DM519 Concurrent Programming

Spring 2013 Re-Exam Project

Department of Mathematics and Computer Science
University of Southern Denmark

June 4, 2013

Introduction
The purpose of the project for DM519 is to try in practice the use of models in the design and implementation of concurrent programs.

Please make sure to read this entire note before starting your work on the project. Pay close attention to the sections on deadlines, deliverables, and exam rules.

Exam Rules
This project is an exam. Thus, the project must be done individually, and no cooperation is allowed beyond what is explicitly stated in this document.

Deliverables

- A short project report in PDF format (4-8 pages excluding front page and appendix) has to be delivered. This report should document the result of has to contain at least the following 4 sections:
  - **Modelling:** design decisions, FSP model, structure diagram
  - **Analysis:** absence of deadlocks, safety, liveness
  - **Implementation:** threads vs monitors, relevant parts
  - **Testing:** correctness of implementation
- FSP model as .lts file
- Java source code as .java file

The deliverables have to be delivered using Blackboard's SDU Assignment functionality. Delivering by e-mail or to the teacher is only considered acceptable in case Blackboard is down directly before the deadline.

Deadlines

June 26, 2013, 12:00

The Problem

With the experience from designing elevator systems for military purposes in North America, IMADA has been chosen to design the new elevator system for the main elevator of the SDU Campus Tower, a new tower to be built on top of main entrance. The board of directors has approved the following specification.

The tower has a total of 6 floors:

U the university level ground floor

1 the 1st regular floor

2 the 2nd regular floor

3 the 3rd regular floor

4 the 4th regular floor

P the professor floor for grumpy old professors

On each floor there is a button to call the elevator to that floor. Likewise, inside the elevator there are six buttons for U, 1, 2, 3, 4, and P. When a call button is pressed, all other call buttons are deactivated until the target floor is reached. Thus, a memory of the buttons pressed is not necessary for this elevator. Here is a possible model of one of these buttons.

```
// example FSP for an elevator call button
BUTTON = (call -> BUTTON).
```

The elevator has a maximum capacity of C=3 persons, which may never be exceeded for safety resasons. To count the number of persons in the elevator and to ensure at most C people are inside the elevator, each floor has a two-way turnstile that can be blocked.

```
// FSP for the turnstiles
TURNSTILE = TURNSTILE[FALSE],
TURNSTILE[locked:B] =
    ( lock -> TURNSTILE[TRUE]
    | unlock -> TURNSTILE[FALSE]
    | when (!locked) in -> TURNSTILE[locked]
    | out -> TURNSTILE[locked]).
```

Obviously, grumpy old professors prefer not to mix with ordinary humans (here: students). Hence, the elevator has some special requirements:

- On the ground floor U, professors have a special entrance and exit on the opposite site of the normal entrance.
- Non-professors do not enter through the special entrance.
- Non-professors do not try to get to the professor floor.
- If the call button at the professor floor or the special entrance is pressed, the elevator delivers it non-professor cargo before answering the request.

Your Tasks

Your task is to implement the elevator system as a Java program with the ability to test its function by letting grumpy old professors and non-professors use the system.

To this end, you are expected to perform the following tasks:

0. Read this description very carefully. You will probably find that a lot of details are underspecified. You can make your own choices, but try to keep them meaningful.
1. First model the elevator without a controller.
2. Observe that too many people may get into the elevator, causing it to fall down. Add a safety property CAPACITY, that would hold if the capacity of the elevator would never be exceeded.
3. Add a controller to the elevator such that the capacity can no longer be exceeded. Verify this using the LTSA tool.
4. Observe that a grumpy old professor might be forced to share elevator with annoying students. Add a safety property SEGREGATION, that would hold if professors would never have to share the elevator with students.
5. Enhance the controller such that professors do not risk to share the elevator with students anymore. Verify this using the LTSA tool.
6. Add a liveness property VIP formally verifying that a professor is always eventually permitted to enter and leave his apartment. You may assume that people eventually move in and out the elevator. Verify this using the LTSA tool.

7. Make a structure diagram of your system (without the safety and liveness properties).

8. Structure your model into an implementation. Which processes should be implemented as (active) threads and which as (passive) monitors?

9. Implement your model in Java. As usual, try to reuse action names as method names in order to make the connection between the model and the implementation obvious.