

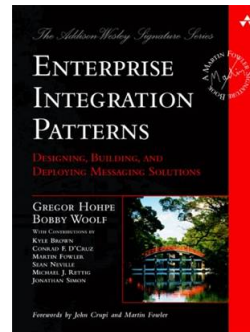
Developing in a Service-Oriented World

Gregor Hohpe
www.eaipatterns.com

SD Forum Web Service SIG
June 28, 2005

About Me

- Integration Architect with ThoughtWorks
 - 650 people consulting firm
 - BizTalk, TIBCO, MQ, Web Services
- Microsoft MVP Solution Architect
- Books
 - Enterprise Integration Patterns (Addison-Wesley)
 - Enterprise Solution Patterns using Microsoft .NET (Microsoft Pattern & Practices)
 - Integration Patterns (Microsoft Pattern & Practices)
 - The Best Software Writing I: by Joel Spolsky (APress)



www.eaipatterns.com

My Advice for Aspiring SOA Developers

ThoughtWorks®
The art of heavy lifting.™

- Forget about SOAP
- Become good at PowerPoint
- PROLOG rocks
- Pay close attention to Starbucks
- Shred “Design Patterns” (or eBay it)
- Look for ADM instead of MDA

www.eaipatterns.com

Copyright ThoughtWorks 2005. All rights reserved.

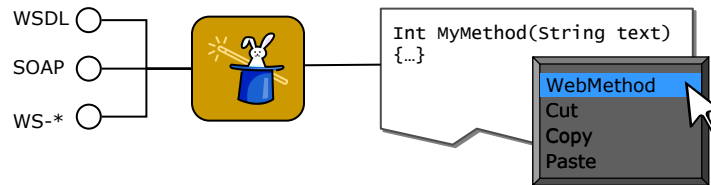
3

ThoughtWorks®
The art of heavy lifting.™

PART I

How Did We Get Here?

Could It Be So Easy?



- Buzzword compliant, but not a service-oriented architecture
- Synchronous call stack mentality
- No interface-implementation separation

Service-Oriented Architecture

- Service
 - Well-defined, Self-contained
 - Independent of consumer context (mostly)
 - Universally accessible without individual deployment
- Service-Oriented Architecture
 - An architectural style
 - A simple, document-oriented interaction model
 - Loose(r) coupling
 - Interface contracts
 - Registry
 - Compose existing services into new services
 - Functional assets reside in services, explicit orchestration across services

SOA is an Architectural Style

- An architectural style is based on patterns and intent, not technology selection.
- SOAP is just a message format and is a rather insignificant part of the SOA puzzle.
- Think about:
 - Asynchrony
 - Document-orientation
 - Granularity
 - Composition
 - Conversations
 - Monitoring and Management
 - Patterns

Architectural Styles

Architectural Style:

- A Vocabulary
- Design rules
- Semantic interpretation
- Analyses

Monroe, Kompanek, Melton, Garlan:
"Stylized Architecture, Design Patterns, and
Objects", 1996

"An **architectural style** is a coordinated set of architectural constraints that restricts the roles/features of architectural elements and the allowed relationships among those elements within any architecture that conforms to that style."

Roy T. Fielding
2000

Common Examples:

Pipes-and-Filters
Objects
Layering
Process control
Interpreters

"A system does not have to follow a single style"

Distributed Component Architectures

- Main driver: transparency to developer
- The Distributed Object approach ignores:
 - Latency (network, marshalling, applications)
 - Disconnected or intermittently connected networks
 - Lack of shared memory access (pointers, references)
 - Partial failure and concurrency
 - Independent variability between systems (coupling)

“95% transparent is not good enough. In fact, it is worse because it deceives developers.”

-- Werner Vogels

SOI – Defining Characteristics

- Main Driver: Simplicity of interaction
- No notion of inheritance, polymorphism, call stack, references etc.
- No lifecycle control. Service provider manages instances / allocations internally to suit its needs.
- Pass fewer, more self-contained documents. A tree structure (e.g., XML) is well suited for this.
- More amenable to asynchronous interaction.

“Objects that interact in a distributed system need to be dealt with in ways that are intrinsically different from objects that interact in a single address space.”

-- Waldo et al, 1994

SOI - Considerations

- Progress through Regress?
- Is the simplified interaction model sufficient? (WS-*)
- Are the contracts expressive enough?
- Are we getting it right this time around?
- When is SOA not appropriate?

PART II

What To Do Now?

The Human Side of Service-Orientation

- An architectural style is based on patterns and intent, not technology selection.
- Conversation models, asynchrony, document-orientation, granularity, decoupling, management, etc. are much more important.
- Loose coupling means fewer assumptions between systems.
- This means shared architectural vision and intent are critical.
- Your compiler can't tell you if you violated SOA principles.
- In the near term, this means documentation. Yes, PowerPoint!

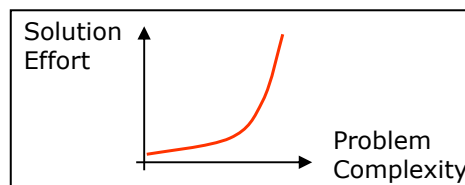
Will Tools Make it Easy?

1st Generation: Retrofit

- New syntax or technology, but old model
- “Lipstick on a pig”, sometimes worse than the old model
- Sometimes no other choice, but should be used with caution
- Example:
 - 3270 to GUI converters
 - Some Web Services Facades

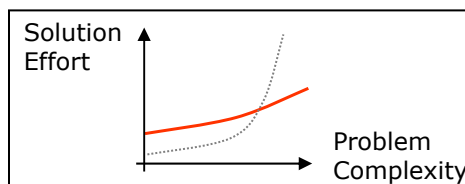
2nd Generation: Simple Tools for Simple Problems

- Use correct model but make many assumptions
- Good for simple problems but usually have “brick wall” syndrome
- Can mislead developers
- Example:
 - 2-tier application frameworks
 - Right-click, make Web service (Generation 1.5)



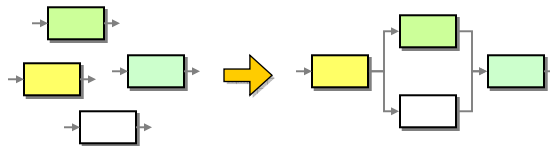
3rd Generation: Efficient Tools for Complex Problems

- Use correct model
- Understand how experienced developers really work: refactoring, automated testing
- Need to solve a range of complex problems before you can build these tools!
- Example:
 - IntelliJ Idea, Eclipse, xUnit, CruiseControl



Composability: A New Programming Model

- The ability to compose individual elements into a larger solution
- Introduces a new layer into the system: the composition layer
- This layer is often based on an alternative architectural style, e.g. Pipes-and-Filters vs. Object-Oriented
- This layer needs to be defined and tested



New Programming Paradigms

- Object-Document Mapping
 - Analogous to O-R mapping: subtle, but important
- Declarative Programming
 - XSLT
 - Rules engines
- Event-based Programming
 - No call stack
 - Continuations
 - Explicit state management
- “Doodlware”
 - Graphical process editors
 - Graphical transformation editors
 - Often a thin veneer over a new programming paradigm

Exception Handling

- “Starbucks does not use two-phase commit”
 - Compensation
 - Retry
 - Write-off
- Optimize throughput over latency
 - “Wider bridges, not faster cars”
- Optimize for happy day scenario

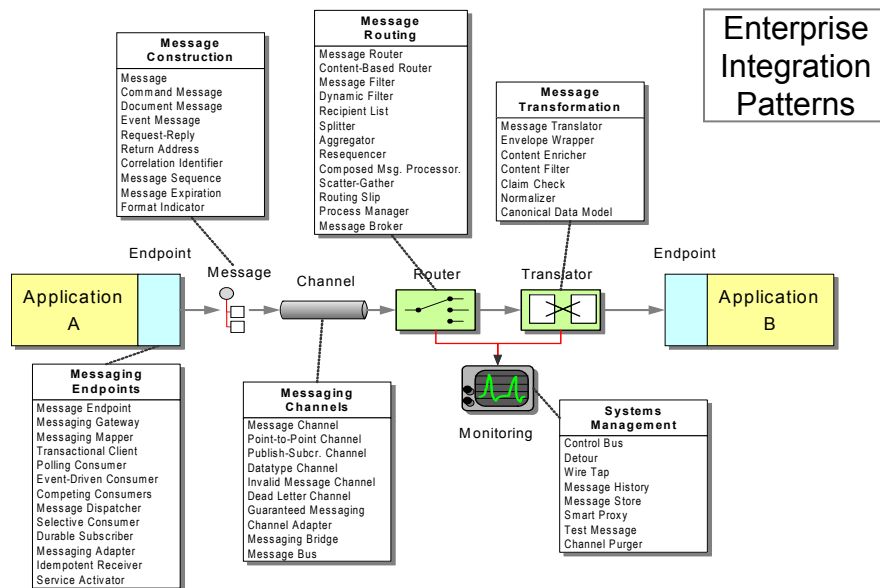
Domain-Specific Languages

- Finding generic languages to support these programming models is hard
- It also makes the languages complex and the learning curve steep (see XSLT)
- “Language Workbenches” may help us create our own domain-specific languages which are smaller in scope
 - Intentional Programming
 - JetBrains Meta Programming System (MPS)
 - Visual Studio 2005
 - See article on <http://www.martinfowler.com/>

Design Patterns Revisited

- Document observed experience – it takes time!
- Express intent (the “why” vs. the “how”)
- “Mind-sized chunks of information”
- Vocabulary / “Pattern Language”
- As pragmatic as possible, as formal as necessary
- Can be visually expressive
- Patterns depend on the underlying architectural style to provide the base constructs and vocabulary
 - OO: class, object, instance, inheritance, polymorphism
 - SOA: service, message, orchestration, conversation

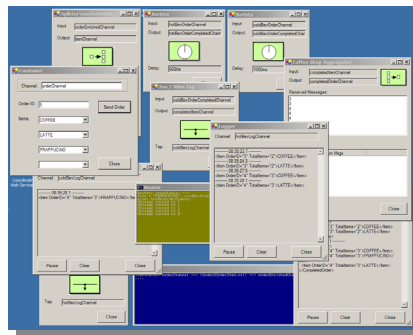
“Language” of 65 Patterns



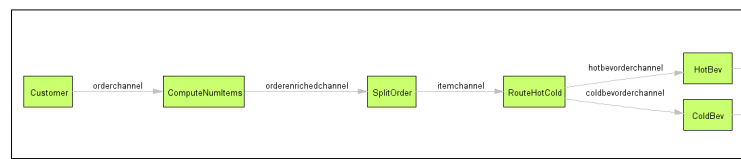
Ongoing Patterns Work

- Conversation / Interaction Patterns
 - Message exchange between loosely coupled systems
- Orchestration Patterns
 - Process engine
- Error Handling Patterns
 - Starbucks etc,
- Complex Transformations
 - Information Integration (EII)
- Rules Engines
 - Prolog (?)

Get the Big Picture



- Loosely coupled systems = evolution without global impact
- This can also lead to chaos
- Harvest a model of the system from the actual system
- Visualize this model
- “Reverse MDA”



In Summary

- SOA brings new and unfamiliar:
 - Architectural Styles
 - Programming Models
 - Best Practices
 - Patterns
 - Testing Approaches
 - Management Approaches
- The collective learning cycle will take some time (remember OO)
- The vendors and specs are sometimes ahead (or amiss) of the real issues
- We need to progress by sharing our experiences

Enterprise Integration Patterns

- Language of 65 patterns
- Vocabulary and notation
- Focuses on asynchronous messaging (pipes-and-filters)
- Many more patterns to harvest:
 - Conversations
 - Orchestrations
 - Error Handling
 - Complex Transformations
 - Rules Engines

