Building LINDA from scratch

Distributed Systems / Hands-on

Sistemi Distribuiti / Laboratorio

Giovanni Ciatto Andrea Omicini

giovanni.ciatto@unibo.it andrea.omicini@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
ALMA MATER STUDIORUM – Università di Bologna a Cesena

Academic Year 2019/2020

Outline

1. Wetting your appetite

2. Linda Recap
   - Overview
   - Primitives
   - The blackboard metaphor
   - Semantics

3. Implementing LINDA in Java
   - Design
   - Tuples & Templates interfaces
   - Text-based Tuple Spaces in Java

Motivation & Lecture Goals

-

Lab 3 Repository on GitLab

- Examples and exercises described in this lecture are provided by means of the following GitLab repository:

  `https://gitlab.com/pika-lab/courses/ds/aa1920/lab-04`

- Clone it on your machine using Git

```
$ git clone <repo URL>
```

- Even if a minimal environment simply relying on a text editor + Gradle is sufficient for this lab, we kindly suggest to import the cloned repository into some IDE, e.g. IntelliJ Idea or Eclipse
  - in case of problems in importing the project on IntelliJ, try to downgrade the gradle wrapper

```
$ ./gradlew wrapper --gradle-version 4.8.1
```

- In order to be able to submit your exercises, please ensure you requested access to the GitLab group of the course

Outline

1. Wetting your appetite

2. Linda Recap
   - Overview
   - Primitives
   - The blackboard metaphor
   - Semantics

3. Implementing LINDA in Java
   - Design
   - Tuples & Templates interfaces
   - Text-based Tuple Spaces in Java

Next in Line. . .

1. Wetting your appetite

2. Linda Recap
   - Overview
   - Primitives
   - The blackboard metaphor
   - Semantics

3. Implementing LINDA in Java
   - Design
   - Tuples & Templates interfaces
   - Text-based Tuple Spaces in Java

The LINDA model

- The LINDA model is built up of five main (sorts of) elements:

Tuples — structured information chunks (such as strings, records, dictionaries, or other kinds of data structures)

Templates — compact notations for expressing sets of tuples adhering to the same pattern (e.g. regular expressions are templates w.r.t. strings). Such patterns express some partial knowledge about one or more tuple

Tuple Spaces — unordered containers (i.e., multisets) of tuples, which may evolve over time since tuples may be added or removed

Agents (or "Processes", or "Activities") — pro-active entities which interact by writing, observing, or taking information from the tuple spaces

Primitives — operations which can be performed by agents on tuple spaces, in order to manipulate or observe the information they contain

Next in Line. . .

1 Wetting your appetite

2 Linda Recap
- Overview
- Primitives
- The blackboard metaphor
- Semantics

3 Implementing Linda in Java
- Design
- Tuples & Templates interfaces
- Text-based Tuple Spaces in Java

LINDA's primitives

- The minimal set of primitives according to the original LINDA's model comprehends the following ones:

  `out` or `write` — let agents insert a tuple into a tuple space

  `rd` or `read` — let agents know if a tuple matching a particular template exists on a tuple space. If it is the case, agents can also read the content of such a tuple

  `in` or `take` — let agents retrieve (or consume) a tuple matching a particular template on a tuple space, if any
- Other primitives will be considered in the future, but for the moment this is all we need
- Think about tuple spaces as collections, about tuples as the the elements contained by such collections, and about primitives as the interface of tuple spaces

Next in Line. . .

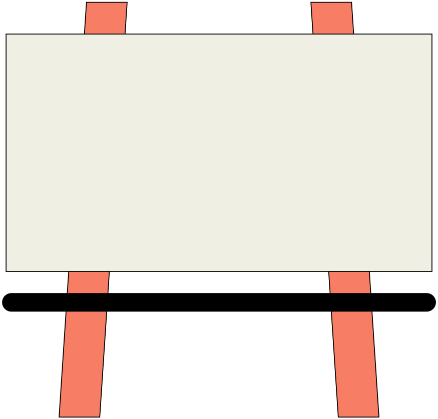
1. Wetting your appetite

2. Linda Recap
   - Overview
   - Primitives
   - The blackboard metaphor
   - Semantics

3. Implementing Linda in Java
   - Design
   - Tuples & Templates interfaces
   - Text-based Tuple Spaces in Java

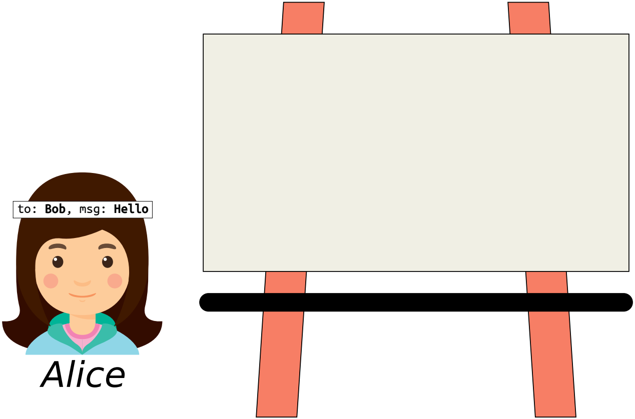
The blackboard metaphor

1. You can imagine a tuple space as a blackboard



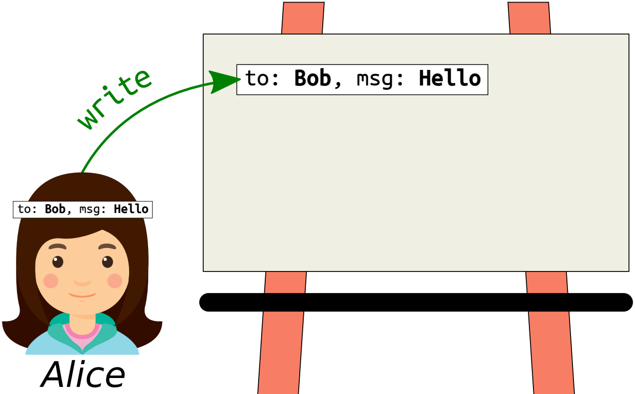
The blackboard metaphor

3. where agents can `write` any sort of information—[i.e.] tuples
    - according to some representation format of choice



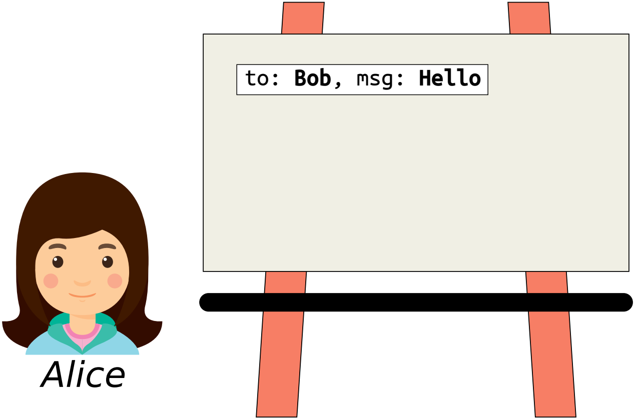
The blackboard metaphor

3. where agents can `write` any sort of information—[i.e.] tuples
   - according to some representation format of choice



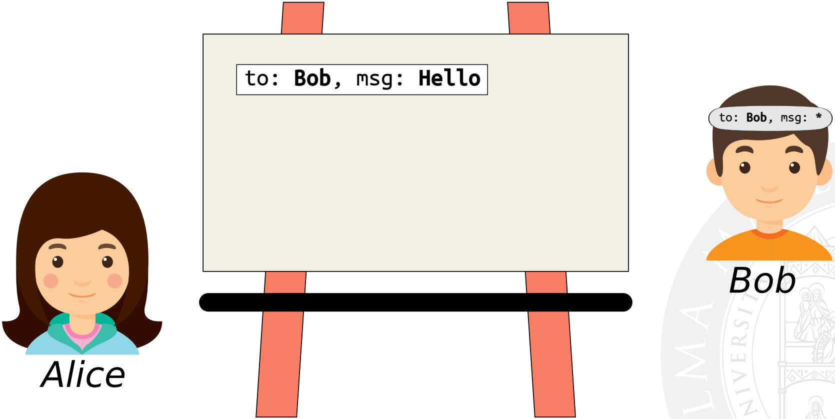
The blackboard metaphor

3. where agents can `write` any sort of information—[i.e.] tuples
   - according to some representation format of choice



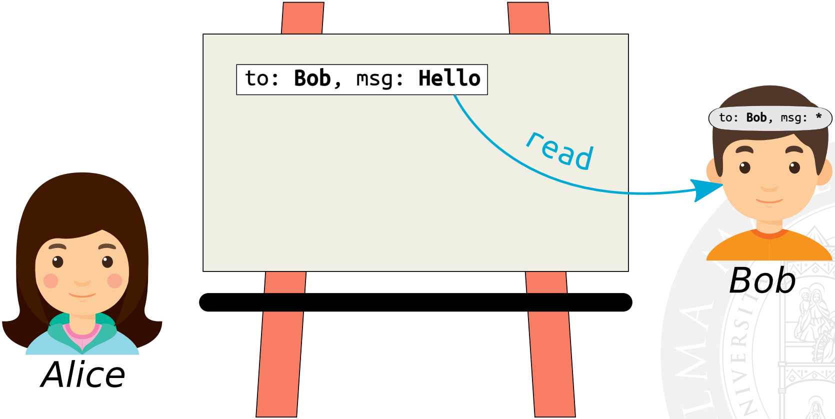
The blackboard metaphor

4. where agents can `read` all information matching a particular template
   - according to some query language of choice



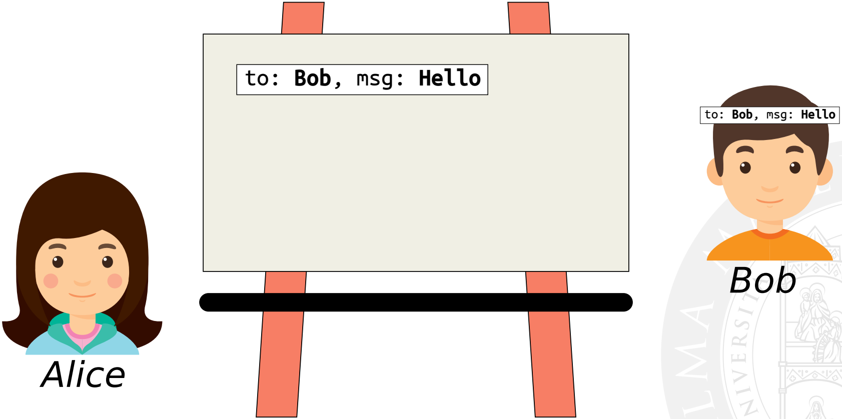
The blackboard metaphor

4. where agents can read all information matching a particular template
   - according to some query language of choice



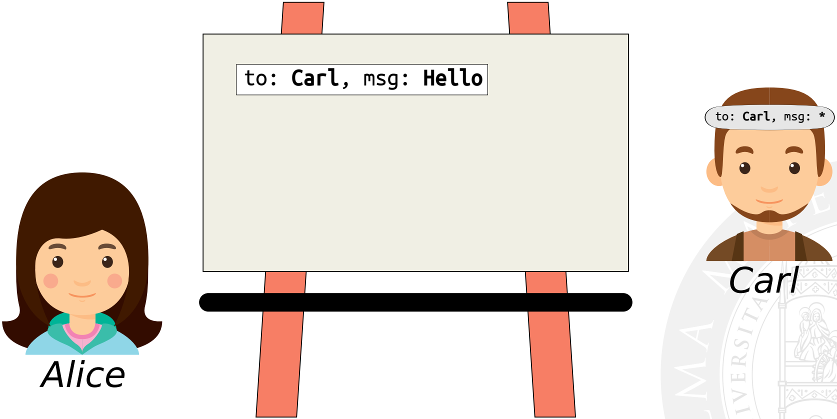
The blackboard metaphor

4. where agents can `read` all information matching a particular template
   - according to some query language of choice



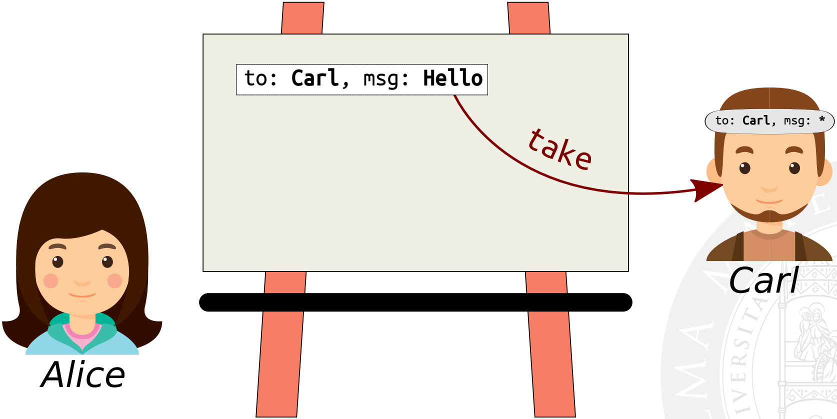
The blackboard metaphor

5. or `take` it, making it unaccessible for other agents



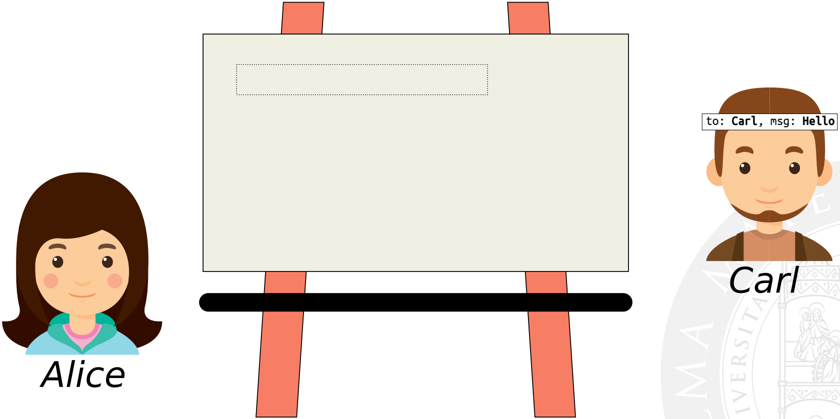
The blackboard metaphor

5. or `take` it, making it unaccessible for other agents



The blackboard metaphor

5. or `take` it, making it unaccessible for other agents



Next in Line. . .

1. Wetting your appetite
2. Linda Recap
   - Overview
   - Primitives
   - The blackboard metaphor
   - Semantics
3. Implementing LINDA in Java
   - Design
   - Tuples & Templates interfaces
   - Text-based Tuple Spaces in Java

(Semi-)Formal notation

In what follows, we will use the following notation:

- Tuples are enumerated by t, or t_i, so, for instance, $t_1 \neq t_2$, but $t_0 = t_0$
- Templates are enumerated by $\bar{t}$, or $\bar{t}_i$. Notice that templates are a compact way to represent sets of tuples
- Matching is written as $t \in \bar{t}$, i.e., tuple t matches template $\bar{t}$
    - unless stated otherwise, we implicitly assume the following notation: tuple t matches template $\bar{t}$, tuple t_1 matches template $\bar{t}_2$, tuple t_i matches template $\bar{t}_i$, and so on ...
- Tuple Spaces are enumerated by TS, or TS_j
- Agents are enumerated by A, or A_k

LINDA's semantics

Generative — after an agent A performs a `write(`t`)` operation on some tuple space TS, tuple t exists regardless of A. If agent A terminates, crashes, or disconnects, t will keep existing on TS

Associative — tuples are accessed (read or taken) in an associative way: instead of using a name, or an address, agents can specify templates in order to access tuples

Suspensive — whenever an agent A invokes the `read(`$\bar{t}$`)` or `take(`$\bar{t}$`)` over a particular template $\bar{t}$, on a particular tuple space TS, if not tuple t matching $\bar{t}$ exists on TS, the operation is suspended until t is inserted into TS by some agent performing a `write(`t`)` operation

Non-deterministic — whenever an agent A invokes the `read(`$\bar{t}$`)` or `take(`$\bar{t}$`)` operation, if more than one tuple t, t', t'' exist matching $\bar{t}$, one is retrieved non-deterministically

Outline

1. Wetting your appetite

2. Linda Recap
   - Overview
   - Primitives
   - The blackboard metaphor
   - Semantics

3. Implementing LINDA in Java
   - Design
   - Tuples & Templates interfaces
   - Text-based Tuple Spaces in Java

Next in Line. . .

1 Wetting your appetite

2 Linda Recap
- Overview
- Primitives
- The blackboard metaphor
- Semantics

3 Implementing LINDA in Java
- Design
- Tuples & Templates interfaces
- Text-based Tuple Spaces in Java

LINDA as a Java interface

- A tuple space can be easily conceived as an object in the OOP sense
- But how should control-flow related aspects be faces?

```
import java.util.concurrent.Future;
import org.apache.commons.collections4.MultiSet;

interface TupleSpace<T extends Tuple, TT extends Template> {

  Future<T> read(TT template);
  Future<T> take(TT template);
  Future<T> write(T tuple);
//       vvvvvvvv collection type provided by Apache
  Future<MultiSet<? extends T>> get();     // Utility primitive
  Future<Integer> getSize();               // Utility primitive

  String getName();   // To discriminate among several tuple spaces
}
```

- Where Future<X> is the return type for asynchronous operations
  - its functioning will be clear in a few slides

Next in Line. . .

1 Wetting your appetite

2 Linda Recap
- Overview
- Primitives
- The blackboard metaphor
- Semantics

3 Implementing LINDA in Java
- Design
- Tuples & Templates interfaces
- Text-based Tuple Spaces in Java

Where Tuples and and Templates are simply:

```
interface Tuple {
  // Just a tag interface
}
```

```
interface Template {
  boolean matches(Tuple tuple);
}
```

- Tuples may be potentially anything
- Templates may be anything able to match a tuple, somehow

Of course, this is just an abstract model

How should we actually implement it?

Next in Line. . .

1. Wetting your appetite

2. Linda Recap
   - Overview
   - Primitives
   - The blackboard metaphor
   - Semantics

3. Implementing LINDA in Java
   - Design
   - Tuples & Templates interfaces
   - Text-based Tuple Spaces in Java

Text-based Tuple Spaces in Java, idea

We will implement the `TupleSpace` interface by means of the `TextTupleSpace` class, where:

- Strings are used as tuples, meaning that the `java.lang.String` class is used to reify tuples

  i.e. `T = String`

- Regular Expressions (regex) are used as templates, meaning that the `java.util.regex.Pattern` class is used to reify templates

  i.e. `TT = Pattern`

- The matching consists of deciding whether a string matches a regex

! If you are not practical with regex, you can acquire some experience or simply test your patterns with `https://regex101.com`

String as Tuples

```java
class StringTuple implements Tuple {
  private final String value;

  public StringTuple(String value) { this.value = value; }

  public String getValue() { return value; }

  // @Override public boolean equals(final Object obj) { /*...*/ }

  // @Override public int hashCode() { /*...*/ }

  @Override public String toString() {
    return "\"" + value + "\"";
  }
}
```

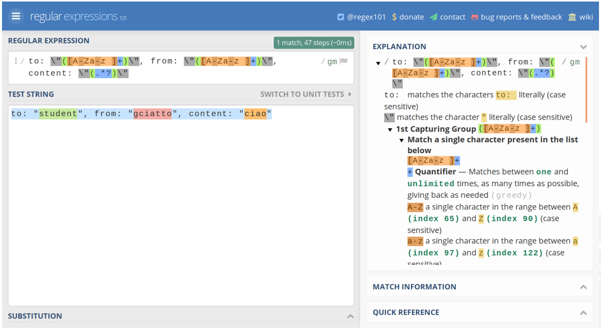
Regex as Templates

```java
class RegexTemplate implements Template {
  private final Pattern regex;

  public RegexTemplate(final String regex) {
    Objects.requireNonNull(regex);
    this.regex = Pattern.compile(regex, Pattern.MULTILINE);
  }

  @Override public boolean matches(final Tuple tuple) {
    if (tuple instanceof StringTuple) {
      StringTuple casted = (StringTuple)tuple;
      return regex.matcher(casted.getValue()).matches();
    }
    return false;
  }

  @Override public String toString() {
    return "/" + regex.pattern() + "/";
  }
}
```

Regex101.com – Example

- Could you say what's the meaning of the following regex?

  `to: \"([A-Za-z ]+)\", from: \"([A-Za-z ]+)\", content: \"(.*?)\"`



REGULAR EXPRESSION

1 match, 47 steps (~0ms)

```
/ to: \"([A-Za-z ]+)\", from: \"([A-Za-z ]+)\", content: \"(.*?)\" / gm
```

TEST STRING

```
to: "student", from: "gciatto", content: "ciao"
```

EXPLANATION

```
/ to: \"([A-Za-z ]+)\", from: \"([A-Za-z ]+)\", content: \"(.*?)\" / gm
```

`to: ` matches the characters `to: ` literally (case sensitive)

`\"` matches the character `"` literally (case sensitive)

- **1st Capturing Group** `([A-Za-z ]+)`
  - **Match a single character present in the list below** `[A-Za-z ]+`

    `+` **Quantifier** — Matches between `one` and `unlimited` times, as many times as possible, giving back as needed `(greedy)`

    `A-Z` a single character in the range between `A` `(index 65)` and `Z` `(index 90)` (case sensitive)

    `a-z` a single character in the range between `a` `(index 97)` and `z` `(index 122)` (case sensitive)

- `https://regex101.com` provides an interactive explaination of your regex (on the right side)
- you can test your regex on the fly against any input string

Next in Line. . .

1. Wetting your appetite
2. Linda Recap
   - Overview
   - Primitives
   - The blackboard metaphor
   - Semantics
3. Implementing LINDA in Java
   - Design
   - Tuples & Templates interfaces
   - Text-based Tuple Spaces in Java

Exercise 3-1: Text-based Tuple Spaces in Java I

1. Clone the initial source code from `https://gitlab.com/das-lab/courses/ds/aa1819/lab-3`
2. Import the project into your favourite IDE as a Gradle project
3. Inspect the project and try to figure out the purpose of the provided code
4. Notice that the project's `build.gradle` file comes with some dependencies. Try to figure out what they are and what's their purpose
5. Notice that the project comes equipped with some tests. Read them and try to understand them

Exercise 3-1: Text-based Tuple Spaces in Java II

```java
class TextTupleSpace
    implements TupleSpace<StringTuple, RegexTemplate> {

  private String name;
  private ExecutorService executor;
  private MultiSet<StringTuple> tuples = new HashMultiSet<>();
  private MultiSet<PendingRequest> pendings = new HashMultiSet<>();

  public TextTupleSpace(String name, ExecutorService executor) {
    this.name = Objects.requireNonNull(name);
    this.executor = Objects.requireNonNull(executor);
  }

  public Future<StringTuple> read(RegexTemplate template) {
    // TODO: implement
  }

  public Future<StringTuple> take(RegexTemplate template) {
    // TODO: implement
  }
```

Exercise 3-1: Text-based Tuple Spaces in Java III

```
  public Future<StringTuple> write(StringTuple tuple) {
    // TODO: implement
  }

  public Future<Integer> getSize() {
    // TODO: implement
  }

  public Future<MultiSet<? extends T>> get() {
    // (Optional) TODO: implement
  }

  public String getName() { return name; }


  private enum RequestTypes { READ, TAKE; }
  private static class PendingRequest { /* ... */ }
}
```

Exercise 3-1: Text-based Tuple Spaces in Java IV

Your solution must satisfy the following constraints:

- The unit tests contained within the `TestTextTupleSpace` class must be satisfied
  - use them usage as examples
  - consider looking at the other `Test*` classes, in order to understand how Executor Services or Futures work
- You shouldn't need any threads, just use Executor Services and Active Objects (to act as Agents)
- The tuple space must work regardless of the particular Executor Service it is initialised with
- The provided implementation must adhere to the Linda semantics described on slide 23

! If you feel confident with these concepts you can start your exercise now. Otherwise, just wait for the teacher's tutorial

Exercise 3-1: Text-based Tuple Spaces in Java V

While solving your exercise on your branch:

It is strictly forbidden

- to alter, remove, ignore, or comment the `.gitlab-ci.yml` file on your branch
- to alter, remove, ignore, or comment the provided test classes

! submissions subject to such kind of problems will be considered late

If you understand some test is faulty or ill-constructed

- you must post the information on the forum as soon as possible

Building LINDA from scratch

Distributed Systems / Hands-on

Sistemi Distribuiti / Laboratorio

Giovanni Ciatto Andrea Omicini

giovanni.ciatto@unibo.it andrea.omicini@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
ALMA MATER STUDIORUM – Università di Bologna a Cesena

Academic Year 2019/2020