# Distributed Algorithms - Exercise

Winter term 2016/17

CIT

## *Exercise sheet 3*

The third exercise will target patterns for Enterprise Application Integration (see lecture slides – Design Patterns). The objective is to develop an integration solution using Apache Camel. Apache Camel is a Java-based integration framework supporting most of the patterns presented in the lecture and the corresponding literature.

**Exercise 3.1: Vector Clocks**

i. Causal Order:
   The concept of the "causal broadcasat" has been introduced in the lecture as an application of vector clocks. In order to guarantee the proper execution of the algorithm it was necessary to send each message to all participating nodes. If we want to avoid sending each message to all nodes, why wouldn't it be sufficient to only use the vector as applied by the causal broadcast to achieve causal order?

ii. Order Relation
   If e is a system event, then V (e) defines its vector time stamp and C(e) $\in$ N is the corresponding sum of the vector's components. Whenever an event e occurs, V (e) is computed as introduced in the lecture.
   1. Show that C(e) together with the "less than"-relation (<) defines a partial order of the set of system events.
   2. Show that the respective logical clock fulfills the clock condition.
   3. Show that the partial order can be extended to a total order by applying process IDs together with the vector time stamp.

**Exercise 3.2:** [(Dive-Surf Inc.) Integration Scenario]
Develop an integration scenario 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

You are free to select an integration scenario of your choice or to implement the Dive-Surf Inc. scenario presented in the lecture. The only requirement is that the aforementioned set of patterns is used. If you decide for the Dive-Surf Inc. scenario, your solution must implement the "take and process order" integrations and comply to the following requirements:

- Dive-Surf Inc. offers at least two kinds of 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 applications 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: <First Name, Last Name, Number of ordered surfboards, Number of ordered diving suits, Customer-ID> - e.g.: Alice, Test, 2, 0, 1
    - **CallCenterOrderSystem**: the Call center order system generates a text file containing new orders every 2 minutes and stores it at a pre-defined destination in the local file system. Each line represents a single order. An order consists of comma-separated entries formatted as <Customer-ID, Full Name, Number of ordered surfboards, Number of ordered diving suits> - e.g.: 1, Alice Test, 0, 1. The full name always consist of the first and the last name separated by a space. It is not defined how many orders are contained in the file. However, it is required that at least one call center order is processed during presentation of the exercise.
    - **BillingSystem**: the billing system 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 optional modifies the validationResult property.
    - **InventorySystem**: the inventory system takes transformed order messages (see below) and simply 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 optional modifies the validationResult property.
    - **ResultSystem**: the result system simply collects processed order and prints a message to the console once a new order is received. The result system 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 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 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
- You may optionally further extend the scenario by integrating more patterns (e.g. Splitter).

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

**Additional notes and assessment:**

- important parts of the implementation have to be annotated with comments
- each exercise has to be completed handled in teams of 3-4 students
- the exercise sheet is successfully completed, if exercise 1 was presented and the solution was explained satisfactorily