Discussion SE Exercise 1

Dustin Wüest and Cédric Jeanneret

Requirements Engineering Research Group
Department of Informatics
University of Zurich

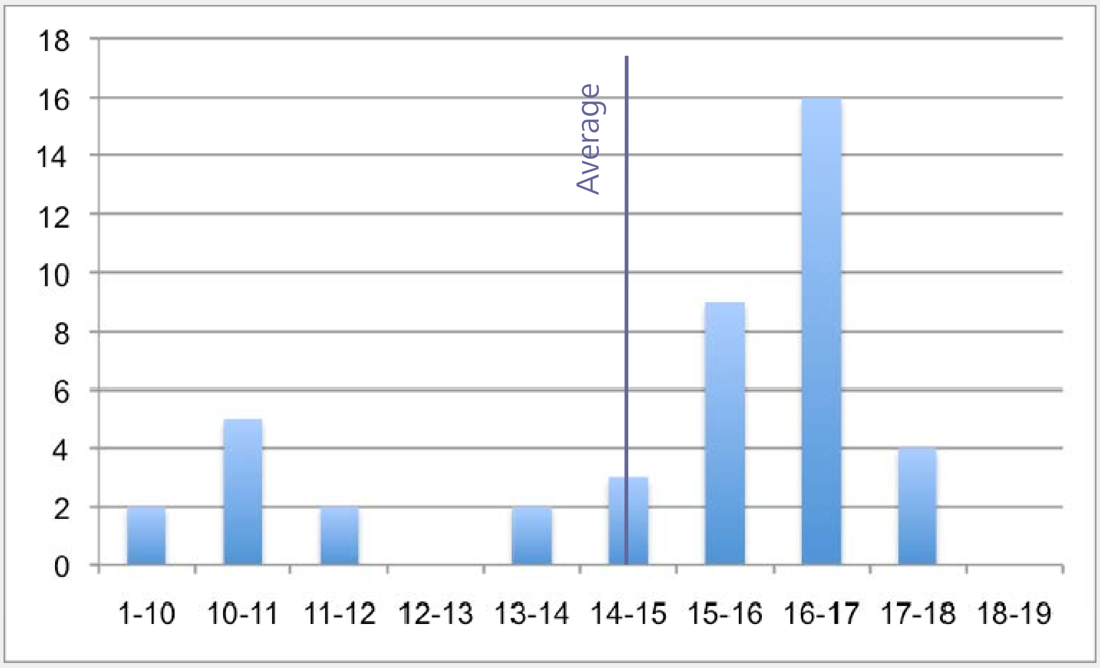
SE Discussion / Interviews

Interviews are not systematic (random sample). They are meant to verify that every member of a group participates in solutions elaboration.

Future discussions may be based on short presentations of students solutions.

In any case, you will be informed in advance. You are free to decline the invitation for a presentation but not for an interview.

SE Exercise 1 Results



Submission Protocol

Archive

- Filename schema: Ex[n]_[NameA]_[NameB]
- Without special characters.
- Example: Ex2_Wueest_Jeanneret
- Content: a document and source code (no libraries, …)

Document

- PDF files only
- Must contain group members name and matriculation number
- If possible, send one document

Email subject begins with [SE EX HS08]

From now, these requirements must be satisfied

Part I – Code Understanding

To apprehend JClusim, you were asked to:

- Understand the concepts used in JClusim
- Describe the generic behavior of an agent
- Figure out how an experiment is configured

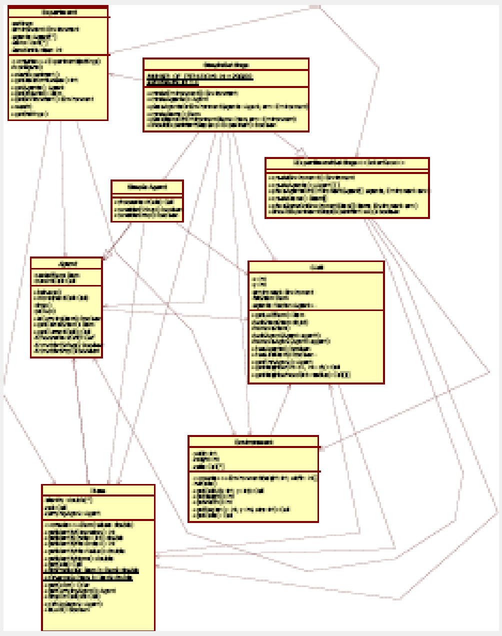
These were prerequisites for the second part of the exercise!

Part I – Code Understanding
Ex 1: Structure

Evaluation:

- The "right" level of details
  - Multiplicities and role names for associations
  - Static and abstract elements
- Correctness of the model

Frequent Problems Ex 1
Layout

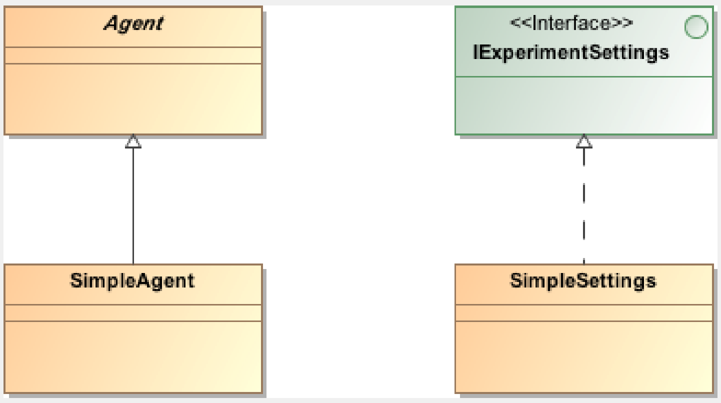


A diagram looses its value if it is poorly layouted.

Frequent Problems Ex 1
Generalization / Realization

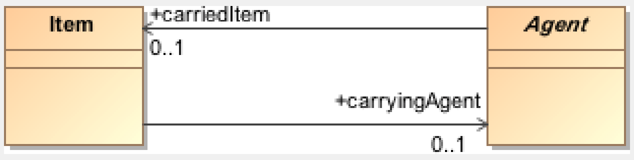
A generalization (left) relates a specific classifier to a more general classifier.

A realization (right) relates an implementation to its specification.



Frequent Problems Ex 1
Navigable Associations

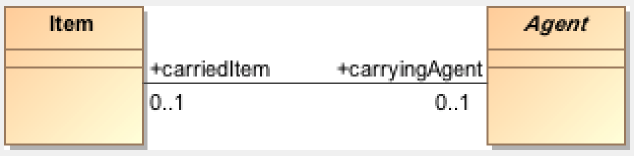
The second diagram (bottom) implies that if an Agent **a** carries an Item **i**, the following constraint holds:



$\neq$

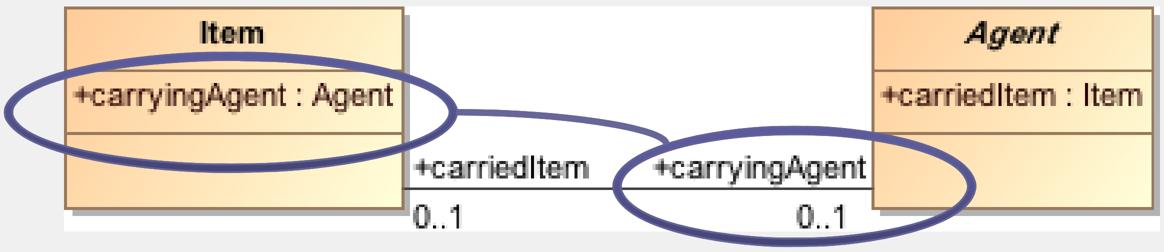
a.carriedItem.carryAgent == a

The first diagram (top) does not.



Frequent Problems Ex 1
Assoc. + properties

The class Item has two **carryingAgent** properties: one as attribute, the other as association end

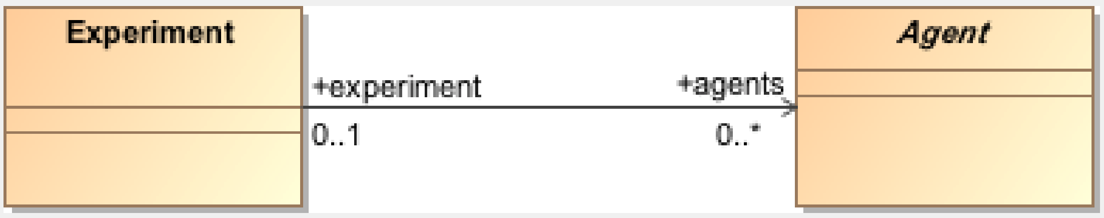


Frequent Problems Ex 1
Assoc. + properties

University of Zurich

In UML, multiplicities are placed differently than in an entity relationship diagram.

In JClusim, several agents work in an experiment (and not vice-versa).



Part I – Code Understanding
Ex 2: Behavior of an Agent

University of Zurich

To understand the generic behavior of an Agent, you had the choice between:

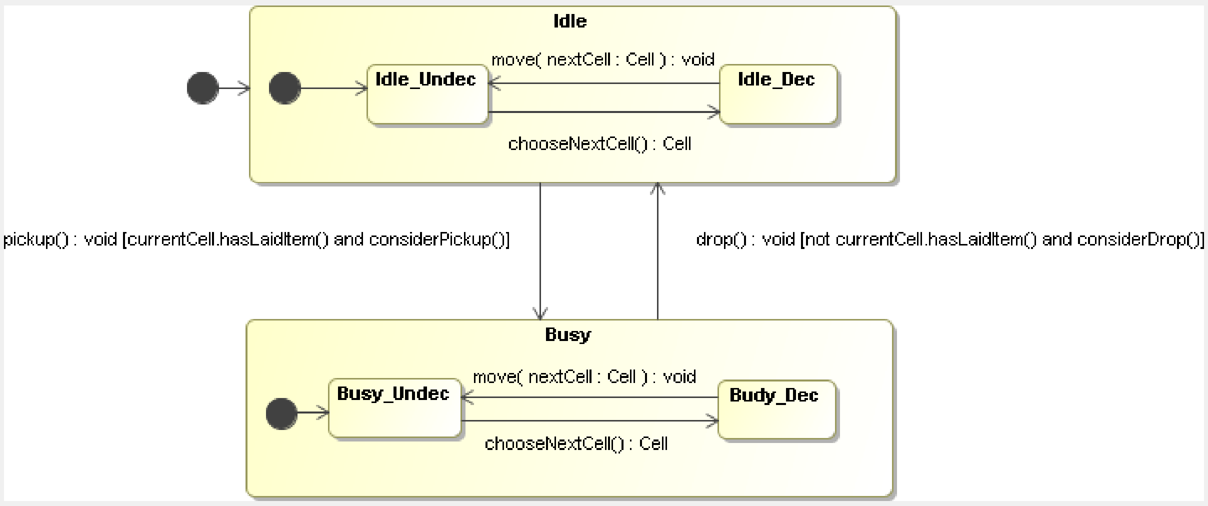
a) Represent an Agent as a state machine
b) Represent the behave() method as an sequence of actions

Evalutation:

- Correctness of the model (including its validity)
- The “scoping”. For example:
  - An agent does not know anything about the end of the experiment
  - We are not interested in the details of methods invoked by behave()

Part I – Code Understanding
Ex 2 a): Agent's behavior

An agent can essentially be in two states: Idle (when it does not carry an item) or busy (when it does).



Idle
Idle_Undec
Idle_Dec
move(nextCell : Cell) : void
chooseNextCell() : Cell
pickup() : void [currentCell.hasLaidItem() and considerPickup()]
drop() : void [not currentCell.hasLaidItem() and considerDrop()]
Busy
Busy_Undec
Budy_Dec
move(nextCell : Cell) : void
chooseNextCell() : Cell

Frequent Problems Ex 2 a)
Transition Labels

Usually, transition labels are written as:

Triggers [Guards] / Effect

A **trigger** is an event that may fires the transition

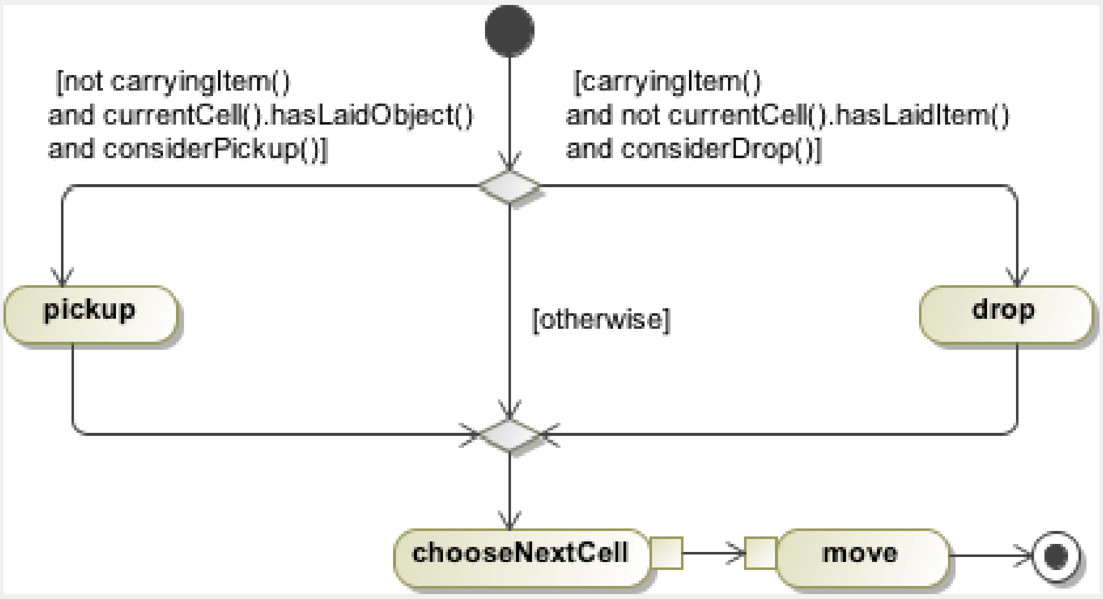
- The invokation of a method, the reception of a signal, …

A **guard** is a boolean expression (e.g. [isCarryingItem()])

- Enables or disables the transition
- Calling a method in a guard is allowed as long as it has no side effects (such a method is called query method)

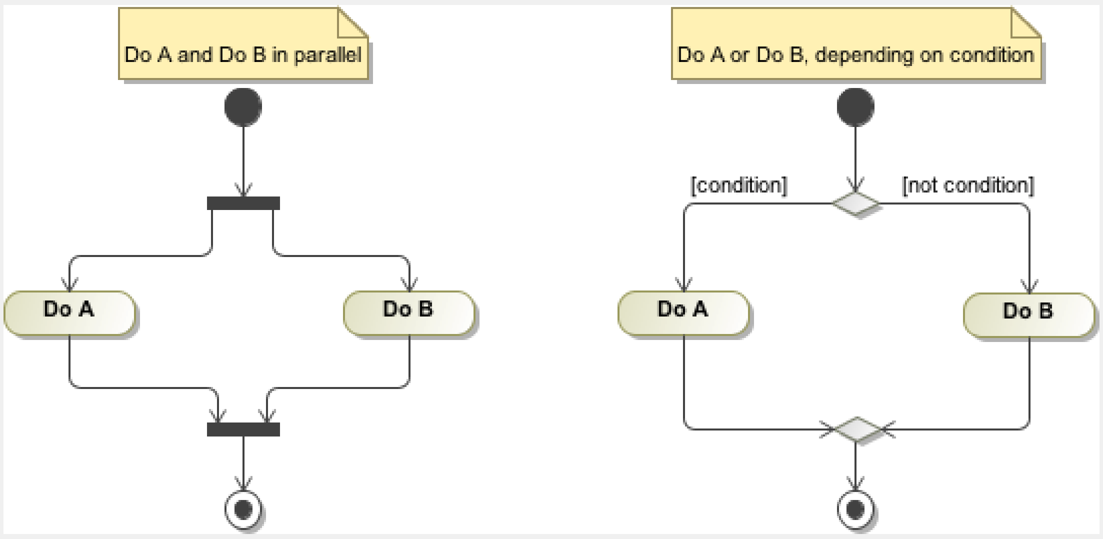
An **effect** is an optional behavior to be performed when the transition fires

Part I – Code Understanding
Ex 2 b): Agent's behavior



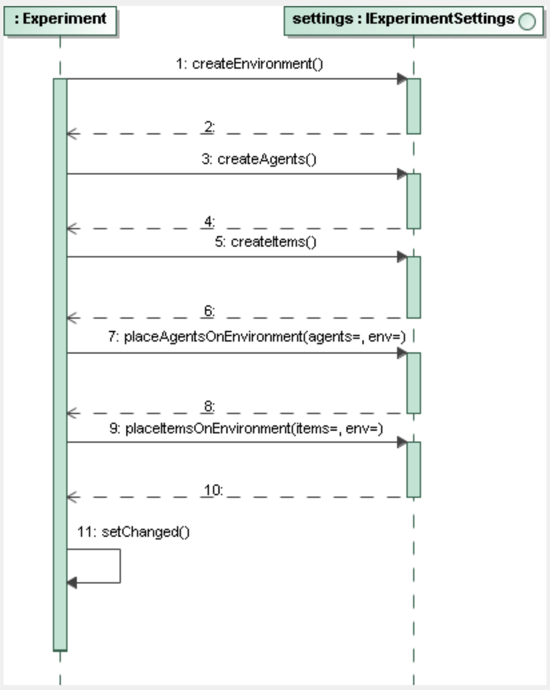
Frequent Problems Ex 2 b)
Fork/Join - Decision/Merge

Do not mix a decision (right, diamond at the top) with a join (left, bar at the bottom), otherwise, your activity will be blocked.


Do A and Do B in parallel
Do A
Do B
Do A or Do B, depending on condition
[condition]
[not condition]
Do A
Do B

Part I – Code Understanding
Ex 3: Experiment Initialisation

You were asked to describe the interaction between an experiment and its settings.

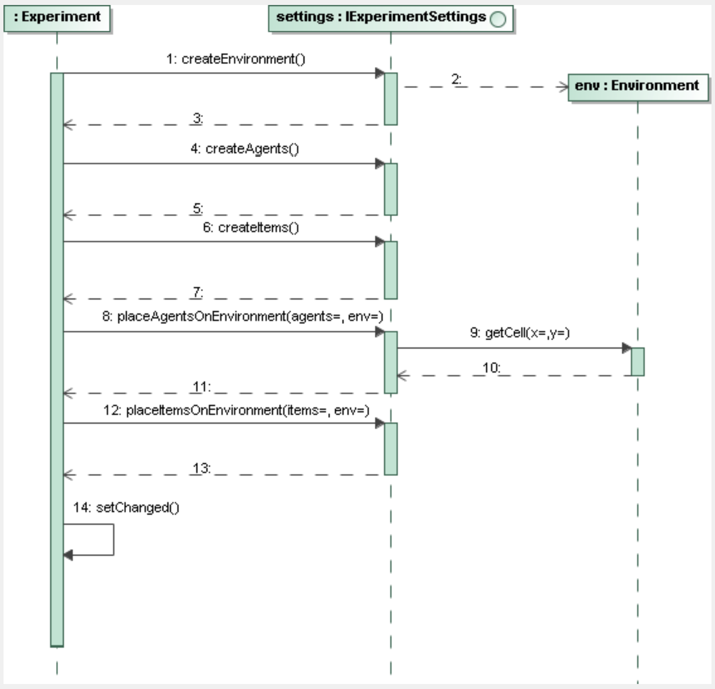


: Experiment
settings : IExperimentSettings
1: createEnvironment()
2:
3: createAgents()
4:
5: createItems()
6:
7: placeAgentsOnEnvironment(agents=, env=)
8:
9: placeItemsOnEnvironment(items=, env=)
10:
11: setChanged()

Frequent Problems Ex. 3

Replies to operation calls go back to the calling lifeline (and nowhere else).

(Creation of objects is usually depicted this way, to highlight the fact they have been created during the interaction.)



Part II – Code Improvement
Systematic Programming

University of Zurich

You were not asked explicitely to apply what you were taught during the first lecture, but you should have. It makes your code more understandable, especially for the correctors. Especially:

- Use meaningful and non-ambiguous names (bad example: maxValue for an attribute and valueMax for a variable)
- Do not write numerical values in the code, but factor them as a variable/constant (e.g. the size of the environment)
- Make your control structure visible with indentation

Part II – Code Improvement
Ex 1: Optimization

University of Zurich

Memoization:

+ querying an item is faster (and it happens often in JClusim)
- An item uses more memory
- The creation of an item takes more time

(-) If not implemented « correctly », the code of the constructor is scattered with optimization code and the public interface of an item may change

Lazy-initialisation has another purpose: spare memory by delaying the creation of a (relatively) large object until it is needed.

Part II – Code Improvement
Ex 1: Optimization

```
public Item(double[] values) {
 …
 // Memoization computations
 …
 color = computeColor();
}

private Color computeColor() {
…
}
```

Part II – Code Improvement
Ex 2: Documentation

```
package jclusim.base;
/**
 * The class represents agents in a JClusim experiment. An agent moves around
 * the environment. He can pickup items, carry them and drop them on an other
 * cell.
 *
 * This class is supposed to be extended to implement the specific behavior of
 * an agent.
 *
 * @author Cedric Jeanneret
 * @copyright Department for Computer Science, RERG
 * @history 2008-08-01 CJ First Version
 * @version 2001-08-01 CJ 1.0
 * @responsibilities This class implements the behavior of an agent
 * @see Experiment
 * @see Item
 * @see Cell
 */
public abstract class Agent {…}
```

Part II – Code Improvement
Ex 2: Documentation

```
/**
 * Reference to the item currently carried by
 * this agent.
 * null if the agent is not carrying an item.
 *
 * @see Item
 */
private Item carriedItem;
```

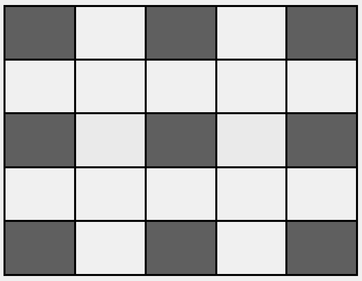
Part II – Code Improvement
Ex 2: Documentation

```
/**
 * This method actually moves the agent to another
cell.
 * This method is not meant to be overridden.
 *
 * @param nextCell The cell the agent is about to
move to.
 *
 * @pre nextCell != null
 * @post currentCell = nextCell &&
nextCell.getAgents().contains(this)
 */
public final void move(Cell nextCell) {…}
```

Part II – Code Improvement
Ex 3: New Experiment

Code reuse is not « blind copy-paste »…

- SimpleAgents used instead of the new Agent
- All agents at the same place instead of being randomly placed
- Items placed regularly instead of randomly



Choose Next Cell

```
int dx = RandomVariable.drawDiscreteUniform(-1, 1) * speed;
int dy = RandomVariable.drawDiscreteUniform(-1, 1) * speed;
```