Distributed Systems - Exercise

Exercise sheet 4

The fourth exercise deals with patterns for Enterprise Application Integration (see lecture slides for further details). The objective is to develop an integration using Apache Camel. Apache Camel is a Java-based integration framework supporting most patterns presented in the lecture and the corresponding literature.

Dive-Surf Inc. Integration Scenario

Implement the *Dive-Surf Inc.* scenario presented in the lecture[1] using at least the following patterns:

- Point-to-Point Channel
- Publish-Subscribe Channel
- Aggregator
- Content-Based Router
- Content Enricher
- Message Translator
- Channel Adapter
- Message Endpoint

The patterns mentioned above must be used. In addition, the solution must implement the "take and process order" integrations and comply with the following requirements:

- Dive-Surf Inc. offers at least two items: surfboards and diving suits.
- Implement five simple applications and integrate them. It must be possible to start each application in a separate Java program – i.e., each application provides its own main() method:
  - WebOrderSystem: the web order system generates a string for each new incoming order. The string needs to be processed further by the integration solution. The string consists of comma-separated entries and is formatted as follows: <Customer-ID, First Name, Last Name, Number of ordered diving suits, Number of ordered surfboards> - e.g., "1, Alice, Test, 2, 0".
  - CallCenterOrderSystem: the Call center order system generates a text file containing new orders every 2 minutes and stores it at a predefined destination in the local file system. Each line represents a single order. An order consists of comma-separated entries formatted as <Full Name, Number of ordered surfboards, Number of ordered diving suits, Customer-ID> - e.g., "Alice Test, 0, 1, 1". The full name always consists of the first and the last name separated by a space. It is not defined how many orders are contained in the file.

[1] Chapter 4.2, Slide 29ff

    - The BillingSystem takes transformed order messages (see below) and simply tests whether the customer is in good credit standing and is allowed to order the requested items. Therefore, the billing system modifies the valid property of the incoming messages and optionally modifies the validationResult property.
    - The InventorySystem takes transformed order messages (see below) and tests whether the requested items are available. Only one inventory exists, which means that the InventorySystems checks both types of items. Therefore, the inventory system modifies the valid property of the incoming messages and optionally modifies the validationResult property.
    - The ResultSystem collects the processed orders and prints a message to the console once a new order is received. It must distinguish between valid and invalid orders.
- The result of the order application integration is a common/transformed format for orders. Each order consists of the following properties (type String):
    - CustomerID
    - FirstName
    - LastName
    - OverallItems (Number of all items in order)
    - NumberOfDivingSuits
    - NumberOfSurfboards
    - OrderID
    - Valid
    - validationResult
- In order to implement the order application integration, use at least the following patterns:
    - Message Endpoint (for the WebOrderSystem), Channel Adapter (for the CallCenterOrderSystem), Point-to-Point Channel, Message Translator, Content Enricher
- In order to implement the order processing application integration (i.e., BillingSystem, InventorySystem, ResultSystem), use at least the following patterns:
    - Publish-Subscribe Channel, Point-to-Point Channel, Aggregator, Content-Based Router
- It is possible to further extend the scenario by integrating more patterns (e.g., Splitter).

Additional notes

- We recommend relying on Java Messaging Services (JMS) and ActiveMQ to provide communication channels.