Based on the files I've reviewed, I can now provide you with information about Use Case 1, which appears to be related to "Reverse Movement Handling" in the Kafka system.

## Use Case 1: Reverse Movement Handling

### Problem Description

In a Kafka-based system where SA (Service Area) teams publish messages about store-DC (Distribution Center) movements, there's a critical issue with reverse movements:

- \*\*Normal Flow\*\*: Items move from DC (origin) → Store (destination)

- \*\*Reverse Flow\*\*: Items move from Store (origin) → DC (destination)

The challenge is that the payload format remains the same in both cases, but the semantic meaning of the fields changes. In reverse movements, what was previously labeled as "destination" is now actually the origin point.

### Example Payload

```json

{

"destination": "2352",

"servicingNodes": ["960", "1001", "1002"],

"sourceLocation": "960",

"destinationLocation": "2352",

"movementType": "NORMAL" // or "REVERSE"

}

```

### Solution Implementation

The system implements two approaches to detect and handle reverse movements:

#### Approach 1: Movement Type Detection

This approach checks if the destination is present in the servicing nodes list to determine if it's a reverse movement:

```java

public class MovementTypeDetector {

public MovementType detectMovementType(KafkaMessage message) {

String destination = message.getDestination();

List<String> servicingNodes = message.getServicingNodes();

// If destination is in servicing nodes, it's reverse movement

if (servicingNodes.contains(destination)) {

return MovementType.REVERSE;

}

return MovementType.NORMAL;

}

}

```

#### Approach 2: Business Rule Based Detection

This approach uses business rules to determine movement type based on location types:

```java

public class BusinessRuleProcessor {

public boolean isReverseMovement(String source, String destination) {

LocationType sourceType = locationService.getLocationType(source);

LocationType destType = locationService.getLocationType(destination);

// If source is STORE and destination is DC, it's reverse

return sourceType == LocationType.STORE && destType == LocationType.DC;

}

}

```

### Key Components

1. [MovementProcessor.java](file:///Users/sarthak/CascadeProjects/kafka\_pain\_areas/MovementProcessor.java) - Main component that handles the detection and processing of reverse movements

2. [LocationService.java](file:///Users/sarthak/CascadeProjects/kafka\_pain\_areas/LocationService.java) - Service that determines location types (Store, DC, etc.)

3. [MovementValidationService.java](file:///Users/sarthak/CascadeProjects/kafka\_pain\_areas/MovementValidationService.java) - Validates processed movements against business rules

### Data Flow for Reverse Movement

```

Original Message: {source: "2352", destination: "960", servicingNodes: ["960", "1001"]}

↓

Detection: LocationService identifies 2352=Store, 960=DC

↓

Processing: MovementType.REVERSE, actualOrigin="960", actualDestination="2352" (SWAPPED!)

↓

Validation: Passes reverse movement rules

↓

Storage: Saved with flowDirection="STORE\_TO\_DC"

```

This solution ensures that regardless of the movement direction, the system correctly identifies and processes the actual origin and destination, maintaining data integrity throughout the process.