

Arquitectura de Sistemas de Software

Departamento de Ciencia de la Computación
Escuela de Ingeniería – PUC
Hans Findel {hifindel@uc.cl}



Conectores

Conectores de Software

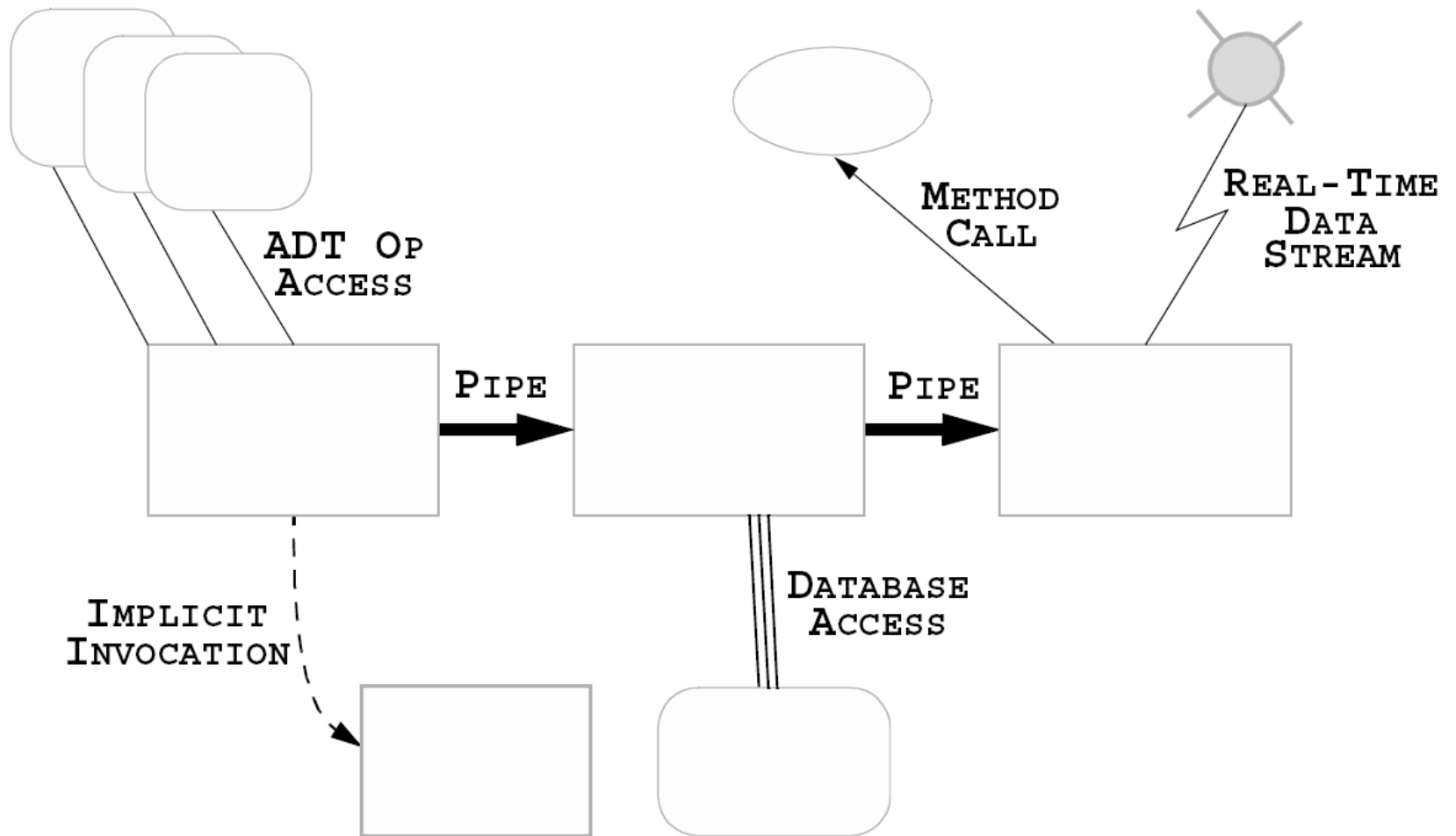
- Elementos arquitecturales que modelan
 - ▣ Interacciones entre componentes
 - ▣ Las reglas que gobiernan esas interacciones
 - ▣ Permiten transferir datos y/o control
- Interacciones simples:
 - ▣ Llamadas a procedimientos
 - ▣ Acceso a variables compartidas
- Interacciones complejas:
 - ▣ Protocolos cliente-servidor
 - ▣ Protocolos de acceso a base de datos
 - ▣ Multicast de eventos asíncronos

Afectan Atributos de calidad

Availability	Downtime/90 days	Downtime/year
99.0%	21 hours, 36 minutes	3 days, 15.6 hours
99.9%	2 hours, 10 minutes	8 hours, 0 minutes, 46 seconds
99.99%	12 minutes, 58 seconds	52 minutes, 34 seconds
99.999%	1 minute, 18 seconds	5 minutes, 15 seconds
99.9999%	8 seconds	32 seconds

Table 1: System Availability Requirements

Ejemplo



Conectores conceptuales (diseño)



- Elementos fundamentales
- Tienen identidad
- Describen las interacciones del sistema
- Tienen una especificación y abstracción

Conectores implementados (código)

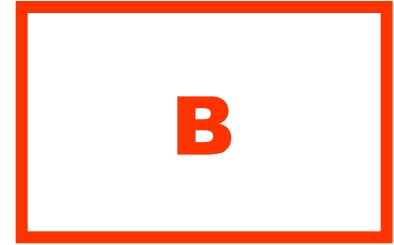
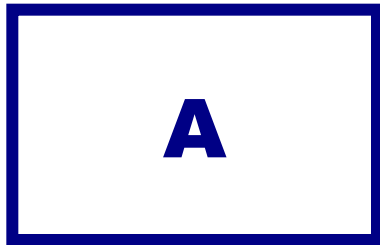
- Generalmente, no tienen un código específico y bien delimitado que los implemente (e.j. Un objeto)
- Generalmente, no tienen identidad
- Generalmente, no corresponden a unidades compilables
- Generalmente, están Implementados de manera distribuida
 - Incluyen varios módulos
 - Abarcan varios mecanismos de interacción

Conectores conceptuales Vs. implementados

- Conector != Componente
 - ▣ Componente: Funcionalidad específica para una aplicación
 - ▣ Conector: Mecanismos de interacción abstractos independientes de la aplicación
- Conector.
 - ▣ Permite *abstraer* reglas de interacción complejas, *parametrizarlas*, *localizarlas*, *reusarlas*, *modificarlas*, etc...
 - Ej: Binario vs. N-ario, asimétrico vs. simétrico, etc...

Rol de los Conectores de Software

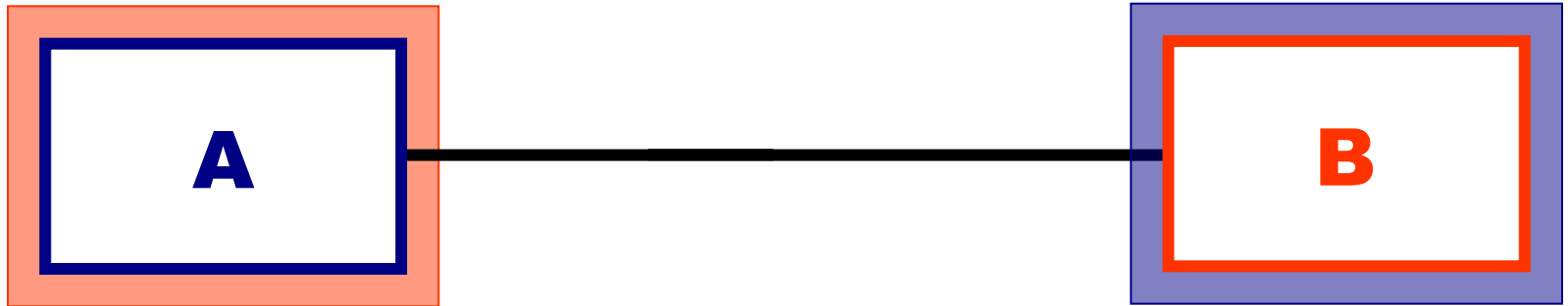
Cómo permitimos que los componentes
de tipo A y B interactúan?



Rol de los Conectores de Software

Añadimos un adaptador a A

o a B



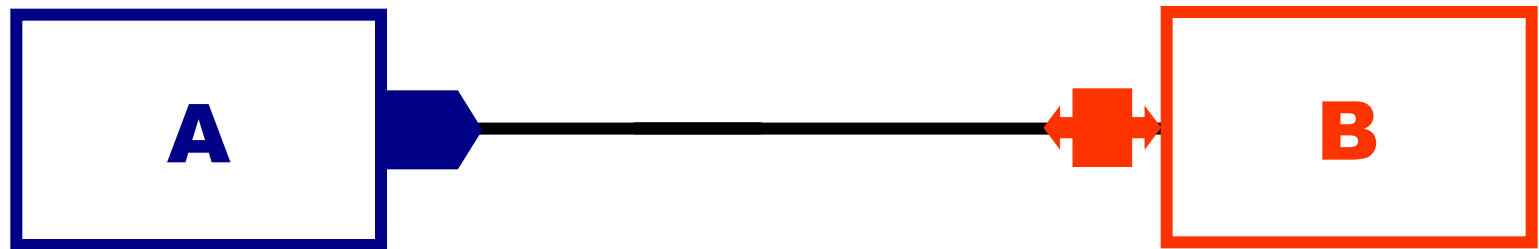
Rol de los Conectores de Software



Cambiamos la forma de A según la de B

Hacemos B multilingüe

Rol de los Conectores de Software

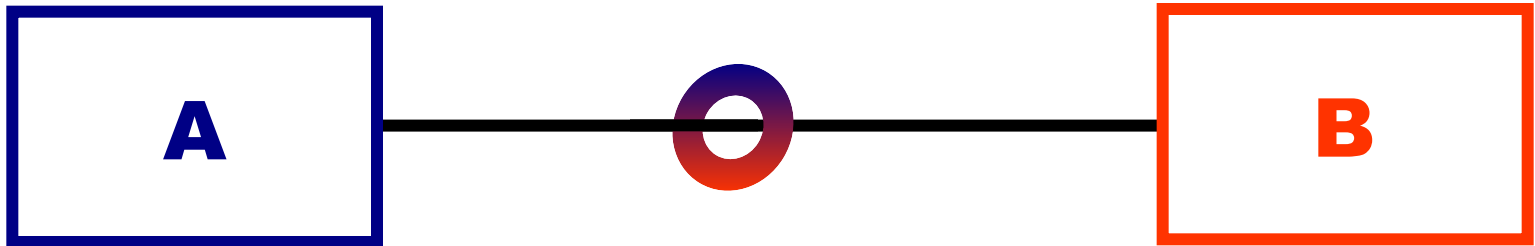


Publicamos una abstracción
De la forma de A

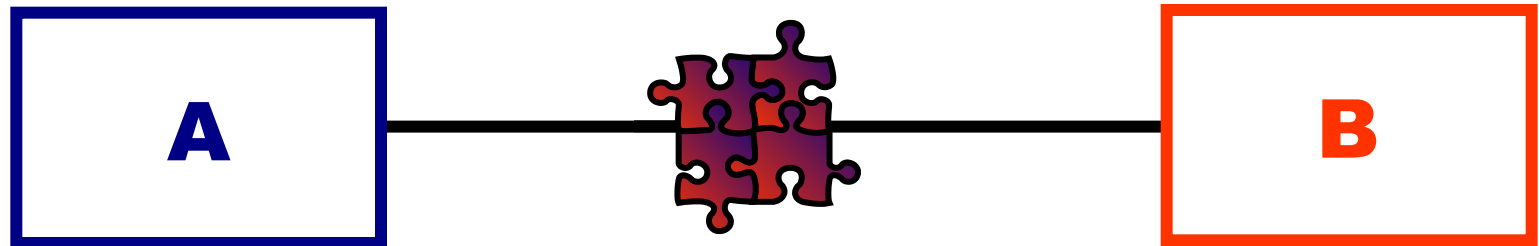
Proveemos a B con un
convertidor de importar/exportar

Rol de los Conectores de Software

Introducimos una forma
intermediaria

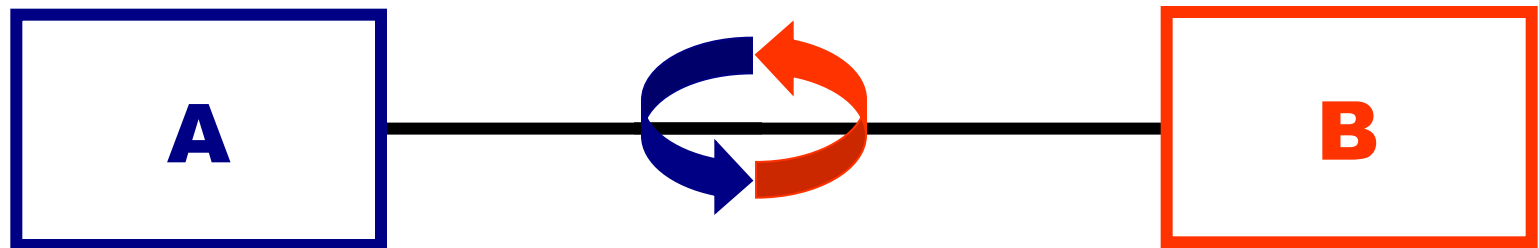


Rol de los Conectores de Software



Negociamos para encontrar
formas comunes para A y B

Rol de los Conectores de Software



Transformamos los datos
Bajo demanda

... Qué es lo correcto?

Roles de conectores



- Comunicación
- Coordinación
- Conversión
- Facilitación

Conectores como Comunicadores

- Permite:
 - ▣ Diferentes mecanismos de comunicación
 - ▣ Restricciones en la dirección y estructura de la comunicación
 - ▣ Restricciones de la calidad del servicio
- Separa la comunicación del procesamiento
- Influencia características no funcionales del sistema
 - ▣ Performance, escalabilidad, seguridad, etc...

Conectores como Coordinadores



- Determinan el flujo de control
- Controlan el flujo de datos
- Separan el control del procesamiento
- Son ortogonales a la comunicación, conversión, y facilitación

Conectores como conversores

- Permiten la interacción de componentes diferentes desarrollados independientemente
 - ▣ Adaptadores, wrappers
- Diferencias basadas en la interacción:
 - ▣ Tipo de datos
 - ▣ Número de parámetros
 - ▣ Frecuencia de
 - ▣ Orden de la interacción

Conectores como facilitadores

- Permiten interacción entre componentes que buscan interoperar:
 - **Interoperar:** Componentes independientes interactúan y colaboran para proveer una funcionalidad conjunta
 - **Integración:** Componentes independientes se amalgaman de manera de convertirse en una nueva unidad funcional
- Definen el acceso a la información compartida
- Aseguran perfiles de desempeño
 - Balance de carga
- Proveen mecanismos de sincronización
 - Secciones críticas, monitores, etc...

Tipos de conectores

	Comunicación	Coordinación	Conversión	Facilitación
Comunicación	Streams	Llamada a procedimientos Eventos	Acceso a datos	
Coordinación	Llamada a procedimientos Eventos			Árbitro
Conversión	Acceso a datos		Adaptador	
Facilitación		Árbitro		Links Distribuidor

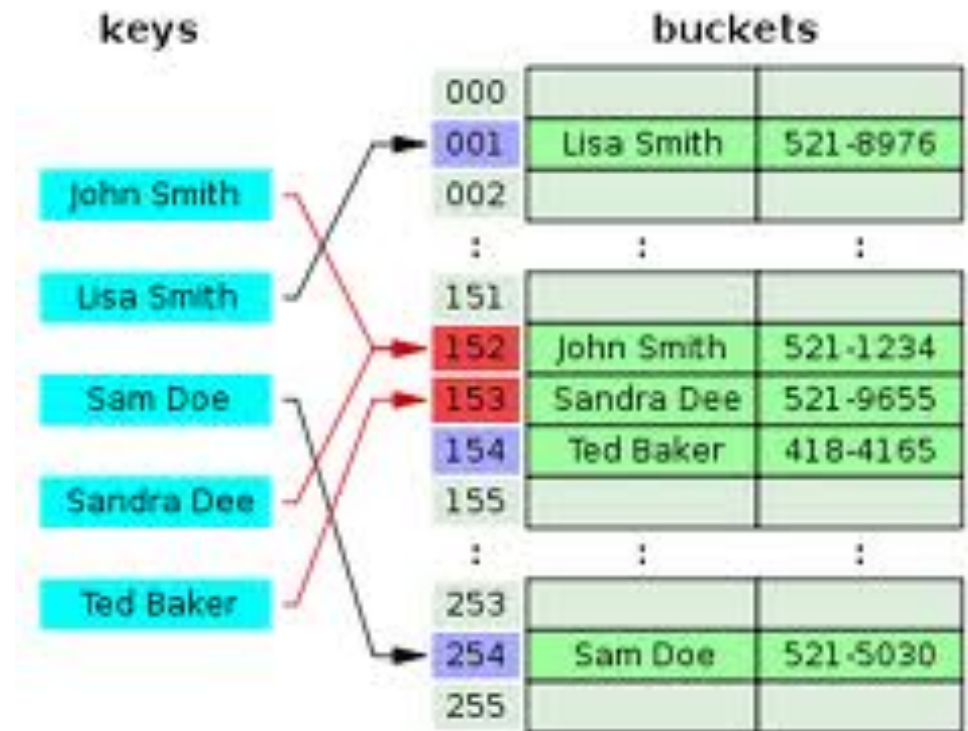
Llamada a procedimientos

Comunicación y Coordinación

Dimensión	Subdimensión	Valor
Parámetros	Transferencia de datos	Referencia, Valor, Nombre
	Semántica	Valores por defecto, Parámetros keyword, Parámetros en línea
	Valores de retorno	
	Registro de invocación	Push L to R, R to L, tabla hash
Punto de entrada	Múltiple, Simple	
Invocación	Explícita	Llamada a método, a macro, a sistema
	Implícita	Excepciones , callbacks , delegación
Sincronicidad		Asíncrona, Síncrona
Cardinalidad	Fan-out, Fan-in	
Accesibilidad		Privada, Protegida, Pública

Ejemplos: Tabla de Hash

- Estructura de datos que usa una *función de hash* para calcular una *clave o índice* a partir de ciertos valores (ej. Nombre) y asociar valores adicionales (ej. Teléfono) a dicha clave
- Puede haber conflictos, en cuyo caso se pueden definir listas enlazadas para el desborde de datos, doble hash, etc...



Ejemplos: Llamadas a procesos de sistema desde Java

```
import java.io.*;

public class doscmd {
    public static void main(String args[]) {
        try {
            Process p=Runtime.getRuntime().exec("cmd /c dir");
            p.waitFor();
            BufferedReader reader = new BufferedReader(new
                InputStreamReader(p.getInputStream()));
            String line=reader.readLine();
            while(line!=null){
                System.out.println(line);
                line=reader.readLine();
            }
        }
        catch(IOException e1) {}
        catch(InterruptedException e2) {}
        System.out.println("Done");
    }
}
```


Ejemplos: Excepciones

- Mecanismo diseñado para manejar la ocurrencia de excepciones, de las condiciones especiales que cambian el flujo de ejecución de un programa

```
try { line = console.readLine();  
    if (line.length() == 0) {  
        throw new EmptyLineException("error!1!"); }  
    console.println("Hello %s!" % line);  
    console.println("Success!!");  
} catch (EmptyLineException e)  
    { console.println("Hello!"); }  
catch (Exception e)  
    { console.println("Error: " + e.message()); }  
finally {  
    console.println("The program terminates now");  
}
```

Ejemplos: Callback

- Un callback es una referencia a código ejecutable que se pasa como argumento a otro código

```
#include <stdio.h>
#include <stdlib.h>

void PrintTwoNumbers(int (*numberSource)(void))
{    printf("%d\n", numberSource()); }

int meaningOfLife(void)
{    return 42;    }




int main(void)
{    PrintTwoNumbers(meaningOfLife); }
```






Ejemplos: Callback URL

```
<?php
    $id = $_REQUEST['id'];
    if ($_SERVER['REQUEST_METHOD'] == 'GET') {
        echo 'CallBack URL : id='.$id; }
    if ($_SERVER['REQUEST_METHOD'] == 'POST') {
        $raw = file_get_contents('php://input');
        echo $raw;
    }
?>
```

```
curl -d "id=2&id3=3" http://localhost/callbackTest.php
```

id=2&id3=3

← → ↻  localhost/callbackTest.php?id=2  

 Atom2RDF  Desktop  OER  REST »  Other Bookmarks

CallBack URL : id=2

Ejemplos: Delegación

- Un objeto depende de otro para proveer un conjunto específico de funcionalidades

```
class A {  
    void foo() { this.bar(); }  
    void bar() { print("a.bar"); }  
}
```

```
class B {  
    private A a;  
    public B(A a) { this.a = a; }  
    void foo() { a.foo(); }  
    void bar() { print("b.bar"); }  
}
```

```
a = new A();  
b = new B(a);  
b.foo();
```

a.bar

Eventos

Comunicación y Coordinación

Dimensión	Subdimensión	Valor
Cardinalidad	Productores, patrones de eventos, observadores	
Entrega		Mejor esfuerzo, exactamente uno, a lo más uno, al menos uno
Prioridad	Salida, Entrada	
Sincronicidad		Síncrono, asíncrono, sincronía con timeout
Notificación		Polling, pub/sub , actualización centralizada, despacho en cola
Causalidad		Absoluta, relativa
Modo	Hardware	Falla de página, interrupciones, traps
	Software	Señales, I/O GUI, triggers

Eventos

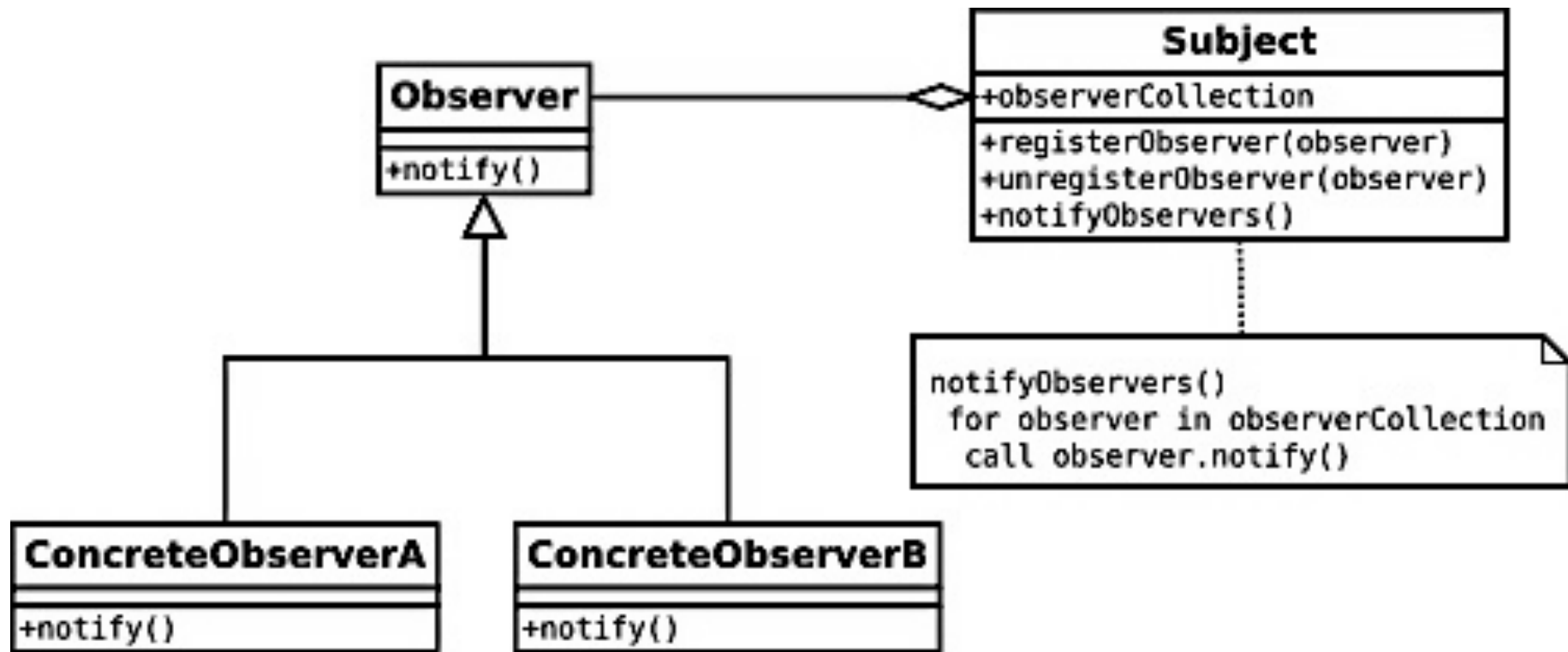
- Comunicación de Servicios:
 - ▣ Transfiere datos sobre el evento a los componentes interesados(Notificación)
 - ▣ Otorga el control a los componentes para procesar el mensaje

- Coordinación de Servicios:
 - ▣ Regula el control de flujo entre los componentes (Flujo es gatillado por un evento)

Patron de Diseño : Observador

- Se notifica a los componentes (observers) el cambio de estado (eventos) de un objeto (Subject)
- Logra independencia de los componentes (Débilmente acoplado)
- Ejemplo MVC: Las vistas (Observers) se actualizan cuando el modelo (Subject) cambia (evento)

Patrón de Diseño: Observador



Polling

- Activamente examinar el estado de un dispositivo mediante un cliente efectuando llamadas síncronas
- Generalmente se usa como sinónimo de **busy-wait** polling (polling con espera ocupada), donde el cliente no hace otra cosa que esperar hasta que el dispositivo esté listo para entregar la respuesta

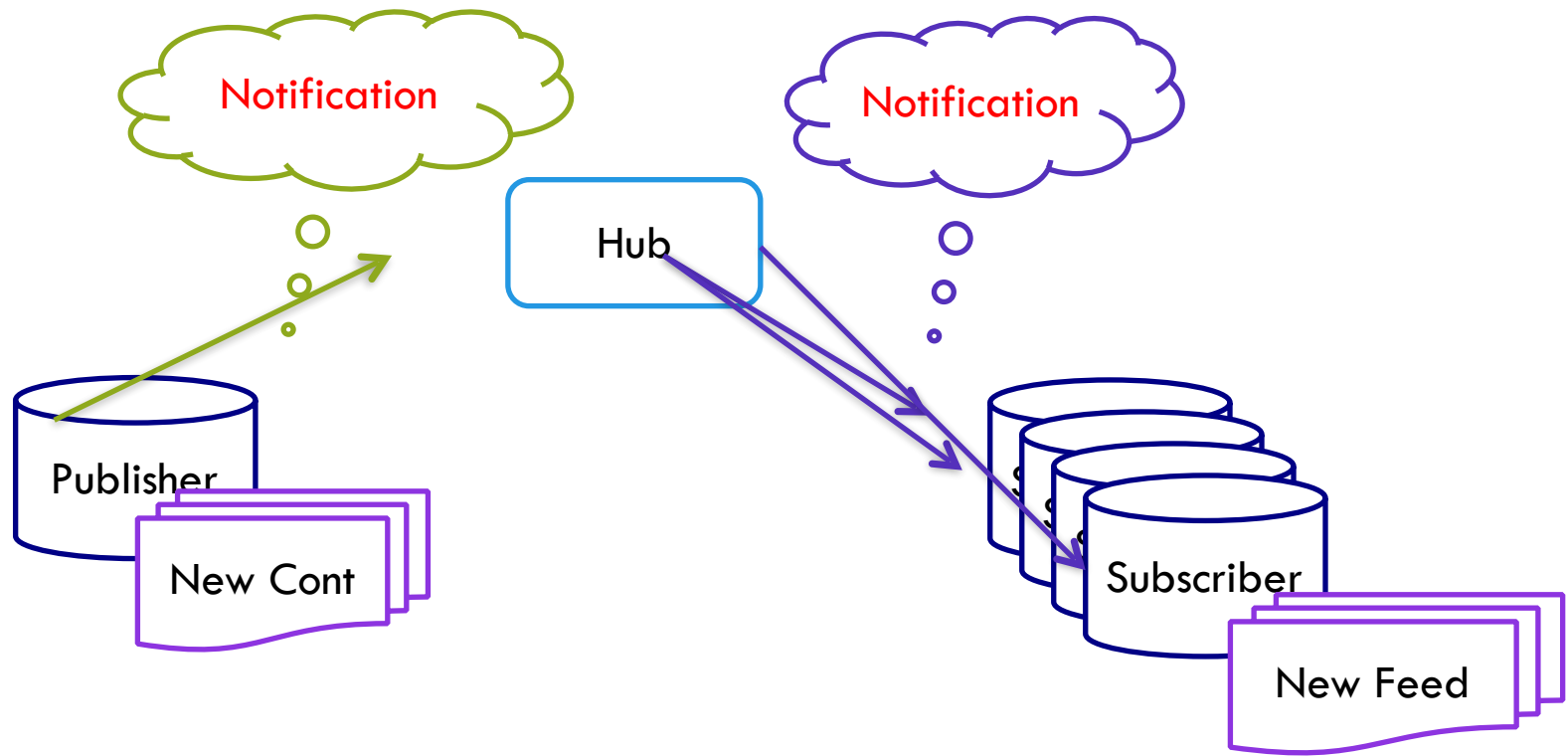
Ejemplo: Polling con busy-waiting

```
public class A
{ public int valueGen()
  { return new Random().nextInt(5); }
}

public class SleepMessages
{ public static void main(String args[])
  throws InterruptedException
  { A a = new A();
    for (;;)
    { Thread.sleep(4000); //Pause for 4 seconds
      if ( (a.valueGen % 2) == 0)
      { System.out.println("done");
        break; }
    }
  }
}
```

Ejemplo: PubSubHubBub (pub/sub)

- Basado en el estilo arquitectónico Pub/Sub
- Se envían y reciben datos via notificaciones(eventos)



Ejemplo: Señales

```
for (;;)
{ try
  { Thread.sleep(4000);
  } catch (InterruptedException e)
    { //We've been interrupted!!
      return; }
  System.out.println(importantInfo[i]);
}
```

```
if (Thread.interrupted())
{ throw new InterruptedException(); }
```

```
for (int i = 0; i < inputs.length; i++)
{ heavyCrunch(inputs[i]);
  if (Thread.interrupted())
  { //We've been interrupted!!
    return; }
}
```

Ejemplo: I/O GUI (java)

```
public class Beeper ... implements ActionListener {  
    ...  
    //where initialization occurs:  
        button.addActionListener(this);  
    ...  
    public void actionPerformed(ActionEvent e) {  
        ...//Make a beep sound...  
    }  
}
```

Ejemplo: Triggers (mysql)

```
DELIMITER $$  
CREATE TRIGGER before_employee_update  
BEFORE UPDATE ON employees  
FOR EACH ROW BEGIN  
INSERT INTO employees_audit  
SET action = 'update',  
employeeNumber = OLD.employeeNumber,  
lastname = OLD.lastname,  
changedon = NOW(); END$$  
DELIMITER ;
```

Acceso a Datos

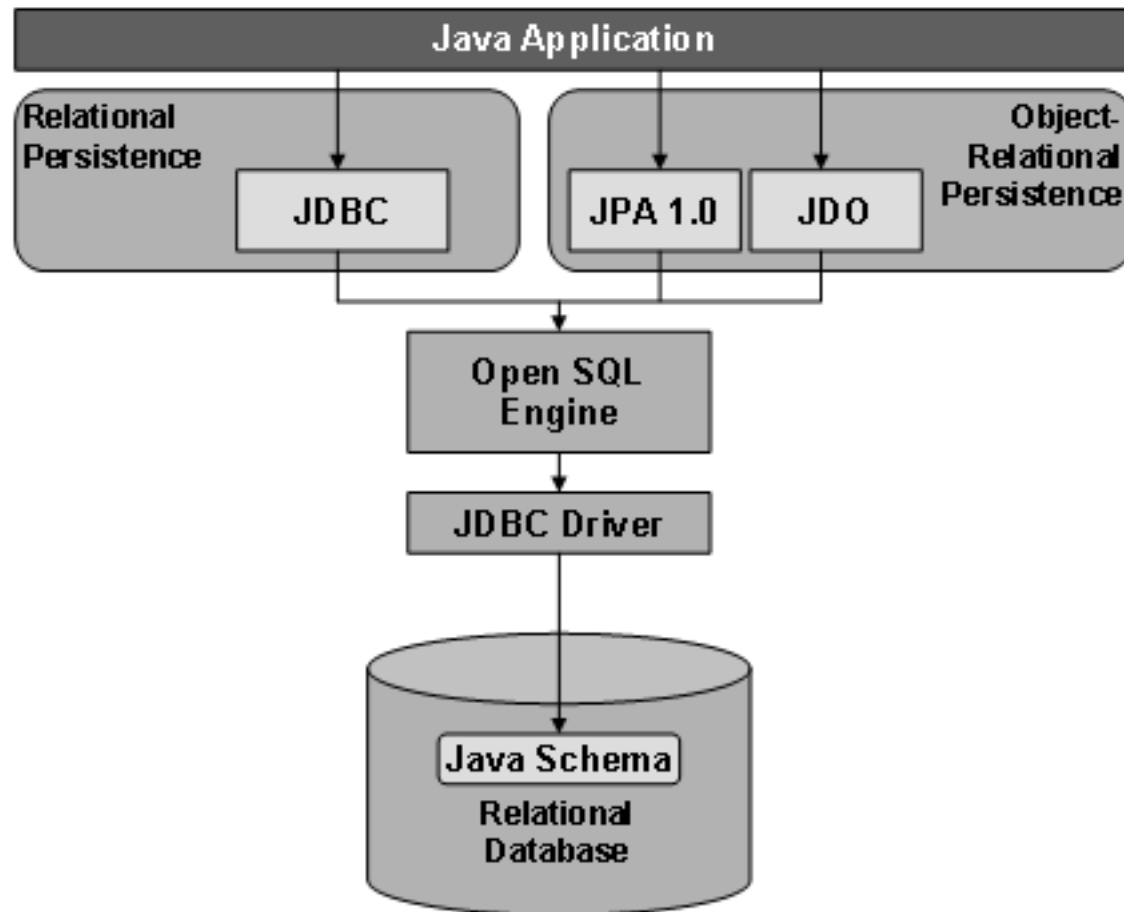
Comunicación y conversión

Dimensión	Subdimensión	Valor
Localidad		Específica al thread, al proceso, global
Acceso		Accesador, mutador
Disponibilidad	Transiente	Registro, Cache, DMA, Heap, Pila
	Persistente	Acceso a repositorio, I/O archivo, intercambio de datos dinámico, acceso a base de datos
Accesibilidad		Privada, protegida, pública
Ciclo de vida		Inicialización, terminación
Cardinalidad	Define	
	Usa	

Acceso a Datos

- Permite a los componentes acceder a fuentes de datos
 - ▣ Diferentes formatos
 - ▣ Diferentes mecanismos
- Conversión:
 - ▣ Traducir la información para eliminar la diferencia de formatos (SQL)
- Conector:
 - ▣ Mutar (cambiar) la información para acceder de forma global a los datos (jDBC, ODBC)

Acceso a Datos (JPA)



Enlace (Binding)

Facilitación

Dimensión	Subdimensión	Valor
Referencia		Implícita, explícita
Granularidad	Unitaria	
	Sintáctica	Variable, procedimiento, función, constante, tipo
	Semántica	
Cardinalidad	Define, usa, provee, requiere	
Enlace		Tiempo de compilación, de ejecución, pre-compilación

Enlace (Binding)

- Enlaza los componentes de un sistema mientras dura una operación
- Puede ser de 3 formas:
 - ▣ Unitaria
 - Un módulo depende de otro
 - ▣ Sintáctica
 - Refina la relación entre módulos tipificando los enlaces
 - ▣ Semántica
 - Define la forma en como los módulos son enlazados (requisitos & restricciones)

Enlace (Binding)

- Un recurso es definido una sola vez e invocado desde varios recursos

- El enlace puede ocurrir:

- Tiempo de pre-compilación (very early)

- **Client client = new Client();**

- Tiempo de compilación (early)

- **Object client = new Client();**

- Tiempo de ejecución (late)

- **Object client = personFactory.createClient();**

Enlace (Binding) - Unitario

- Define la relación entre dos unidades del sistema
 - Unidades, entendidas como componentes (módulos o archivos)
 - Común en relaciones de dependencia
 - $\text{Unit-IM} = (\{\text{units}\}, \{\text{"depends on"}\})$
- Ejemplos
 - Determinar contexto de compilación
 - preprocesador
 - $\text{IM} = \{\text{files}\}, \{\text{"include"}\}$
 - Determinar estrategias de recompilación
 - $\text{IM} = \{\text{compile_units}\}, \{\text{"depends_on"}, \text{"has changed"}\}$
 - Modelar sistemas
 - $\text{IM} = \{\text{systems}, \text{files}\}, \{\text{"is composed of"}\}$

Enlace (Binding) - Unitario

- Interconexiones estáticas
 - A nivel de componentes
- No describen la interacción de los componentes
 - Se centran en el tipo de dependencia

Enlace (Binding) - Sintáctico

- Describe la relación entre “elementos sintácticos” del lenguaje de programación
 - Definición de variable / uso
 - Definición de método / invocación
 - $IM = (\{ \text{método, tipo, variable} \}, \{ \text{“is def at”, “is set at”, “is used at” ...} \})$
- Ejemplos
 - Manejo automatizado de cambios del software
 - Interlisp masterscope
 - Análisis estático
 - Detección de código “inalcanzable” (unreachable code, lanzado por el compilador)
 - “Recompilación inteligente”
 - Recompilar solo los cambios de las unidades
 - “Modelaje de sistemas”
 - Es más “específico” en su granularidad que unit-IM

Enlace (Binding) - Sintáctico

- Interconexiones de granularidad más fina
 - A nivel de objetos individuales
- Interconexiones pueden ser estáticas y dinámicas
- Especificación incompleta de la interconexión
 - Pueden ser sintácticamente válidas, pero no permitidas semánticamente
 - Ordenación de operaciones, transacción de comunicaciones
 - ej: “pop” de un stack vacío
 - “Violación de operaciones semánticas” (operaciones sin sentido)
 - ej: Llamadas a “calendario” para realizar operaciones de cálculo numérico

Enlace (Binding) - Semántico

- Expresar la relación entre los componentes
 - La intención de diseño de cada componente
- Capturar el uso (real) de los componentes del sistema
- La interconexión semántica puede ser especificada formalmente:
 - Pre- & Post- condiciones
 - Protocolos de interacción (CSP, FSM...)
 - IM = (método, tipos, variables... predicados}, {"is set at", "is used at", "calls", ..., "<relación>"})

Enlace (Binding) - Semántico

- Se construye por sobre las interconexiones sintácticas
- Interacciones pueden ser estáticas o dinámicas
- Especificación completa de la interconexión
 - Valida tanto la relación (interconexión) sintáctica como la semántica
- Es necesaria al nivel de arquitectura
 - “Grandes” componentes
 - Interacciones complejas
 - Heterogeneidad (de interconexiones)
 - Reusabilidad de componentes
- Se desprenden propiedades de la interacción
 - Robustez
 - Seguridad
 - Disponibilidad
 - “reliability”

Streams

Comunicación

Dimensión	Subdimensión	Valor
Entrega		Mejor esfuerzo, exactamente uno, a lo más uno, al menos uno
Límites		Acotado, sin cota
Buffering		Almacenado en buffer, sin buffer
Throughput		Unidades atómicas, de orden superior
Estado		Sin estado, con estado
Identidad		Nombrada, sin nombre
Localidad		Local, remoto
Sincronicidad		Síncrono, asíncrono, sincronía con timeout
Formato		Raw, con estructura
Cardinalidad	Binaria	
	N-aria	Multi-emisor, multi-receptor, multi emisor/receptor

Stream Connectors

- Utilizados para realizar transferencia de grandes cantidades de datos entre procesos autónomos
- Ejemplos:
 - ▣ Sockets TCP/UDP
 - ▣ Protocolos Cliente-Servidor
- Se pueden combinar con otros conectores:
 - ▣ Data Access connectors: Acceso a bases de datos y repositorios de archivos
 - ▣ Event connectors: Multiplexar la entrega de un gran número de eventos

Stream Connectors

- Algunas opciones
 - ▣ Puede proveer transferencia síncronas o asíncronas
 - ▣ Puede garantizar al menos una entrega
 - ▣ Su cardinalidad puede ser binaria (un sender y un receiver) o teniendo multi sender y multi receiver
 - ▣ Puede ser
 - Statefull: Guarda información sobre la conexión entre el sender y el receiver. Ej: Conexión TCP
 - Stateless: No se guarda información, ni por el sender ni por el receiver
 - El sender envía el paquete y no espera confirmación del receiver
 - El receiver recibe el paquete sin previamente configurar la conexión

Árbitro

Coordinación y facilitación

Dimensión	Subdimensión	Valor
Manejo de fallas		Con autoridad, por votación
Concurrencia	Mecanismo	Semáforo, rendezvous, monitor, lock
	Peso	Ligero, pesado
Transacciones	Anidación	Simple, múltiple
	Awareness	Ninguno, soportado, requerido, nuevo
	Aislamiento	Lee, escribe, lee/escribe
	Autenticación	
Seguridad	Autorización	Capacidades, lista de control de acceso
	Privacidad	Encriptación, padding, screening
	Integridad	Verificación de redundancia, certificados
	Durabilidad	Sesión única, sesión múltiple
Planificación	Tiempo	
	Peso	

Manejo de fallas

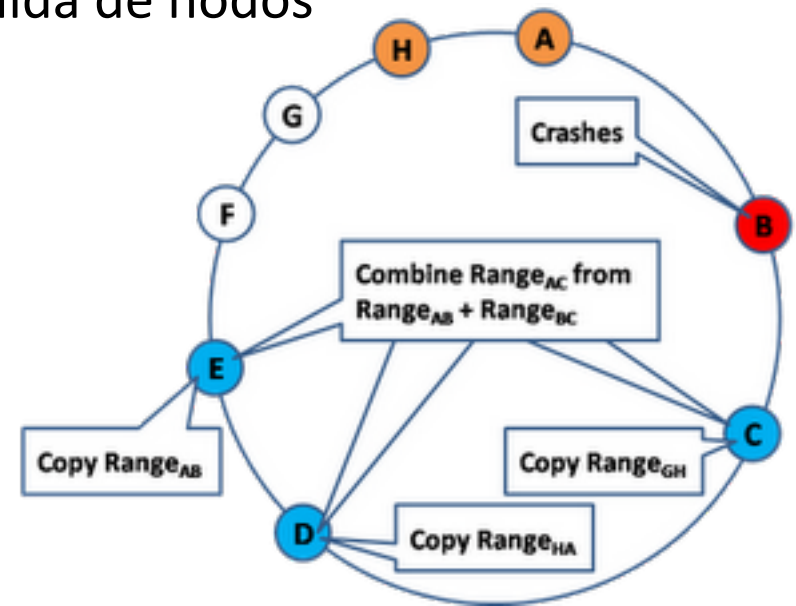
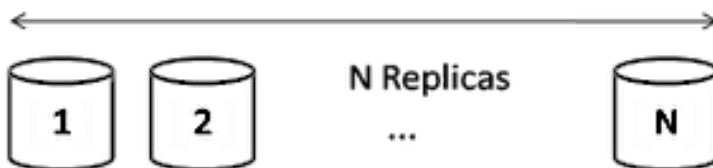
- **Tolerancia a fallas, degradación graciosa (graceful degradation) o mejora progresiva**, son enfoques que permiten que un sistema siga operando correctamente a pesar de la falla de uno o varios de sus componentes.
 - ▣ **Replicación**
 - Un componente se replica y una de sus instancias se elige como valor válido
 - ▣ **Redundancia**
 - Un componente se replica y en caso de falla se elige una de las instancias activas (failover)

Manejo de fallas

- Replicación
 - ▣ Se soportan varias instancias del mismo sistema o subsistema (hardware, memoria, cpu, datos, etc.)
 - ▣ Se dirigen las tareas o solicitudes a cada instancia en paralelo
 - ▣ La respuesta correcta se elige por votación
- ▣ Ejemplos: Bases de datos NoSQL de escalabilidad masiva
 - GoogleBigTable, HBase, Hypertable, CouchDB, MongoDB, Voldemort, etc...

Manejo de fallas

- La DB se replica en cientos o miles de máquinas (PCs) en red, con configuración similar
- Cada máquina aloja n nodos virtuales (VN)
- Una tabla de hash conteniendo direcciones (*membresía*) de cada VN se distribuye entre los VNs
- Crítico: Hash consistente, entrada/salida de nodos
- Consistencia:
 - *2 Phase Commit* basado en quorum

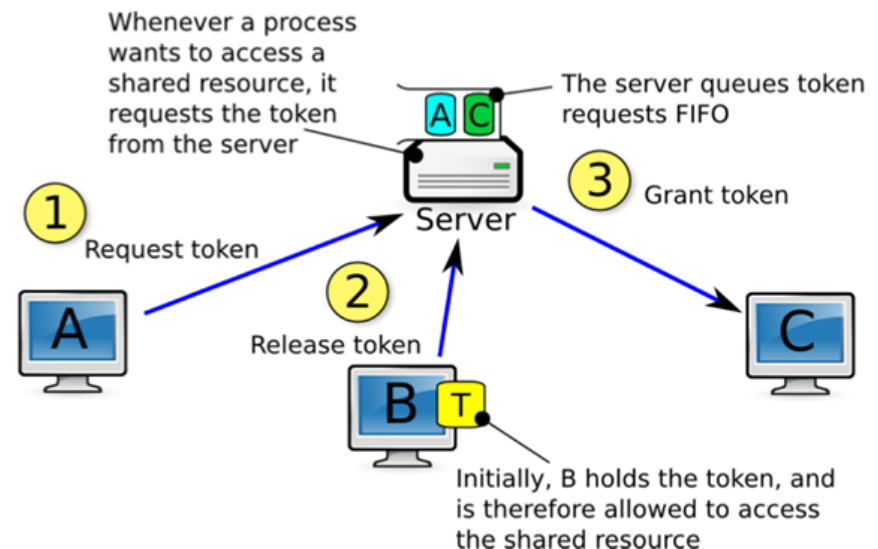


Concurrencia

- Los programas son procesos que interactúan y *podrían* ejecutarse en paralelo (secuencialmente en un solo procesador, multicore, distribuidos sobre una red)
- Problema:
 - ▣ Asegurar la secuencia correcta de interacciones o la secuencia correcta de intercambio de mensajes entre los procesos
 - ▣ Coordinar el acceso a recursos compartidos por los procesos

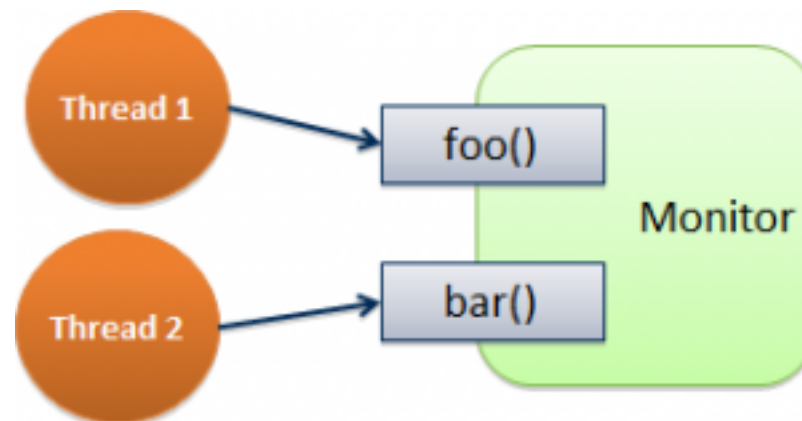
Concurrencia sobre memoria compartida

- **Semáforo** es una variable o tipo abstracto de datos que permite controlar el acceso a un recurso compartido.
 - ▣ Contadores (número de recursos disponibles)
 - ▣ Binarios (locks o mutexes)



Concurrencia sobre memoria compartida

- **Monitor** es un objeto o módulo que garantiza la seguridad de uno o varios threads, puesto que sus métodos se ejecutan con exclusión mutua
 - ▣ A lo más un thread puede ejecutar sus métodos en determinado tiempo



Seguridad

- Capacidades
 - ▣ Una capacidad o llave (key) es un *token* (estructura de datos) de autoridad comunicable y no falsificable
 - ▣ El token se a un objeto y describe un conjunto de derechos de acceso sobre ese objeto

```
/etc/passwd  
O_RDWR
```

- Access Control List (ACL)
 - ▣ Especifica que usuarios o procesos de sistema tienen acceso a objetos, así como la lista de operaciones permitidas sobre los objetos

Encriptación: MD5

- MD5 Message Digest
 - ▣ Función de hash criptográfica que produce un valor de 128 bits (16 Byte, número de 32 dígitos). RFC 1321
 - ▣ Puede presentar colisiones (no se puede usar en certificados SSL o firmas digitales únicamente)

Mac OS X:

```
$ md5 sample.xlsx  
MD5 (sample.xlsx) = 14b2b2e14afc6456705dc1e3010591f1
```

```
<?php  
    $md5 = md5(utf8_encode('Städte'));  
?>
```

Encriptación: sha1sum

- ▣ Programa que calcula y verifica hashes SHA-1
- ▣ Se usa para verificar la integridad de los archivos

Mac OS X

```
$ openssl sha1 sample.xlsx
```

```
SHA1(sample.xlsx)= 2d4c223e6eac0fe38a13ced3cbbb9aa58e43e4ea
```

```
<?
```

```
public function hashSSHA($password)  
{    $salt = sha1(rand());  
    $salt = substr($salt, 0, 4);  
    $hash = base64_encode(  
        sha1($password . $salt, true) . $salt );  
    return $hash;  
}  
?>
```


Planificación (scheduling)

- Procesos deben ejecutarse en cierto tiempo específico o a intervalos regulares
- Java Timer API, Quartz
- Temas:
 - ▣ Cancelar tareas
 - ▣ Excepciones
 - ▣ Tareas concurrentes
 - ▣ One shoot
 - ▣ Triggers

Scheduler

```
public static void main(String[] args) {
    Timer timer  new Timer();
    Calendar date = Calendar.getInstance();
    date.set(
        Calendar.DAY_OF_WEEK,
        Calendar.SUNDAY
    );
    date.set(Calendar.HOUR, 0);
    date.set(Calendar.MINUTE, 0);
    date.set(Calendar.SECOND, 0);
    date.set(Calendar.MILLISECOND, 0);
    // Schedule to run every Sunday in midnight
    timer.schedule(
        new ReportGenerator(),
        date.getTime(),
        1000 * 60 * 60 * 24 * 7
    );
}
```

Adaptador

Conversión

Dimensión	Subdimensión	Valor
Conversión de la invocación	Mapeo de direcciones	
	Marshalling	
	Traducción	
Conversión de paquete		Wrapper
		Empaquetadores
Conversión de protocolo		
Conversión de presentación		

Marshalling

- Serializar:
 - Convertir el estado (valores de los atributos) de un objeto en un stream de bytes (**java.io.Serializable**) de manera que luego pueda volver a convertirse en una copia del objeto original
- Marshalling
 - Sinónimo
 - En Java: Se registra el estado (atributos) y el código (métodos) de manera que cuando el objeto es “unmarshalled” una copia del objeto original se obtiene cargando la definición de la clase. El objeto a transferir es Remoto (**java.rmi.Remote**) y serializable

Serialize

```
import java.io.Serializable;
public class Person implements Serializable
{   private String name;

    public Person(String name)
    {   this.name = name;   }

    public String getName()
    {   return name;   }
}
```

Marshalling

```
JAXBContext jc =  
    JAXBContext  
        .newInstance( "com.acme.foo" );  
  
Unmarshaller u = jc.createUnmarshaller();  
  
FooObject obj = (FooObject)u.unmarshal( new  
    File( "foo.xml" ) );  
  
Marshaller m = jc.createMarshaller();
```

Ejemplo: Traducción ... de XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  <cd>
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <country>UK</country>
    <company>CBS Records</company>
    <price>9.90</price>
    <year>1988</year>
  </cd>
</catalog>
```

Traducción (XSLT)

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html> <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32"> <th>Title</th>
                                <th>Artist</th></tr>
      <xsl:for-each select="catalog/cd">
        <tr> <td><xsl:value-of select="title"/></td>
              <td><xsl:value-of select="artist"/></td> </tr>
      </xsl:for-each>
    </table>
  </body> </html>
</xsl:template>

</xsl:stylesheet>
```


XSLT en el lado del servidor (Java)

```
public void transform(String xmlfile, String xsltfile)
{ File xmlFile = new File(xmlfile); //validar ok
  File xsltFile = new File(xsltfile); //validar ok

  try { Source xmlSource = new StreamSource(xmlFile);
        Source xsltSource = new StreamSource(xsltFile);

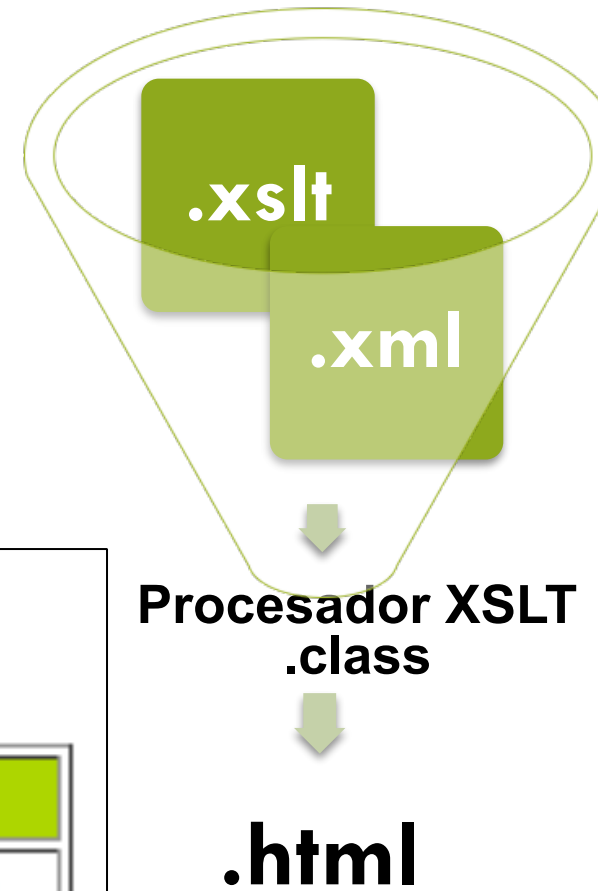
        ByteArrayOutputStream out =
            new ByteArrayOutputStream();
        Result result = new StreamResult(out);
        TransformerFactory transFact =
            new net.sf.saxon.TransformerFactoryImpl();
        javax.xml.transform.Transformer trans =
            transFact.newTransformer(xsltSource);
        trans.transform(xmlSource, result);
        System.out.println( out.toString() );
    } ...
```

Traducción ... resultado HTML

```
<html>
<body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32"> <th>Title</th>
                                <th>Artist</th></tr>
    <tr> <td>Empire Burlesque</td>
        <td>Bob Dylan</td> </tr>
    <tr> <td>Hide your heart</td>
        <td>Bonnie Tyler</td> </tr>
  </table>
</body>
</html>
```

My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler



Distribuidor

Facilitación

Dimensión	Subdimensión	Valor
Nombramiento	Estructurado	Jerárquico
	Basado en atributos	Plano
Entrega	Semántica	Mejor esfuerzo, exactamente uno, a lo más uno, al menos uno
	Mecanismo	Ninguno, soportado, requerido, nuevo
Ruteo	Membresía	Acotada, ad-hoc
	Ruta	Estática, cacheada, dinámica

Componiendo Conectores Básicos

- No todos los conectores pueden componerse
 - Algunos son naturalmente interoperables
 - Algunos son incompatibles
 - Todos presentan trade-offs
- Análisis de Taylor sobre conectores



Requiere – Una dimensión obliga el uso de la otra



Prohibe – Dos dimensiones no pueden componerse en un único conector



Restringe – Las dimensiones no siempre se usan juntas, algunas combinaciones pueden ser inválidas



Precaución – Las combinaciones pueden ser inestables o no confiables

Recomendaciones (Taylor)

		Procedure call			Event			Data access			Stream			Linkage			Arbitrator					Adaptor			Distributor		
		Availability	Accessibility	Cardinality	Delivery	Format	Directionality	Cardinality	State	Granularity	Cardinality	Resolution	Fault handling	Concurrency	Transactions	Logging	Security	Scheduling	Pooling	Invocation	Data	Presentation	Deployment	Naming	Delivery	Routing	
Procedure call		&	&																								
			&																								
Event		&	&																								
		&	&																								
		&	&																								
		&	&																								
Data access	Availability																										
	Accessibility																										
	Cardinality																										
Stream	Delivery																										
	Format																										
	Directionality																										
	Cardinality																										
Linkage	State																										
	Granularity																										
Arbitrator	Cardi																										
	Reso																										
	Fault																										
	Conc																										
Adaptor	Trans																										
	Loggi																										
	Secur																										
	Sched																										
Distributor	Poolin																										
	Invoc																										
	Data																										