Enterprise Integration Patterns

Gregor Hohpe
「グレガー ホペ」
www.eaipatterns.com

パターンワーキンググループ

April 20, 2004

"As applications become more connected, asynchronous messaging will be the next big architectural trend.

Similar to the shift procedural to Object-Oriented Programming, we need patterns and best practices to help us master the new paradigm."

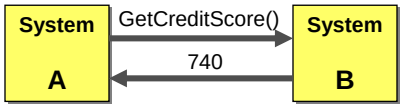
Coupling

- Coupling = Measure of dependency between applications
  - Technology Dependency
  - Temporal Dependency
  - Location Dependency
  - Data Format Dependency
- Coupling is a Range, not a Binary Property
- Coupling is not inherently good or bad

> "How do you make two systems loosely coupled? Don't connect them."
>
> -- David Orchard, BEA

Remote Procedure Calls (RPC)

- "Natural" extension of local programming model to remote applications



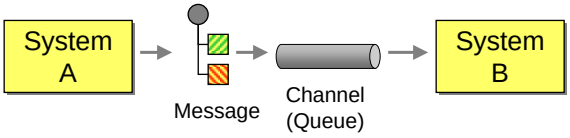
- Allows one application to invoke a method inside another application
- Details of remote communication are largely hidden from programmer
- Java RMI, .Net Remoting, DCE RPC

RPC's Can Be a Dangerous Illusion

- Tight coupling just as local method call
- The RPC approach ignores:
  - Latency
  - Lack of shared memory access
  - Partial failure and concurrency

> "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

Message-Oriented Communication

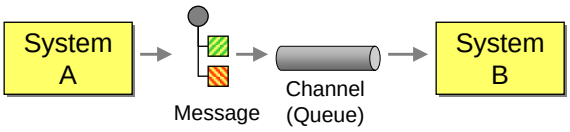


- Systems communicate via *Channels*
- Channels have logical (location-independent) addresses
- Data is encapsulated in messages in a technology-neutral format, e.g. XML
- The sending application places a message into the Channel and goes on to other work ("fire-and-forget")
- The Channel queues the data until the receiving application is ready to consume it (FIFO)

Message-Oriented Communication

ThoughtWorks

Message-oriented communication provides loose coupling and reliability



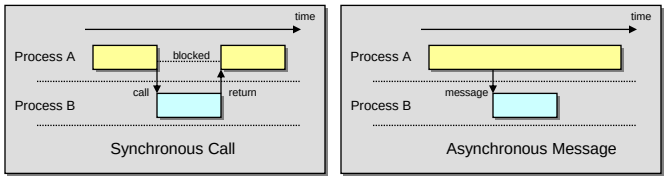
- Channels are separate from applications
- Channels are asynchronous & reliable (queues)
- Use a common data representation, e.g. XML
- Messages are self-contained

- ➢ Removes location dependencies
- ➢ Removes temporal dependencies
- ➢ Removes technology dependencies
- ➢ Removes data format dependencies

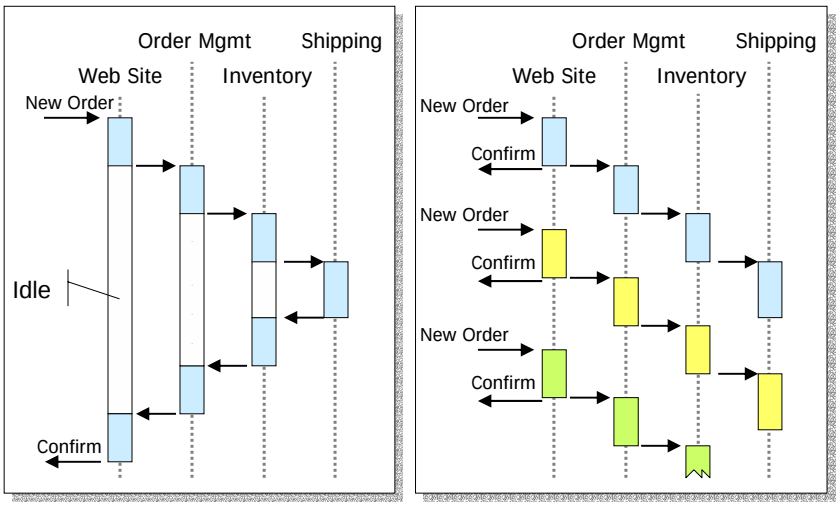
Asynchrony

ThoughtWorks

- Most applications are synchronous
  - The caller waits until a method completes
  - The call stack manages local state and keeps track of the return address
- Messaging is inherently asynchronous
  - The calling program continues after the message is sent
  - Need to manage state manually
  - Results may come back at any time via an asynchronous message

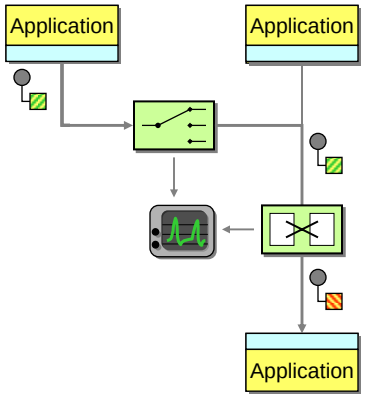


Thinking Asynchronously



Enterprise Integration Patterns

1. Transport messages
2. Design messages
3. Route the message to the proper destination
4. Transform the message to the required format
5. Produce and consume messages
6. Manage and Test the System



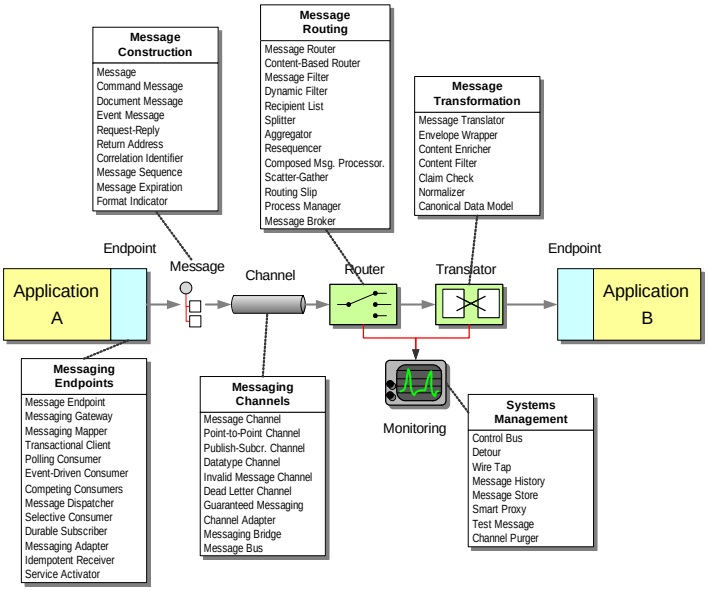
Enterprise Integration Patterns

ThoughtWorks
The art of heavy lifting

1. Transport messages → *Channel Patterns*
2. Design messages → *Message Patterns*
3. Route the message to the proper destination → *Routing Patterns*
4. Transform the message to the required format → *Transformation Patterns*
5. Produce and consume messages → *Endpoint Patterns*
6. Manage and Test the System → *Management Patterns*

"Language" of 65 Patterns

ThoughtWorks
The art of heavy lifting



Icon Language ("Gregorgrams")


Content-Based Router


Message Filter


Recipient List


Splitter


Aggregator

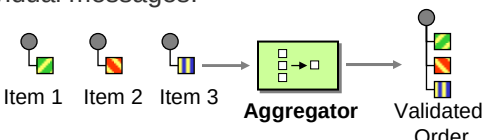

Resequencer


Routing Slip (Itinerary)


Process Manager

Example: *Aggregator* (abbrev.)

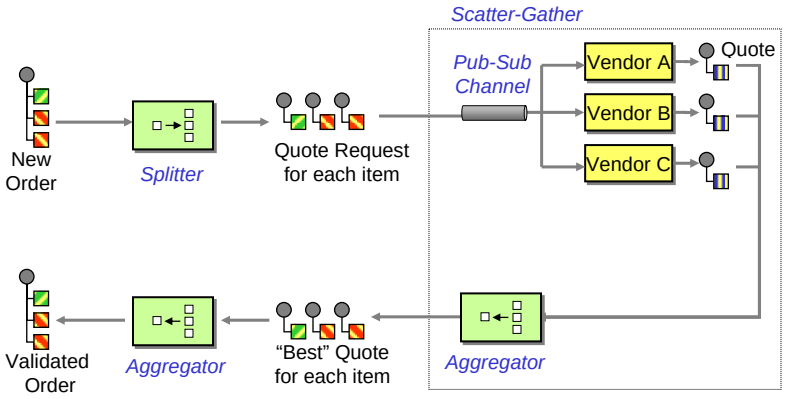
- Problem
  - How do you combine the results of individual, but related messages back into a single message?
- Forces
  - Messages may be out of sequence
  - Messages may be delayed, when to stop?
  - Avoid burdening receiver with these issues
- Solution
  - Use a stateful filter, an Aggregator, to collect and store individual messages until it receives a complete set of related messages. Then, the Aggregator publishes a single message distilled from the individual messages.
- Sketch



Aggregator Design Decisions

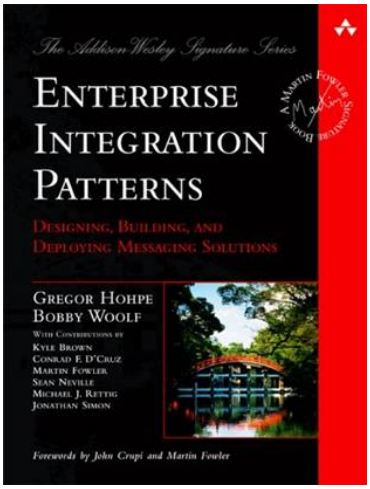
- Correlation
  - Which incoming messages belong together?
- Completeness Condition
  - Wait for all
  - Time out (absolute, incremental)
  - First best
  - Time box with override (Buy Now")
  - External event
- Aggregation Algorithm
  - Single best answer
  - Condense data (e.g., average)
  - Concatenate data for later analysis

Composing Patterns



Enterprise Integration Patterns

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



For more information…

ghohpe@thoughtworks.com
www.eaipatterns.com
www.thoughtworks.com