An Object-Oriented Software Architecture for The Control of Flexible Parts Feeding Systems

Greg Causey
M.S. Thesis Defense

Dept. of Electrical Engineering & Computer Science
Case Western Reserve University
Nov 7, 2001

Outline

• Introduction
• Control Hardware / Setup
• Overall Software Architecture
• High-Level
• Mid-Level
• Server-Level
• Hardware-Level
• Results / Future Work
Introduction

- Goal: Software architecture for all flexible feeders
  - Easier programming
  - Easier hardware up-grades / replacements
  - New designs ready quicker
  - Reduced in-house expertise
  - Focus on problem

- How
  - Break feeder into common sub-systems
    - Presentation sub-system
    - Location sub-system
    - Retriever sub-system

Introduction

- Philosophy
  - Vertical hierarchy
  - Horizontal segregation

- Top-level
  - System state
  - Current part

- Mid-level
  - Present parts
  - Locate parts
  - Get parts

- Server-level
  - Control hardware
  - Robot move to ...
  - Camera grab image ...
Controller Hardware

- **PC: WinNT, PIII 600, 128mb**
  - Overall control
  - Vision processing (MIL)
  - User interaction
- **Galil 1822 motion control card (PCI)**
  - Conveyor control
  - I/O
- **Matrox vision system**
  - Meteor II (PCI)
  - MIL
  - Pulnix 6702, prog scan, async reset camera
- **Adept AWC controller (Ethernet)**
  - Robot control
  - I/O

Controller Hardware

- **Communications**
  - **Galil: PCI Bus**
    - Shared Memory
    - DMCWriteData()
    - DMCReadData()
  - **Matrox: PCI Bus**
    - MIL
  - **Adept: Ethernet**
    - Sockets
- **Time critical communications**
  - Conveyor encoder <-> Adept controller
  - Camera trigger <-> Conveyor encoder latch
Controller Hardware

Overall Software Architecture

- **Top-level**
  - Remote users (GUI)
  - remote_user objects
  - main_control object
- **Mid-level**
  - Asynchronous threads
  - Parts specific objects
  - Utility objects
- **Server-level**
  - Robot server
  - Robot position server
  - Conveyor server
  - Vision server
  - I/O server
Top-Level

• **GUI**
  - Enables remote control of the feeder
  - Connection via Ethernet sockets
  - Java: portability
  - Two threads
    - Watch for events, send requests
    - Update GUI from system updates

Top-Level

• `remote_user` object
  - One per logged-in user
  - Verify credentials
  - Handles log-in and log-out
  - Two threads
    - Listen for user requests
    - Send system updates

• `main_control` object
  - Oversees system operation
  - Maintain system state and status
  - Create mid-level asynchronous threads
  - Separate thread for updates
Top-Level

- **System start-up / shutdown**
  - `main()`
    - Initializes system
    - Listen for users login requests
    - Shuts down system
  - Create all singleton based server objects
    - Enforce creation order
    - Catch failures
  - Open user login socket
  - Listen for login’s or shutdown request
  - "q" key press to shutdown
  - Destroy singleton based servers
    - Enforce destruction order

```c
if(start_up_servers() != SUCCESS)
    exit(-1)

soc = open_socket()

while(continue) {
    listen for login or keypress

    if(login)
        *temp_user = new remote_user(soc)
        create_thread(temp_user)

    if('q' keypress)
        continue = FALSE
}

shutdown_servers()

exit 0
```
Mid-Level

- Asynchronous threads
  - Presentation, Location, Retrieval, Adjustment
  - Create by `main_control` at system start
  - Always running
    - `RUNNING`: perform task
    - `STOPPED, PAUSED`: sleep
  - Use part specific objects to perform actions
  - Hand-shaking with shared variables

Mid-Level

- Part specific objects
  - Correspond with asynchronous threads
  - Tailored to a specific feeding situation
    - Part specific requirements and algorithms
    - System parameters and settings
  - Common functionality in base classes
    - E.g. `too_far()`
  - Abstract factory design pattern
  - New part introduction
    - ~500 - 1000 lines
    - Including in-source documentation
    - 2 - 8 hours
Utility objects
  Encoder synchronization
    Part location
    Different sized encoder counters
      Galil: 32 bit signed
      Adept: 24 bit signed
    Singleton creation
    Sync at start-up
    Add difference with roll check
    Trouble if larger counter loops without update
Parts queue
  FIFO between locator and retriever
  Holds part locations
  Singly linked list
  Singleton creation
  put_front()
Server-Level

- Five servers
  - Robot motion
  - Robot position
  - Vision
  - Conveyor motion
  - Digital I/O
- Singleton design pattern
- Encapsulate hardware / communications
  - Proxy / Adapter design patterns
- Provide access control
  - Mid-level threads do have to coordinate
  - Monitor construct

Server-Level

- Robot motion server
  - Open socket to Adept system at creation
  - Blocking and non-blocking methods
  - Methods
    - Robot motion
    - Parameter set/get
    - Tool offset
    - Utility
- Robot position server
  - Open socket to Adept system at creation
  - Reports current robot location
  - Reports Adept conveyor encoder counter and belt transform
  - Separate from robot motion server due to blocking methods
Server-Level

- **Vision server**
  - Initialize Matrox system at creation
  - Encapsulate MIL functions
  - Performs robot <-> vision calibration
  - Methods
    - Image acquisition
    - Image analysis
    - Parameter set/get
    - Calibration

- **Conveyor motion server**
  - Control of inclined and horizontal conveyors
  - Millimeter <-> encoder count conversion
  - Galil digital I/O
  - Conveyor distance notification
  - Notification
    - Distance
    - Input change
    - Event -> Interrupt -> Galil lib -> Win msg -> message thread -> Registered objects
  - Methods
    - Conveyor motion
    - Parameter set/get
    - Digital I/O
    - Registration
Server-Level

- **I/O server**
  - Consistent access point and numbering
    - Physically: Galil and Adept
  - Maintain current output state
    - `set_io()` checks state before action
- Input change notification
  - Two threads and helper class
  - Thread 1 listens for Adept changes
    - Notification via socket
    - Forwards notification to thread 2
  - Thread 2 listens for Galil and thread 1 changes
    - Message queue
  - `io_notify` maintains list of registered objects
    - Array of linked lists
    - Each bin -> input point
    - Multiple objects per input point
    - Multiple input points per object
Hardware-Level

- Guaranteed timing of servo-level actions
- Leverage vendor controllers
- Two methods of interaction
  - Locally executing programs
  - Single commands
- Three controllers
  - Galil motion controller
  - Matrox Meteor II frame grabber
  - Adept AWC controller

Galil motion controller
- Two letter command and programming language
- Conveyor motions and parameters via single commands
  - \texttt{move(100)} \rightarrow \texttt{"AM X 22190"}
  - \texttt{set\_speed(50)} \rightarrow \texttt{"SP X 11095"}
- Distance and I/O notifications via local programs
- Latch input via built-in function
  - \texttt{"CN ,,1"}
  - \texttt{"AL X"}
  - \texttt{"_RLS"}
Hardware-Level

- Matrox Meteor II
  - No localized processor
  - MIL is effectively the programming language
  - Vision server makes MIL or series of MIL function calls
- Adept AWC controller
  - V+
  - All locally executing programs
  - Four independent processes
    - Robot motion server
    - Robot position server
    - Digital I/O server
    - Input change notification process

Hardware-Level

- Robot motion server
  - Read socket
  - `decode_and_do()`
    - Determine request type
    - Decodes request parameters
    - Perform action
    - Return result
  - Reset socket after closure / PC crash
- Robot position server
  - Read socket
  - Return requested information
- Digital I/O server
  - Starts and kills Input notification process
  - Read socket
  - Change output or report input state
Results

- 2 different parts, 3 different scenarios
  - Lenses
  - Squares
  - Combined lenses and squares
- Over 60 parts per minute throughput

Results

- Two auto-adjuster algorithms
- Conveyor speed dependent on parts in queue
  - Slow conveyor + full queue = low throughput
- Modification of initial algorithm
  - High mark = slow conveyor
  - Low mark = speed up conveyor
  - Between = pull towards median
- Very dependent on parameters
Future Work

- Functionality
  - Error handling
  - Communications
  - Abstract server base classes
  - Data logger
  - Multi-part queue
  - Vision server MIL encapsulation
  - Utility programs
    - Conveyor <-> robot calibration
    - Camera <-> robot calibration
    - Tool offset determination
  - GUI
  - Naming conventions / file names
Future Work

- Extending capabilities
  - New hardware components
    - Seiko robot
    - Cognex vision system
  - New feeder layout / design
    - Vibrational elements to move parts
  - Different system operation
    - Serial vs. parallel
    - Port to original CWRU feeder

- Auto-adjustment
  - Very little research to date, lots of opportunities
  - What parameter are important
  - How does different part families change important parameters
  - Modeling
  - Example: "Smart queue"
    - Does robot retrieval speed depend on part location
    - Can this be used to increase throughput
    - Always get the better located part first
Future Work

Acknowledgments

• Andy Podgurski
  • Advisor
• Committee
  • Mike Branicky
  • Gultekin Ozsoyoglu
• CAMP
  • Bill Stellhorn
• Roger Quinn
  • Post-doc stipend
• Nick Barendt
  • Initial vision system work