OO Approach to Control Apps and Machine Physics with Java Beans

M. Kadunc, I. Kriznar, M. Plesko, I. Verstovsek, G. Tkacik, J. Stefan Institute
Outline

- Product Description
- List of Features
- Technical Details
- Price List (ICALEPCS promotion special)
Problem Description

- Industry average of 60% project time on client development
  - a good design will have the largest impact here

- Application support takes more time than actual interaction with control system
  - SCADA, GUI-builders are mostly closed systems
  - standalone Widgets ignorant of control system issues (e.g. the fact that it’s a distributed system)
Goal: Be User Friendly

- Enable final users - application developers to develop applications for accelerator control (1) easily, (2) safely and (3) quickly
  1. professional programmers write the code, people who design GUI’s “build” applications like Lego blocks
  2. Present a clean, object oriented view of the accelerator to the final user
  3. vision: not to write a single line of code when producing an application

- Be platform independent and pluggable
  - so others can use our approach, too
The Result

- **Abeans library**
  - wrap communication with control system
  - model device/property/field: ps.current().units()
  - services for common application support
  - connect GUI components with control system in RAD (Rapid Application Development) tools

- **CosyBeans – complex GUI components**

- **DataBush library**
  - Control of machine in physics quantities
  - Intuitive access to devices for physicists
  - Separation of physics code from control system
**DataBush Framework**

- Maps optics device (magnet, etc.) ↔ Java object
- Bush-like structure of machine optics devices
- set/read to/from CS
- 1st order machine model calculation
- Initialization
- Configuration saving/loading
- Magnet calibration
- CS connection
- User interaction
Data Structure

DataBush
- init
- update
- apply
- CS Engine
- Calibration
- Factory
- Simulation
- Model

Optics List

Bends List

BPMs List

...
DataBush Elements

Quadrupole

double getPosition()
PositionedData getBeta(),
getAlpha(), etc.
TransferMatrix getMatrix()
double getLength()
double getCurrent()
double getQuadrupoleStrength()

- Use of RTTI for more flexibility
- Two modes:
  - connected: receives/sends values
  - simulated: holds values set by user

ICALEPCS 2001  M. Kadunc, I.Kriznar, M.Plesko, I.Verstovsek, G.Tkacik

Abbeans, CosyBeans and DataBush
Intuitive Usage

- **find element by name, index, through iterator**
  - `bpm = db.getBPMonitors().get("Name");`
  - `bpm1 = db.getBPMonitors().get(index);`
  - `bpm2 = db.getBPMonitors().elementIterator(bpm).previous();`

- **navigate through lattice**
  - `OpticalElement el = db.getOptics().elementIterator(bpm).next();`

- **get/set data from/to control system**
  - `result = db.update();`
  - `str = bpm.getName() + " " + bpm.getBeamPos().x;`
  - `while (correctorIterator.hasNext()) {`
    - `correctorIterator.next().setAngle(some_value);`
    - `elementList.add(c);`
  - `} db.applyCorrectors(elementList); /*ignore exception handling */`
With a Little GUI...
CosyBeans

- CosyBeans in PSpactal
  - Aframe (with glasspane)
  - a) selector
  - b) gauger
  - c) slider
  - d) LEDder
  - e) ATextPane

- Extensive features
  - initialize from CS at run-time
  - monitors -> events
  - alarms, timeout, shortcuts
  - resize-able, adjust shape,…
  - trend: history, save as CSV, histogram, zoom, …
Visual Composition in RAD Tool
Abeans

- Application framework

- Hide communication layer
- Provide application services
- Provide CS models
- RAD, easier maintenance

- Applications
  - Device server
  - Hardware

Σ:
- RAD, easier maintenance
Visual Programming

- Panels in RAD (no hand-written code)
  - implement default behavior
    - notifications to user when communication errors occur
      - timeouts
      - exceptions
      - interpret error numbers
    - callback, monitor and alarm queued & dispatched as event
  - default lifecycle management
    - initialization when GUI becomes visible
    - connection when all necessary parameters have been set
    - destruction when application closes
- ServiceBean provides access to services in “visual programming” mode
Serious Programming

- Manual programming with Abean device Beans
  - device manual = API (Bean properties & methods)
  - compile-time error checking through strong typing
  - sync and async program flow (through lock object)
  - families of Abeans for same behaviour
  - ~10 event types, ~15 system properties

- Abeans discover vis/man mode => change behaviour

- Aggregate data for display of large number of similar devices
Common Services

- Resource initialization and destruction
- Connection management
  - Data delivery (timeouts, threading options)...
- Logging (to outside world)
- Reporting (inside application)
- Exception and error handling
- Policy and configuration management
- Authentication (to be done)
Is it possible or is it just a dream?

Reality:
- Most of the models (device beans) can be shared, at least as superclasses
- Certain types of applications can be shared (with RAD this is not a major drawback)
Plugs II

- Pluggable architecture (green part)
  - Functionality declared by blue API must be provided
  - Simulation plug for quick tests
  - XSLT generator makes plug implementations (ESO, ANKA)
  - Nicely implemented using Java interfaces
Highlight of Release 3

- Allow both wide and narrow interface to the devices
  
  - “Abeans for TINE” (THAP025)
  - Extensive use of Java libraries (JAAF, JAF, InfoBus)
  - As R2 separated plugs from beans, R3 separates models (beans) from framework
  - In R3 even services are pluggable
**Highlight of Release 3**

- **Composite Pattern:**
  - Put all components into a tree
  - Apply Visitor pattern

- **Gain:**
  - Easy tree traversals (recurse)
  - Independence of “virtual applications”
  - All applications run in one JVM
  - Well-defined startup / shutdown order
Beans in Practice (1)

- A power supply Bean
  
  ```java
  public class PowerSupplyBean extends DeviceBean {
      
      // properties
      RWDoubleProperty current;
      RODoubleProperty readback;
      ROPatternProperty status;
      
      // methods
      Completion on() {...}
      Completion off() {...}
  }
  ```

- Trivial to add any new device (or generator does it)
Beans in practice (2)

- A read-write double property type (EPICS AO)

```java
public class RWDoubleProperty extends DoubleProperty {
    // monitor handling
    void addPropertyChangeListener(PropertyChangeEvent e) {...}
    // methods
    Completion set(double value) {...}
    double getMinValue() {...}
    String getUnits() {...}
    double get() {...}
    ...
}
```
Beans in practice (3)

- Narrow interface support (e.g. TINE)

```java
BasicRemoteInfo info = new BasicRemoteInfo("/HERA/HEPBPM/WL197_MX/ORBIT.X","Channel-TINE");
DoubleChannel channel = new DoubleChannel(defaultFamily, info);

double singleValue = channel.getValue(new Completion());
double[] multipleValues = channel.getValue(41, new Completion());
double minimum = channel.getMinimum();
String units = channel.getUnits();
```
Conclusion

- Framework greatly improves development of reliable clients, but is a large investment
  - Abeans must be reused to justify their creation
    - ANKA: Abeans R2 on Visibroker CORBA
    - Riken: Abeans R2/R3 on Orbacus CORBA (test)
    - ESO: Abeans R2 on Orbacus CORBA/BACI (prototype)
    - DESY: Abeans R3 on TINE (test system)

- Next step: generators (see THCT004)
  - IDL (to BACI standards) -> XML, XSLT -> Plugs, Device and Property Beans
ICALEPCS Promotion

- E-mail-in rebate of $10
  - if Abeans (old release) or Databush demo installed and tested at your site
  - (Windows-only installer)
  - valid for registered ICALEPCS participants
  - offer expires Dec 31st, 2001

- Proof of ownership: screenshot
  - e-mail with IP number of your machine to: info@cosylab.com
Universal Containment

- Problems
  - Multiple panels require multiple JVMs → no memory
  - Control of a large amount of device beans (iterations)
  - Optimizations possible for large amount of device beans (packed monitoring)
  - Transparent containment and organization of programs
  - Life cycle management (when to create the framework, the order of creation and destruction)
Composite Pattern

- Put all components into a tree
- Gain:
  - Easy tree traversals (recurse)
  - Independence of “virtual applications”
  - Apply Visitor pattern
  - Unique string names of tree nodes
  - Well-defined startup / shutdown order
Composite Pattern II

Organization of Abeans into two parts
- Fixed (framework) – managers, loaders, services
  - Independent of CSs! Reusable application building framework
  - Independent of the variable part
- Variable – applications, device beans
  - New nodes added / removed during RT
  - Use framework services

Root

Framework

Variable part of the tree
Components

Problems
- If you have a logging (configuration) library, how do you change or modify it?
- If you have an application that uses authentication mechanism, how do you make the mechanism available to all applications?

Design issues
- Modifying / extending framework functionality
- Solving framework thin vs. heavy dilemma
Components II

- **Create components:** tree nodes with well-defined management accessible only through their interfaces

- Components allow you to change implementation, even at RT (state transfer)

- Keep default implementations simple as placeholders for future (COTS) implementations
Components III

- **Existing components in R3**

- **Gain for application programmers**
  - Uniform and prescribed way of adding new functionality
Quality of Service

Problem

- Create uniform way of applying settings and monitoring if they are still valid

Example

- Binding data delivery to conditions: timer, delta change etc.
- Browsing the set of effective settings
Quality of Service II

- Typical framework programming:
  - Short term: bigger investment because of centralization
  - Long term: profit because there is no need to do separate designs for each application (single solution)
How Do We Define the Device Interfaces

- "Narrow"
  
  ```java
  ctrlObj.remote(device, "msg", dataIn, dataOut)
  ```

- "Wide"
  
  ```java
  dataOut = device.msg(dataIn)
  ```
An Accelerator Device Maps to an Object

interface PS {
    ADC: double RO
    DAC: double RW
    status: bits RO
    on(): void
    off(): void
}

RampedPS : PS {
    start(): void
    stop(): void
}
Properties are Objects by Themselves

Properties are:
- RO properties
- RW properties

Methods:
- get()
- getHistory()
- defaultValue()
- resolution(), minStep()
- graphMin(), graphMax()
- description(), units()
- newAlarmMonitor()
- alarmLowOn()
- alarmLowOff()
- alarmHighOff()
- alarmHighOn()
- set()
- setNonBlocking()
- increment()
- decrement()
- minValue()
- maxValue()
core Abeans package

- policy manager
- component manager
- loaders
- basic log service
- containment hierarchy
- static methods to managers
framework package

- support for applications
- manage application context
  - life cycle
  - request services
models (controlled interfaces)

- Connectable :: Linkable
- all connect/disconnect events
- events with data and timestamps
- services
  - JNDI
- concrete models n subpackages
  - models.baci
  - models.banking
plugs

- propagate quality of service requests
- user can enforce optimizations
  - caching
  - join requests
    - buffering
    - AMI
  - use dynamic CORBA