Reactive and Event Based Systems

Assignment 2: Actors and Databases

Part 1: Programming with Virtual Actors

In this part of Assignment 2, you will implement a simple game using virtual actors and test whether your implementation respects an invariant of the game. You should deliver for this part:

a) A ZIP file containing your code for Exercises 1 and 2 below (please use Microsoft Orleans 2.4.2 for your implementation).
b) A two-page PDF report discussing:
    1. your implementation of the game (please include an overview of your main decisions regarding concurrency and asynchronous communication among actors) ;
    2. your approach to test the invariant (please include an argument for why your approach is correct).

    The two-page report must use 11-point font of the Times family and be formatted single-column with 2 cm margin.

Exercise 1: The Ball-Passing Game

In this exercise, you will implement a simple ball-passing game. The game comprises N players, who pass $K < N$ balls among each other. When the game starts, the K balls are allocated at random among the players such that a player can have at most one ball. Moreover, each player is given a list of other player names that it is aware of and with whom it can exchange balls.

After initialization, the players perform the following behaviors concurrently:

1) If a player has a ball, it first chooses a random amount of time as a period length. It then repeats continuously the following steps:
    a. The player waits for the period to expire.
    b. When the period is up, the player then decides at random to either keep the ball for another period or to pass the ball.
    c. If the player decides to pass the ball, it chooses one of its known other players at random and sends the ball to that player.
2) If a player does not have a ball, it lies dormant until it receives a ball from another player (or needs to respond to any other communication event).
3) If a player receives a ball from another player, it first gets the ball passed, but then reacts in the following ways:
    a. If the player is now holding more than one ball, it enters a state in which it passes all but the ball received to known other players selected at random. It can then move to the next step below.
    b. If the player is now holding only the received ball, it executes step (1) above.

In your game implementation, each player must be modeled as a virtual actor. Balls, on the other hand, are not actors; you are supposed to simply keep track of them as part of player state. Your implementation must be performed with Microsoft Orleans (where virtual actors are called grains) and follow the handout code provided together with this assignment text.

Moreover, we expect that the game implementation be resilient to grain deactivations, i.e., the behaviors of the game should continue to be executed consistently and the grain should be reactivated if necessary. However, you do not need to ensure that the behaviors will execute consistenty across silo failures.

Exercise 2: Testing an Invariant

In the ball-passing game above, it must be true that there be K balls in the game in total at any point of time. Design and implement a program that checks this invariant, i.e., when the program is called, it communicates with the player actors to establish a global state of the game and then tests if there are really only K balls in

this global state. The program outputs whether the ball-passing game implementation generated a global state that respects the invariant or not. Note that you may need to extend the logic of the player actors to support the checking of the invariant.

Part 2: Modeling an Actor-Oriented Database Application

In this part of Assignment 2, you will model an actor-oriented database application scenario. You should deliver for this part:

a) A two-page PDF report documenting for Exercise 1:
    1. your UML model for the scenario (please format your diagram so that it fits in a single page maximum);
    2. an overview of your main decisions regarding which entities in the domain should be captured as actors and which as non-actor objects and why (please relate your discussion to actor granularity and the principles in the article of Wang et al.[1]).

   The two-page report must use 11-point font of the Times family and be formatted single-column with 2 cm margin.

Exercise 1: A Simplified Global Logistics Scenario

In this exercise, you will model a simplified global logistics scenario as an actor-oriented database. The resulting model should be represented as *a single class diagram in UML*, where classes are marked explicitly as actor or non-actor. Note that we expect that in your model, a non-actor class must *always* be (transitively) associated via aggregation relationships to an actor class, i.e., state must be represented in actor classes, even if non-actor classes are used to model it.

The scenario comprises the sea transportation network of a global logistics operator.[2] The operator owns vessels, which carry containers with various goods. A vessel has an identifier (e.g., "Milan Maersk"), a capacity (i.e., the number of containers it can hold), a cruising speed, and a schedule by which it visits several port terminals, e.g., Shanghai or Amsterdam. In each terminal, the vessel can either: (1) Get containers loaded; (2) Get containers unloaded; (3) Stop for other reasons (refueling, crew changes, etc). A container has an identifier (e.g., "TGHU 759933 0 45G1"), an origin terminal and a destination terminal, as well as deadlines to leave the origin and arrive at the destination. For simplicity, you may assume that containers are carried by a single vessel from their origin to their destination. However, a vessel schedule may be longer than that of any of the containers that get loaded and unloaded from it.

The operator and its customers wish to have visibility on the dynamic conditions of the transportation network. For that purpose, the operator needs to be able to keep track of the positions of vessels as well as perform changes to schedules to adapt to dynamic transportation conditions (e.g., vessel delay due to inclement weather).

The operator and customers wish to pose the following queries to the system:

1) Operator: With the current schedules, will any vessel get overloaded at any terminal?
2) Operator: Which vessels are expected to arrive at a given terminal (e.g., Amsterdam) in the next hour, given their current positions and their schedules? Note that some vessels may end up not respecting the deadlines in their schedules depending on their cruising speed and the distance between their current position and the port terminal.
3) Customer: What is the expected time to arrival of a given container number to its destination terminal? Note again that this time is influenced by potential delays.

The system is expected to scale to large numbers of terminals, vessels, and containers while supporting the queries above.

---

[1] Yiwen Wang, Júlio César dos Reis, Kasper Myrtue Borggren, Marcos Antonio Vaz Salles, Claudia Bauzer Medeiros, Yongluan Zhou: Modeling and Building IoT Data Platforms with Actor-Oriented Databases. EDBT 2019: 512-523.

[2] This scenario is inspired by personal communication with Harry Huang of Harry Consulting ApS.