A NEW EXTENSIVE SOURCE FOR WEB BASED CONTROL EDUCATION – CONTLAB.EU

Martin ČECH, Miloš SCHLEGEL, Pavel BALDA, Ondřej SEVERA

NTIS / Department of Cybernetics,
University of West Bohemia in Pilsen,
Czech Republic

IFAC ACE 2013, Sheffield, UK
Content

- Virtual laboratories in general
- ContLab and PIDlab history
- Three platforms idea
- Implementation details
- Illustrative examples
  - PID autotuners
  - Advanced PID control schemes
- ContLab as a first step of rapid development cycle
- Conclusions
Virtual laboratories in general

PROS

- Allow students, researchers and technicians to do virtual experiments with fundamental physical or control problem on their PC
- Interactivity
- Local vs. remote simulation core

CONS

- Unfortunately, virtual labs often present only simple problems, the development tools oriented to process modeling (e.g. EJS)
- Usually coded manually, straightforward migration to RT control is very difficult
- Rarely present advanced control structures
ContLab and PIDlab history

- For more than 7 years, PIDlab works as a popular tool for robust PID controller design and tuning.
- It presents several unique algorithms for process identification and PID controller design which are not available anywhere else.
- These tools are used by hundreds of universities and companies worldwide.

- The PIDlab popularity was an inspiration to provide more interactive tools with direct applicability at a real-time platform.
- Moreover, there is a clear motivation to present more automation topics above the scope of PID control – predictive control, vibration damping, motion control, etc.
Mutual connection of three platforms

1. Simulation, research and development platform
2. Real-time platform
3. Presentation platform
Implementation details - Local simulation core

- more programming on the client side (development platform is necessary)
- provide maximum possible interactivity (immediate response)
- can be run also off-line (without internet connection)
Implementation details – Java programming language

- **Advantages**
  - Platform independence – the same code can be run on different operating systems with Java Running Environment - Windows, Linux, Solaris, MacOS
  - Integration in web browsers (Java applets)
  - Large possibilities of GUI layout configuration (Swing, now Java FX)
  - Free development tools and running environment
  - A lot of supporting libraries
  - Java is taught as a basic programming language at a lot of universities.

- **Disadvantages**
  - The process control experts must be also excellent programmers
  - One cannot use high level libraries and advanced mathematical support like in Matlab or Sysquake
  - You have to create your own simulation core
  - To implement joint plots interactivity, you have to create your own package or look for some free or commercial Java packages
  - The development time is bigger
In the past – manual coding (first applets on PIDlab.com)

- very successful (hundreds of companies and universities registered)
- helped us to transfer the particular technology from academic to practical world
- motivation to develop routine procedure for virtual lab development
Advanced function block library in JAVA (automatically generated from C-coded original)

PID autotuners, predictive controllers, sliding mode controllers, active vibration damping, motion control, sequence logic, state automats, advanced input shaping filters, PWM, process models, switches, … (more than 200 blocks)

```java
public class ADD extends Block{
    public static XInitVar[] xInInitVars = new XInitVar[]{
        new XInitVar("ul", -1,
            new XInitCfg((short)0, (short)XAV.XATM_XANY_VAR, XAV.MIN_DOUBLE, XAV.MAX_DOUBLE,
                new XInVar(XAV.XATC_XDOUBLE)) ),
        new XInitVar("u2", -1,
            new XInitCfg((short)0, (short)XAV.XATM_XANY_VAR, XAV.MIN_DOUBLE, XAV.MAX_DOUBLE,
                new XInVar(XAV.XATC_XDOUBLE, (double) 0)) ),
    }

    public static XOutInitVar[] xOutInitVars = new XOutInitVar[]{
        new XOutInitVar("y", -1,
            new XOutCfg((short)0, XAV.MIN_DOUBLE, XAV.MAX_DOUBLE,
                new XOutVar(XAV.XATC_XDOUBLE)) ) );

    public static XStatInitVar[] xStateInitVars = new XStatInitVar[]{};

    public static XArrInitVar[] xArrInitVars = new XArrInitVar[]{};

    public ADD(String name) {
        super(name);
        nInCount = (short) xInInitVars.length;
        nOutCount = (short) xOutInitVars.length;
        nStateCount = (short) xStateInitVars.length;
        nArrCount = (short) xArrInitVars.length;
        nExtInCount = 2;
    }

    public void init() throws JReXLibException {
    }

    public void main() throws JReXLibException, RexServerException {
        super.main();
        getOutAt(0).xDouble = getInAt(0).xDouble + getInAt(1).xDouble;
    }

    public void exit() throws JReXLibException {
    }
}
```
public class TaskExample extends Task{
    public TaskExample(String name) { // class constructor
        super(name);
    }

    public void init() throws JRexLibException, RexServerException{
        // Blocks declaration, allocation. By declareBlock() the block 'tels' the task number of its variables
        CNR SP = new CNR("SP"); declareBlock(SP);
        CNR HV = new CNR("HV"); declareBlock(HV);
        PSMPC PSMPC = new PSMPC("PSMPC"); declareBlock(PSMPC);
        MDL Model = new MDL("Model"); declareBlock(Model);
        // the task allocates arrays of inputs, outputs, states, and buffers for all declared blocks
        allocateMemory();
        // in addBlock(), the block inputs, outputs, states and buffers are created and inited
        addBlock(SP);
        SP.setParamDouble(0, 5.0);
        SP.setPeriod(0.01);
        SP.init(); // HV block initialization was removed from the code listing
        addBlock(PSMPC);
        PSMPC.setParamDouble(0, 1);
        //.... // For easy reading, the remaining PSMPC params were removed from code listing
        PSMPC.setPeriod(0.01);
        PSMPC.init();
        addBlock(PSMPC);
        Model.setParamDouble(0, 10);
        //.... // For easy reading, the remaining MDL params were removed from code listing
        Model.setPeriod(0.01);
        Model.init();
        connectBlock(0, 0, 1, 0);
        connectBlock(1, 0, 2, 0);
        connectBlock(2, 0, 1, 1);
        connectBlock(1, 0, 1, 2);
        this.setPeriod(0.01);
    }
}
Implementation details - JAVA classes structure

- automatic building and packaging system was developed
- automatic generation of JNLP and JAR file
- Java WEB start is used for deploying virtual labs

Function blocks used in the lab

JAR FILE

TASK

GUI Panel

Frame (application)

Resources (Czech, English, …)

Applet

GUI library

GUI classes
Example 1: PID pulse autotuner demo
Example 2: Pulse-step predictive controller demo
Example 3: Smith predictor demo
Example 4: Floating control
ContLab as a first step of rapid development cycle

MIL - ContLab virtual laboratory, non RT Java engine

SIL - soft RT SW scheduler (notebook) + remote I/O (ARDUINO)

PIL/HIL - RT SW scheduler Raspberry PI + remote I/O (ARDUINO)

... will be explained more precisely in the next presentation
Conclusions & further work ideas

- JavaREX platform for automatic generation of advanced virtual laboratories was presented
- Examples of new labs based on the JavaREX platform were presented (PID pulse tuner demo, predictive controller demo, …)
- The labs are available on [www.ContLab.eu](http://www.ContLab.eu) and [www.PIDlab.com](http://www.PIDlab.com) (only PID related labs)
- All the labs are accomplished by short illustrative videos
- They are used at UWB in master courses „Industrial control systems“
- Future idea - present PLCopen motion control blocks

This work was supported by the European Regional Development Fund (ERDF), project NTIS { New Technologies for Information Society, European Centre of Excellence, CZ.1.05/1.1.00/02.0090.. The support is gratefully acknowledged.