Script **generated by TTT**

Title: Distributed_Applications (16.07.2012)

Date: Mon Jul 16 09:15:51 CEST 2012

Duration: 47:43 min

Pages: 14

Distributed Shared Memory

Issues of the section

implicit communication via shared memory

what is the Linda tuple space?

Javaspaces as modern tuple space

Consistency model

The content of DSM may be replicated by caching it at the separate computers;

- data is read from the local replica.
- updates have to be propagated to the other replicas of the shared memory.

Approaches to keep the replicas consistent

Write-update

- updates are made locally and multicast to all replicas possessing a copy of the data item.
  - the remote data items are modified immediately.

Write-invalidate

- before an update takes place, a multicast message is sent to all copies to invalidate them;
  - acknowledgement by the remote sites before the write can take place.
  - other processes are prevented to access the blocked data item.
  - the update is propagated to all copies, and the blocking is removed.

Distributed Shared Memory

Issues of the section

implicit communication via shared memory

what is the Linda tuple space?

Javaspaces as modern tuple space

Introduction
Programming model
Consistency model
Tuple space
Object Space

Generated by Targeteam

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).

Tuple space implementation

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.

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

Features of JavaSpaces

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.

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 Targeteam

Object Space

object space for sharing and exchanging objects between components of a distributed application

JavaSpaces supports an object space.

based on the Linda tuple concept.

Tuples are references to Java objects

Generated by Targeteam

Write - operation

```
Lease write (Entry e, Transaction txn, long lease) throws RemoteException,
TransactionException
```

Parameter semantics

Entry e is entered into the space; e is transmitted, as well as stored, in a serialized form in the space.

Transaction txn allows to group several operations to a transaction; the parameter value ***null*** represents a transaction with only one operation.

long lease specifies how long the entry e is to be stored in the space before the space automatically removes the entry e.

The result Lease specifies how long the space will store the entry e.

Write can trigger the exceptions RemoteException (communication problems) and TransactionException (transaction txn not valid).

Example

```
space.write(global_counter, null, Lease.FOREVER);
```

Generated by Targeteam

Read and take - operation

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 Targeteam

Matching rules

An access template matches an object in the space if the following rules hold true

- the template class matches the object class, or else the template class is a super class of the entry's class.
- if a template field has a wildcard (null), then it matches the corresponding object field.
- if a template field is specified, then it matches the object's corresponding field if the two values are the same.

Example Java Spaces

A process is notified when a new message is deposited in the object space. The process retrieves the new message from the object space.

Message Entry

```
import net.jini.core.entry.Entry;
public class Message implements Entry {
    public String content;
    public Message() { }
}
```

Listener

```
import java.rmi.server.*;
import java.rmi.RemoteException;
import net.jini.core.event.*;
import net.jini.space.JavaSpace;

public class Listener implements RemoteEventListener {
    private JavaSpace space;
    public Listener(JavaSpace space) throws RemoteException {
       this.space = space;
       UnicastRemoteObject.exportObject(this);
    }

    public void notify(RemoteEvent ev) {
```

Example Java Spaces

```
       }
    }
```

HelloWorld

```
import jsbook.util.SpaceAccessor;
import net.jini.core.lease.Lease;
import net.jini.space.JavaSpace;

public class HelloWorldNotify {
    public static void main(String[] args) {
       JavaSpace space = SpaceAccessor.getSpace();
       try {
          Listener listener = new Listener(space);
          Message template = new Message();
          space.notify(template, null, listener, Lease.FOREVER, null);
          Message msg = new Message();
          msg.content = "Hello World";
          space.write(msg, null, Lease.FOREVER);
       } catch (Exception e) { e.printStackTrace(); }
    }
}
```