Evaluation Of A Structured Design Methodology For Concurrent Programming

Lex Bijlsma – Kees Huizing – Ruurd Kuiper
Harrie Passier – Harold Pootjes – Sjaak Smetsers

Open University – Radboud University
Eindhoven University of Technology

CSERC 2019
18 November – Lanarca

Evaluation Of A Structured Design Methodology For Concurrent Programming

Lex Bijlsma – **Kees Huizing** – Ruurd Kuiper
Harrie Passier – Harold Pootjes – Sjaak Smetsers

Open University – Radboud University
Eindhoven University of Technology

CSERC 2019
18 November – Lanarca

CONTEXT

- First year university course on OO programming
- Students have knowledge of classes, interfaces, inheritance, basic familiarity with UML
- Concurrency part introduces threads, time-slicing, non-determinism, atomicity, race condition, synchronization, locking and deadlock

CONCURRENT PROGRAMMING

- Notoriously difficult
- familiar constructs get new semantics:
  `x:=x+1` may result in **`x`** not changing value
- objects may be accessed in inconsistent state
- non-deterministic

DIFFICULTIES

- sequential execution model is well understood (after some time)
  straightforward relation between execution steps to statements
- concurrent execution model (interleaving) more complex *and*
  relation between execution and program is much less direct
- ⇒ harder to track bugs and harder to derive program
  from intended execution

CONSEQUENCE IN EDUCATION

- students get stuck while programming or are wildly trying
- ⇒ don't complete the exercises ⇒ learning-by-doing fails
- non-determinism ⇒ errors in the program may go unnoticed ⇒ learning by doing fails
- vicious circle

APPROACH: PROCEDURAL GUIDANCE

- prevent students getting stuck by providing them with a step-wise construction approach (Merriënboer & Kirschner: supportive information)
- every step produces an artifact
- during each step one design issue is solved
- program is developed during the process

ONE STEP

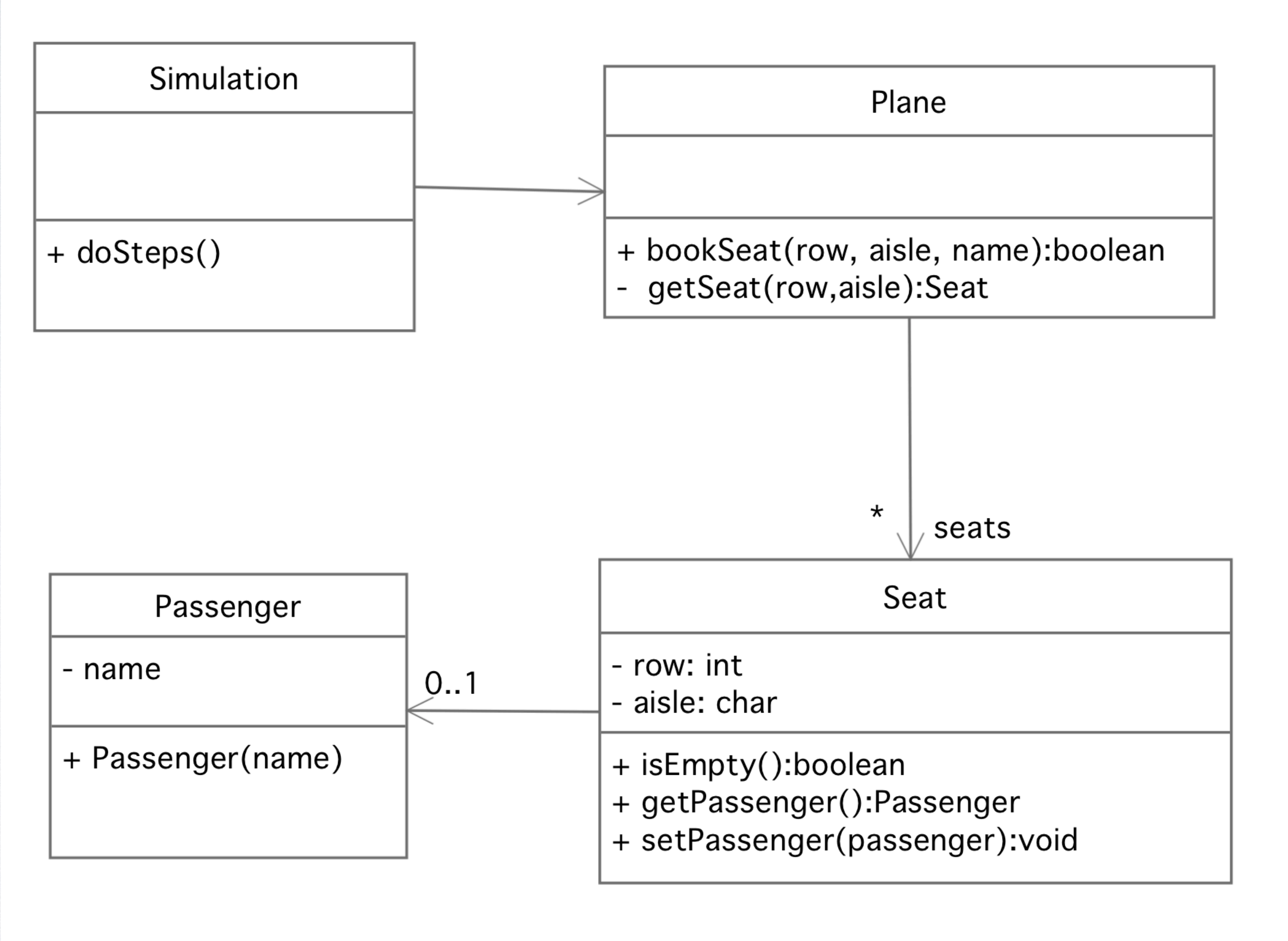
1. Analysis
2. Design decision
3. Implementation or other artifact, documenting result of step
4. Reflection

RUNNING EXAMPLE
(not the example of the experiment)

- System for booking seats in airplanes
- with concurrent simulation

STEP 1

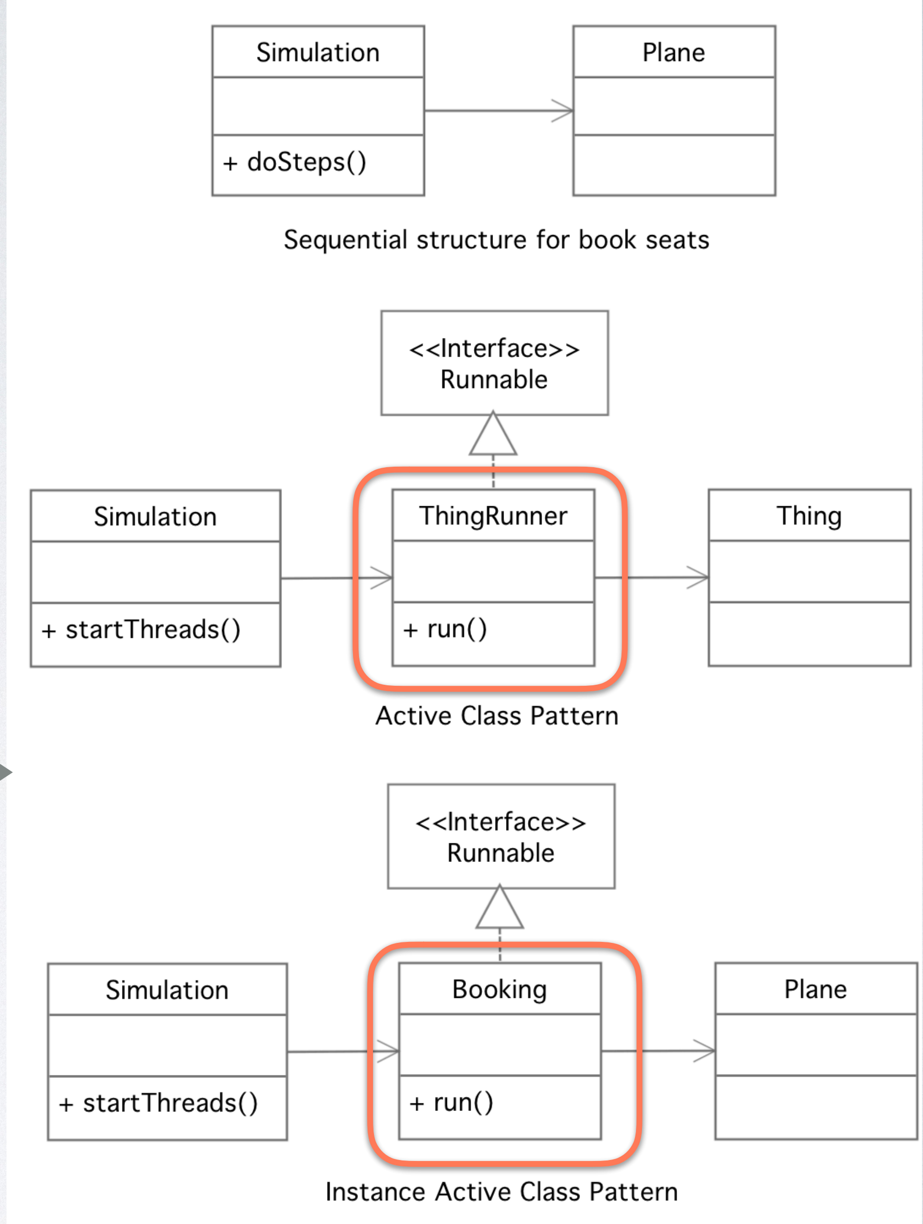
- OO structuring of the problem domain



STEP 2: CONCURRENCY OF THE PROBLEM

- does concurrency apply here?
- identify concurrent activities
- implement the *active classes*

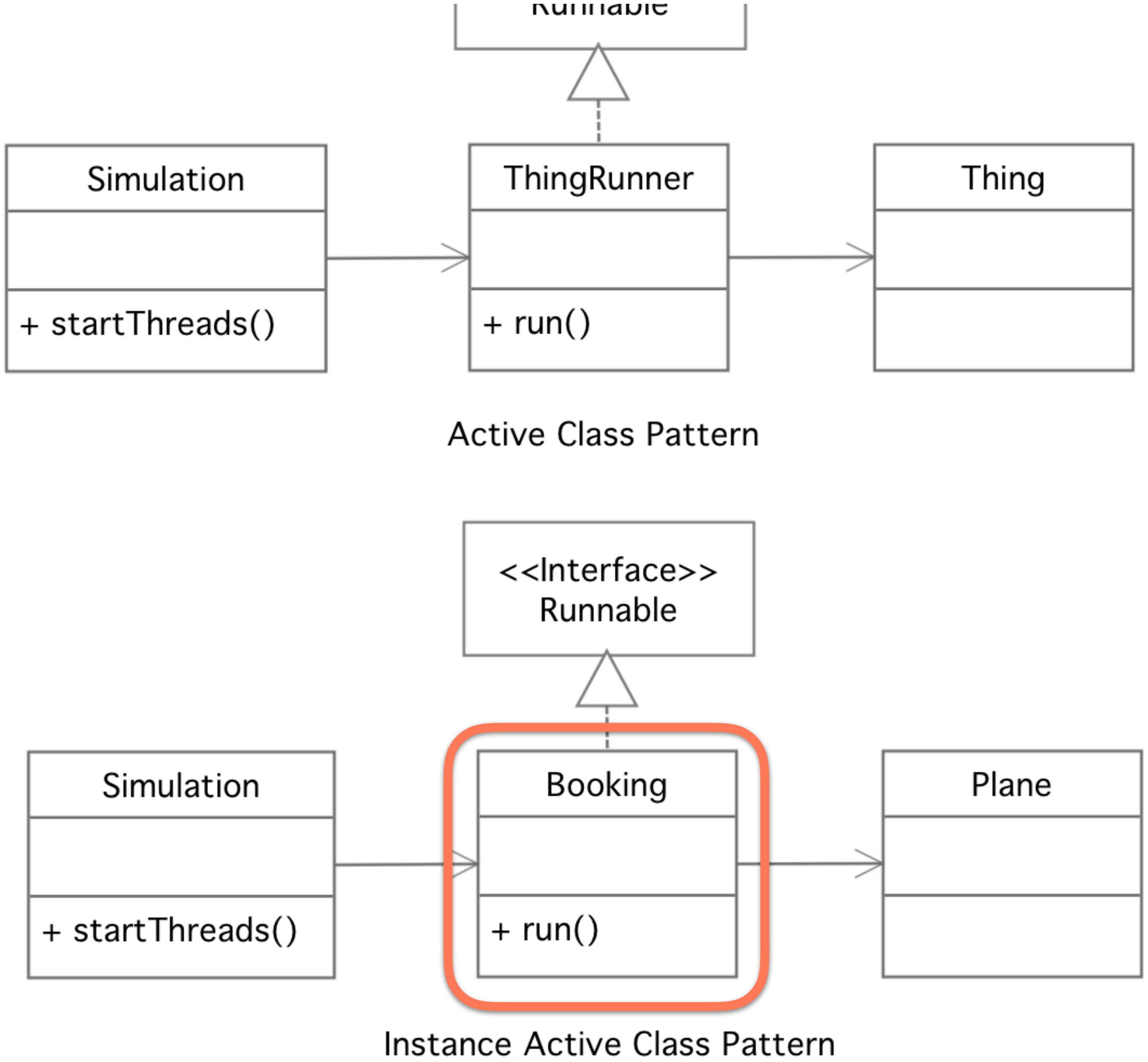
active class pattern



STEP 2: CONCU
OF THE PROBLE

- does concurrency apply here?
- identify concurrent activities
- implement the *active classes*

active class pattern →



STEP 2: (IMPLEMENTATION)

- instances of Booking will run concurrently

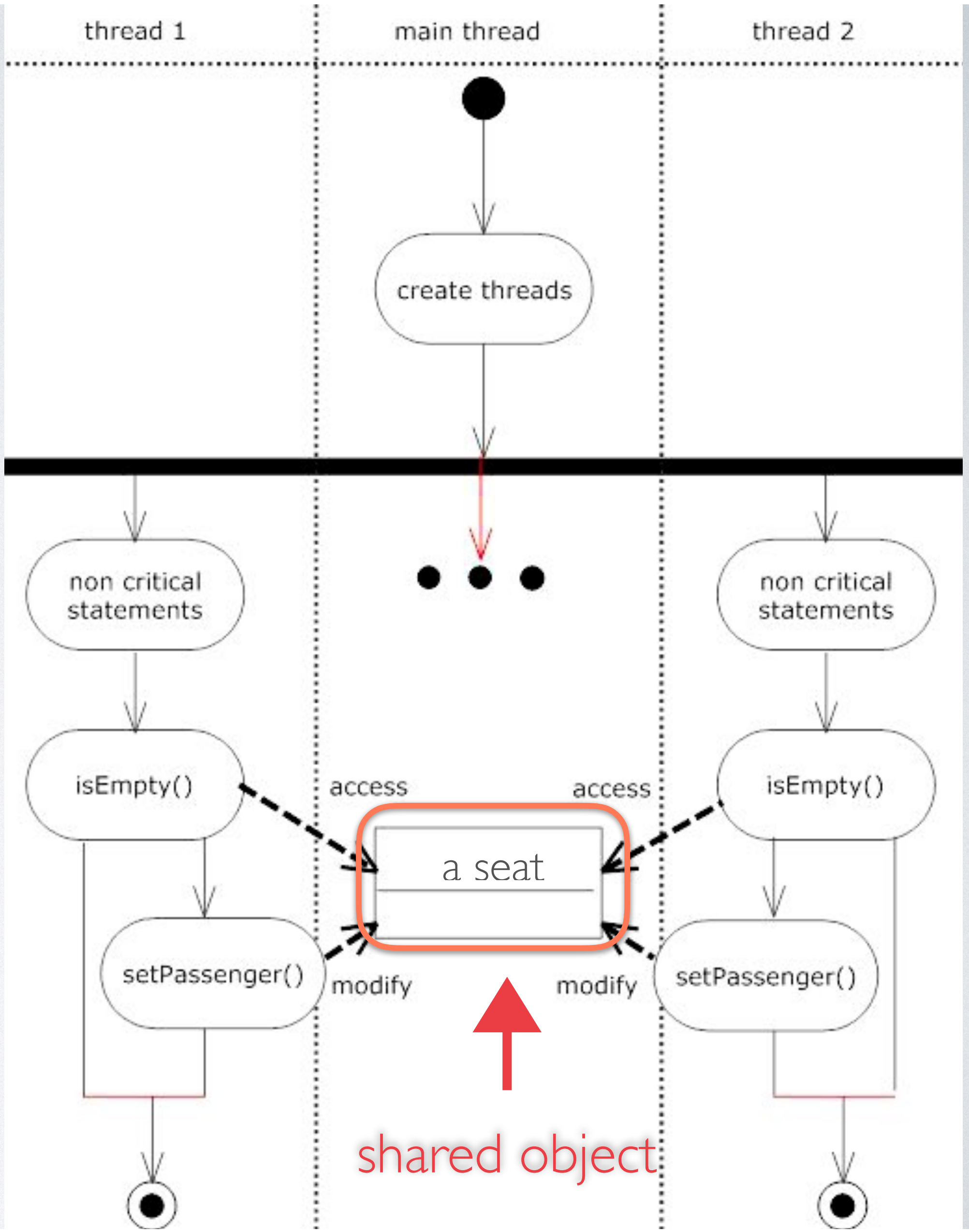
```
public class Booking implements Runnable {
   ...

   @Override
   public void run() {
      ...
      boolean success = plane.bookSeat( row, aisle, name );
      if ( success ) {
         ...
      }
   }
}

public boolean bookSeat( int row, char aisle, String name ) {
   Seat seat = seats.getSeat( row, aisle );
   if ( seat.isEmpty() ) {
      seat.setPassenger( new Passenger( name ) );
      return true;
   } else {
      return false;
   }
}
```

STEP 3: RACE CONDITIONS

- analyze using extended activity diagram
- with *swim lanes* denoting threads
- threads accessing same variables in *shared objects*
- have all access & modification to shared objects in *synchronized* methods



STEP 4: CHECK-THEN-ACT

another thread may grab the seat here

- thread checks a vari
and then changes it based upon that condition

- no guarantee that condition still holding at change, *due to other threads*

- reorganize check and change into one synchronized method or block

```java
public boolean bookSeat(
    int row, char aisle, String name ) {

    Seat seat = seats.getSeat(row, aisle);

      if ( seat.isEmpty() ) {
        seat.setPassenger(
            new Passenger( name )
        );
        return true;
      } else {
        return false;
      }

}
```

STEP 4: CHECK-THEN-ACT

another thread *can not* grab the seat here

- thread checks a vari
and then changes it based upon that condition
- no guarantee that condition still holding at change, *due to other threads*
- **reorganize check and change into one synchronized method or block**

```
public boolean bookSeat(
    int row, char aisle, String name ) {

    Seat seat = seats.getSeat(row, aisle);
    synchronized( seat ) {
        if ( seat.isEmpty() ) {
            seat.setPassenger(
                new Passenger( name )
            );
            return true;
        } else {
            return false;
        }
    }
}
```

STEP 5: REFLECTION ON PREVIOUS STEPS

- concurrent programs are not showing their bugs easily
- critical evaluation of the decisions
  - synchronization on the right objects?
  - is there enough concurrency?
  - etc.

OVERVIEW

step	topic	result
STEP 1	OO Structuring of problem domain	class diagram
STEP 2	Concurrency of the problem	- enhanced activity diagram - active classes implemented as threads
STEP 3	Race conditions	`synchronized` methods
STEP 4	Check-then act	code reorganized into `synchronized` methods or blocks
STEP 5	Reflection	results of reiterated steps

RESEARCH QUESTIONS

- What problems did the students encounter with the Steps Plan (related to issues with the steps or combinations of steps)?
- What problems remain after the Steps Plan (general issues with concurrency)?

EXERCISE

- Simulation of taxi service at a station. Passengers arrive by train and take taxis.
- A sequential solution is provided, students should turn this into a concurrent solution (with taxis being threads, etc.)

DATA COLLECTION AND ANALYSIS

- students make exercise (three institutes)
  - think-aloud sessions recorded on video
  - in-depth interviews with students
- analysed with qualitative techniques
  - pair coding
  - categorizing codes

RESULT OF ANALYSIS

category	*observations*
Concepts	
Synchronization	Synchronizing the wrong code; synchronizing too much or too little code.
Shared resources	Identifying the wrong object as shared.
Race conditions	Misunderstanding the concept of race condition and/or of check-then-act.
Correctness	
Testing	Assuming that concurrent programs are deterministic and that tests are reproducible.
Input/Output analysis	Assuming the program is correct once it produces some output.
Procedural guidance	
Active Class design pattern	Misunderstanding the design pattern: failure to identify actor; referring to the design pattern at the wrong time.
Following the steps	Starting a next step before the preceding one has been completed; taking the steps in the wrong order.
Performing the steps	Confusing the active class with the class that creates the thread; being unable to perform the refactoring required by a correctly identified check-then-act situation; unclear how the domain model classes correspond to the thread model.

TYPES OF ISSUES

- Steps Plan weaknesses
- Problems with understanding concurrency

RESULT OF ANALYSIS CTD

category	*observations*
Implementation	
Emphasis on code	Inspecting code rather than design; ignoring the procedure and starting on the code right away.
Sequential simulation	Reproducing the limitations of the sequential simulation in the problem statement; trying to adapt the sequential simulation rather than designing afresh.
Other (unexpected) activities	
Anthropomorphisms	Ascribing human motives to threads and objects; detection of final state by observing prolonged inaction.
Unsuccessful approaches	Trial and error, (random) googling; having no plan at all; just responding to IDE error messages

EXAMPLE 1

- Student:
  "The thread has to be created afresh every time. I just happen to know that. […] There are four taxis and there will never be more. But each taxi is inserted into a thread as a task, and when it is finished its work it should go for a new ride. Then you should start a new thread, hence also create one."

- Issue: Steps Plan too high level (active class identification and thread creation not clearly separated).

EXAMPLE 2

Student A:

"Why don't you make the whole thing synchronized?"

Student B:

"Because the synchronized part should not be made too large."

Student A:

"What's too large?"

Student B:

"You should not sleep within the synchronized block. Because there may be no people waiting at the station."

Student A:

"Let's measure how long the sleep lasts."

- Issue: Struggle with concurrency granularity. General problems with concurrency.

EXAMPLE 3A

- Passengers are waiting, many taxis are created, but no passenger is taken. Nevertheless program produces some output in the right form and students seem satisfied.
- Issue: Incorrectness not observed.

EXAMPLE 3B

Student A:

"While not station is closed, ... well, ...But, in that case he should close. The train will close the station ... Look at this!"

Student B:

"All passengers have been transported."

Student A:

"I think it is ok so. We finished the job. We have to write our report."

- Issue: Correctness not properly checked.

EXAMPLE 4

Student :

"1 2 3 4 1 2. Hey! How is that possible? That is strange. Why didn't it do that a moment ago?"

- Issue: Not aware of consequences of non-determinism. General concurrency problems.

EXAMPLE 5

Student:

"Yes, that is wrong. There should be something … indeed. […] Can you say that after a number of taxi rides he simply stops? Or, that after a long time of waiting, in case he has waited ten times and there still aren't passengers there, he goes home?"

- Possible issue: anthropomorphism.

ANTHROPOMORPHISM

- anthropomorphism: important faculty of human coginition (my view)
- nevertheless here possibly detrimental:
  objects in context of concurrency lead too easily too anthropomorphic miscinceptions?

CONCLUSIONS / LESSONS

- Sequential solution to be concurrified was not helpful. Better (if we want to give them a flying start): provide a framework of domain classes
- Exercise to be more specific about which activities to be concurrent
- Steps Plan should separate task defintion (active classes) and task creation
- Steps of Plan to be refined into micro steps *when needed* (use adaptivity)

CONCLUSIONS / LESSONS

- Amount of concurrency (nr. of threads and granularity) is a struggle for the students. Exercise needs to find a balance between giving away and letting students swim.
- Self-critical attitude should be elicited: Reflection step of Plan to be extended with means of how to check the output for correctness
- Exercises should be realistic (ideally, concurrency should be implied by the problem). (Taxi exercise had its problems.)

CONCLUSIONS

- Steps Plan does help students (evidence in results)
  - overall structuring in steps *and* structured approach per step
- A Steps Plan helps in education analysis, since it makes the structure of the exercise solving process more explicit and uniform
  - and helps the teacher in her student support

FUTURE WORK

- refine the procedure, deal with weaknesses that appeared
- larger practice runs and evaluation with new exercises
- extend with more advanced concurrency constructs
- long-term goal: comprehensive procedural guidance with rules, notation, and steps; supporting analysis and program design