

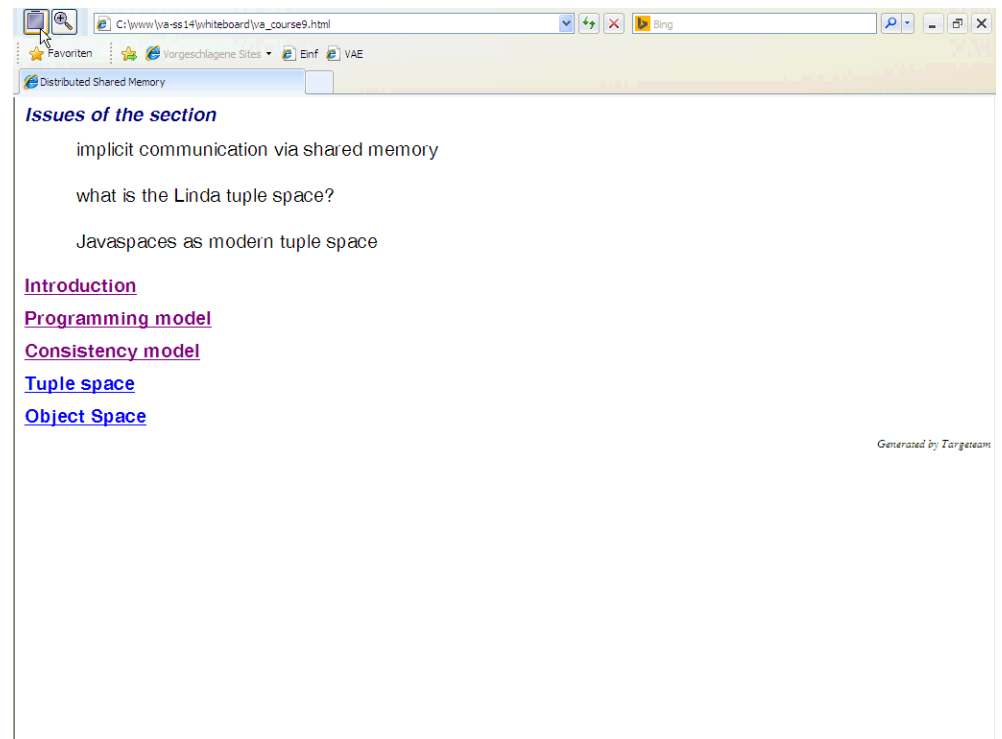
Script generated by TTT

Title: Distributed_Applications (07.07.2014)

Date: Mon Jul 07 09:15:51 CEST 2014

Duration: 47:04 min

Pages: 14



The screenshot shows a web browser window with the address bar containing 'C:\www\lva-ss14\whiteboard\lva_course9.html'. The browser has several tabs, including 'Distributed Shared Memory'. The main content area displays a slide with the following text:

Issues of the section

- implicit communication via shared memory
- what is the Linda tuple space?
- Javaspaces as modern tuple space

Below the list are several hyperlinks: [Introduction](#), [Programming model](#), [Consistency model](#), [Tuple space](#), and [Object Space](#). The text 'Generated by Targeteam' is visible in the bottom right corner of the slide.



Tuple space



The tuple space was invented by Gelernter (Yale University) as an object-oriented approach to managing distributed data. It was specially designed for Linda language.

Tuple space consists of a set of tuples that could be interpreted as lists of typed fields.

A tuple space has the following basic characteristics:

it is based on the shared-memory model.

tuples represent information, e.g. ("Linda", 3).

[Atomic operations](#)

[Tuple space implementation](#)

[Example for client-server communication](#)

Generated by Targeteam



Atomic operations



Tuple space supports read and write operations on the shared memory.

1. Operations on a tuple t

out (t): creates a new tuple t in the tuple space.

in (t): reads and simultaneously removes a tuple from the tuple space.

read (t): reads a tuple; t remains in the tuple space and subsequent operations can refer to it.

2. Read access is associative, e.g. **in** ("order", ?i, ?j).

3. **in**, **read** are synchronous.

4. **inp**, **readp** are asynchronous.

5. Generation of new processes: **eval** (t).

Generated by Targeteam



Implementation alternatives

1. central tuple space.
2. replicated tuple space,
each computer maintains a complete copy of the tuple space.
3. distributed tuple space; division into subspaces
each computer owns part of the tuple space; **out** operations are executed locally.



Generated by Targeteam



The tuple space was invented by Gelernter (Yale University) as an object-oriented approach to managing distributed data. It was specially designed for Linda language.

Tuple space consists of a set of tuples that could be interpreted as lists of typed fields.

A tuple space has the following basic characteristics:

it is based on the shared-memory model.

tuples represent information, e.g. ("Linda", 3).

[Atomic operations](#)

[Tuple space implementation](#)

[Example for client-server communication](#)

Generated by Targeteam



The following simple example shows how the client-server style of communication could be programmed in the tuple space model.

[/* Client */](#)

[/* Server */](#)

Generated by Targeteam



The JavaSpaces programming interface is simple; a space provides the following key features.

Objects in a space are passive.

processes do not manipulate objects directly in the space.

processes do not invoke methods of objects in the space.

Spaces are **shared**: they represent a network-accessible memory that many remote processes can interact with concurrently.

Spaces are **persistent**: objects are stored until a process explicitly removes them or until their **lease** time expires.

Spaces are **associative**: objects are accessed via associative lookup, rather than by identifier or by memory address.

Spaces are transaction oriented: access operations to the space are atomic.

Spaces support the exchange of executable code.

Generated by Targeteam



Objects in a space are realized via the `Entry` interface (net.jini.core.entry package).

Interface Definition

```
public interface Entry extends java.io.Serializable {
    // this interface is empty
}
```

Example of an object representing a shared variable in the distributed system

```
public class SharedVar implements Entry {
    public String name;
    public Integer value;
    public SharedVar() {
    }
    public SharedVar(String name, int value) {
        this.name = name;
        this.value = new Integer(value);
    }
}
```

Instantiation of a shared variable within a process

```
SharedVar global_counter = new SharedVar("counter", 0)
```

Generated by Targeseam

The shared space is identified via the method `getSpace` of the `SpaceAccessor` class.

```
JavaSpace space = SpaceAccessor.getSpace();
```

Access to the space identifier; there are two options

the space is registered as Jini service, i.e. Jini lookup services may be used.

the space is registered in the [RMI](#) registry.

Generated by Targeseam



Overview

- read
- take, i.e. read and remove
- write
- notify, i.e. inform the process when an entry matching the given pattern has arrived.

[Write - operation](#)

[Read and take - operation](#)

[Matching rules](#)

[Atomicity](#)

Generated by Targeseam

The methods `read` and `take` access an object in a space. `read` copies the object into the local process environment while `take` removes it from the space.

For remote access, a process needs a template. A template is a kind of entry: containing some specified and some empty fields (i.e. the value `null`).

matching associatively the relevant objects in the space.

If several objects in the space match the template, then an object is selected at random.

Example

```
SharedVar template = new SharedVar("counter");
SharedVar result = (SharedVar) space.take(template, null, Long.MAX_VALUE)
```

The `take` operation waits until there is a suitable entry in the space available.

Generated by Targeseam



Overview

- read
- take, i.e. read and remove
- write
- notify, i.e. inform the process when an entry matching the given pattern has arrived.

[Write - operation](#)

[Read and take - operation](#)

[Matching rules](#)

[Atomicity](#)

Generated by Targeteam



Overview

- read
- take, i.e. read and remove
- write
- notify, i.e. inform the process when an entry matching the given pattern has arrived.

[Write - operation](#)

[Read and take - operation](#)

[Matching rules](#)

[Atomicity](#)

Generated by Targeteam



Overview

- read
- take, i.e. read and remove
- write
- notify, i.e. inform the process when an entry matching the given pattern has arrived.

[Write - operation](#)

[Read and take - operation](#)

[Matching rules](#)

[Atomicity](#)

Generated by Targeteam