Script **generated by TTT**

Title: Distributed_Applications (08.07.2014)

Date: Tue Jul 08 14:31:14 CEST 2014

Duration: 76:33 min

Pages: 31

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

Introduction

Features of JavaSpaces

Data structures

- Entry interface
- SpaceAccessor

Basic operations

Events

Example Java Spaces

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

Introduction

Features of JavaSpaces

Data structures

- Entry interface
- SpaceAccessor

Basic operations

Events

Example Java Spaces

Generated by Targeteam

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(); }
    }
}
```

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

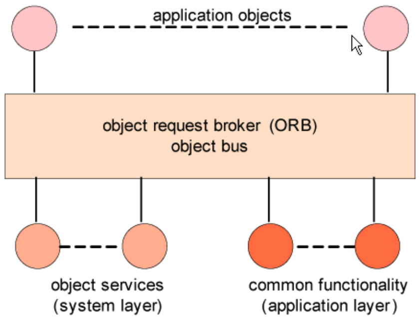
Object-based Distributed Systems

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

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

Generated by Targeteam

General features

ORB supports the following general characteristics

1. static and dynamic invocation of object methods

   static : method interface is determined at compilation time.

   dynamic : method interface is determined at runtime.

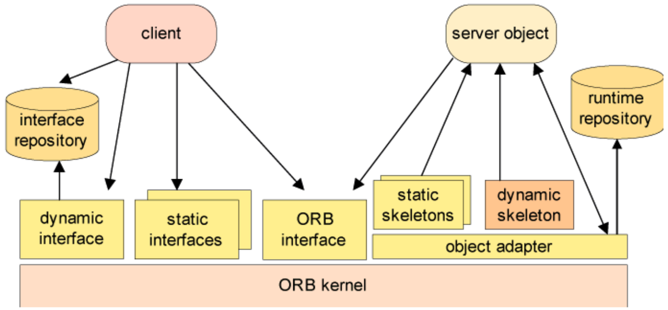
2. interfaces for higher programming language, e.g. C++, Smalltalk, Java.
3. a self-descriptive system.
4. location transparency.
5. security checks, e.g. object authentication.
6. polymorphic method invocation, i.e. the execution of the method depends on the specific object instance. Difference between RPC and ORB

   RPC calls a specific server function; data are separated.

   ORB calls the method of a specific object.

7. hierarchical object naming.

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

- 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.

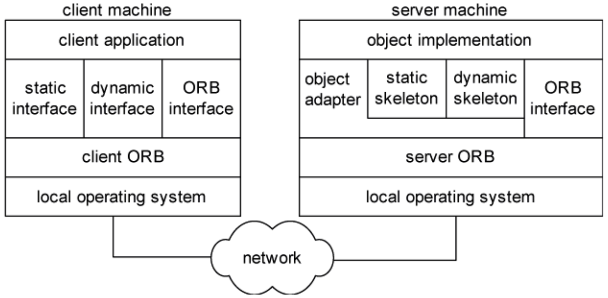
- Interface repository

  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.

Embedding in distributed Applications

Usually the ORB is embedded as a library function.



- ORBIX by Microfocus .

    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.

GIOP message types

GIOP supports the following message types:

1. ***Request*** : request the execution of an operation at the remote object, e.g. access of an attribute; the message contains the call parameters.
2. ***Reply*** : answer to a request message.
3. ***CancelRequest*** : termination of a request; the calling ORB does not expect an answer to the original request.
4. ***LocateRequest*** : is used to determine

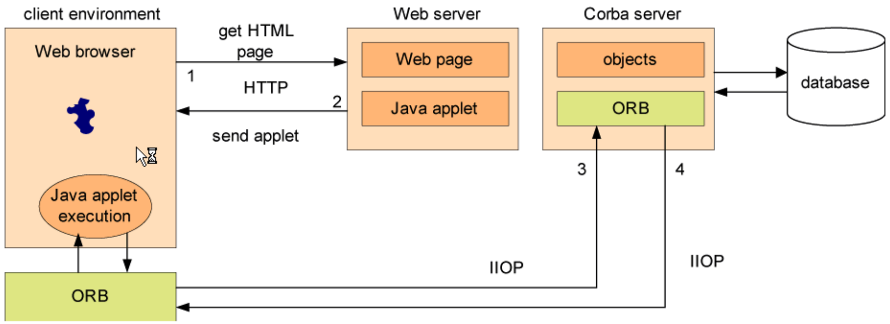
    whether the given object reference is valid, or

    whether the destination ORB processes the object reference, and if not, to which address requests for the object reference are to be sent.

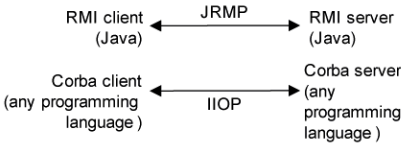
5. ***LocateReply*** : answer to LocateRequest
6. ***CloseConnection*** : the destination ORB notifies the calling ORB that it closes the connection.
7. ***MessageError*** : exchange of error information.
8. ***Fragment*** : if, for instance, the request consists of several parts, then first a request message is sent, and then the remaining parts are sent using fragment messages.

Example for IIOP use

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



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

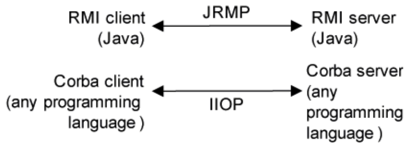


[Extension of RMI to RMI-IIOP]

Generated by Targeteam

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.


RMI client
(Java)
JRMP
RMI server
(Java)
Corba client
(any programming
language)
IIOP
Corba server
(any
programming
language)

[Extension of RMI to RMI-IIOP]

Generated by Targeteam

DCOM grew out of COM (Component Object Model)

tightly integrated into Windows OS.

Goal of COM: support the development of components that can be dynamically activated and that can interact with each other.

component : executable code either contained in a DLL or in form of an executable program.

COM is offered in form of a library that is linked to a process.

DCOM : extension of COM which allows a process to communicate with components that are placed on another machine.

DCOM provides access transparency.

[Object Model]

[Architecture]

[Object Invocation Model]

DCOM was combined with Microsoft Transaction Server (MTS) and Microsoft Message Queue Server (MSMQ) to ***COM+*** .

supports distributed transactions to enable transactional distributed applications

integrated into Windows OS

Generated by Targeteam

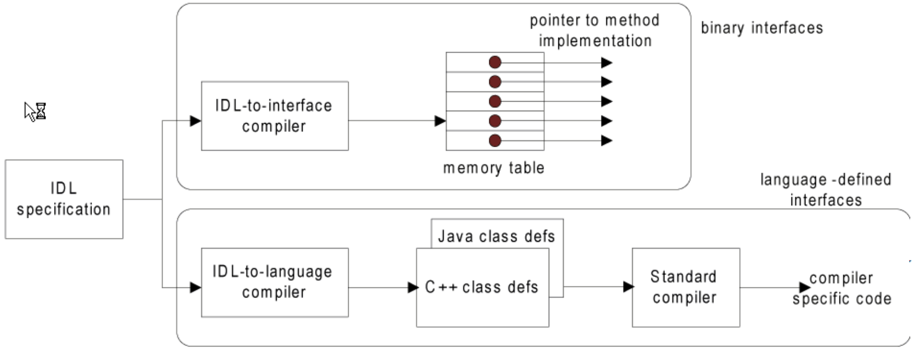
DCOM adopts the remote-object model.

a DCOM object is simply an implementation of an interface

each interface has a unique 128-bit identifier, called Interface Identifier (IID).

each IID is globally unique.

DCOM supports only binary interfaces; essentially a table with pointers to the implementations of the methods which are part of the interface.

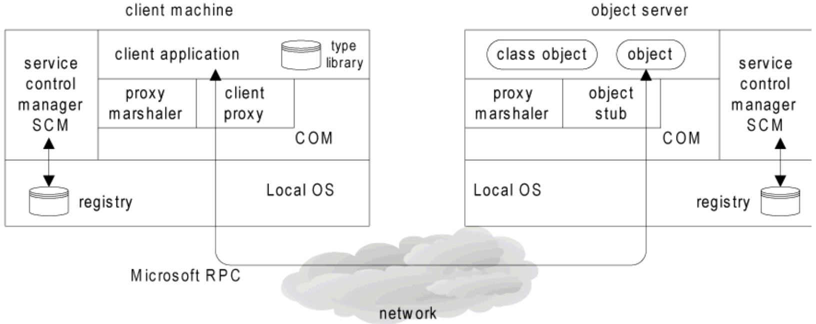


A DCOM object is created as an instance of a class.

DCOM objects are transient.

Generated by Targeteam

The overall architecture of DCOM in combination with the use of class objects, objects and proxies has the following form.



the type library specifies the exact signature of the method to be invoked dynamically.

the registry records the mappings of a call identifier to a local file name containing the implementation of that class.

the service control manager (SCM) is responsible for activating objects.

port for incoming requests and object identifier are registered by SCM.

the proxy marshaler deals with transforming the code of a proxy into a series of bytes for network transmission.

the client proxy represents the object's interface on the client side; responsible for (un)marshaling of object

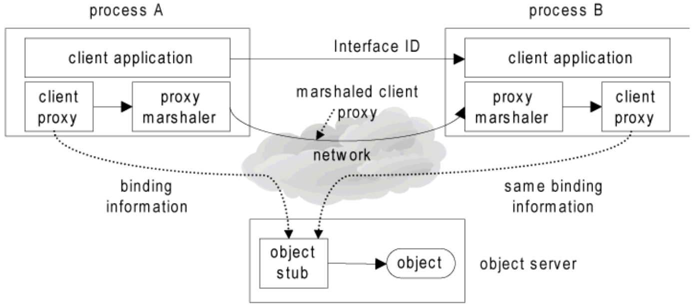
DCOM supports the remote-invocation model, i.e. a client is blocked until it receives an answer from the remote object.

use of a ***cancel object*** to cancel a pending synchronous call.

Passing of Object References

a client references a remote object via an interface pointer; an interface is implemented by means of a proxy.

how does a process A pass an object reference to process B?



DCOM grew out of COM (Component Object Model)

tightly integrated into Windows OS.

Goal of COM: support the development of components that can be dynamically activated and that can interact with each other.

component : executable code either contained in a DLL or in form of an executable program.

COM is offered in form of a library that is linked to a process.

DCOM : extension of COM which allows a process to communicate with components that are placed on another machine.

DCOM provides access transparency.

Object Model

Architecture

Object Invocation Model

DCOM was combined with Microsoft Transaction Server (MTS) and Microsoft Message Queue Server (MSMQ) to ***COM+*** .

supports distributed transactions to enable transactional distributed applications

integrated into Windows OS

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

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

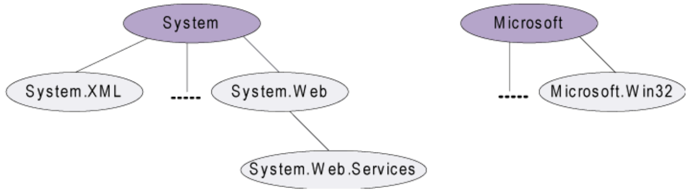
- data structures are uniformly interpreted on the MSIL layer.
- enables interoperability between the languages supported by .NET

Frame Class Library

object-oriented library of common functions available to all languages using the .NET Framework.

e.g. file access, XML document manipulation, or database interaction.

organized in a hierarchy of namespaces



`System.Object` is the base of all library classes and application classes.

.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.

Distributed Applications - Verteilte Anwendungen

- Prof. J. Schlichter
  - Lehrstuhl für Angewandte Informatik / Kooperative Systeme, Fakultät für Informatik, TU München
  - Boltzmannstr. 3, 85748 Garching

    Email: schlichter@in.tum.de

    Tel.: 089-289 18654

    URL: http://www11.in.tum.de/