Lecture 8: Reactive and Real-time System Modelling

Exercises:

1. Use the **Finite State Machine Explorer** to model the "Bank-box Code" example (pp.20 of slides). If your model is not a reduced one, try to determinize and minimize it using the corresponding buttons.

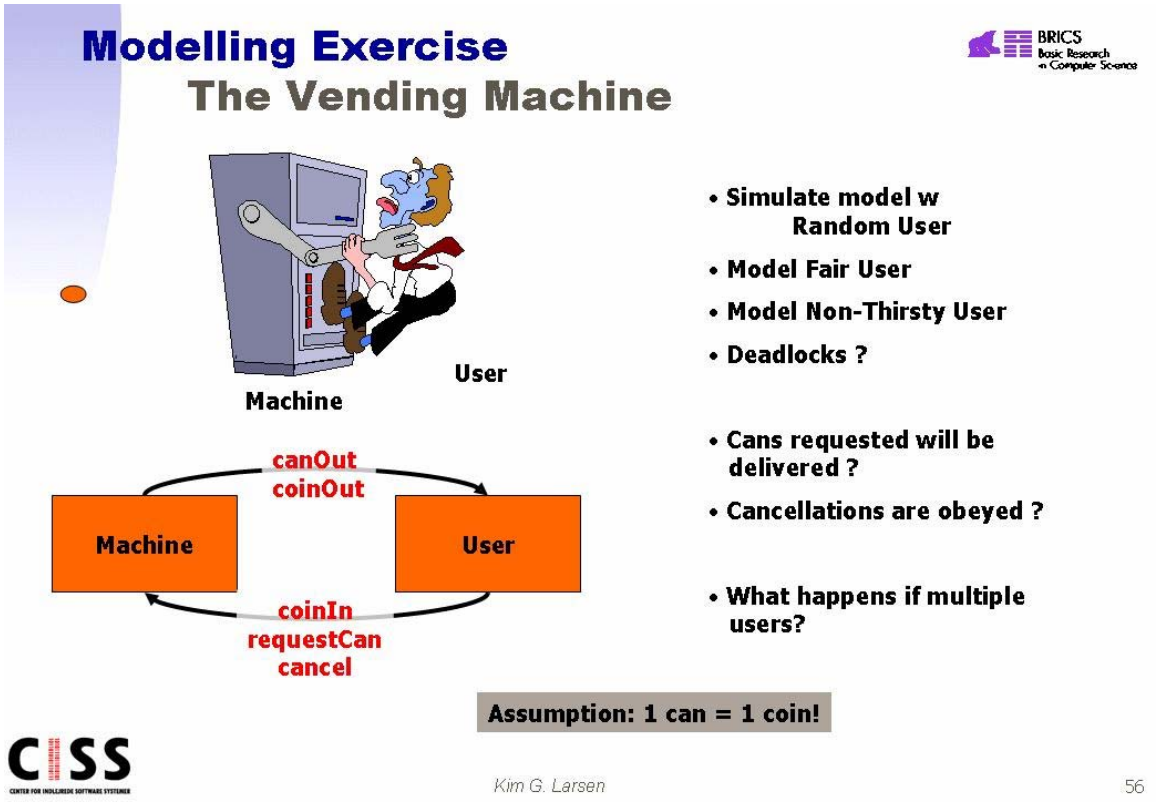
2 (Simple Vending Machine).

In this exercise you are required to model a (very) simplified vending machine for beverage cans. The vending machine is supposed to sell cans (containing some interesting substance) to customers. In this simplest version, the machine only sells a single kind of cans where a can cost one single unit (of the currency of your choice). We also assume that the vending machine has an unlimited number of available cans. Purchasing a can is usually performed as follows:

1. The customer inserts a coin into the machine
2. The customer may now either
   - request a can, or
   - cancel
3. Depending on the order of the customer the vending machine should either provide a can or return the coin.

Model in UPPAAL the vending machine. Also make models of various types of customers including
- a random customer (may at any given moment non-deterministically try to insert a new coin, cancel an order, ...etc),
- a fair customer (behaving as intended by the machine),
- a non-thirsty user (always cancelling after insertion of coin).

Validate the various configurations (machine and a particular customer) by simulation. Check for (absence) of deadlock in your model. Try to verify that cans requested will be delivered, and cancellations are obeyed. What happens if the configuration involves multiple users (possibly of different type)?

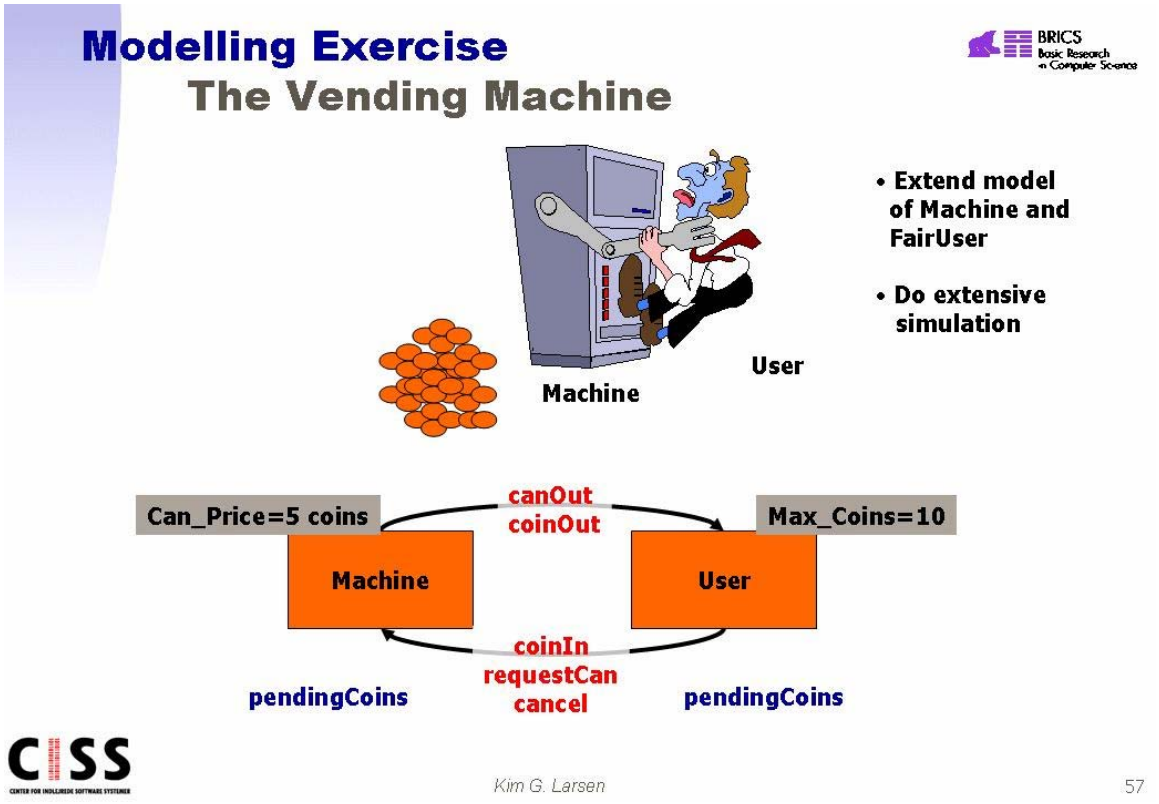


3 (Extended Vending Machine).

Extend the model of the vending machine from the previous exercise so that the price of a can is 5 units (or coins). Also change the behaviour of the customers so that multiple coins may be inserted before requesting a can is attempted -- both the machine and the user should have their own local variable (pendingCoins) for recording the credit of the customer. When a can is requested the surplus of coins should be given back to the customer. To make matters simpler you may assume that the user has a limited number of coins (say 10).

Try to establish the following properties either by simulation or verification:

1. If the pending amount of money is sufficient (=5) and (only) the button "requestCan" is pressed then a can will be given out.
2. A can is only delivered if the pending amount of money is sufficient (=5).
3. As soon as a can is delivered, the change is given to the customer.
4. The machine does not lose or produce money.
5. The amount of pending coins in non-negative.
6. If cash is pending and (only) the "cancel" button is pressed then the machine will output all pending coins and no can.

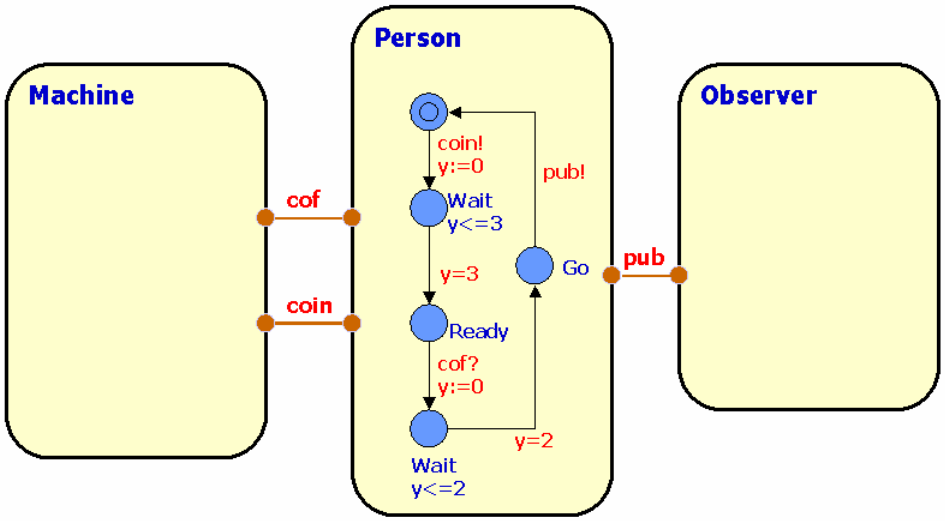


4 (Timed Coffee Machine).

In this exercise you are asked to design the control of a **Machine** (the control program) which will serve a coffee craving **Person** (the environment). As you can see below the person repeatedly (tries to) insert a coin, (tries to) extract coffee after which (s)he will make a publication. Between each action the person requires a suitable time-delay before being ready to participate in the next one.

The machine takes some time for brewing the coffee and will time-out if coffee has not been taken before a certain upper time-limit.

As a requirement we want the overall behaviour to ensure that the indicated **Observer** experiences a constant flow of publications from the system. In particular we want the Observer to complain if at any time more than 8 time-units elapses between two consecutive publications. Model the **Machine** and **Observer** in UPPAAL and analyze the behaviour of the system. Try to determine the maximum brewing time allowed by the Machine in order that the above requirement is met.


Person
Machine
Observer
coin!
y:=0
Wait
y<=3
y=3
Ready
cof?
y:=0
Wait
y<=2
y=2
pub!
Go
cof
coin
pub