Composition

Distributed Systems Paradigms
Lab Guide 5

Use stream composition operators to convert sequential to reactive code and obtain applications.

Steps

1. Convert the following sequential code to its reactive equivalent:[1]

   ```
   int op1(int i) { Thread.sleep(i); return i+1; }
   int op2(int i) { return i*2; }
   int m1() { int i=1; i=op1(i)+1; i=op2(i); return op1(i); }
   ```

2. Convert also the following method:

   ```
   int m2() { int i=op1(1)+op1(2); return op2(i); }
   ```

3. Using the selector main loop (from Lab 3 or Lab 4) and stream composition, implement a chat server.
4. Test the chat server with the non-interactive client (Lab 1).

Questions

1. How to call `op1(...)` in parallel from `m2()`?
2. What would be the equivalent programs in a language with *async/await* primitive (e.g., Python)? How would these programs be implemented directly on a `Selector`?
3. How does the final chat program deal with all the scale and performance concerns (memory usage, scheduling overhead, ...)?

Learning Outcomes Apply reactive composition to translate sequential code. Recognize the opportunities for parallelism and how to exploit them.

[1]Hint: See the `delay(...)` operator.