Desarrollo de Aplicaciones Distribuidas

AUTORES:
Alejandro Calderón Mateos
Javier García Blas
David Expósito Singh
Laura Prada Camacho

Departamento de Informática
Universidad Carlos III de Madrid
Julio de 2012

PARADIGMAS AVANZADOS DE COMPUTACIÓN DISTRIBUIDA

Contenidos

1. Sistema de colas de mensajes
2. Agentes móviles
3. Servicios de red
4. Espacio de objetos

Contenidos

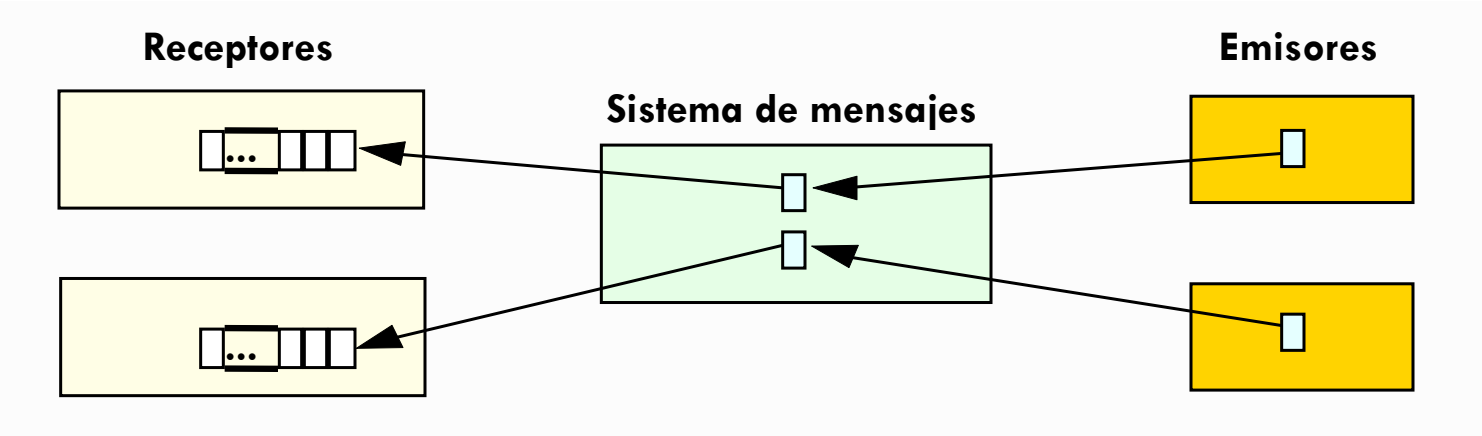
1. **Sistema de colas de mensajes**
2. Agentes móviles
3. Servicios de red
4. Espacio de objetos

Paradigma de sistemas de colas de mensajes

Espacio de objetos, aplicaciones colaborativas
Servicios de red, *object request broker*, agentes móviles
procedimientos remotos, métodos remotos
Cliente-servidor, *peer-to-peer*, MOM
Paso de mensajes

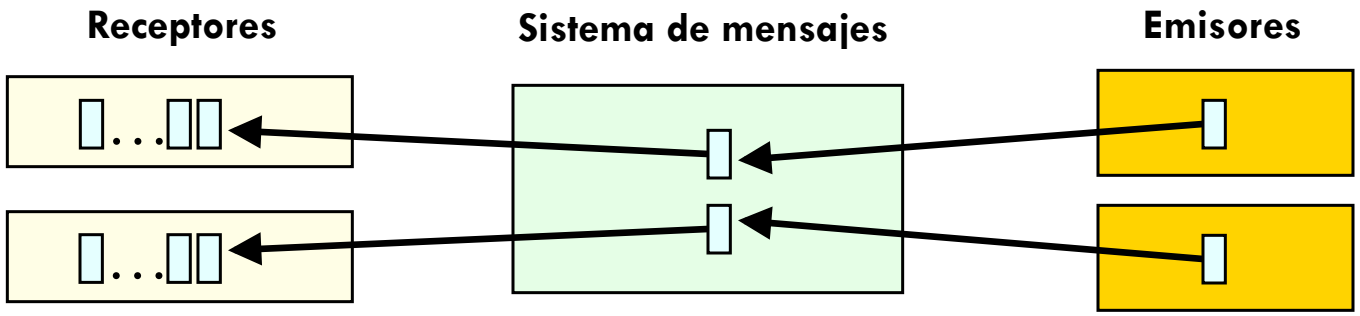
Paradigma del sistema de mensajes

- También denominado *middleware* orientado a mensajes (MOM)
- El sistema de mensajes actúa de intermediario entre los procesos que se comunican
- Proceso:
  - Emisión al sistema de mensajes
  - Almacenamiento en la cola asociada al receptor
  - Envío al proceso receptor

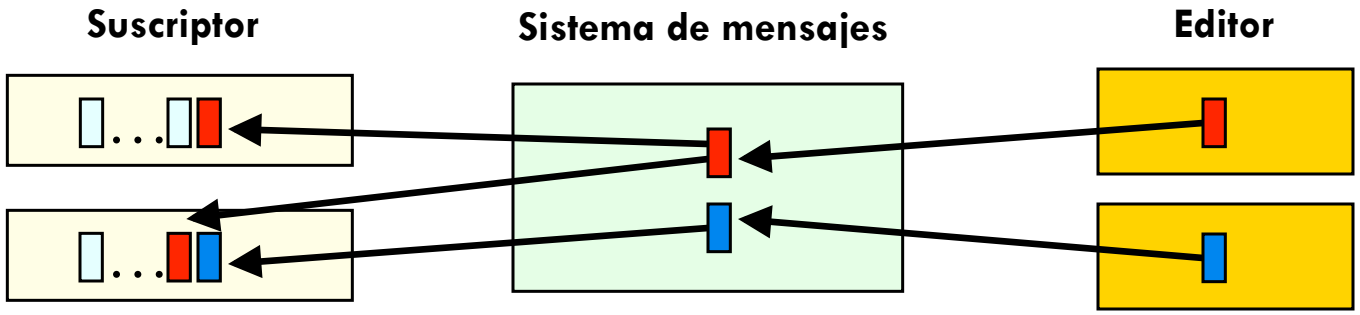


Paradigma de sistemas de colas de mensajes

- Clasificación:
  - Modelo de mensajes punto a punto

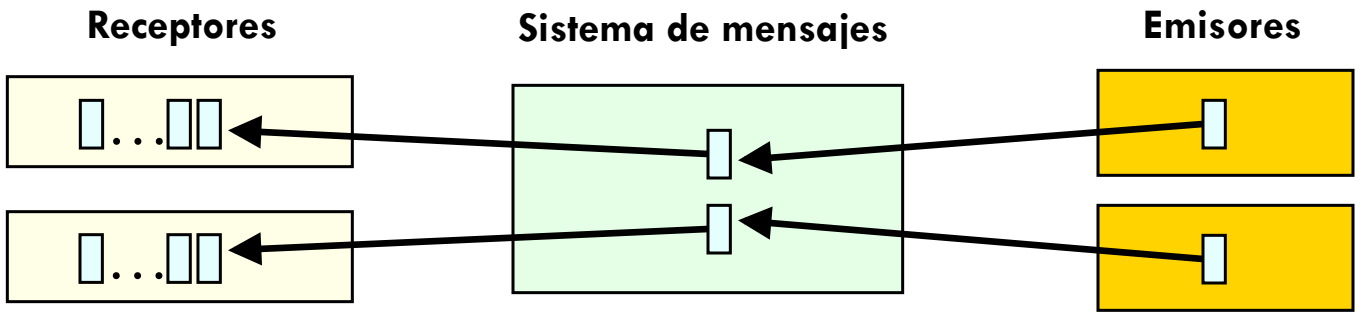


  - Modelo de mensajes publicación/suscripción

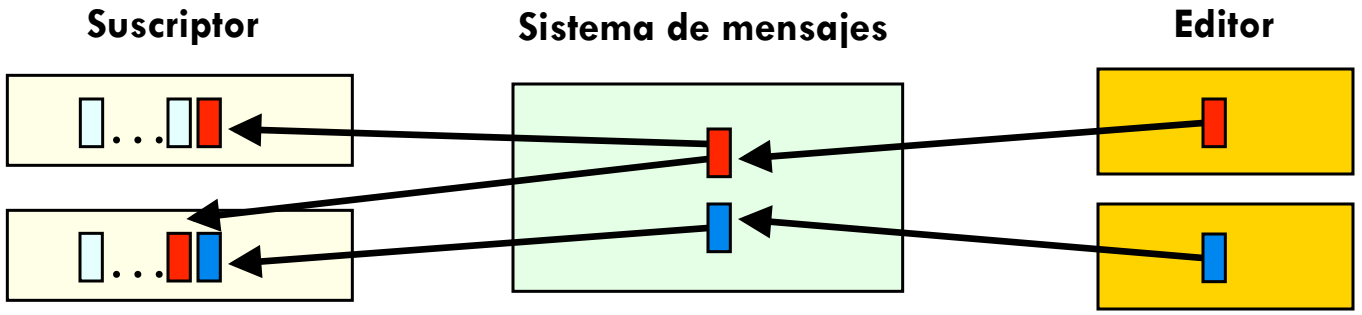


Paradigma de sistemas de colas de mensajes

- Clasificación:
  - Modelo de mensajes punto a punto



  - Modelo de mensajes publicación/suscripción



Paradigma de sistemas de colas de mensajes

- Modelo de mensajes publicación/suscripción:
  - *WebSphere MQ*
  - *IBM MQ*Series* (antes)
    - *http://www-01.ibm.com/software/integration/wmq/*
  - *Microsoft's Message Queue (MSQ)*
    - *http://msdn.microsoft.com/en-us/library/ms711472(VS.85).aspx*
  - *Java's Message Service*
    - *http://java.sun.com/products/jms/tutorial/*

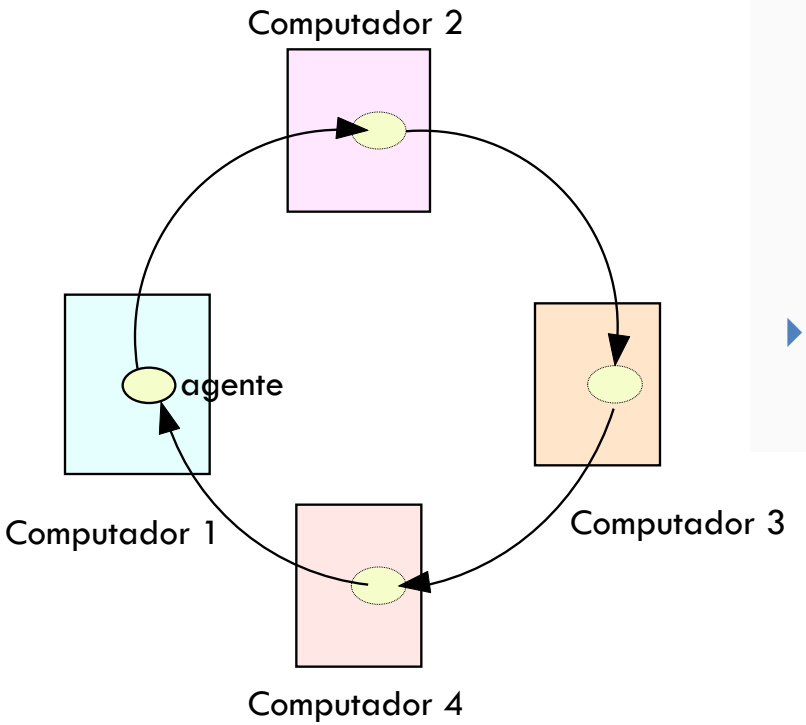
Contenidos

1. Sistema de colas de mensajes
2. **Agentes móviles**
3. Servicios de red
4. Espacio de objetos

Paradigma de agentes móviles

Espacio de objetos, aplicaciones colaborativas
Servicios de red, *object request broker*, agentes móviles
procedimientos remotos, métodos remotos
Cliente-servidor, *peer-to-peer*
Paso de mensajes

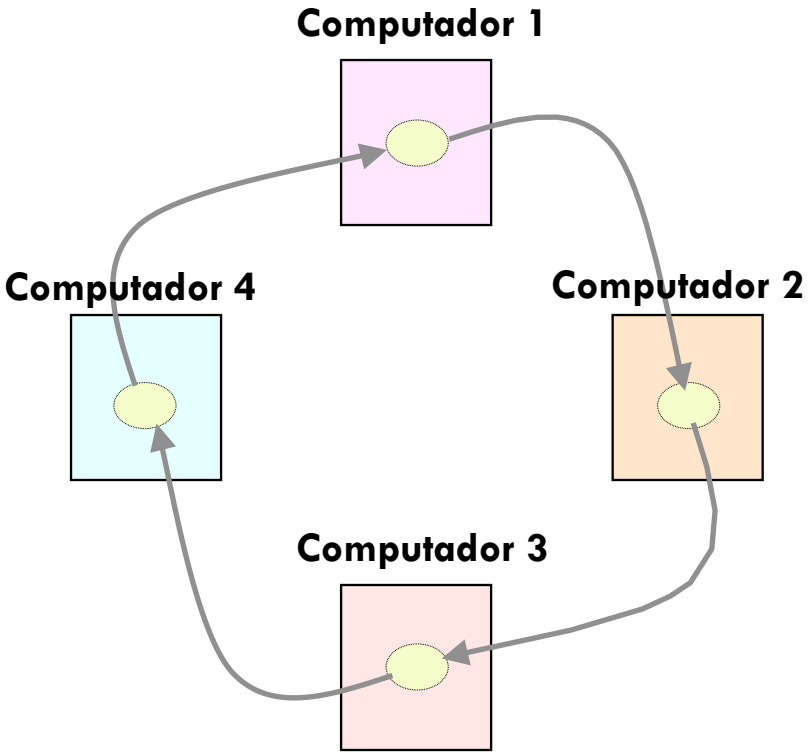
Paradigma de agentes móviles



- Agente móvil:
  programa u objeto transportable.
  - Un agente se lanza desde un ordenador
  - Viaja de forma automática de acuerdo con un itinerario
- Accede a los recursos o servicios de cada sistema que visita

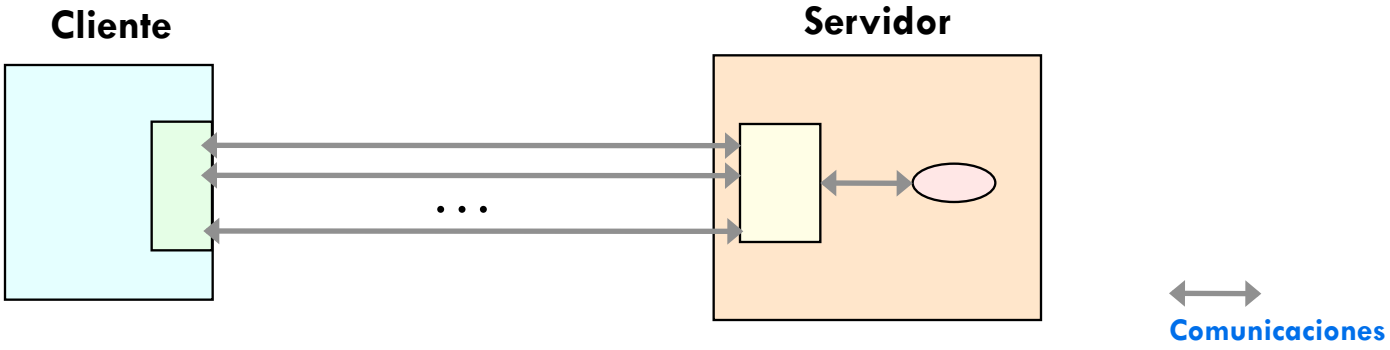
Paradigma de agentes móviles

- Agente: objeto serializable.
- Arquitectura:
  - Identidad.
  - Itinerario.
  - Datos de la entrega.
  - Código.

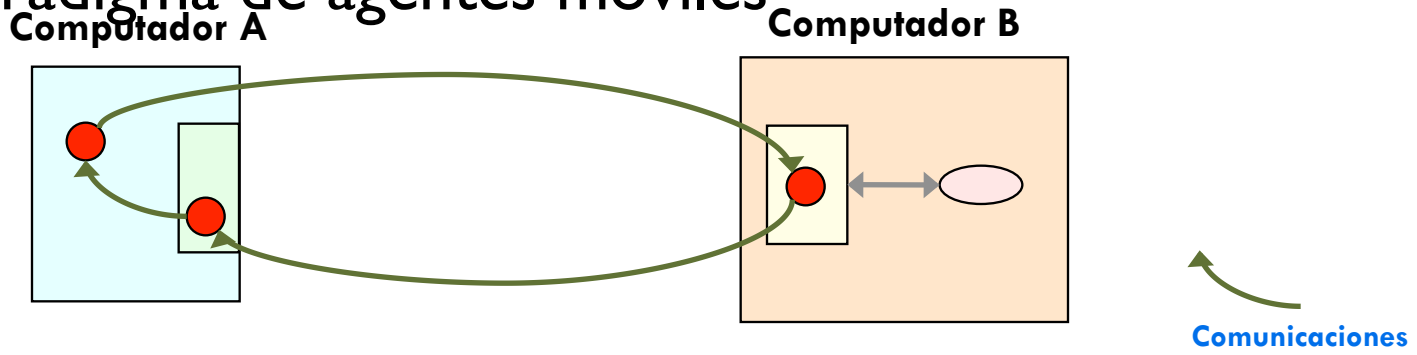


Cliente/servidor *vs* agentes móviles

- Paradigma cliente/servidor



- Paradigma de agentes móviles



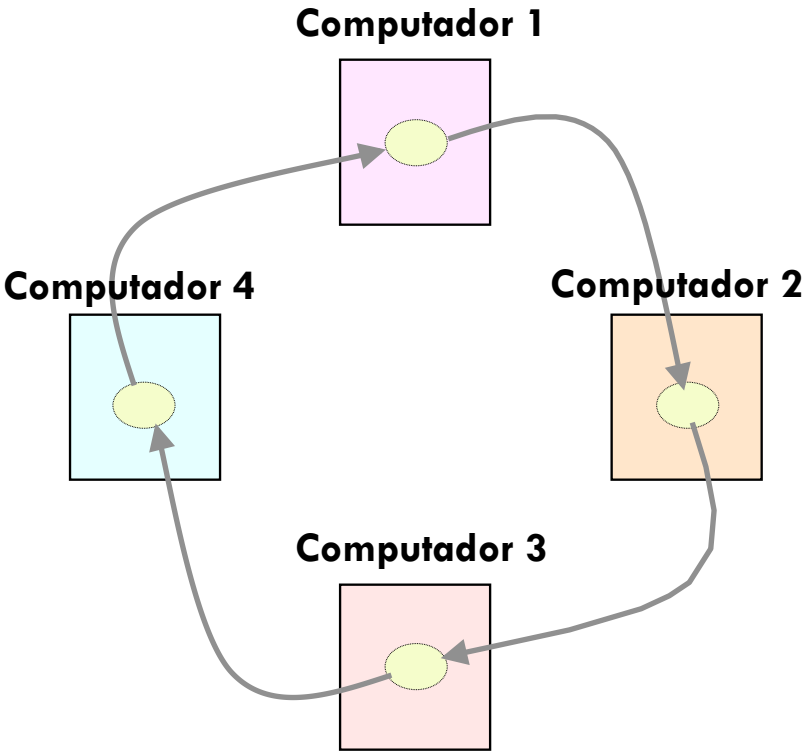
Paradigma de agentes móviles

- Ventajas en el empleo de agentes móviles:
  - Uso eficiente y económico de los canales de comunicación
  - Permite el uso de dispositivos portátiles de bajo coste
  - Permiten operaciones asíncronas y descentralizadas
- Riesgo de los agentes móviles: seguridad
- Técnicas de aumento del nivel de seguridad:
  - Autenticación
  - Acceso a recursos
  - Cifrado

Para más información...

- Mobile Agents Introductory http://www.docstoc.com/docs/13114805/Java-Mobile-Agents---Aglets
- The Mobile Agent List
  http://mole.informatik.uni-stuttgart.de/mal/mal.html
- Mobile Agents and the Future of the Internet http://www.cs.dartmouth.edu/~dfk/papers/kotz:future2/
- Mobile Agents http://cs.gmu.edu/~yhwang1/SWE622/Slides/MobileAgent.ppt

Ejemplo en Java RMI



Ejemplo: interfaz para un agente

```
import java.io.Serializable;

public interface AgentInterface  extends Serializable
{
      void execute() ;
}
```

Ejemplo: agente (1/3)

```
import java.io.*;
import java.util.*;
import java.rmi.*;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;

public class Agent implements AgentInterface
{
      int  hostIndex;
      String name;
      Vector hostList;
      int RMIPort = 12345;

      public Agent (String myName, Vector theHostList,  int theRMIPort )  {
          name = myName;
          hostList = theHostList;
          hostIndex = 0;
          RMIPort = theRMIPort;
      }
```

Ejemplo: agente (2/3)

```
public void execute()  {
      String thisHost, nextHost;
      sleep (2);
      System.out.println("007 here!");
      thisHost = (String) hostList.elementAt(hostIndex);
      hostIndex++;
      if (hostIndex < hostList.size()) {
         nextHost = (String) hostList.elementAt(hostIndex);
         sleep (5);
         try {
                Registry registry = LocateRegistry.getRegistry ("localhost", RMIPort);
                ServerInterface h = (ServerInterface)  registry.lookup(nextHost);
                System.out.println("Lookup for " + nextHost + " at " + thisHost + " completed " );
                sleep (5);   // delay for visibility
                h.receive(this);
         }  catch (Exception e)  {
                System.out.println ("Exception in Agent execute: " + e);
         }
```

Ejemplo: agente (3/3)

```
        } else {
            sleep (5);
            System.out.println("Agent 007 has come home");
            sleep (5);
        }
    } // execute


    static void sleep (double time )  {
        try {
            Thread.sleep( (long) (time * 1000.0));
        } catch (InterruptedException e)  {
            System.out.println ("sleep exception");
        }
    }

}
```

Ejemplo: interfaz para el servidor

```
import java.rmi.*;

public interface ServerInterface   extends Remote
{
      public void receive(Agent h)
                  throws java.rmi.RemoteException;
}
```

Ejemplo: servidor (1/2)

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.net.*;
import java.io.*;

public class Server extends UnicastRemoteObject  implements ServerInterface
{
      static int RMIPort = 12345;

      public Server() throws RemoteException {
         super();
      }

      public void receive (Agent h) throws RemoteException {
         sleep (3) ;
         System.out.println ("*****Agent" + h.name + " arrived." ) ;
         h.execute();
      }
```

Ejemplo: servidor (2/2)

```
public static void main(String args[])  {
  InputStreamReader is = new InputStreamReader(System.in);
  BufferedReader br = new BufferedReader(is);
  String s;
  String myName = "server" + args[0];
  try {
     System.setSecurityManager(new RMISecurityManager());
     Server h = new Server();
     Registry registry =  LocateRegistry.getRegistry(RMIPort);
     registry.rebind( myName, h);
     System.out.println("**********************************");
     System.out.println("     Agent " + myName + " ready.");
     System.out.println("**********************************");
   } catch (RemoteException re) {
         System.out.println("Exception in AgentServer.main:“ + re);
   }
}

static void sleep (double time ) {
     try                                  {Thread.sleep( (long) (time * 1000.0));}
     catch (InterruptedException e) { System.out.println ("sleep exception");}
}
}
```

Ejemplo: cliente (1/2)

```
import java.io.*;
import java.util.*;
import java.rmi.*;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;

public class Client
{
    static int RMIPort = 12345;

    public static void main (String args[])
    {
        System.setSecurityManager(new RMISecurityManager());
        try {
         ...
```

Ejemplo: cliente (2/2)

```
try {
        Registry registry = LocateRegistry.getRegistry ("localhost",  RMIPort);
        ServerInterface h = (ServerInterface)  registry.lookup("server1");
        System.out.println(" Lookup for server1  completed " );
        System.out.println(" ***Have a good trip, " + " agent 007.");
       Vector hostList = new Vector();
        hostList.addElement("server1");
        hostList.addElement("server2");
        hostList.addElement("server3");
       Agent a = new Agent("007", hostList, RMIPort);
        h.receive(a);
        System.out.println("***Nice job, agent 007");

    } catch (Exception e)  {
       System.out.println("Exception in main: " + e);
    }
  } // main

}
```

Contenidos

1. Sistema de colas de mensajes
2. Agentes móviles
3. **Servicios de red**
4. Espacio de objetos

Paradigma de servicios de red

Espacio de objetos, aplicaciones colaborativas
Servicios de red, *object request broker*, agentes móviles
procedimientos remotos, métodos remotos
Cliente-servidor, *peer-to-peer*
Paso de mensajes

Paradigma de servicios de red

- Red se presenta como una infraestructura con servicios
- Cliente accede a servicios:
  - Detección
  - Localización
- Servicios se añaden y retiran de forma dinámica
- Paradigma no centralizado

JSON

- JSON, acrónimo de *JavaScript Object Notation*
- Es un formato ligero para el intercambio de datos
  - JSON es un subconjunto de la notación literal de objetos de JavaScript
  - Es una alternativa simple y ligera de XML puesto que un analizador léxico, sintáctico y semántico es mucho más sencillo.
  - Dada su integración en JavaScript, es fácil su uso con AJAX puesto que con la función eval() es simple recrear el objeto representado por JSON

http://es.wikipedia.org/wiki/JSON

JSON vs. XML

- JSON:

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

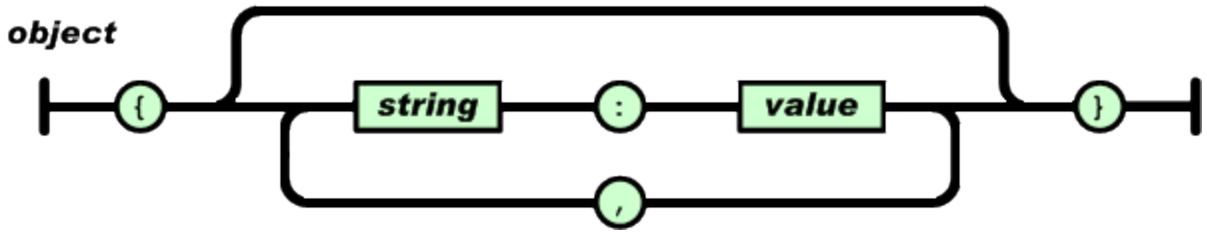
- XML:

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

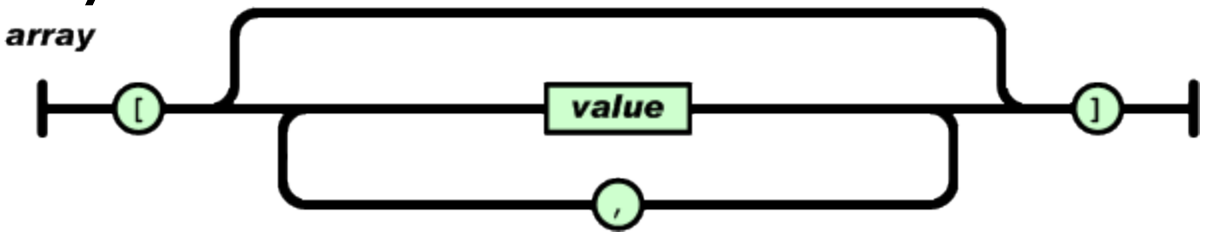

http://es.wikipedia.org/wiki/JSON

JSON: formato (1/4)

- Objeto:



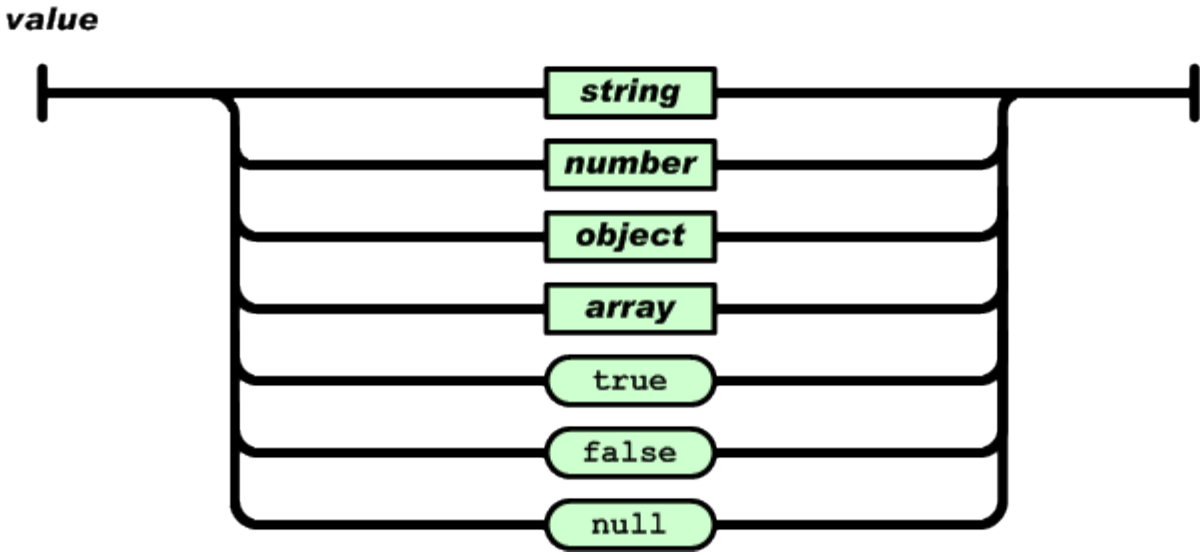
- Array:



http://www.json.org/

JSON: formato (2/4)

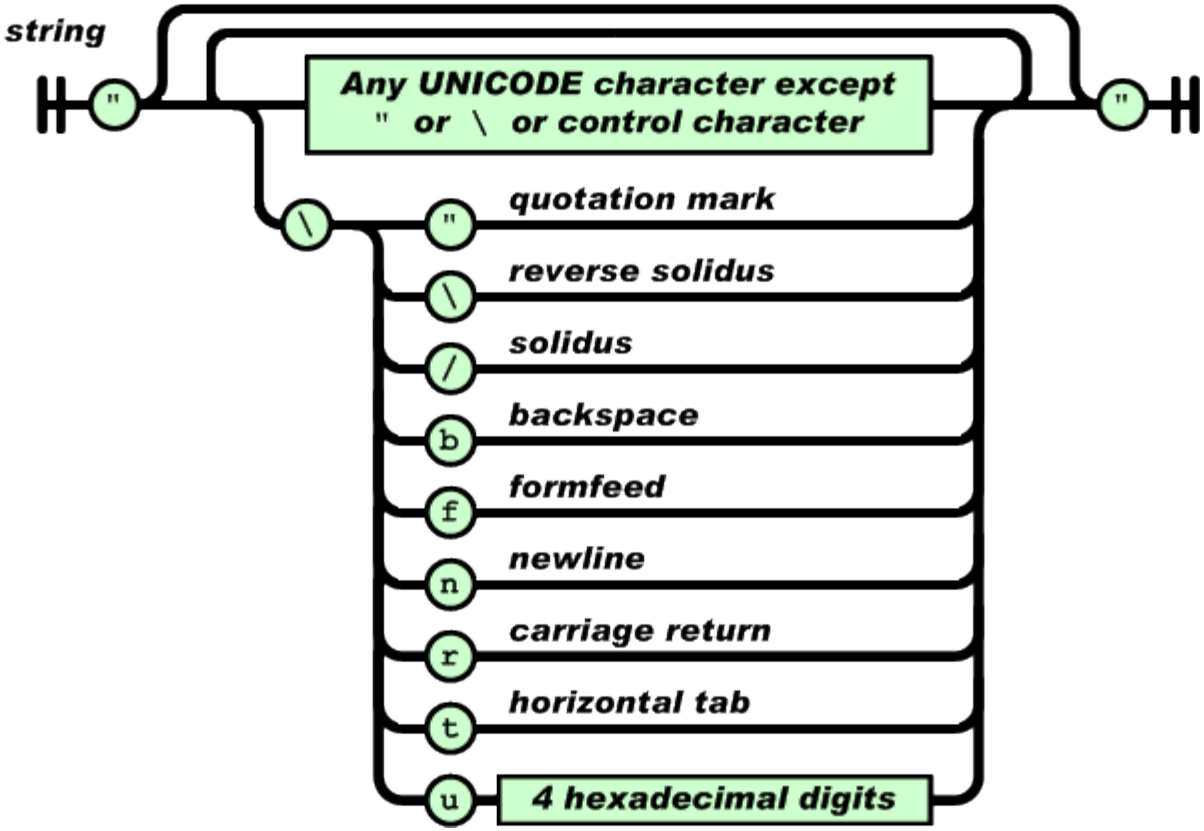
- Valor:



http://www.json.org/

JSON: formato (3/4)

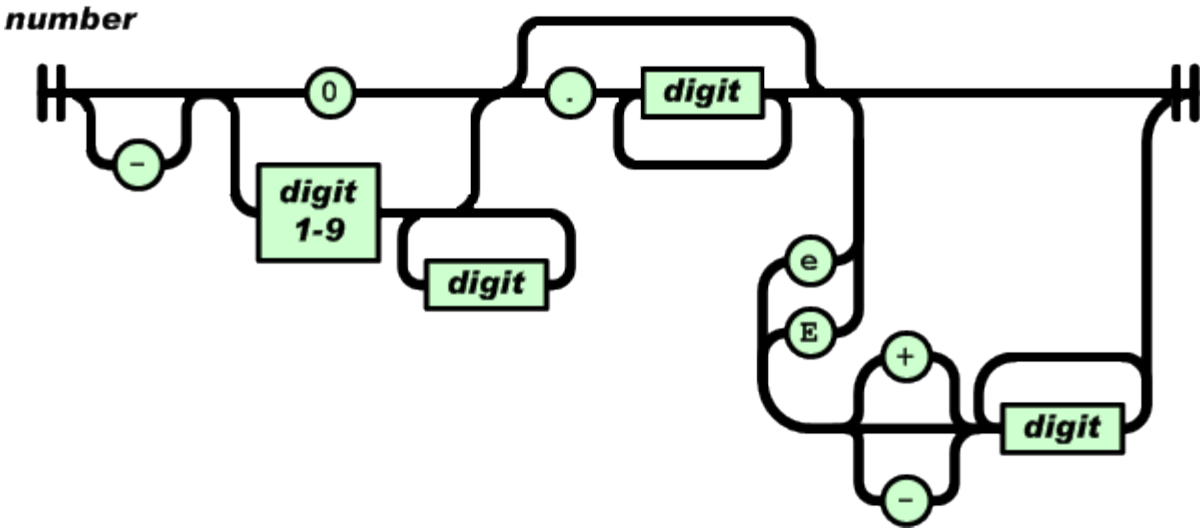
- String:



http://www.json.org/

JSON: formato (4/4)

- Número:



http://www.json.org/

JSON: procesamiento

- De objeto *JavaScript* a JSON:

```
var myJSONText = JSON.stringify(myObject, replacer);
```

- De JSON a *JavaScript*:

```
var myObject = eval('(' + myJSONtext + ')');
```

```
var myObject = JSON.parse(myJSONtext);
```


http://json.org/js.html

Ejemplo de *JavaScript* que usa JSON…

```
var the_object = {};
var http_request = new XMLHttpRequest();

http_request.open( "GET", url, true );

http_request.onreadystatechange = function ()
{
    if ( http_request.readyState == 4 && http_request.status == 200 )
      {
        the_object = JSON.parse( http_request.responseText );
      }
      http_request = null;
};
```


http://json.org/js.html

Ejemplo de JSP que genera JSON…

JSON-JSPExample.jsp

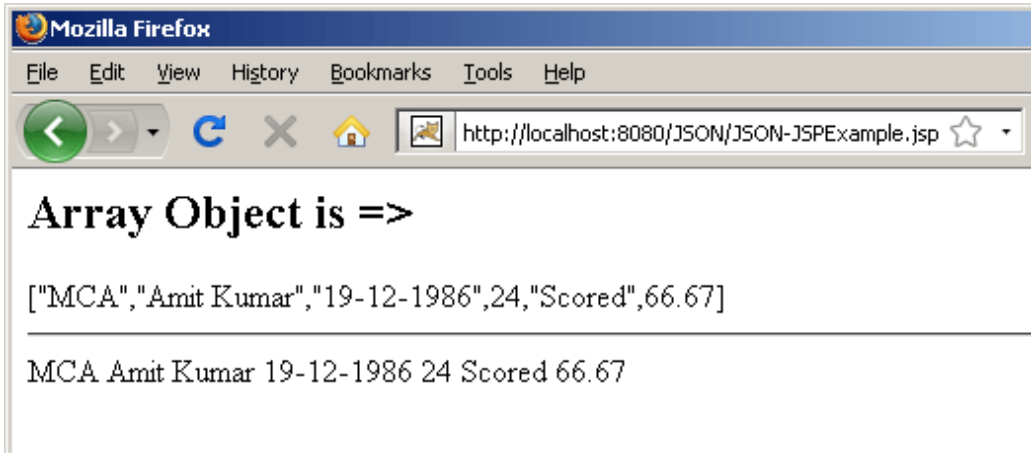
```
<%@ page language="java" import="net.sf.json.JSONArray" %>

 <%
   JSONArray arrayObj = new JSONArray();
   arrayObj.add("MCA");
   arrayObj.add("Amit Kumar");
   arrayObj.add("19-12-1986");
   arrayObj.add(24);
   arrayObj.add("Scored");
   arrayObj.add(new Double(66.67));
%>
<h2>Array Object is =></h2> <%=arrayObj%>
<br><hr>
<%  for(int i=0;i<arrayObj.size();i++){  %>
        <%=arrayObj.getString(i)%>
<%
  }
%>
```


http://www.roseindia.net/tutorials/json/json-servlet-example.shtml

Ejemplo de JSP que genera JSON…

- Crear el archivo JSON-JSPExample.jsp
- Colocarlo en el directorio WEB-INF
- Obtener las JSONLibraries y situarlas en el directorio lib de Tomcat
- Arrancar el servidor Tomcat
- Escribir la siguiente dirección en la barra de direcciones del navegador Web:
  - http://localhost:8080/JSON/JSON-JSPExample.jsp



http://www.roseindia.net/tutorials/json/json-servlet-example.shtml

REST

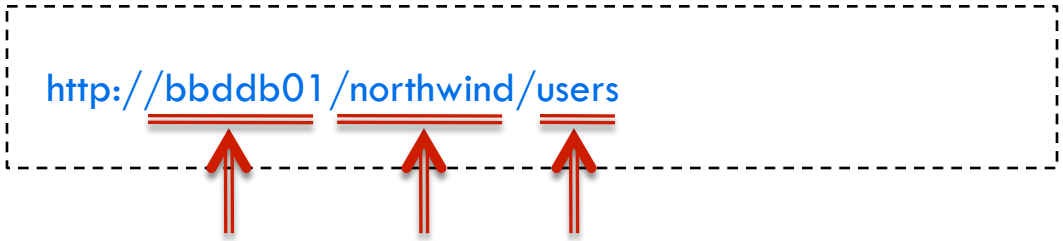
- REST, acrónimo de *Representational State Transfer*
- Es un estilo de organizar la arquitectura software para sistemas distribuidos
  - Aparece como parte de la tesis doctoral de Roy Fielding en el año 2000
  - Se basa en el uso de URL para localizar los recursos y en HTTP para la gestión de los recursos

http://es.wikipedia.org/wiki/JSON

Ejemplo simple con REST (1/3)

- Tenemos como recurso una tabla (users) de una base de datos (northwind) que reside en un servidor de base de datos (bbddb01).

- Localización del recurso:



- El contenido de los datos puede ser XML o JSON
  - En SOAP es obligatorio XML

http://www.slideshare.net/rmaclean/json-and-rest

Ejemplo simple con REST (2/3)

- Tenemos como recurso una tabla (users) de una base de datos (northwind) que reside en un servidor de base de datos (bbddb01).
- Gestión del recurso:

REST	CRUD (Create, Read, Update, Delete)
POST	Crear
GET	Leer
PUT	Actualizar o crear
DELETE	Borrar

- Se usan las acciones de HTTP
  - En SOAP el programador se crea métodos extras

http://www.slideshare.net/rmaclean/json-and-rest

Ejemplo simple con REST (3/3)

- Ejemplos de gestión del recurso:
  - http://bbddb01/northwind/users[firstname="rob%"]

POST	Error
GET	Obtenemos todos los usuarios que comienzan por rob
PUT	Error
DELETE	Borramos todos los usuarios cuyo nombre comience por rob

  - http://bbddb01/northwind/users + datos de entrada

POST	Crear un nuevo usuario
GET	Obtenemos los usuarios que cumplen el criterio
PUT	Creamos/Actualizamos un usuario
DELETE	Borramos todos los usuarios que cumplan el criterio

http://www.slideshare.net/rmaclean/json-and-rest

REST vs. SOAP

- Las acciones de HTTP pueden sustituir la creación de métodos HTTP (SOAP) a la hora de gestionar los datos:
  - Ahorro a la hora de codificar métodos por cada tipo de dato
  - Forma más familiar de tratar con datos:

```
recurso = new Recurso('http://bbddb01/northwind/users') ;
recurso.delete() ;
```

  - Se puede usar tanto JSON como XML
  - Se puede usar fácilmente desde *JavaScript*, Java, etc.

http://blog.feedly.com/2009/05/06/best-practices-for-building-json-rest-web-services/

Ejemplo en ASP.NET Ajax

- Llamada a un servicio Web remoto desde un cliente con JavaScript
- Uso del servicio Web *findNearByWeatherJSON* de *Geonames* que toma la longitud y latitud como parámetros y nos retorna un objeto JSON que describe el tiempo (situación meteorológica) obtenido del observatorio más cercano

WeatherService.cs

```
using System;
using System.Web.Script.Services;
using System.Web.Services;

namespace Mashup
{
    /* To allow this Web Service to be called from script using ASP.NET AJAX,
     * we need to set the following attribute. */
    [ScriptService]
    [WebService(Namespace = "Mashup")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    public class WeatherService : WebService
    {
        [WebMethod]
        public string GetWeatherByLocation (double lat, double lng)
        {
            return GeonamesWeather.GetWeatherByLocation(lat, lng);
        }
    }
}
```


http://geekswithblogs.net/JuanDoNeblo/archive/2007/10/24/json_in_asp.net_ajax_part2.aspx

GeonamesWeather.cs (1/2)

```
using System;
using System.Net;
using System.Globalization;
using System.IO;

namespace Mashup {
    public class GeonamesWeather {

        private readonly static string
                    FindNearbyWeatherUrl = "http://ws.geonames.org/findNearByWeatherJSON?lat={0}&lng={1}";

        public static string GetWeatherByLocation(double lat, double lng)
        {
            string formattedUri = String.Format(CultureInfo.InvariantCulture, FindNearbyWeatherUrl, lat, lng);

            HttpWebRequest webRequest = GetWebRequest(formattedUri);
            HttpWebResponse response = (HttpWebResponse)webRequest.GetResponse();
            string jsonResponse = string.Empty;
            using (StreamReader sr = new StreamReader(response.GetResponseStream()))   {
                jsonResponse = sr.ReadToEnd();
            }
            return jsonResponse;
        }
```


http://geekswithblogs.net/JuanDoNeblo/archive/2007/10/24/json_in_asp.net_ajax_part2.aspx

GeonamesWeather.cs (2/2)

```
    private static HttpWebRequest GetWebRequest(string formattedUri)
    {
        // Create the request's URI.
        Uri serviceUri = new Uri(formattedUri, UriKind.Absolute);

        // Return the HttpWebRequest.
        return (HttpWebRequest)System.Net.WebRequest.Create(serviceUri);
    }

  }
}
```


http://geekswithblogs.net/JuanDoNeblo/archive/2007/10/24/json_in_asp.net_ajax_part2.aspx

WeatherService.asmx

```
<%@ WebService
        Language="C#"
        CodeBehind="WeatherService.cs"
        Class="Mashup.WeatherService"
%>
```

WeatherService.asmx

```
<%@ WebService
        Language="C#"
        CodeBehind="WeatherService.cs"
        Class="Mashup.WeatherService"
%>
```

JsonWeather.aspx (1/2)

```
<body>
<form id="form1" runat="server">
<div>
    <asp:ScriptManager ID="ScriptManager1" runat="server">
        <Services>
            <asp:ServiceReference Path="~/WeatherService.asmx" />
        </Services>
    </asp:ScriptManager>
    <div id="legend">
        <b>If you don't have a clue</b>:
        <br />
        <b>Rome</b>: Lat: 41.9 - Lng: 12.5          <br />
        <b>London</b>: Lat: 51.5 - Lng: 0           <br />
        <b>Paris</b>: Lat: 48.84 - Lng: 2.35        <br />
        <b>Redmond</b>: Lat: 47.68 - Lng: -122.13   <br />
        <b>New York</b>: Lat: 40.7 - Lng: -74
    </div>

    <label for="txtLat“>        Latitude:</label><input type="text" id="txtLat" />
    <label for="txtLng“>        Longitude:</label><input type="text" id="txtLng" />
    <input type="button" value="Get Weather Info" onclick="callService();" />
    <div id="divResult“></div>
```

JsonWeather.aspx (2/2)

```
<script type="text/javascript">
        function callService() {
            var latitude = $get('txtLat').value;
            var longitude = $get('txtLng').value;
            Mashup.WeatherService.GetWeatherByLocation(latitude, longitude,GetWeather_success, onFailed);
        }
        function GetWeather_success(e) {
            var result = Sys.Serialization.JavaScriptSerializer.deserialize(e, true);  //var result = eval('(' + e + ')');
            var weatherData = new Sys.StringBuilder();
            var line;
            for(var property in result.weatherObservation) {
                line = String.format("<b>{0}:</b> {1}<br/>", property, result.weatherObservation[property]);
                weatherData.append(line);
            }
            $get('divResult').innerHTML = weatherData.toString();
        }
        function onFailed()     {
            $get('divResult').innerHTML = 'Something went terribly wrong!';
        }
    </script>
  </div>
  </form>
</body>
```


http://geekswithblogs.net/JuanDoNeblo/archive/2007/10/24/json_in_asp.net_ajax_part2.aspx

Contenidos

1. Sistema de colas de mensajes
2. Agentes móviles
3. Servicios de red
4. **Espacio de objetos**

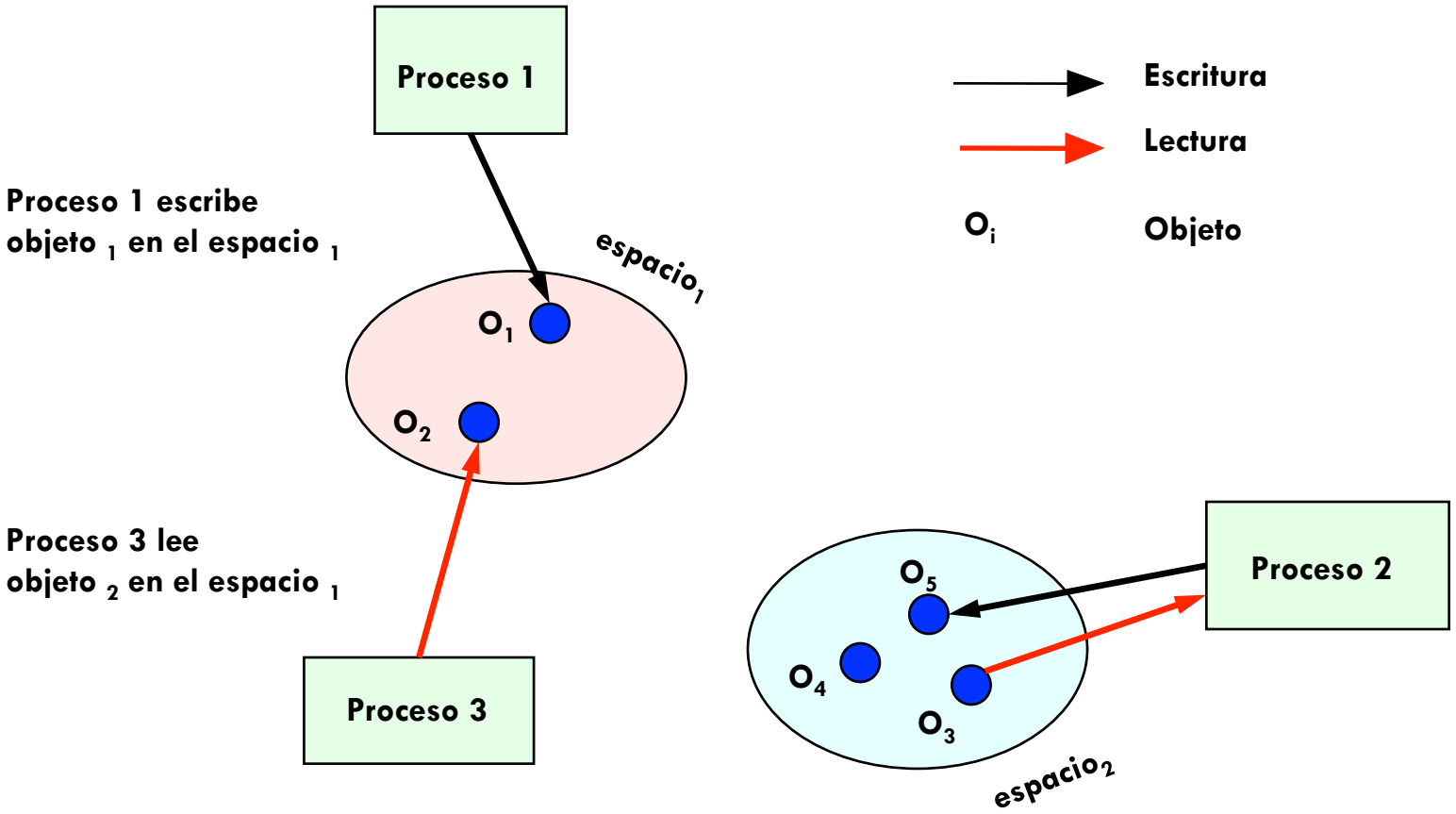
Paradigmas de
Espacio de objetos y aplicaciones colaborativas

Espacio de objetos, aplicaciones colaborativas
Servicios de red, *object request broker*, agentes móviles
procedimientos remotos, métodos remotos
Cliente-servidor, *peer-to-peer*
Paso de mensajes

Paradigma de espacio de objetos

- Base del sistema de pizarras
- Espacio de objeto: repositorio de objetos
- Los procesos se coordinan intercambiando objetos
- Los objetos no modifican los objetos en el espacio ni invocan los métodos directamente
- Operaciones de extracción y reinserción del objeto

Paradigma de espacio de objetos



Paradigma de espacio de objetos

- JavaSpace:
  - Basado en Jini
  - Simple y potente:
    "*Using a small set of operations, we can build a large class of distributed applications without writing a lot of code*"
  - Da soporte a protocolos estrechamente relacionados
  - Simplifica el diseño del servidor

Para más información…

- The Nuts and Bolts of Compiling and Running JavaSpaces Programs: http://www.jiniworld.net/document/javaspace/The Nuts and Bolts of Compiling and Running JavaSpaces(TM).html
- JavaSpaces(TM) Principles, Patterns, and Practice: http://www.cswl.co.uk/whiteppr/tutorials/jini.html
- O'Reilly Network: First Contact: Is There Life in JavaSpace?