The CERN PS/SL Controls Java API

P. Charrue, K. Kostro, M. Vanden Eynden
(CERN-SL) SPS & LEP

J. Cuperus, I. Deloose, F. Di Maio
(CERN-PS) Linacs (p+, Pb), Booster, PS, LPI & AD

W. Watson
(TJNAF)
Motivations

An initiative of the PS/SL Controls groups towards a common controls architecture

- Define a device model and the capabilities seen by the applications via the API
- Produce a stable interface for Java application developers
- Enter into the exciting Java world
PS/SL Java API Project
(April 98 - Dec. 99)

- Specify a Java API
- Deliver a first implementation on top of the *mature* PS & SL equipment access.

- The Java API is **CDEV** (new Java version)
  - TJNAF’s OO Controls API (C++, EPICS)
- TJNAF & CERN use the same Java version
- Some applications in operation (June 99)
Selected Milestones

May 1998:
Device model discussed in SOSH context

July 1998:
Why not using Jlab’s work instead of doing a similar work @ CERN?

October 1998: Vs “beta1” available

December 1998: Vs “1.0” ready (all PS devices)
PS/SL Java API Highlights

- Device-oriented & Device/Property Model
- Asynchronous, event-based I/O
- Connectivity to different communication infrastructures
- Directory Service (Next talk by J. Cuperus)
Named Devices
- Have Properties
- Organised in Classes

Device I/O Methods:
- get or set a property
  \textit{get(...)}, \textit{set(...)}
- monitor a property
  \textit{monitorOn(...)}
- execute a command
  \begin{itemize}
    \item take parameters
    \item return results
  \end{itemize}
  \textit{send(…)}

“BTP.D VT10” is a Magnet

status: int
command: int
current: double
...
// Create a Device
Device dev = new Device("BTP.DVT10");

// Get a Property
Data data = new Data();
DeviceError err = dev.get("Current", data);

if (err != null) throw err;
float current = data.getFloatValue();
Class `Magnet` extends `Device` {
  void `Magnet` (String name) {
    `super`(name);
  }
  float `getCurrent`() throws `DeviceError` {
    `Data` `data` = new `Data`();
    `DeviceError` `err` = `this`.get ("Current", `data`);
    if (`err` != null) throw `err`;
    return `data`.getFloatValue();
  }
}
Synchronous / Asynchronous Methods

- **“Synchronous”** Method
  - User thread waits for I/O completion
  - Exchange data and return errors

- **“Asynchronous”** Method
  - Separate thread activated as a result of the method invocation
  - Generate events
// Activate property monitoring
dev.monitorOn("Status", listener);

<<interface>> DeviceListener
deviceChanged(event: DeviceEvent)

DeviceEvent
getDevice(): Device
getProperty(): String
getOPCode(): int
getError(): DeviceError
ggetValue(): DataEntry
ggetCycleStamp(): long

Java Event Model
3-tier Architecture

I/O Service

User’s Java Virtual Machine

RMI, CORBA

Middle-tier Server

Separate Process

RPC, CORBA

Execute I/ O

Equipment Server

Implement virtual devices

Real-Time Environment
Conclusion

✓ A Java API has been specified and delivered

(Displayed next: new communication services

Another PS/SL project

✓ Applications and components can be developed

Another collaboration domain

★ A very productive collaboration