Event-Based Systems:
Architect's Dream or Developer's Nightmare?

Gregor Hohpe
Software Engineer
Google, Inc.
www.eaipatterns.com
### Architect's View
- Loosely coupled
- Asynchronous
- Composable
- Scalable
- Real time

### Developer's View
- Good bye, call stack
- Timing uncertainty
- Evolving system
- Complex Run-time
- Real headache
Call Stack = Developer's Comfort Zone

- Command-and-control scheme
- Single thread of execution, easy to debug
- Predictable, we know who does what when
- What they taught us in CS 101

"Welcome to the real world…"

The Real World Is Full Of Events

- Hide the new programming model
- "Doodleware": programming in pictures
- Make building simple solutions easy
- Vendor-specific terminology

- Expose the new programming model
- Better support in languages and tools
- Make building real solutions efficient
- Improve collective understanding

"Remember: I'm offering you the truth, nothing more."
## Taking the Red Pill

- Separate API from architecture
- Capture knowledge in design patterns
- Configuration is programming
- Use pictures, but not for programming
- Shift attention to run-time
- Think beyond development
- Look for the Killer App

**"I know what you’re thinking: Why didn’t I take the blue pill?"**

---

### API ≠ Architecture

- New technologies and programming models often proliferated by big platform & framework vendors.
- Often more attention is paid to the programming interface as opposed to the programming model.
- Examples: Java JMS, Microsoft WCF ("Indigo"),
- Counter-example: Microsoft Workflow Foundation.
- Difficult to interest developers in architectural considerations.
- Trick: teach developers without them noticing.
Design Patterns – 10 Years After GoF

- “Mind sized” chunks of information (Ward Cunningham)
- Human-to-human communication
- Good solution to a common problem within a specific context
- Expresses intent (“why” vs. “how”)
- Observed from actual experience
- NOT:
  - A firm rule – always a time when not to use
  - Copy-paste solution

Why Revisit Patterns?

- New programming models bring new patterns.
- Patterns are expressed using the constructs of the underlying architectural style (e.g. OO, SOA).
- Patterns can help discover higher levels of abstraction.
- Patterns are fuzzy around the edges and work well in the absence of formalisms.
- Pattern language forms important vocabulary.
- Ultimately some of these patterns can be implemented in the platform. This is an iterative process.
Composability

"The ability to build new things from existing pieces."

Composition Is Programming

- Introduces a new layer into the system: the composition layer.
- Often euphemistically called "Configuration only. No coding needed."
- Deserves to be a 1st class citizen:
  - Language
  - Tools
  - Testing
- Programming with XML files tedious and error prone

“Great composers are few and far in between.”
For example
- Graphical process editors
- Graphical transformation editors

We love pictures

Programming in pictures tedious
- Scalability issues
- Diff, Merge mostly unsupported

Often a thin veneer over a complex (or unfamiliar) programming paradigm

Use Pictures, But Not For Programming

Loosely coupled systems enable independent variability

System can evolve locally without breaking

Evolution can lead to surprises

Therefore, extract accurate state of the system:
- Design-time analysis
- Run-time observation
- “Reverse” MDA
Shift Attention to Run-time

- Loose coupling and composability mean the compiler and static type checking play a smaller part
- Need tools to perform similar functions at run-time
- Validate system configuration
- Validate dynamic system behavior

Model Validation

[Diagram showing a model validation process with nodes labeled Customer, Logger, and Channel, and messages flowing between them.]

© 2007 Google, Inc. All rights reserved.
Think Beyond Development

- More time spent debugging, testing, maintaining, understanding existing solutions
- New testing approaches
  - Testing individual components easier but not sufficient.
  - More aspects into play: network, marshalling etc.
  - Use non-distributed, synchronous versions for initial testing.
- Debugging Tools
  - System monitoring, visualization, model validation
- Documentation needed to convey architectural intent.

“Your compiler does not tell you if you violate architectural principles.”

Look for the Killer App

- Some programming environments are naturally event-driven
- User Interface frameworks, e.g. Visual Basic, Swing
  - Using events feels natural: "user clicks a button"
- Data replication / change tracking for large data sets
  - Financial industry
- The Internet?
  - E.g., feed readers, alerts, ....
Conclusion

• Balance architectural benefits with development effectiveness.

• Do not hide architectural style. Provide tools to work with it effectively.

• Harvest patterns to share knowledge about building "good" event-based systems.

www.EnterpriseIntegrationPatterns.com

• Pattern catalog
• Articles
• Blog ("Ramblings")

© 2007 Google, Inc. All rights reserved.