RCP for Industrial Automation and Real Time Control

S. A. Robenalt
steve@webcircuit.com

Project Definition

- Startup company
  - Developing a new manufacturing process
  - Company has extensive industry experience
- Machine Control via standard controllers
  - PLCs (Programmable Logic Controllers)
  - Lab Instruments and Measurement devices
- Need software to run the operations
  - Define (and refine) the processes
  - Control, display, and monitor the processes

State of the Industry

- Custom applications for process control
  - Usually built with Visual Basic
  - PLC interface via OLE for Process Control (OPC)
  - User interaction via PC or dedicated control device
- Java is still immature in the market
  - Real-Time Java not widely deployed
  - Hardware (device) interfaces not there yet

So Why RCP?

- Experienced developer
  - Extensive background with Java and Eclipse
- Integrates well with platform native code
  - Important for device communications
- OSGi module layer
  - Provides better module control
  - Declarative services option for devices
- EMF for core program logic
  - Rapid iterative development of internal models

Application Architecture

- Process Controller is an RCP Application
  - Contains ~40 custom plugins and ~40 RCP plugins
  - Core models built with EMF
    - EMF Model instances are configuration files
  - Realtime execution is single dispatch multi-thread
    - Dispatcher thread runs at 1 ms interval
    - Worker threads are JSR 166 ThreadPoolExecutors
  - User Interface based on UI forms, SWT, JFace
  - Push most disk interaction to host platform

Eclipse and RCP Frameworks

- Use EMF to build 5 core models
  - Recipe, Process, Operation, Facility, Visualization
  - EMF XMI persistence to store models
- UI built from SWT, JFace, UI Forms
  - Controls wired primarily with databinding
  - Not yet declarative, but working toward it
  - Custom graphical displays for monitoring
- Use BIRT for offline reporting
  - Process reports on server platform, not controller

Realtime Performance

- Sequencing via central dispatch thread
  - Start each task as close as possible to exact time
  - Most tasks complete within 1-2 ms of end time
- Worker threads are pooled
  - Created once, used for many tasks
  - Single thread often necessary for native code
- Garbage collection minimized
  - Pool and reuse objects where possible
  - GC pauses have not been significant factor

Development Process

- Generally agile methodology
  - 2 primary developers
  - 1 tester/developer
  - 1 manager/tester
  - Extensive testing in simulated mode
  - Final testing is live, takes a long time
- New versions produced every 2-4 weeks
- Many issues addressed by configuration

Problems and Workarounds

- Primary developers had no access to hardware
  - Simulate hardware where possible
  - Short test programs to validate hardware interface
- Threading issue with device interface
  - Use SWT-style execution model (single task thread)
- EMF issues with ids
  - Spaces in variable names broke EMF ids
  - Numeric EMF ids must be unique (CCEx)
  - Control EMF ids – use unique names only

Future Directions

- Migrate to fully declarative process definitions
  - Partially implemented for process modeling
  - User Interaction needs more work
- Push functionality to supporting applications
  - Database and reporting server
  - Model editor for system configuration
  - Remote monitor for support personnel
- OSGi Declarative services
  - Need to have better control of devices

What about the code?

- Will it be released?
  - We have verbal agreement to allow us to do so
  - Terms are not yet determined
- Will it be commercial or open source?
  - Open source if it makes sense to do so
  - Need to identify a community to do it right
  - Some relevant specifications are not open
- If open source, under what license?
  - Most likely EPL or BSD style license

Questions?

Thanks for your time!