Script **generated by TTT**

Title: Distributed_Applications (16.07.2013)

Date: Tue Jul 16 14:30:50 CEST 2013

Duration: 83:12 min

Pages: 35

Issues of the section

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

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

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).

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.

Entry interface

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)
```


SpaceAccessor

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.

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);
```


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.

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.

Basic operations

Overview

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

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

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(); }
    }
}
```


Example Java Spaces

```
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) {
        Message template = new Message();
        try {
            Message result =
                (Message)space.read(template, null, Long.MAX_VALUE);
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```

HelloWorld

Object Space

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

JavaSpaces supports an object space.

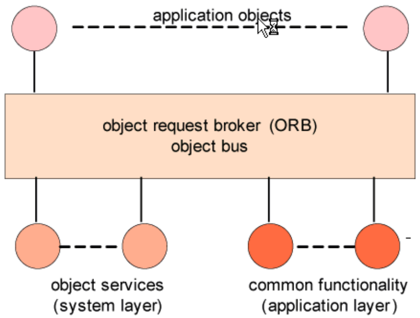
based on the Linda tuple concept.

Tuples are references to Java objects

Object Management Architecture - OMA

The architecture is also referred to as CORBA ("Common Object Request Broker Architecture").

OMA is a possible middleware for object-oriented distributed applications.



ORB supports the communication among the objects through a request/reply protocol.

ORB includes object localization, message delivery, method binding, parameter marshalling, and synchronization of request and reply messages.

ORB itself does not execute methods. Rather, it mediates between application objects, service objects, and shared functionalities of the application layer ("application frameworks").

Object-based Distributed Systems

The OMG (Object Management Group) was founded in 1989 by a number of companies to encourage the adoption of ***distributed object systems*** and to enable ***interoperability*** for heterogeneous environments (hardware, networks, operating systems and programming languages).

Object Management Architecture - OMA

Object Request Brokers ORB

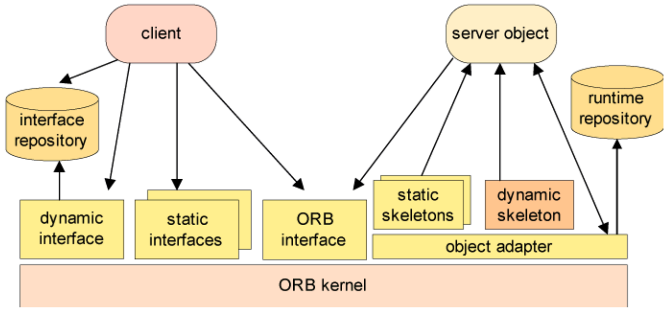
Common object services

Inter-ORB protocol

Distributed COM

.NET Framework

Structure of ORB



ORB components

Embedding in distributed Applications

ORB components

- ORB core (kernel): mediates requests between client and server objects; handles the network communication within the distributed system.

  operations to convert between remote object references and strings.

  operations to provide argument lists for requests using dynamic invocation.

- Static invocation interface

  at compile time, operations and parameters are determined.

  an object class may have several different static interfaces.

- Dynamic invocation interface

  Procedures and parameters are determined at runtime; the interface is identical for all ORB implementations, i.e. there is ***only one*** dynamic invocation interface.

- ORB interface

  supports ORB service calls, e.g. conversion of object references to strings and vice versa; the interface is determined by the ORB.

- Interface repository

  stores at runtime the signatures of the available methods; the signatures are described by the IDL notation; in case of the dynamic invocation interface a lookup within the interface repository is performed.

- Object adapter: bridges the gap between Corba objects with IDL interfaces and the programming language interfaces of the server class.

ORB components

  at compile time, operations and parameters are determined.

  an object class may have several different static interfaces.

- Dynamic invocation interface

  Procedures and parameters are determined at runtime; the interface is identical for all ORB implementations, i.e. there is ***only one*** dynamic invocation interface.

- ORB interface

  supports ORB service calls, e.g. conversion of object references to strings and vice versa; the interface is determined by the ORB.

- Interface repository

  stores at runtime the signatures of the available methods; the signatures are described by the IDL notation; in case of the dynamic invocation interface a lookup within the interface repository is performed.

- Object adapter: bridges the gap between Corba objects with IDL interfaces and the programming language interfaces of the server class.

  forwards client calls to the appropriate server object using the skeletons.

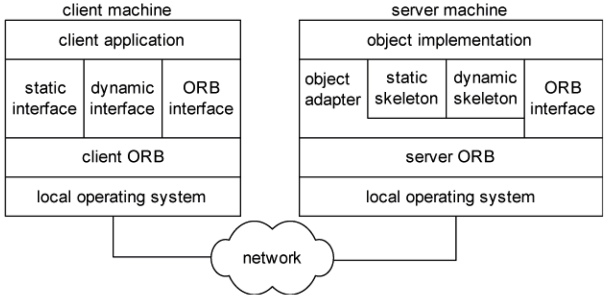
  defines a runtime environment for initialization of server objects and assignment of object identifiers.

  activates objects.

- Runtime repository

Embedding in distributed Applications

Usually the ORB is embedded as a library function.



- ORBIX by Progress (former Iona Technologies).

  available as libraries: client and server library.

  based on TCP/IP transport mechanism

- the TAO system by Doug Schmidt is an implementation of the Corba model.

  exists as a free platform and as a commercial product.

Common object services

A collection of system level services which can be utilized by the application objects; they are extending the ORB functionality.

Life-cycle Service : defines operations for object creation, copying, migration and deletion.

Persistence Service : provides an interface for persistent object storage, e.g. in relational or object-oriented databases.

Name Service : allows objects on the object bus to locate other objects by name; integrates existing network directory services, e.g. OSF's DCE, LDAP or X.500.

Event Service : register the interest in specific events; producer and consumer of events need not know each other.

Concurrency Control Service : provides a lock manager.

Transaction Service : supports 2-phase commit coordination for flat and nested transactions.

Relationship Service : supports the dynamic creation of relations between objects that know nothing of each other; the service supports navigation along these links, as well as mechanisms for enforcing referential integrity constraints.

Query Service : supports SQL operations for objects.

Inter-ORB protocol

Communication between ORBs is based on GIOP ("General Inter-ORB Protocol").

GIOP Features

External data representation

Object reference

GIOP message

Example for IIOP use

RMI over IIOP

External data representation

Distinction between primitive and complex data types, so-called typeCodes; assignment of integer values to identify data types

primitive : char, octet, short, long, float, double, boolean

complex : struct, union, sequence, symbol chains, fields

The format of complex data types is described in the interface repository.

Example

```
tk_struct (Typecode struct):
    string : repository_ID
    string : name
    ulong: count
       {string : member name
       TypeCode: membertype }
```

Inter-ORB protocol

Communication between ORBs is based on GIOP ("General Inter-ORB Protocol").

GIOP Features

External data representation

Object reference

GIOP message

Example for IIOP use

RMI over IIOP

GIOP message head

The GIOP message head has the same format for all message types; it identifies the message type sent to another ORB.

GIOP message head structure

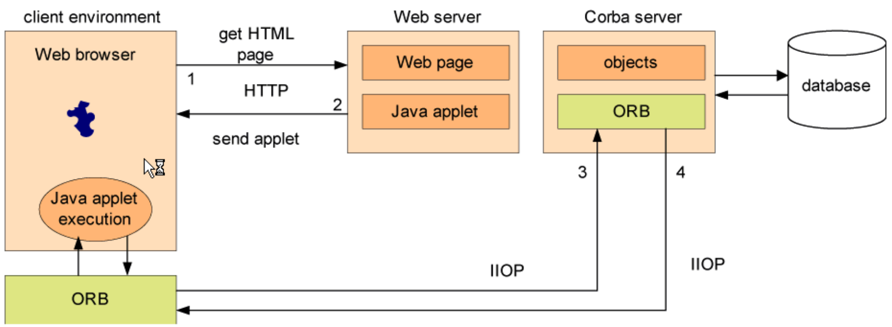
```
module GIOP
    struct Version {octet: major; octet: minor};
    enum MsgType {Request, Reply, CancelRequest, LocateRequest,
    LocateReply, CloseConnection, MessageError, Fragment}
    struct Message_Header {
        char magic[4]; this is the string "GIOP"
        Version GIOP_version;
        octet flag
        octet message_type
        usigned long message_size
    }
```

the component `flags` determines the used byte ordering (big/little endian) and whether or not the entire message has been divided into several fragments.

`message_type` is an element of the MsgType enumeration; it identifies the message type.

Example for IIOP use

Web access of a database using a Java applet and Corba.



Inter-ORB protocol

Communication between ORBs is based on GIOP ("General Inter-ORB Protocol").

GIOP Features

External data representation

Object reference

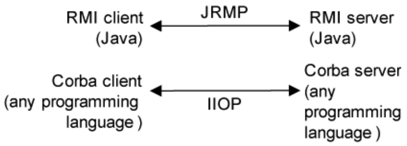
GIOP message

Example for IIOP use

RMI over IIOP

RMI over IIOP

RMI uses JRMP (Java Remote Method Protocol) for the communication between client and server objects, i.e. there is no interoperability with Corba.



Extension of RMI to RMI-IIOP

.NET Framework

The Microsoft .NET Framework is a software framework available with Windows OS for building distributed applications.

- represents a strategy change from the product-oriented desktop world to the service-oriented component world.
- goal: many future Windows applications should be built using .NET.

.NET has 2 core elements: Common Language Runtime (CLR) and the Framework Class Library.

Common Language Runtime (CLR)

Frame Class Library

.NET-Remoting

technology for remote method invocation provided by the framework.

- relevant classes are in the namespace `System.Runtime.Remoting`.
- support of different transport protocols, e.g. TCP (binary formatting) or HTTP (SOAP formatting).
- activation of remote objects.

Common Language Runtime (CLR)

provides a runtime environment for applications which may be developed in different languages, e.g. C#, C++, Java, Perl or Python. CLR supports the following services

- memory management.
- thread management.
- libraries encapsulate access to OS functions.
- common intermediate Language (MSIL).

All .NET programs execute under the supervision of the CLR.

Common Type System (CTS)

CTS defines all possible datatypes and programming constructs supported by the CLR

- data structures are uniformly interpreted on the MSIL layer.
- enables interoperability between the languages supported by .NET

.NET Framework

The Microsoft .NET Framework is a software framework available with Windows OS for building distributed applications.

- represents a strategy change from the product-oriented desktop world to the service-oriented component world.
- goal: many future Windows applications should be built using .NET.

.NET has 2 core elements: Common Language Runtime (CLR) and the Framework Class Library.

Common Language Runtime (CLR)

Frame Class Library

.NET-Remoting

technology for remote method invocation provided by the framework.

- relevant classes are in the namespace `System.Runtime.Remoting`.
- support of different transport protocols, e.g. TCP (binary formatting) or HTTP (SOAP formatting).
- activation of remote objects.

Generated by Targeteam