Elements of Control System Longevity

Stephen Lewis

ICALEPCS'07
19 October 2007
Knoxville, TN

"In theory, there is no difference between theory and practice. But in practice, there is."

—Yogi Berra

"Success comes from experience; but experience comes from failure."

—Mark Twain

LONG TENURE

- We build controls for '30-year' systems (job security?)
- You *will* upgrade many times…
  - …and the first one may precede commissioning (job pain!)
- Goal: upgrades are not disruptive

PITFALLS

- Language
  - Avoid fads…the mainstream may not be what you like
- Ditto for the fancy IDE

PITFALLS

- Operating System?
  - Few last as long as your system…
    - …and an old OS needs old {hardware, people}
  - How many are you supporting?
  - Use a 'glue' layer for essential services

PITFALLS

- Transport
  - This is your middle-ware 'backbone'
  - Don't distort your architecture: map *your* own concepts (name discovery, congestion control, graceful recovery, etc) to it
  - Let it be asynchronous…

PITFALLS

- Hardware and Network
  - The *rate* of change here has been astounding
  - Plan to mix it up—like 'crates' with free-standing 'smart devices'
    - Is each device its own server?

PITFALLS

- Your shopping list:
  - Hardware
  - OS
  - Language
  - IDE
  - Libraries
  - For all host/target combinations…forever

DECOUPLING

- "Decoupling, decoupling, decoupling."
- It's the *web of dependencies* that get you…
- How far does a change 'ripple'?
  - Solution: a few layers, and fewer protocols
  - No 'cheating' (reaching around)

LAYERS

- Avoid middle layers
  - 'Manager' and 'Supervisor' belong in your org chart, not your architecture
    - Use self-configuring 'gateways' (bridges) to solve simple fan-out issues
  - Re-publish any 'value-added' in same layer
  - A 'flat' system is easy for clients
    - The hierarchy should be in the naming

DECOUPLING

- Pick a 'narrow' protocol/API:
  - Easy to code to
  - Allows clients to be generic ('tools')
  - It rarely changes, thus…
  - Decouples server and client teams
    - They work in parallel
    - They don't talk much
  - May support multiple versions

DECOUPLING

- Use a *text file* (sure, XML) between major 'stages', such as RDB and processes
  - Can easily create one (for consumer); can inspect one (from producer)
  - Insulates you from temporary failures, version mismatches, etc
  - Compatible with your code repository

DECENTRALIZATION

- Gives scaling
  - No single-point of failure
  - Graceful degradation without cascading failures
  - No congestion points
  - Supports incremental build-up
- Allows parallel life-cycles for subsystems

ASYNCHRONOUS

- Have you had any deadlocks lately?
  - Very hard to avoid (or recover from) with 3 or more layers of synchronous elements: the cascade effect
  - There are only two kinds of timing values: those that have changed and those that will
  - Non-blocking protocol/API (message passing) avoids this
    - Use a call-back for the hand-shake (transaction)

REQUIREMENTS

- Most are implicit, not explicit
  - 50:1 worst case
- There are two kinds: those that have changed and those that will change
- Don't code directly to them:
  - Use (reusable) building blocks
    - Most are nearly universal for all controls

SUMMARY

- Decouple
- Decentralize
- Go 'flat'
- Use text files
- Go 'narrow'
- Be asynchronous

CONCLUSION

- "All problems in computer science can be solved by adding one more level of indirection."
- "But that just creates another problem."

—David Wheeler