Title: Distributed_Applications (16.07.2013)
Date: Tue Jul 16 14:30:50 CEST 2013
Duration: 83:12 min
Pages: 35

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.
**Tuple space**

A tuple space is a memory region that contains tuples, which are collections of heterogeneous data. A tuple space is a nonpersistent, shared memory region accessed by processes to store and retrieve data.

**Tuple space implementation**

Tuple space is implemented as a process communication mechanism in distributed systems. It enables processes to share and access data concurrently. Processes can read, write, and synchronize on data stored in the tuple space. The tuple space acts as a communication channel between processes, allowing them to exchange data without explicitly passing messages.

**Features of TupleSpaces**

- **Non-persistent**: Tuples are not stored persistently, making tuple space memory transient.
- **Asynchronous**: Tuples can be accessed by multiple processes simultaneously, ensuring data availability for concurrent access.
- **Dynamic**: Tuples can be added or removed from the tuple space dynamically, allowing for flexible data management.

**Examples of communication**

- Processes can read tuples from the tuple space and update them simultaneously.
- Processes can synchronize on tuple values, ensuring mutual exclusion and data consistency.

**Implementation**

The implementation details of tuple space typically involve managing tuple allocation, deletion, and synchronization. Efficient tuple space management is crucial for performance and scalability in distributed systems.
Objects in a space are realized via the Entry interface (net.jini.core.entry package).

**Interface Definition**

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

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

```java
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

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

---

**Write - operation**

```java
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 `null` not valid).

**Example**

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

---

**Read and take - operation**

```java
space.take(SharedVar.class, lease, null, null);
```

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**

```java
SharedVar template = new SharedVar("counter");
SharedVar result = space.take(template, lease, null, Lease.FOREVER);
```

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

---

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

```java
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 Torquejava
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.

Write - operation
Read and take - operation
Matching rules
Atomicity

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

Listener
```java
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) {
```
```java
```
HelloWorld

import javax.rmi.util.SpaceAccessor;
import net.jini.core.lease.Lease;
import net.jini.space.JavaSpace;

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

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(); }
        }
    }
}
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**

**ORB components**

**Embedding in distributed Applications**

- ORB kernel
  - client
  - server object
  - runtime repository

**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.

- Runtime repository
  - 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.

- Runtime repository
Usually the ORB is embedded as a library function.

```
<table>
<thead>
<tr>
<th>client machine</th>
<th>server machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>client application</td>
<td>object implementation</td>
</tr>
<tr>
<td>static interface</td>
<td>object adapter</td>
</tr>
<tr>
<td>dynamic interface</td>
<td>static skeleton</td>
</tr>
<tr>
<td>ORB interface</td>
<td>dynamic skeleton</td>
</tr>
<tr>
<td>client ORB</td>
<td>ORB interface</td>
</tr>
<tr>
<td>local operating system</td>
<td>local operating system</td>
</tr>
</tbody>
</table>
```

- ORBIX by [Progress](https://www.progress.com) (former Iona Technologies).
- Available as libraries: client and server library.
- Based on TCP/IP transport mechanism.

- The TAO system by [Doug Schmidt](https://www.cs.wisc.edu/~schmidt) is an implementation of the Corba model.
- Exists as a free platform and as a commercial product.

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. CSF’s DOE, LDAP or X.500.
- **Event Service**: registers 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.

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

```
tk_struct (Typecode struct):

  string : repository_ID
  string : name
  ulong : count

  {string : member name
   TypeCode : membertype }
```
Communication between ORBs is based on GIOP ("General Inter-ORB Protocol").

**GIOP Features**

- External data representation
- Object referencing
- GIOP messages
- Example for IIOP use
- RMI over IIOP

**GIOP Message Header**

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

**GIOP Message Header Structure**

```c
module GIOP

struct Version {octet: major; octet: minor;}
enum MessageType {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
    unsigned long message_size
}
```

The component `flag` 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 `MessageType` enumeration; it identifies the message type.

---

Example for IIOP use

Web access of a database using a Java applet and Corba

1. Get HTML page
2. HTTP
3. Send applet
4. ORB

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

**GIOP Features**

- External data representation
- Object referencing
- GIOP messages
- 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.

Corba client (any programming language) \rightarrow \text{IIOP} \rightarrow \text{Corba server (any programming language)}

Extension of RMI to RMI-IIOP

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.
- It is the 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)

- 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 Remoting

- Technology for remote method invocation provided by the framework.
- Relevant classes are in the namespace \texttt{System.Runtime.Remoting}.
- Support of different transport protocols, e.g., TCP (binary formatting) or HTTP (SOAP formatting).
- Activation of remote objects.