type of hardware.

**Beam component** is based upon the `IProbe` interface. Objects exposing this interface represent aspects of the particle beam, e.g., centroid, RMS envelopes, ensembles

The entire model is encapsulated by the Scenario class which manages the components

# Online Model Design

We can get as detailed as we need in out designs to elaborate point of concern.  However, attempting to model the code in every detail usually constitutes over-engineering.

The XAL online model is built upon the Element/Algorithm/Probe design pattern developed by N. Malitsky
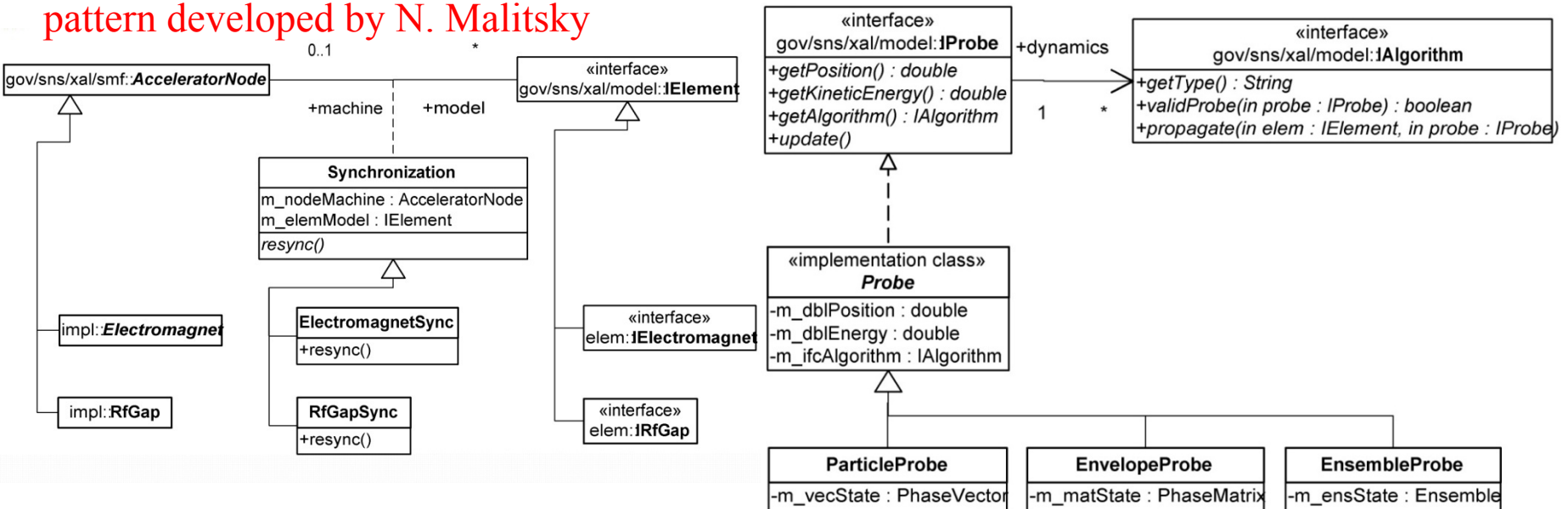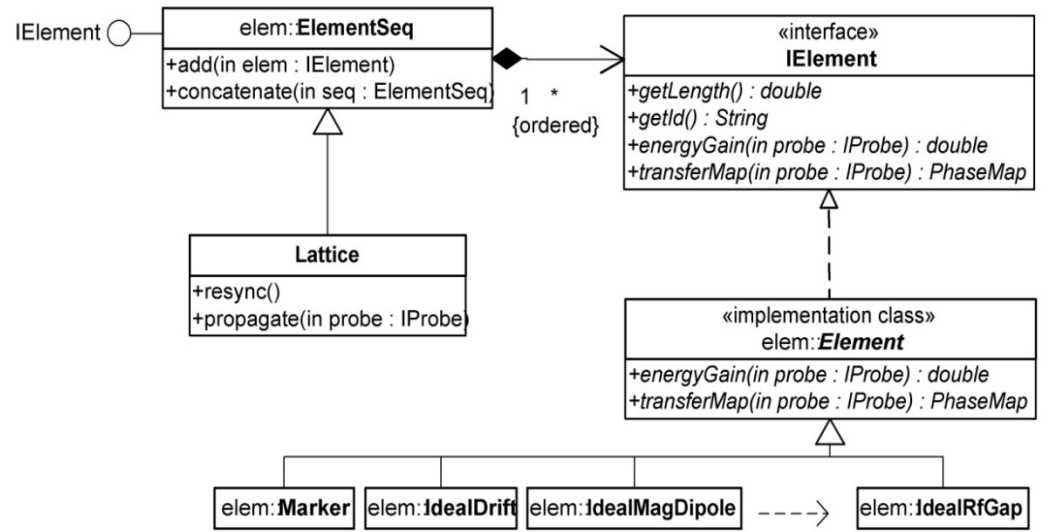
# XAL

## A Case Study in the High-Level Application Framework

- XAL is the progenitor of Open XAL
- XAL was designed and built at SNS with the intent that it be general accelerator application framework
- Discussed and designed from the beginning
  - Commonality of high-level applications was noted from previous accelerator projects
  - Many possible design implementation were discussed, at length, on paper, before any code was written.
  - Extensive analysis of existing systems and data formats

- The following are several early design illustrations (some in UML, most not)

# Example: Software Engineering a Framework

XAL — A Framework for Portable High-Level Control of Charged Particle Accelerators

Conceptual (Pre-Design) XAL Mechanism Diagram
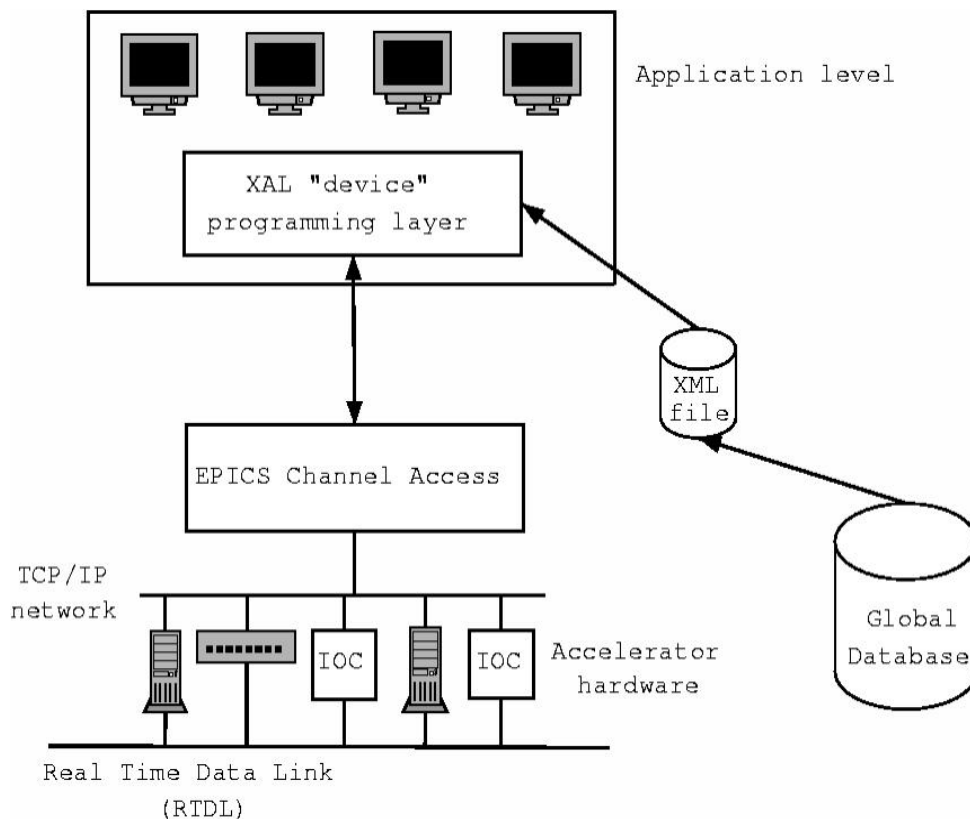
Use cases at the highest level
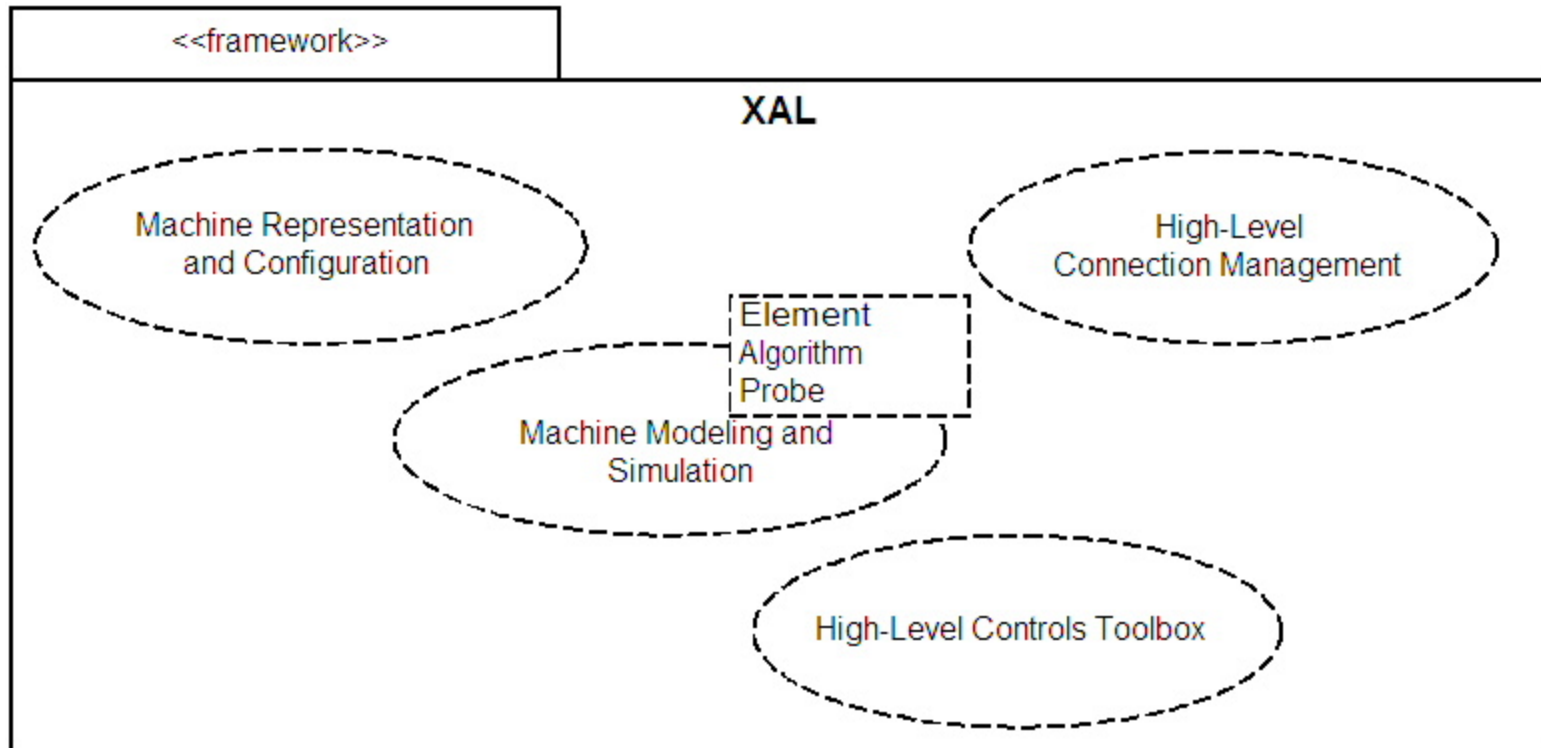
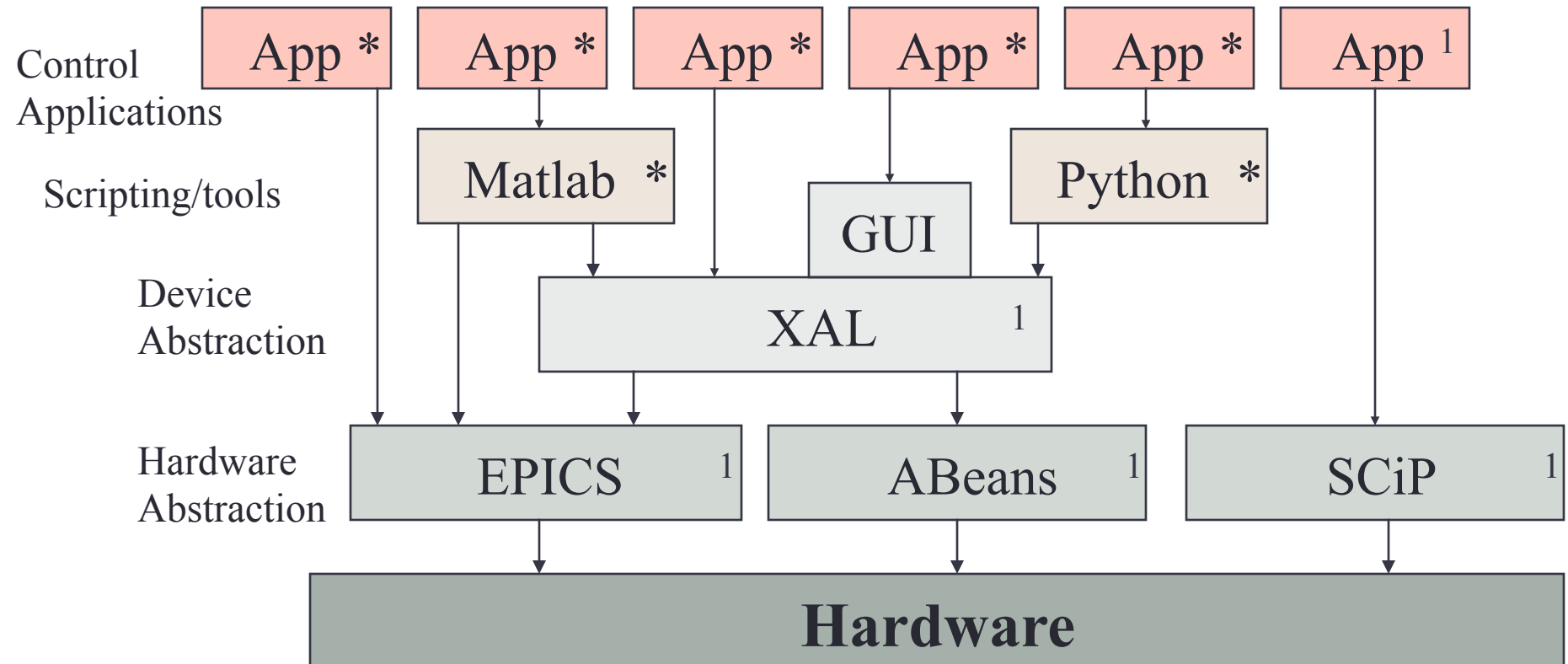# Example: Software Engineering

XAL

A Framework for Portable High-Level Control of Charged Particle Accelerators
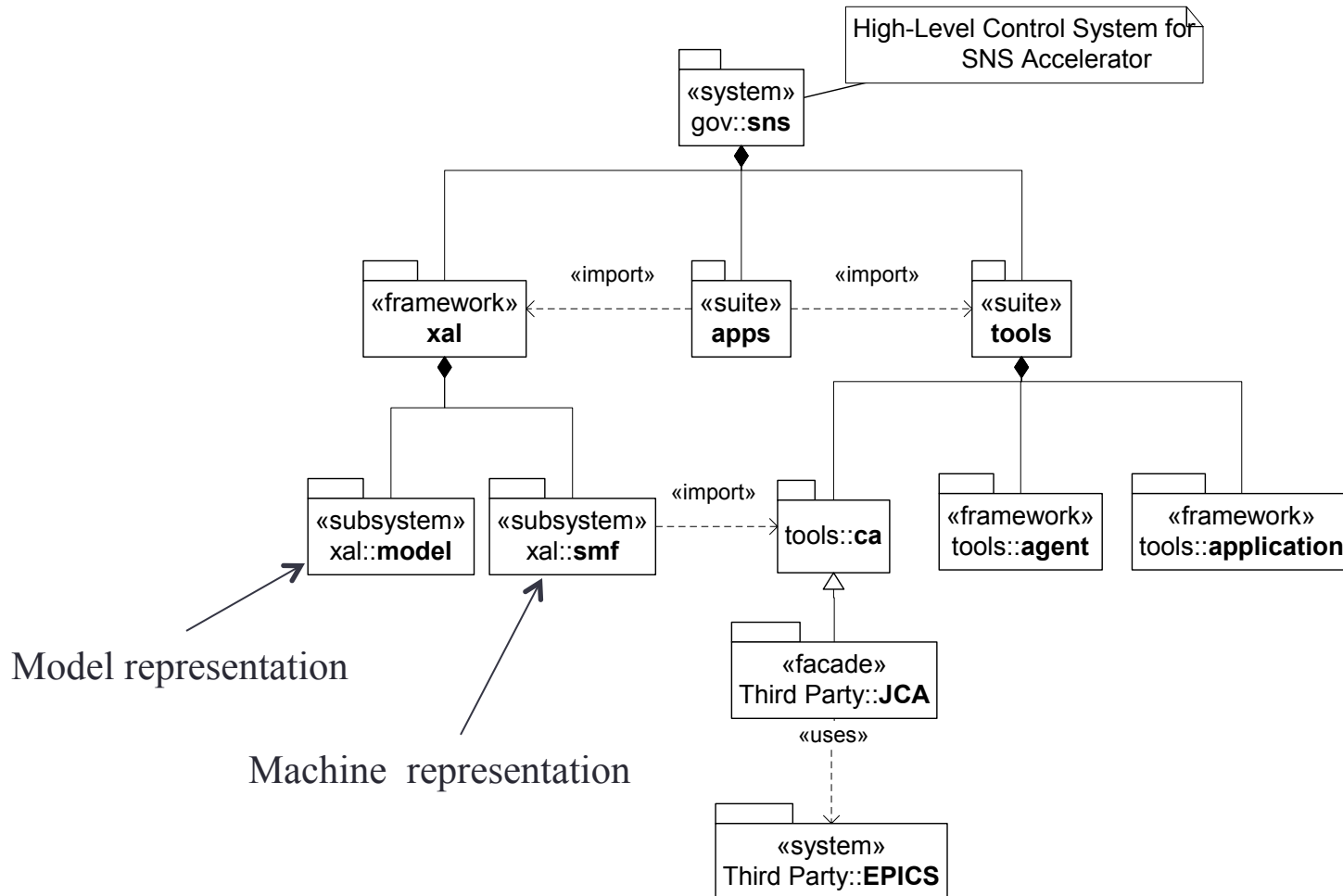
Conceptual (Pre-Design) Deployment Diagram

# XAL in the Control System Hierarchy

System Engineering and XAL

| | | | | | |
|---|---|---|---|---|---|
| **Control Applications** | App * | App * | App * | App * | App * | App [1] |

Scripting/tools: Matlab *, Python *

GUI

Device Abstraction: XAL [1]

Hardware Abstraction: EPICS [1], ABeans [1], SCiP [1]

**Hardware**

# XAL Architecture
## Subsystem Diagrams



Model representation

Machine representation
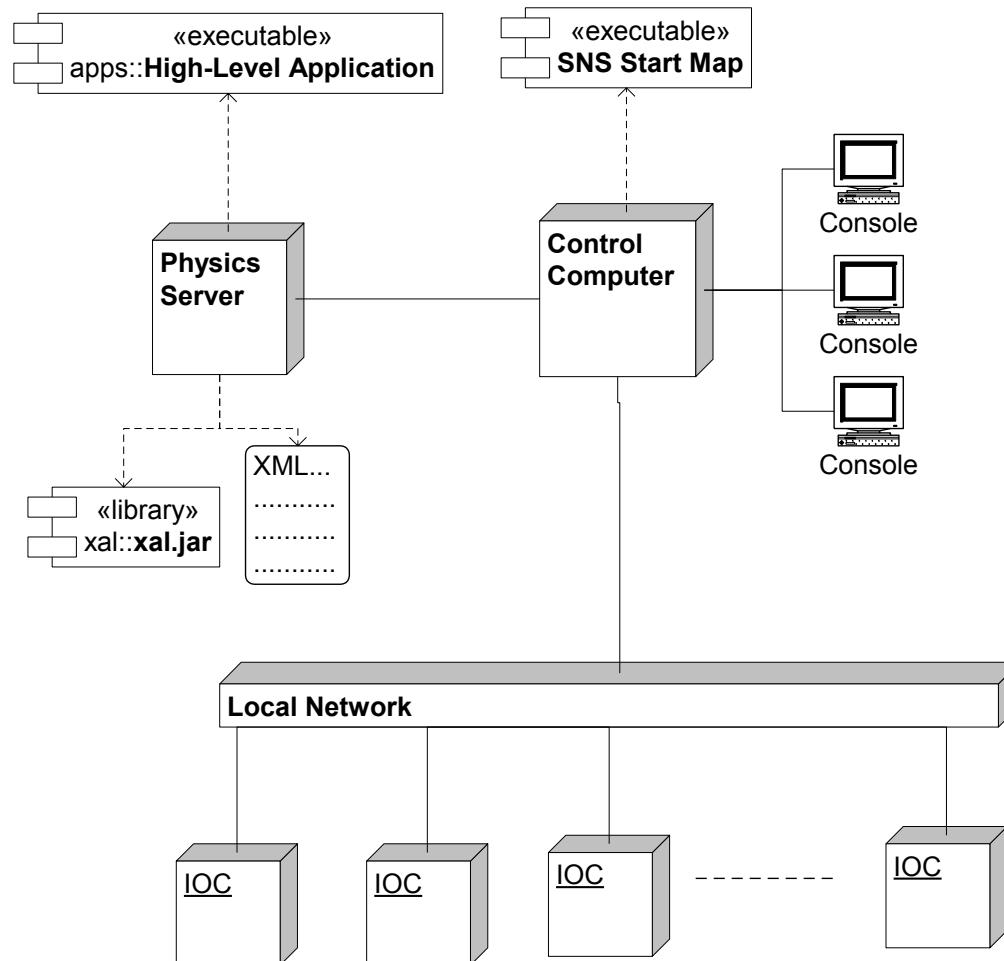
# XAL Architecture
## Class Diagrams

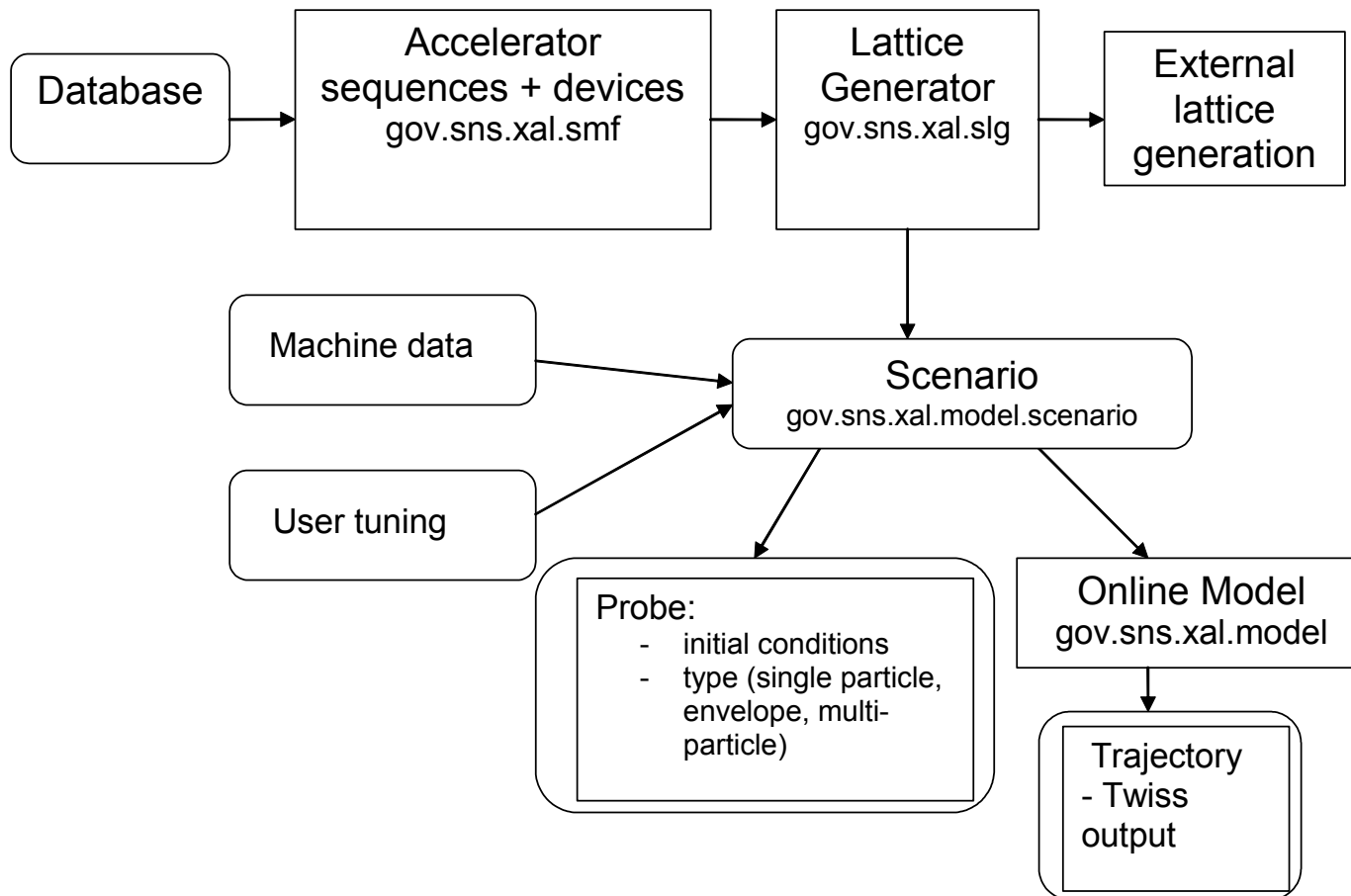Modeling Accelerator Hardware – Sector Tree
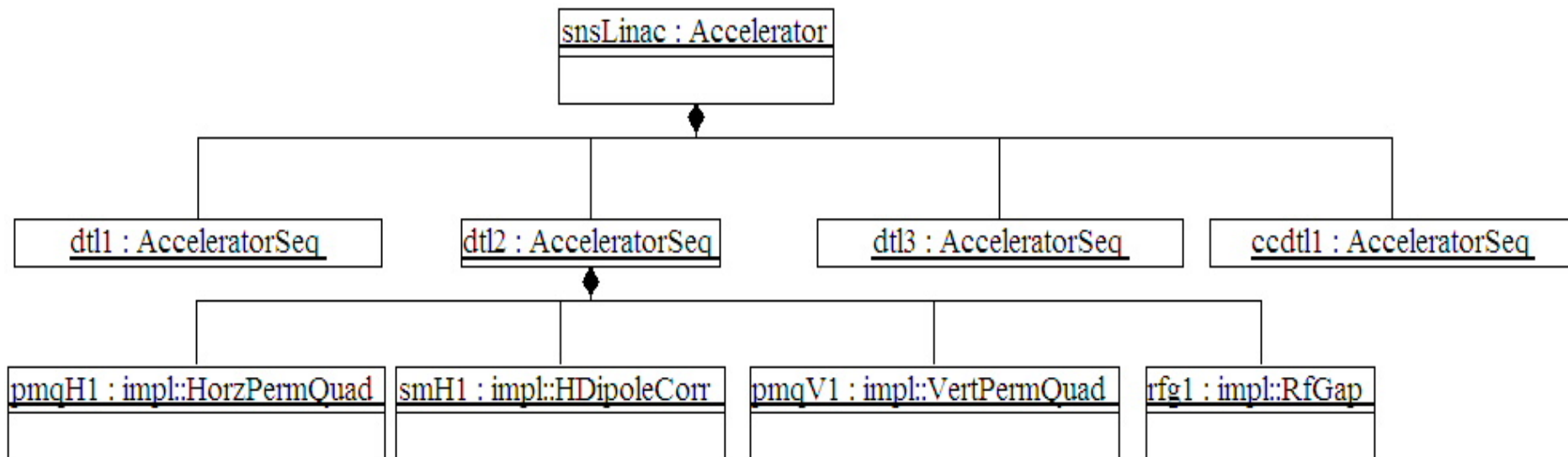
# XAL Architecture
## Deployment Diagram

# XAL Architecture
## Interaction Diagrams

(Here we are focusing on the online model)

# GUI Component
## GUI Application Framework

The GUI Application Framework was based upon the Document/View/Controller design pattern

- Provides a consistent "look and feel for all XAL applications"
- Avoid "re-learning" common operations in separate applications
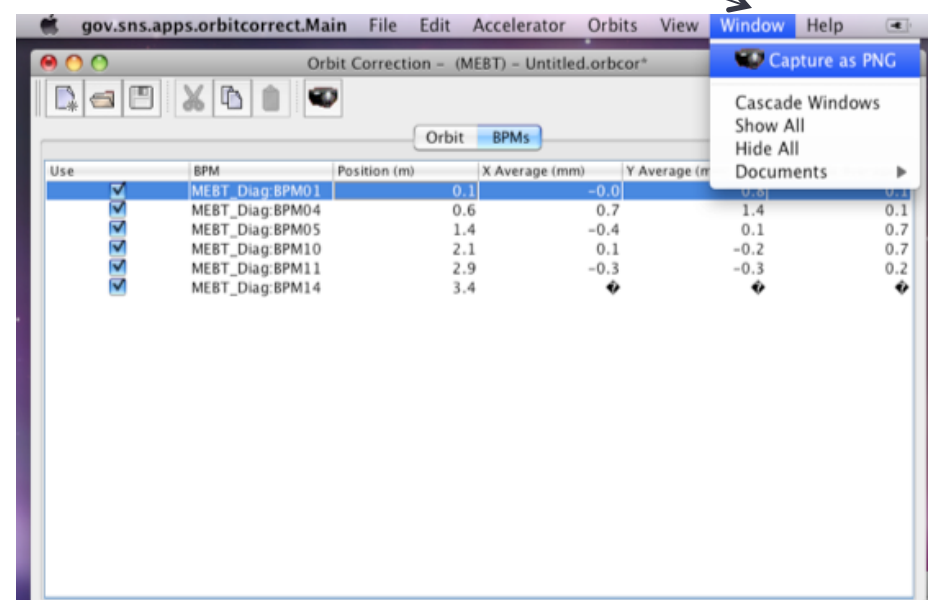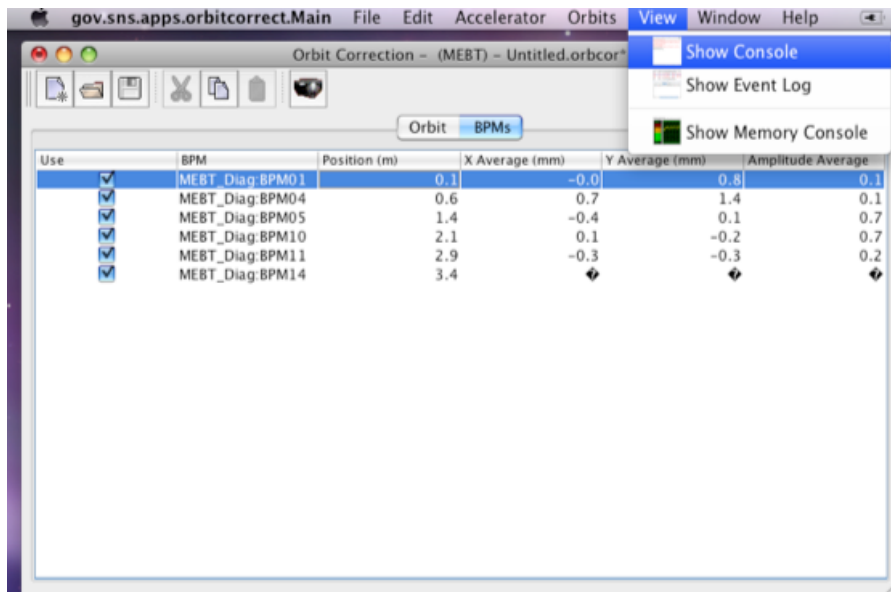- Upgrades available to all applications simultaneously

Standard menu items
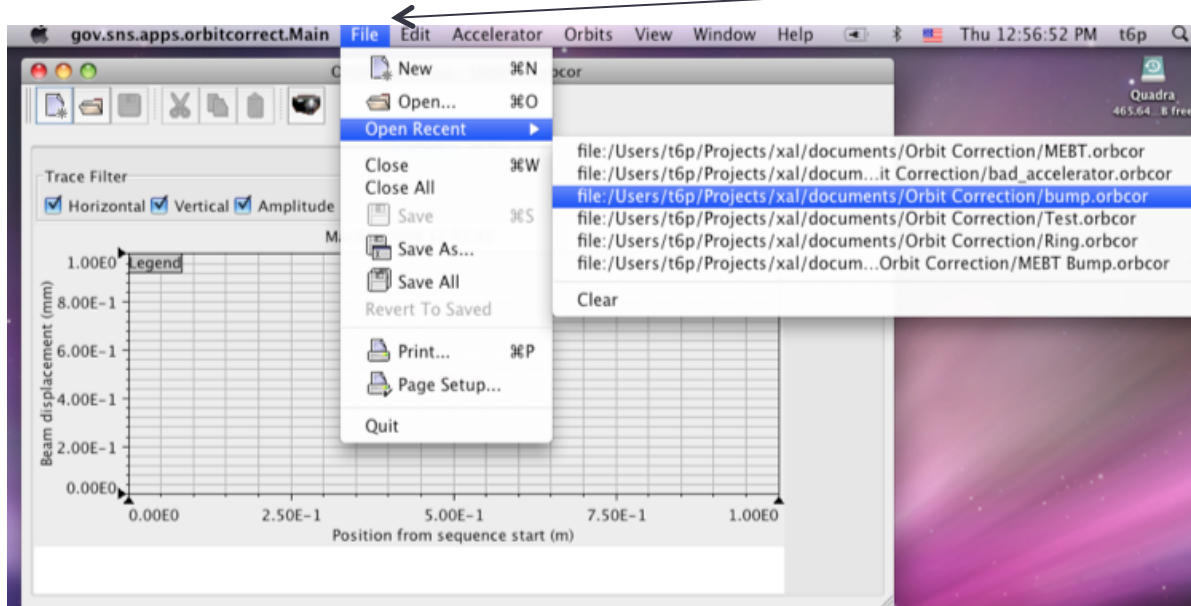
# GUI Component
## GUI Application Framework

Additional features available to all applications

# Application Framework
## Summary

- Application frameworks are a front-loaded approach to high-level control application implementation

  - They require substantial effort to build, but easy to maintain and upgrade, i.e., they are robust

- A major advantage to application frameworks is their dynamic configuration capability

  - Adapt quickly to hardware changes

- Use cases can help you visualize the needs of your high-level control software to design your framework

  - Common scenarios warrant implementation by single software components

  - The number of incoming edges of a scenario indicates its criticality – a failure in this scenario could be devastating to your system