

# Coordinación de Componentes mediante Conectores Implementados con Servicios Web



*J.L. Pastrana* ([pastrana@lcc.uma.es](mailto:pastrana@lcc.uma.es))

*M. Katrib* ([mkm@matcom.uh.cu](mailto:mkm@matcom.uh.cu))

*E. Pimentel* ([ernesto@lcc.uma.es](mailto:ernesto@lcc.uma.es))

# Introducción

- **Arquitecturas Software**

- **Relaciones de Implementación: cómo realizan los componentes su computación**



**COMPONENTE**

- **Relaciones de Interacción: cómo se combina la computación en todo el sistema**



**CONECTOR**

# ¿Qué Proponemos?

- **Implementación Real del concepto de Conector para Sistemas Abiertos, Basados en Componentes y Distribuidos**
- **Mejoras sobre el concepto “Tradicional” de Conector:**
  - **Uso del Diseño por Contrato en Conectores**
  - **Conectores Activos (con comportamiento)**
  - **Incorporar Requisitos No Funcionales**
  - **Posibilidad de uso de propiedades de Tiempo Real**
  - **Adaptación Dinámica**

# Características de Los Conectores

- **Orientados al Cliente :** Son definidos por los clientes como extensión de la interfaz del servidor.
- **Son Servicios Web**
- **Generados automáticamente.**
- **Usan contratos para establecer los requisitos no funcionales que desea el cliente.**
- **Permiten que un cliente solicite el uso en exclusión mutua de un servidor.**
- **Permiten que cada cliente tenga un comportamiento no funcional distinto (si lo desea) de un mismo servidor.**
- **Permiten que sean enriquecidos tanto estática como dinámicamente con una ontología del dominio del problema (en formato XML) con el fin de resolver problemas de adaptación de servicios.**

# Definición de Conectores

PFC Enriquecedor de Interfaces de Webservices

Archivo Ayuda

Abrir fichero Abrir URL Abrir definición Guardar definición

Servicios Invariantes

**Propiedades**

Nombre:

Connect to:

External:

**Invariante**

Invariant:

On failure:

Actualizar

9 servicios WSDL: http://150.214.108.138/ChessMaster/Ch Definición: Sin título

PFC Enriquecedor de Interfaces de Webservices

Archivo Ayuda

Abrir fichero Abrir URL Abrir definición Guardar definición

Servicios Invariantes

**WebService**

**Precondición**

Require:

On failure:

**Postcondición**

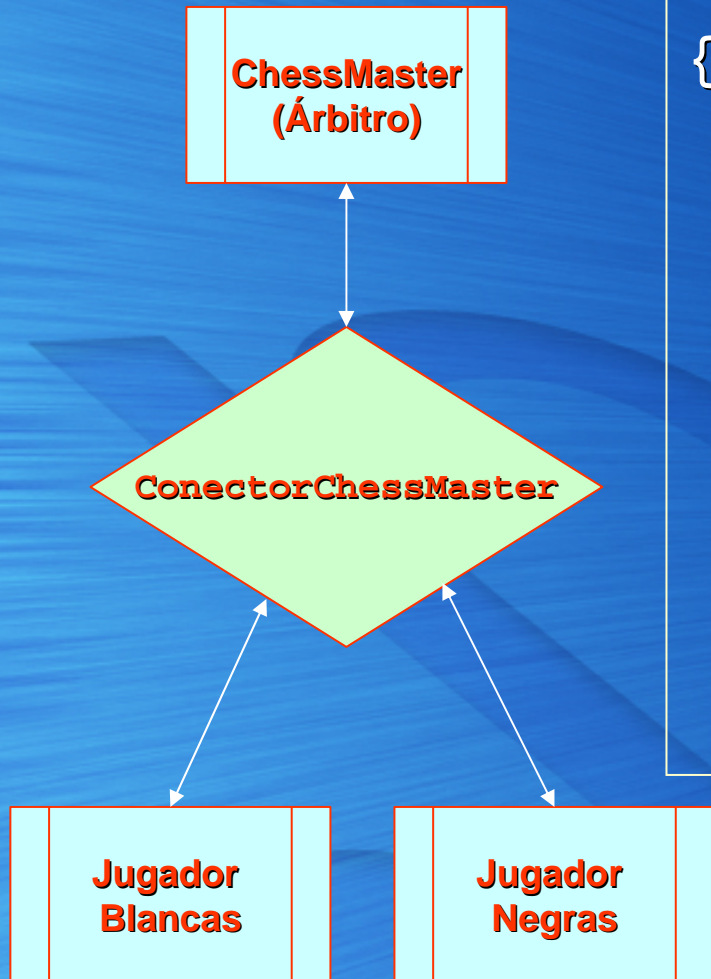
Ensure:

On failure:

Actualizar

9 servicios WSDL: http://150.214.108.138/ChessMaster/Ch Definición: Sin título

# Ejemplo: ChessMaster



```
interface ChessMaster
```

```
{  
    . . .
```

```
    bool ValidMovement(string mov);
```

```
    void SendMovement(int who,  
                      string move);
```

```
    string GetMovement(int who);
```

```
    bool IsMyTurn(int who);
```

```
    void ResignGame(int who);
```

```
    . . .
```

```
    string GetMovement(int who)
```

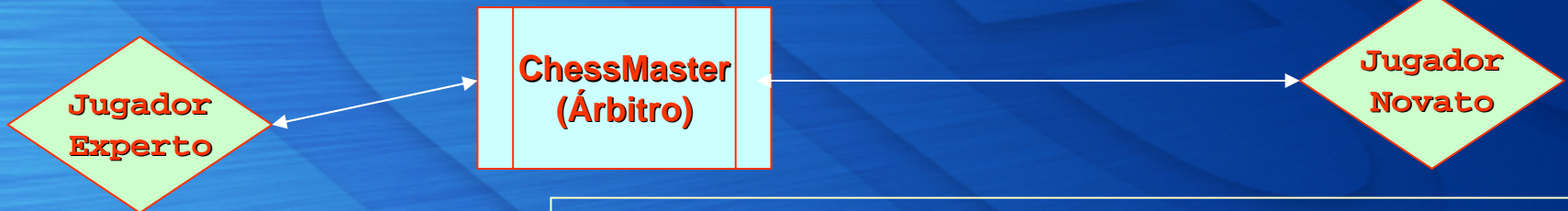
```
    require: (!IsMyTurn(who))
```

```
    on failure: Suspend;
```

# Ejemplo: ChessMaster

```
void SendMovement(int who, string move)
require:      (IsMyTurn(who)) &&
              (ValidMovement(mov))

on failure:
  if (!ValidMovement(mov)) ResignGame(who);
  else                      Suspend;
```

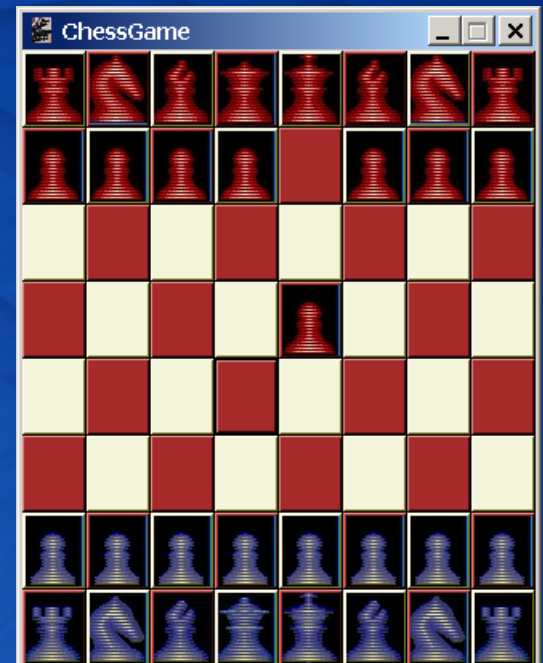
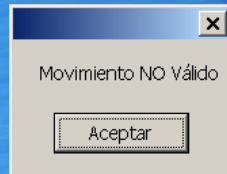
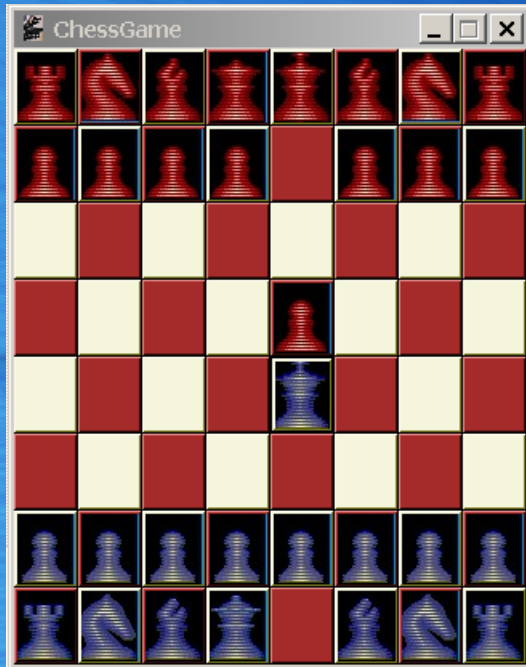


```
void SendMovement(int who, string move)
require:      (IsMyTurn(who)) &&
              (ValidMovement(mov))

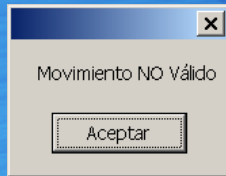
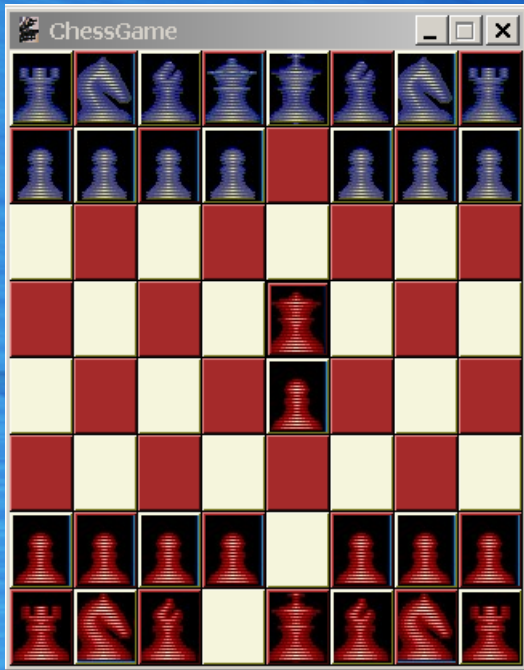
on failure:
  if (!ValidMovement(mov))  Fail;
  else                      Suspend;
```

# Ejemplo: ChessMaster

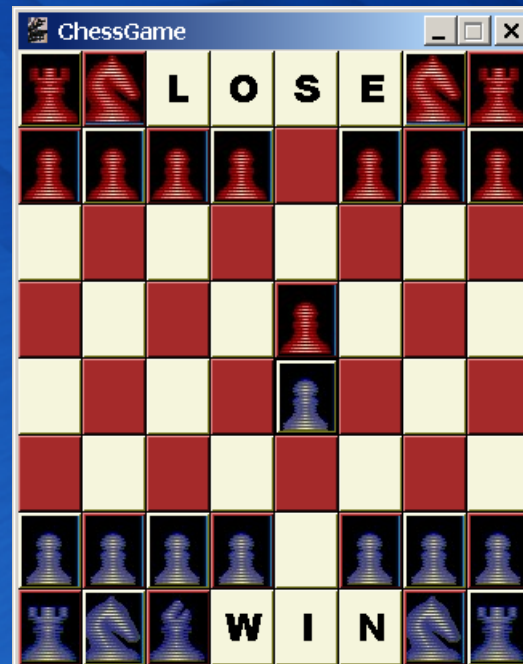
El jugador Novato (negras) intenta un movimiento de apertura No Válido (E8-E5)



# Ejemplo: ChessMaster



El jugador Experto (blancas) intenta un movimiento No Válido (D1-E5)



# Adaptación Dinámica

Analizando Posibles situaciones podríamos resumir en 3 tipos de conflictos:

1. Conflicto de Nombre. Dos servicios que tienen la misma funcionalidad y argumentos pero nombres distintos. Por ejemplo: **SendMovement** y **EnviaMovimiento**.

2. Conflicto de Parametros. Parámetros con tipos y/u orden diferente pero que suministran la misma información. Por ejemplo:

```
void SendMovement(int who, string move)
```

```
void SendMovement(char[] m, long me).
```

3. Conflicto de Composición. Un servicio puede modelarse como la composición de dos o más servicios (o viceversa). Por ejemplo:

```
void SendMovement(int who, string move)
```

```
void ListenToMe(int who) + void NewMovement(string move)
```

# Implementación de Conectores

- Los Conectores han sido implementados mediante servicios web.
- Un Servicio Web no es muy diferente de una aplicación tradicional cliente-servidor: Un cliente necesita datos o un servicio y para ello realiza invocaciones al servidor.
- Los Servicios Web utilizan tecnologías, lenguajes y protocolos estándar como son HTTP, XML, SOAP.
- Los Servicios Web son componentes sin estado, debido a la naturaleza desconectada de las aplicaciones web.
- Los Conector propuestos precisan estado: referencias remotas, cola de llamadas suspendidas, etc.
- Solución: el diccionario `Application` de la clase `HttpApplication`

```
// Application CBehavior beh;  
  
CBehavior beh // Variable translated to HttpApplication  
{  
    get { return (CBehavior)Application["beh"];}  
    set { Application["beh"] = value; }  
}
```

# Implementación de Conectores

- Los Contratos se Transforman en Métodos Lógicos:

```
public bool SendMovement_precondition(int who, string move)
{
    bool _result;
    _result = (IsMyTurn(who)) && (ValidMovement(move));
    return _result;
}
```

- Las Sentencias de Comportamiento se Transforman en Métodos:

```
public void SendMovement_on_failure_precondition(int who,
                                                string move)
{
    if (!ValidMovement(move)) ResignGame(who);
    else                        beh.Suspend();
}
```

# Implementación de Conectores

- Todo servicio se gestiona de forma homogénea:

[WebMethod]

```
public void SendMovement( int who , string move )
{
    object[] parametres = new object[2];
    parametres[0] = who ;
    parametres[1] = move ;
    this.doCall(server,MethodBase.GetCurrentMethod().Name
               ,parametres);
}
```

# Implementación de Conectores

- Y la gestión del comportamiento se hace a través de reflexión:

```
public object doCall(System.Web.Services.Protocols.  
                    SoapHttpClientProtocol server,  
                    string name , object[] parameters)  
{  
    . . .  
    string serviceName = ontology.getName(name, server);  
    MethodInfo remoterService=remoteType.GetMethod(serviceName);  
    MethodInfo preService = connectorType.GetMethod(name  
        + "_precondition");  
    MethodInfo postService = connectorType.GetMethod(name  
        + "_postcondition");  
    MethodInfo on_failure_pre = connectorType.GetMethod(name  
        + "_on_failure_precondition");  
    MethodInfo on_failure_post= connectorType.GetMethod(name  
        + "_on_failure_postcondition");
```

# Implementación de Conectores

```
if (!invariant())    on_failure_invariant();  
if (!(bool)preService.Invoke(this,parameters))  
    on_failure_pre.Invoke(this,parameters);  
  
if (availableFlag) Monitor.Enter(server);  
_result = remoterService.Invoke(server,parameters);  
if (availableFlag) Monitor.Exit(server);  
  
if (!(bool)postService.Invoke(this,parameters))  
    on_failure_post.Invoke(this,parameters);  
if (!invariant()) on_failure_invariant();  
beh.wake_up();  
return _result;  
}
```

¿Dudas, Preguntas?

**Muchas  
Gracias**

*J.L. Pastrana*

[pastrana@lcc.uma.es](mailto:pastrana@lcc.uma.es)

