Asynchronous Messaging Style

<table>
<thead>
<tr>
<th>Systems send messages</th>
<th>Simplified Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channels have logical, location-independent addresses</td>
<td>Location Decoupling</td>
</tr>
<tr>
<td>Placing a message into the Channel is quick (“fire-and-forget”). The Channel queues messages until the receiving application is ready to consume.</td>
<td>Temporal Decoupling</td>
</tr>
</tbody>
</table>

An "honest" architectural style that does not try to deny the limitations of the underlying medium.
Asynchronous Architectural Style

Asynchronous Messaging Benefits

Temporal decoupling
- Sender does not have to wait for receiver to process message

Limit Failure Propagation
- Sender not affected by intermittent failure
- Can be reliable over unreliable transports

Throttling
- Receiver can consume messages at its own pace
- Processing units can be tuned independently

Insertion of intermediaries (Pipes-and-Filters)
- Composability: Transformation, routing etc.

Throughput over latency
- “Wider bridges not faster cars”
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

Message 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

Channel Patterns
Message Patterns
Routing Patterns
Transformation Patterns
Endpoint Patterns
Management Patterns
Patterns in Google Cloud Pub-Sub
Serverless Content-based Router

const Pubsub = require('@google-cloud/pubsub');
const pubsub = Pubsub({projectId: 'eaipubsub'});
exports.contentBasedRouter = function contentBasedRouter(event) {
  const pubsubMessage = event.data;
  const payload = Buffer.from(pubsubMessage.data, 'base64').toString();

  order = JSON.parse(payload)
  outChannel = getOutChannel(order.type);
  return pubsub.topic(outChannel).get({autoCreate: true}).then(function(data) {
    var topic = data[0];
    return topic.publish(order);
  })
};

function getOutChannel(type) {
  switch(type) {
    case 'widget': return 'widgets';
    case 'gadget': return 'gadgets';
    default: return 'unknown';
  }
}

```
gcloud beta functions deploy contentBasedRouter --stage-bucket eaipubsub-functions
--trigger-topic orders
```

http://www.enterpriseintegrationpatterns.com/ramblings/google_cloud_functions.html

© 2017 Gregor Hohpe

www.EnterpriseIntegrationPatterns.com
Messaging and Conversations

Message Patterns

- Focus on message
- Follows the message
- One-way
- Deals with transport
- “Stateless”

Conversation Patterns

- Focus on participants
- Follows time
- Two-/multi-way
- Deals with resources
- “Stateful”

Conversation State and Endpoint State

- Legal message sequences defined through the protocol
- Each conversation instance has a state
- Each participant has a (potentially different) process definition
- Local process state maps to conversation state
Subscriptions

Publish-Subscribe Channel
How can the sender broadcast an event to all interested receivers?

Subscribe-Notify
How can one participant receive information from another participant if that information cannot easily be packaged into a single message?

Conversation Design & Challenges

**Conversation Design:**
- Participant Roles
- Message Type
- Protocol

**Challenges:**

**Protocol design:**
- Completeness
- Deadlocks
- Error States
- Map conversation state to participant state

**Visualization:**
- Difficult to represent state space
- Hard to capture the essence in an evocative “sketch”
- UML 2 difficult to decipher
- BPMN2 suitable, but visually verbose
BED ACOVE

Design problem
Bedrooms make no sense.

Forces
First, the bed in a bedroom creates awkward spaces around it: dressing, working, watching television, sitting, are all rather foreign to the side spaces left over around a bed. (...)
Second, the bed itself seems more comfortable in a space that is adjusted to it.

Solution
Don't put single beds in empty rooms called bedrooms, but instead put individual bed alcoves off rooms with other nonsleeping functions, so the bed itself becomes a tiny private haven.

Related Patterns
Communal Sleeping, Marriage Bed
Ceiling Height Variety, Half-open Room, Thick Walls
Patterns Revisited

- A good solution to a problem within a specific context?
- Recurrence (relevant)
- Teaching ("why" and "how")
- Name (forms a language)

- “Chicken Mc Nuggets of Information” (Ward Cunningham)
- Human-to-human communication
- Harvested: observed from actual experience

Pattern Relationships

<table>
<thead>
<tr>
<th>Design Pattern Space</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Creational</td>
</tr>
<tr>
<td>Scope</td>
<td>Factory method</td>
</tr>
<tr>
<td>Class</td>
<td>Adapter (object)</td>
</tr>
<tr>
<td></td>
<td>Decorator</td>
</tr>
<tr>
<td>Object</td>
<td>Prototype</td>
</tr>
<tr>
<td></td>
<td>Singleton</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Pattern Relationships

Pattern Sequence

“The patterns are ordered, beginning with the very largest, for regions and towns, the working down through neighborhoods, clusters of buildings, buildings, rooms and alcoves, ending finally with details of construction.

This order, which is presented as a straight linear sequence, it essential to the way the language works.”
Towards Pattern Languages

- Pattern Complements
  - Competition
  - Cooperation
  - Combination

- Pattern Compounds

- Pattern Sequences

- Pattern Collections
  - Level, Domain, Intent

Pattern Language: Message Flow

- Message Construction
  - Message
  - Command Message
  - Document Message
  - Event Message
  - Request-Reply
  - Reply-Only
  - Notification
- Message Routing
  - Message Router
  - Content-Based Router
  - Message Filter
  - Dynamic Filter
  - Match Filter
  - Splitter
  - Aggregator
  - Slicer

- Message Transformation
  - Message Translator
  - Envelope Wrapper
  - Content Extractor
  - Content Filter
  - Checksum
  - Normalize

- Message Format
  - Canonical Data Model

- Messaging Endpoints
  - Messaging Gateway
  - Messaging Tripwire
  - Messaging Requestor
  - Messaging Consumer
  - Messaging Publisher
  - Messaging Submitter
  - Messaging Adapter
  - Messaging Bus

- Messaging Channels
  - Message Channel
  - Point-to-Point Channel
  - Publish-Subscribe Channel
  - Reliable Channel
  - Invited Message Channel
  - Direct Message Channel
  - Guaranteed Messaging Channel
  - Messaging Bridge
  - Messaging Bus

- Monitoring

- Systems Management
  - Server Management
  - Load Balancer
  - Message Hub
  - Message Store
  - Event Propagation
  - Test Message Channel
Pattern Story: Root Patterns

Integration Styles

Messaging

Messaging Systems
Message Channel
Message
Pipes and Filters
Message Router
Message Translator
Message Endpoint

Pattern Competition: Routing Patterns

Simple
- Process one msg at a time (stateless)
  - Single msg out
  - Zero or One
  - Parallel
  - Sequential

Composed
- Process multiple msgs at a time (stateful)
  - Multi. msgs out
  - Less msgs out
  - Same number of msgs out
  - Split Message
  - Parallel
  - Sequential
  - Predetermined, Linear
  - Any Path

Content-Based Router
Message Filter
Recipient List
Splitter
Aggregator
Resequencer
Compos. Msg. Processor
Scatter-Gather
Routing Slip
Process Manager
Conversation Patterns

Conversation Pattern Language

Setting Up
- Discovery
- Initiation

Participants
- Basic Conversations
- Intermediaries

Application-level
- Resource Mgmt.
- Ensuring Consistency
## Conversation Pattern Language

### Discovery
- Dynamic Discovery
- Advertise Availability
- Consult Directory
- Referral
- Leader Election

### Basic Conversations
- Fire-and-Forget
- Asynchronous Req-Resp
- Req-Resp with Retry
- Polling
- Subscribe-Notify
- Quick Acknowledgment

### Resource Management
- Incremental State
- Lease
- Renewal reminder

### Initiation
- Three-way Handshake
- Acquire Token First
- Rotate Tokens
- Verify Identity
- User Grants Access

### Intermediaries
- Proxy
- Relay
- Load Balancer
- Scatter Gather

### Ensuring Consistency
- Ignore Error
- Compensating Action
- Tentative Operation
- Coordinated Agreement
Discovery Considerations

• Division of responsibility between provider and consumer

• Arrival/departure rate
  • Arriving/departing providers
  • Arriving/departing consumers

• Centralized vs. Distributed
  • Is there an element dedicated to discovery?

• Addressability
  • What’s the result of discovery?

• Trust
  • Is the consumer happy with just any provider?

• Matching
  • Unique Identifiers
  • Interface Definition / Type
  • Attributes
  • Keyword match (search)

How can a conversation initiator find a partner when it has no knowledge whatsoever about available partners?

Point-to-point communication requires knowledge of the conversation partner (or channel).
Central services for discovery are bound to get out of sync with reality.
Centralized administration may result in a single point of failure.
Even in the presence of a central lookup service, a new participant has to first establish a connection to the lookup service.
**Dynamic Discovery**

1. Broadcast *Lookup* request
2. Interested providers send *Available* responses
3. Requestor initiates interaction with chosen provider

Examples: DHCP

How can a participant let others know that it is available?

*Dynamic Discovery* can flood the network with requests. The number of available providers is often small compared to the number of conversations.
Advertise Availability

Available participants broadcast their existence and capabilities

Examples: Wi-Fi

How can a conversation initiator find a partner across a large network without flooding the network with requests?

Many networks do not route broadcast packets beyond the local network.
Centralized administration is often needed or desired in setting up a new service.
**Consult Directory**

Directory answers request for conversation partner.

Requires *addressability*, i.e. to embed addresses in messages

Directory may store additional metadata about the service

May use *Dynamic Discovery* to find the Directory

Example: UDDI Directory, DNS

---

**YOW!**

The choice of conversation partner may depend on the context of a conversation or may change over time.

How can an initiator discover the right conversation partner?

A participant may be required to interact with the same partner that another participant is already interacting with.

Directories are generally context free, i.e. they do not keep track of existing conversations and when assigning an initiator to a partner.

Some participants may not want to be "discovered". However, "friends of friends" are allowed to interact with them.
Referral

One participant passes a reference to another participant.

Consult Directory is a specialized case of Referral

Example: HTTP 302

Pattern Language: Discovery

<table>
<thead>
<tr>
<th></th>
<th>No Intermediary</th>
<th>Intermediary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consumer-driven</td>
<td>Dynamic Discovery</td>
<td>Referral</td>
</tr>
<tr>
<td>Provider driven</td>
<td>Advertise Availability</td>
<td>Consult Directory</td>
</tr>
</tbody>
</table>

© 2017 Gregor Hohpe
Multi-Party Conversations: Intermediaries

Peer-to-peer

Intermediaries

Connectors

Coordinators
**Proxy**

How can a participant communicate with a partner that is not visible or not reachable?

Initiator can hide identity using a *Proxy*

Proxy can monitor conversations

Proxy may need to be stateful for two-way conversations

Proxy can become a bottleneck

**Relay**

How can participants engage in a two-way communication when each participant is limited to outbound requests?

High overhead when using *Polling*

All other conversations can be layer on top of *Relay*

Needs to be stateful

Example: Amazon SQS
Scatter-Gather (Aggregator)

How can a participant solicit responses from a number of participants without connecting to all of them?

Widespread business model, e.g. “Aggregators”
Composing Conversations

Sequencing
One conversation follows another.

Expansion
A single message in one conversation consists of another conversation.

Overlay
Conversations occur concurrently.

Conclusion
Conclusions

- Enterprise Integration is more than messaging
- A Pattern is more than a solution to a recurring problem
- A Pattern Language is more than a set of patterns