Spring Integration: Using Channel Adapters to Integrate with External Systems

INTEGRATING WITH CUSTOM EXTERNAL SYSTEMS

Steven Haines
PRINCIPAL SOFTWARE ARCHITECT

@geekcap www.geekcap.com

Overview

Custom Integration Strategy

Custom Inbound Channel Adapter

Custom Outbound Channel Adapter

Inbound and Outbound Channel Adapters

Inbound Channel Adapter

MessageSource

Outbound Channel Adapter

MessageHandler

```
@FunctionalInterface
public interface MessageSource<T> extends IntegrationPattern {
    @Nullable
    Message<T> receive();

    @Override
    default IntegrationPatternType getIntegrationPatternType() {
        return IntegrationPatternType.inbound_channel_adapter;
    }
}
```

MessageSource

Implement our functionality using a Lambda expression

Implement this functional interface and implement a receive() method

Extend AbstractMessageSource and implement a doReceive() method

```
@FunctionalInterface
public interface MessageHandler {
   void handleMessage(Message<?> message) throws
MessagingException;
}
```

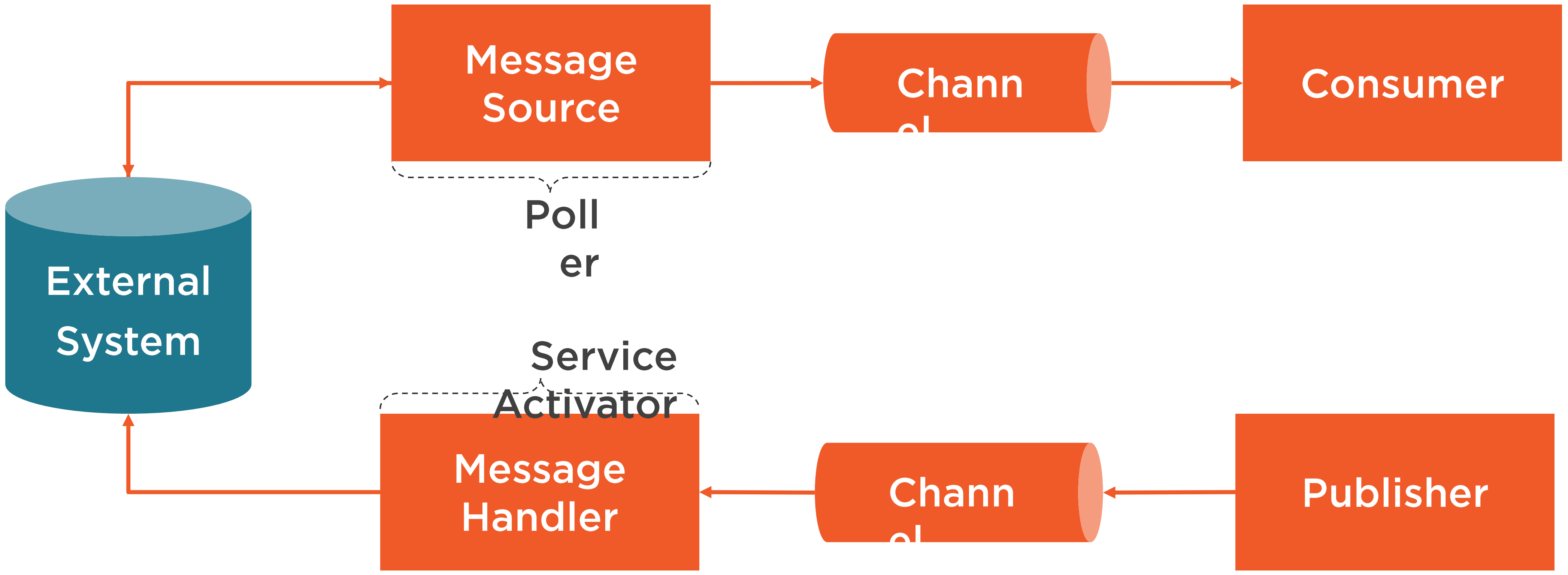
MessageHandler

Implement our functionality using a Lambda expression

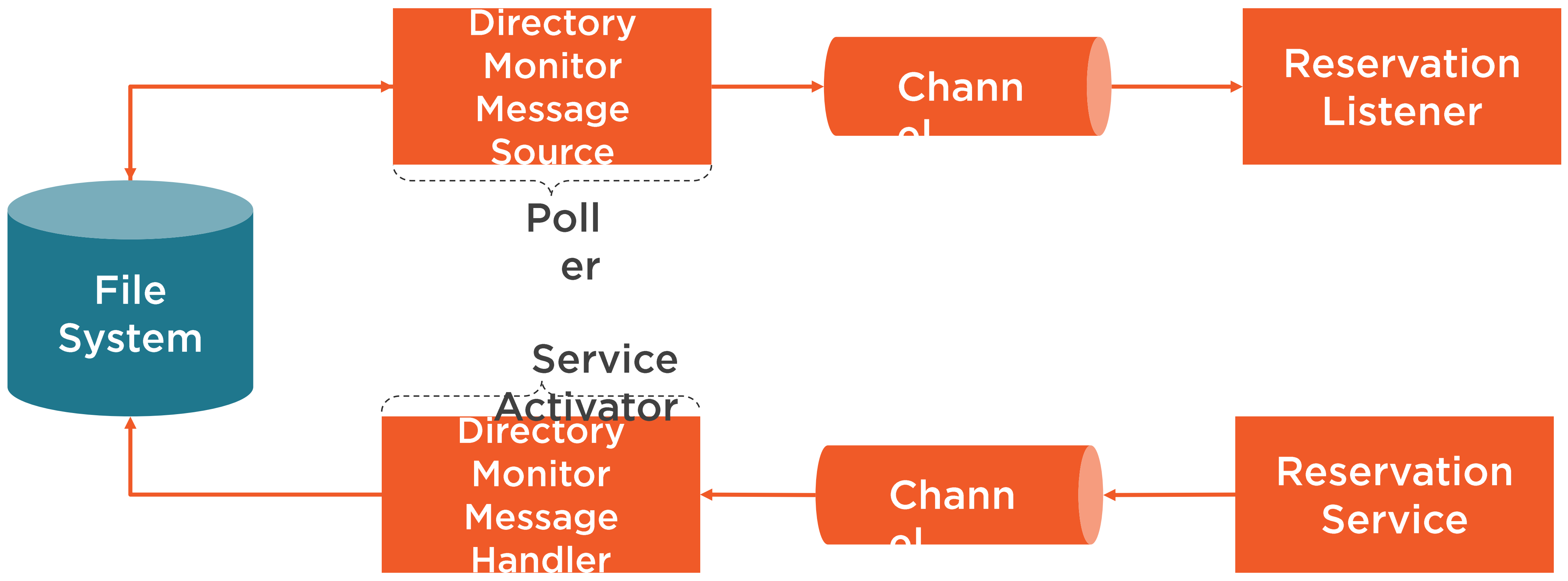
Implement this MessageHandler and implement a handleMessage() method

Extend AbstractMessageHandler and implement a handleMessageInternal() method

Custom Inbound and Outbound Adapters



Example: Directory Monitor



Custom Inbound and Outbound Channel Adapters

```
@FunctionalInterface
public interface MessageSource<T> extends IntegrationPattern {
    @Nullable
    Message<T> receive();

    @Override
    default IntegrationPatternType getIntegrationPatternType() {
        return IntegrationPatternType.inbound_channel_adapter;
    }
}
```

MessageSource

Implement our functionality using a Lambda expression

Implement this functional interface and implement a receive() method

Extend AbstractMessageSource and implement a doReceive() method

```
@Bean
@InboundChannelAdapter(value = "reservationListFromCustomChannel",
                                              poller = @Poller(fixedDelay = "5000"))
public MessageSource<List<Reservation>> customReservationSource() {
    return () -> {
       List<Reservation> reservations = new ArrayList<>();
       reservations.add(new Reservation(1, "Smith", "None"));
       reservations.add(new Reservation(2, "Jones", "None"));
       return MessageBuilder.withPayload(reservations).build();
    };
}
```

MessageSource Using a Lambda Expression

Define a function that receives no arguments and return a Message

```
@IntegrationManagedResource
public abstract class AbstractMessageSource<T> extends
AbstractExpressionEvaluator
        implements MessageSource<T>,

org.springframework.integration.support.management.MessageSourceMetrics,
                           NamedComponent,
                                    BeanNameAware {
    …
}
```

AbstractMessageSource

Provides an implementation of the MessageSource interface

Provides support for the Spring lifecycle and Bean management

```
public class DirectoryMonitorMessageSource
                         extends
AbstractMessageSource<Object> {
    @Override
    protected Object doReceive() {
        List<Object> results = new
ArrayList<>();
        File dir = new File(this.directory);
        for (File file : dir.listFiles()) {
            try {
                results.add(

objectMapper.readValue(file, entityClass));
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        return
MessageBuilder.withPayload(results).build();
    }
}
```

- Extend AbstractMessageSource
- Override the doReceive() method
- Return our results as a message payload

```
@FunctionalInterface
public interface MessageHandler {
   void handleMessage(Message<?> message) throws
MessagingException;
}
```

MessageHandler

Implement our functionality using a Lambda expression

Implement this MessageHandler and implement a handleMessage() method

Extend AbstractMessageHandler and implement a handleMessageInternal() method

```
@Bean
@ServiceActivator(inputChannel = “outboundReservationChannel")
public MessageHandler outboundReservationMessageHandler() {
    return message -> {
        Reservation reservation = (Reservation)message.getPayload();
        // Implement our business logic
    };
}
```

MessageHandler Using a Lambda Expression

Define a function that receives a message and performs its business logic

```
public class DirectoryMonitorMessageHandler

extends AbstractMessageHandler {
@Override
protected void
handleMessageInternal(Message<?> message) {
    UUID uuid = UUID.randomUUID();
    Path path = Paths.get(directory, uuid +
".json");
    try {
            objectMapper.writeValue(

path.toFile(), message.getPayload());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

- Extend AbstractMessageHandler
- Override the handleMessageInternal() method
- Implement business logic

Demo

Build our application

- DirectoryMonitorMessageSource
- DirectoryMonitorMessageHandler
- Configuration
- Reservation Listener and Reservation Service

Run the application

Validate the results

Conclusion

Inbound and Outbound Channel Adapters

Inbound Channel Adapter

MessageSource

Outbound Channel Adapter

MessageHandler

Inbound and Outbound Channel Adapters

Inbound Channel Adapter

AbstractMessageSource

Outbound Channel Adapter

AbstractMessageHandler

Summary

You should understand how to build custom inbound and outbound channel adapters

You should be prepared to integrate with any system for which Spring Integration does not have native support