Enterprise Integration Patterns

Lab Exercises

Objectives

- Demonstrate the role of messaging in decoupling applications so that they can be more scalable.
- Illustrate some of the challenges that need to be addressed when adopting EDA.
- The role of messaging in addressing non-functional requirements.
- Uses pattern language described at http://www.eaipatterns.com/
Simple Messaging Toolkit

- Easy composition of solutions from a batch file or the command line
- Messaging Domain Specific Language
- See Tutorial Reference Chart

```
call Customer orderChannel
call Enricher orderChannel orderEnrichedChannel
call Splitter orderEnrichedChannel itemChannel "/Order/Item"
call Router itemChannel coldBevChannel "Item = 'FRAPPUCINO'" hotBevChannel
call Logger coldBevChannel
call Logger hotBevChannel
```

Available Pattern Components

- Message Translator
- Content Enricher (special case of Translator)
- Sequence Tagger
- Router
- Splitter
- Aggregator
- Resequencer
- Wire Tap (Tee)
- Delay
Convenience & Test Components

Customer

Sends order messages to specified channel

Manual Step

Allows inspection of messages and out-of-sequence completion

Logger

Display messages and time stamps

Coffee Shop Scenario

• Customer places order for drinks
• Barista prepares drinks
• Run the following commands from the command line:
  • Customer orderChannel
  • Barista orderChannel orderCompletedChannel
  • Logger orderCompletedChannel
• Place some orders
• Close the components by hitting ESC or ENTER.
• Channel names are arbitrary as long as alphanumeric
• Make sure to close all components before trying a new exercise
• You can run components directly or from a batch script
Exercise One

• Part 1
  • Connect a Customer, a Barista, and a Logger.
  • Order one drink. Track the completion time.
  • Place 10 orders. Time it again.
• Part 2
  • Add a second Baristas in parallel
  • Order one drink. Track the completion time.
  • Place 10 orders. Time it again.
  • Feel free to start more Baristas if time permits
• Observations?

<table>
<thead>
<tr>
<th># Baristas</th>
<th>1 Order [secs]</th>
<th>10 Orders [secs]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exercise One - Observations

• Single Barista
  • One coffee takes about 1 second
  • Ten coffees take about 10 seconds
  • Throughput 1 coffee per second

• Two Baristas
  • One coffee still takes about 1 second
  • Ten coffees take about 5 seconds
  • Throughput 2 coffees per second
Exercise One - Discussion

- Messaging architectures scale out through Competing Consumers
- Scalability: Adding more baristas did not require any changes to the architecture or existing components
- Distinguish Throughput from Latency
- They are different, though both elements of perceived performance
- Messaging architectures can provide very high throughput

- Latency can be longer than non-distributed solutions
  - Networking overhead
  - Serialization / deserialization

Exercise Two

- Let's assume we can tune the second Barista for higher throughput
- Simulate by using SecondBarista command (400 ms)

![Diagram showing a sequence of orders through the Baristas and a Logger.]

- Send a rapid series of orders through the Baristas and observe the sequence of messages
- How can you tell the proper sequence?
  - You can start some components later than others
  - You can use a manual step (“Complete All” preserves order)
  - You can use two customers
Exercise Two - Example

```java
call Customer orderChannelTemp
call ManualStep orderChannelTemp OrderChannel "/Order/Item"
call Barista orderChannel orderCompletedChannel
call SecondBarista orderChannel orderCompletedChannel
call Logger orderCompletedChannel
```

Exercise Two - Observations

- Parallel processing causes messages to get out of order
- Correct the problem (using available patterns) so orders arrive at final logger in order
- Verify correct behaviour by inserting a manual step to reshuffle the orders
Exercise Two - Example Solution

Exercise Two - Learnings

- Parallel processing causes messages to get out of order
- Use a Resequencer to bring messages back into order
- Resequencing increases latency because it holds messages
- A Resequencer is a stateful component and needs to persist messages to be robust
- One missing message can stall everything
- Carefully consider scope of resquencing -- sometimes correlation suffices.
Exercise Three

- Processing a whole order at one time limits our scaling options
- Creating a specialized Barista each for iced beverages and for hot beverages allows us to fine-tune baristas
- Create a new solution using the following commands
  - HotBevBarista (400 ms)
  - ColdBevBarista (800 ms)
- These Baristas can only process a single <item>, not a complete <order>
- Still deliver complete orders in one piece to the customer

Exercise Three – Example Solution

```
call Customer orderChannel
call SequenceTagger orderChannel orderTaggedChannel "/Order/@OrderID"
call Enricher orderTaggedChannel orderEnrichedChannel
call Tee orderEnrichedChannel orderEnrichedChannel2 logEnrichedChannel
call Logger logEnrichedChannel
call Splitter orderEnrichedChannel2 orderItemChannel "/Order/Item"
call Tee orderItemChannel orderItemChannel2 logItemChannel
call Logger logItemChannel
call Router orderItemChannel2 orderItemColdChannel "Item = 'FRAPPUCINO'" orderItemHotChannel
call ColdBevBarista orderItemColdChannel orderItemCompletedChannel
call HotBevBarista orderItemHotChannel orderItemCompletedChannel
call Aggregator orderItemCompletedChannel orderCompletedChannel
call Logger orderCompletedChannel
```
Exercise Three - Tuning

• Assume each customer orders one Coffee and one Frappucino
• Assume the Hot Bev Barista is limited to one instance (for now)
• How many Cold Bev Baristas should you run for optimum performance?
• Discuss the optimum number first, then run tests with 10 orders in rapid sequence

<table>
<thead>
<tr>
<th># Cold Bev Bar.</th>
<th>First Test [sec]</th>
<th>Second Test [sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exercise Three - Discussion

• Splitting allows different message types to be processed individually.
• Separating tasks into smaller pieces can improve throughput for the application and support greater scalability.
• Messages will get out of order and have to be re-correlated and re-aggregated.
• Global sequencing constraints can hurt performance.
• Loosely coupled systems can be hard to debug.
• Flexibility and composability make it hard to diagnose problems.
• You don’t need a huge solution to realize the complexities.
For Extra Credit

- Copy your solution scripts to folder ExercisesViz
- Start Visualizer.bat
- Open Graph.htm
- Run your script
- Watch Graph.htm

Summary

- Patterns are a good way to describe messaging solutions
- Messaging solutions can be highly dynamic
- Messaging architectures can address scalability issues
- Messaging introduces new problems

- Pattern references: http://www.eaipatterns.com/