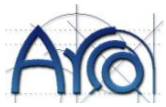
Infraestructura Hw-Sw para la Gestión Uniforme de las Comunicaciones en Sistemas en Chip

Grupo de Arquitectura y Redes de Computadores



Escuela Superior de Informática
Universidad de Castilla-La Mancha

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Juan Carlos López
juancarlos.lopez@uclm.es

Contenidos

- Motivación
- Sistemas distribuidos heterogéneos
- El SoC como Sistema Distribuido
- El concepto de middleware
  - RMI
  - Transparencia
- La Arquitectura de Comunicaciones a nivel de sistema
- Flujo unificado de diseño: metodología
- Ejemplos, resultados y aplicaciones
- Conclusiones

Motivación

- Aplicaciones actuales (multimedia, computación móvil, computación grid, ambient intelligence...)
  - Elementos distribuidos HW y SW que interactúan unos con otros (on-chip y off-chip)
- Complejidad de los sistemas (SoC)
  - Nuevas metodologías que favorezcan:
    - Reuso de modelos y componentes
    - Prototipado rápido
    - Exploración del espacio de diseño
    - Test
    - ...

¿Qué es un SD?

Un sistema distribuido es una **colección de dispositivos** de cómputo que pueden **comunicarse** entre sí-, [. . .] desde un chip VLSI [. . .], hasta un cluster de estaciones de trabajo en una red de área local.

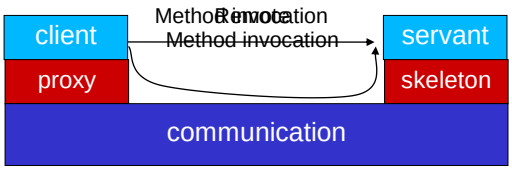
H. Attiya & J. Welch (1998)

Definimos sistema distribuido como una **colección** de **computadores autónomos conectados** mediante una red, con software diseñado para producir una instalación informática **integrada**.

G. Coulouris, J. Dollimore & T. Kindberg (1994)

El paradigma de objeto distribuido

- Arquitectura
  - Elementos de computación heterogéneos
  - Unidos por una infraestructura de comunicaciones
- Modelo de programación: paradigma orientación a objeto
  - Todas las entidades del sistema son objetos
  - Objectos se comunican por paso de mensajes (invocación de métodos, MI)
  - MI se extiende a RMI para objetos remotos



Diseño de SoC's

- El diseño de SoC's complejos requiere el manejo de:
  - Heterogeneidad HW-SW
  - Modelos de diferentes de programación
  - Gran cantidad de componentes
- Por otra parte los diseñadores de aplicaciones demandan mayor transparencia sobre la función que se necesita:
  - Localización : Dónde reside
  - Implementación: Cómo está implementado (lenguaje, O.S., Hw/Sw).
  - Estado de ejecución: está activo?
  - Mecanismo de comunicación: Cómo se realiza la comunicación (mem. compartida, llamada procedimiento local o remoto, red)

El SoC como Sistema Distribuido

- El SoC es un sistema distribuido heterogéneo
  - Elementos de computación: procesadores (SW), cores (HW)
  - Comunicación
- Componentes:
  - Relativa autonomía entre ellos
  - No necesariamente homogéneos (i.e. Bloques IP)
- Conexiones:
  - Punto a punto
  - Buses o jerarquía de buses
  - Redes en chip (NoC)
- Funcionamiento integrado
  - ¡Son parte del mismo chip!

El SoC como Sistema Distribuido

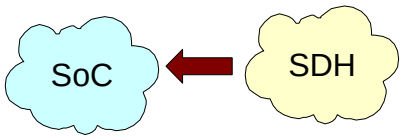
- Componentes HW pueden considerarse objetos, porque:
  - Tienen estado
  - Se encapsulan a través de una entidad
  - El estado se manipula mediante operaciones de la entidad
  - Aunque hay importantes diferencias con los objetos SW
    - Creación/destrucción en tiempo de ejecución
    - *Dynamic binding*
  - Pero desde el punto de vista del diseñador
    - Cualquier componente con una clara separación entre funcionalidad y estado es un objeto

El SoC como sistema distribuido

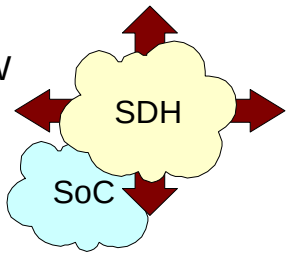
- Así pues, todas las entidades del sistema se consideran componentes distribuidos
- La responsabilidad de la comunicación se ha de pasar a otra parte
  - comunicación transparente
- Desde el punto de vista lógico sólo es necesario conocer la interfaz de un componente para utilizarlo

Aprovechar la analogía

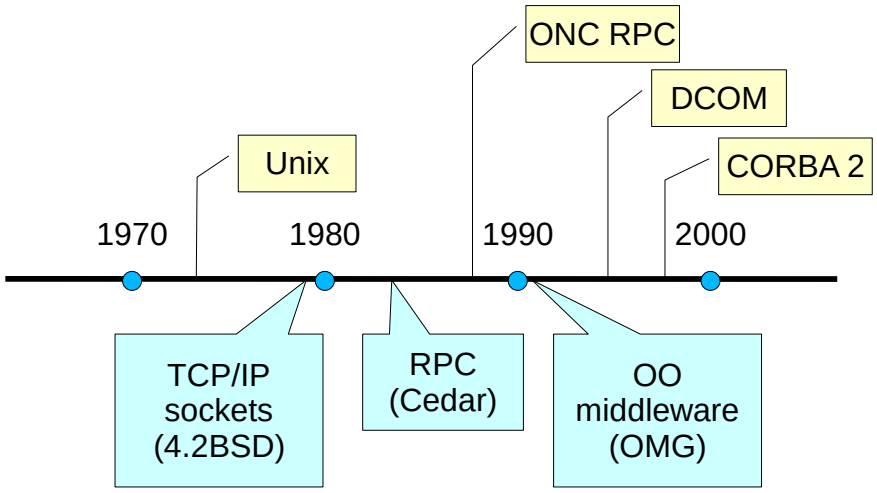
- Buscar en SDH respuesta a problemas en SoC



- Adaptar SDH estándar a SoC
  - Interoperabilidad sencilla entre HW y SW
  - ¿Demasiada sobrecarga?
  - ¿Todos los servicios tienen sentido?



La evolución de los SDH



El boom de los objetos

- Desde ~1995 multitud de arquitecturas para SDH
  - DCOM, Microsoft
  - CORBA, Object Management Group
  - JavaRMI, Jini y EJB, Sun Microsystems
  - WebServices, W3C
  - .NET Remoting, Microsoft
  - ICE, ZeroC Inc.

Middleware de comunicaciones

Estrato software que proporciona una **abstracción** de programación así como un enmascaramiento de la **heterogeneidad** subyacente de las *redes*, *hardware*, *sistemas operativos* y *lenguajes de programación*

¿Qué problemas aborda?

- Heterogeneidad
  - Compromete reusabilidad
- Transparencia
  - Ocultar al usuario la existencia de multitud de componentes: mostrar como un todo
- Modelos de programación
  - No es fácil adaptar una aplicación centralizada
- Prueba y validación del sistema
  - Simular el entorno distribuido es más complejo que la mayoría de las aplicaciones
  - Limitaciones de algoritmos de depuración distribuida

Heterogeneidad

- Red de comunicaciones
  - Adaptación de tasas de transferencia
  - Protocolos
  - Gestión de errores
- Arquitectura del µP
  - Formato de datos
  - Alineamiento
- Lenguajes de programación
- Sistemas operativos

¡Problemas aplicables a SW, SoC, NoC!

¿Dónde nunca habrá acuerdo?

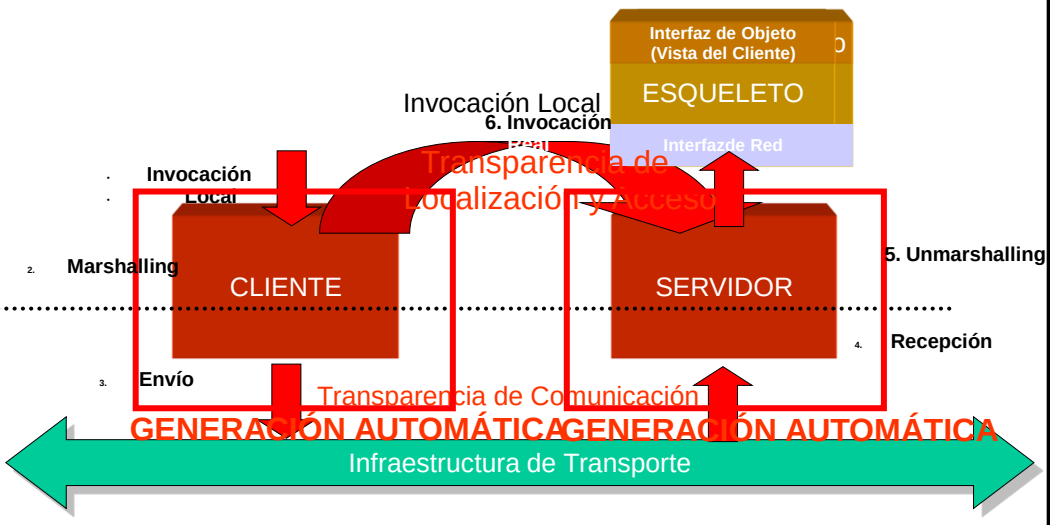
- No habrá acuerdo en el hardware
- No habrá acuerdo en el sistema operativo
- No habrá acuerdo en los protocolos
- No habrá acuerdo en el lenguaje de programación

¡Tiene que haber acuerdo en los interfaces y en los mecanismos de interoperabilidad!

Transparencia de...

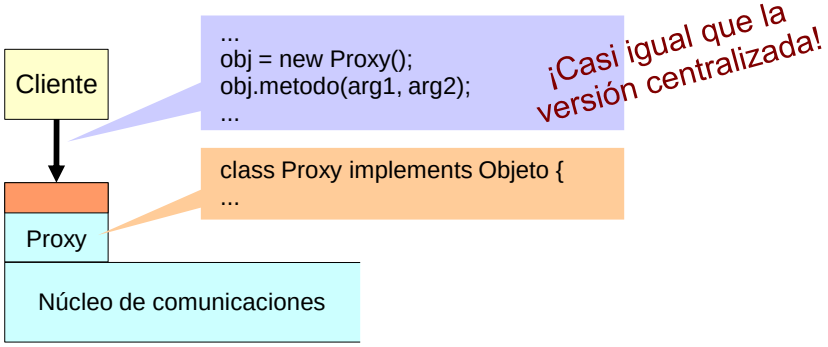
- Localización
  - Independencia de **dónde** reside el objeto
- Implementación
  - Independencia de **cómo** se implementa (lenguaje, OS, arquitectura, ...)
- Estado de ejecución
  - Activación implícita de objetos
- Red de comunicaciones
  - Cualquier mecanismo es posible
    - Shared mem, NoC, bus, ...

Invocación a Método Remoto (RMI)



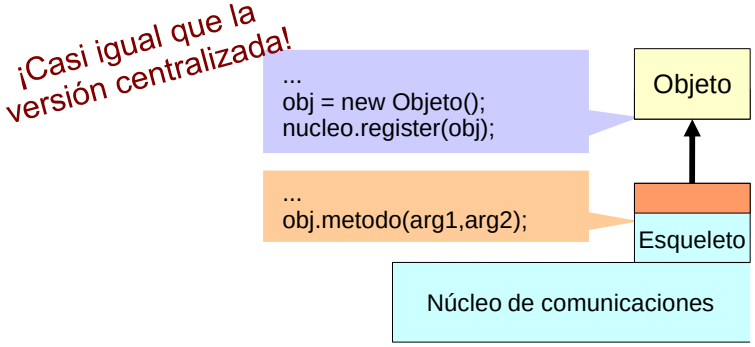
Programación del cliente

- El cliente utiliza un objeto Proxy con el mismo interfaz que el destino, que actúa de intermediario



Programación del servidor

- El servidor utiliza un Esqueleto que traduce los eventos de la red a invocaciones sobre el objeto



Proxy/Esqueleto: Responsabilidades

- Proxy es responsable de
  - Codificar invocación y argumentos en un mensaje
  - Esperar respuesta y decodificar valor de retorno
- Esqueleto es responsable de
  - Esperar invocación y decodificar método y argumentos
  - Invocar el método real
  - Codificar el valor de retorno en mensaje de respuesta

Codificando datos para la red

- Los mensajes son estructuras lineales mientras que los datos pueden ser jerárquicos o incluso cíclicos
- Un mismo tipo de datos puede tener diferentes representaciones
  - little-endian/big-endian, 32bits/64bits, relleno entre campos de una estructura, ...
- La solución:
  - Definir una **Representación Externa** canónica
  - Transformar formato binario a/desde la RE
    - **marshalling** y **unmarshalling**

Y además, las comunicaciones

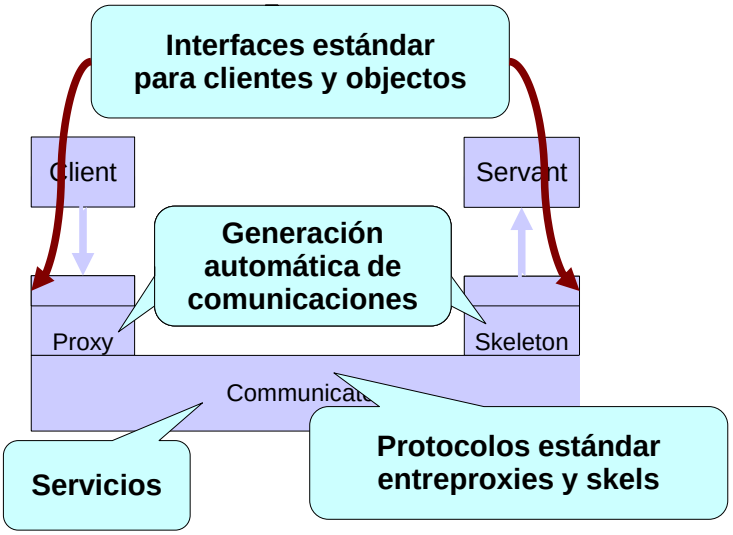
- Identificar mensajes
  - Números de secuencia, filtrado de duplicados, mensajes de reconocimiento, ...
- Gestionar retransmisiones
  - Timeouts, ...
- Gestión de conexiones (si es orientado a conexión)
  - Conexiones, reconexiones, análisis de conectividad
- Identificar objetos
  - Asignar puertos, direcciones, encaminar mensajes, ...

¡Necesitamos un *middleware!*

- El programador de aplicaciones no tiene que ser un *gurú* de las redes, sino un experto en el dominio de la aplicación
- Para un entorno determinado el código de *proxies* y esqueletos es muy similar

Solución: Un núcleo de comunicaciones genérico y un generador automático de proxies y esqueletos

¿Qué proporciona el middleware?

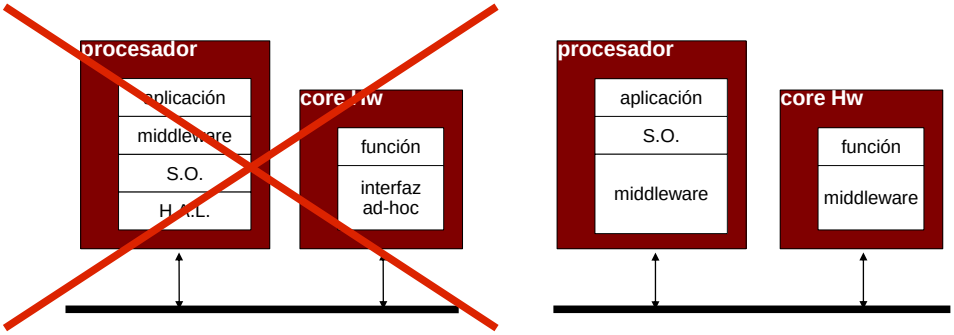


Nuestra propuesta

- Middleware a nivel de sistema
  - Principal preocupación: interoperabilidad
    - Interoperable con middlewares SW estándar
  - Sin interferir el flujo de diseño
  - Sin imponer un tipo particular de arquitectura
  - Los diseñadores pueden usar un subconjunto de sus características para evaluar el compromiso flexibilidad-prestaciones
- Vista integrada de todo el sistema
  - ¡Hay middleware HW y SW de bajo peso también!
  - Interoperación transparente de todos los objetos, HW y SW

El papel del middleware en SoC's

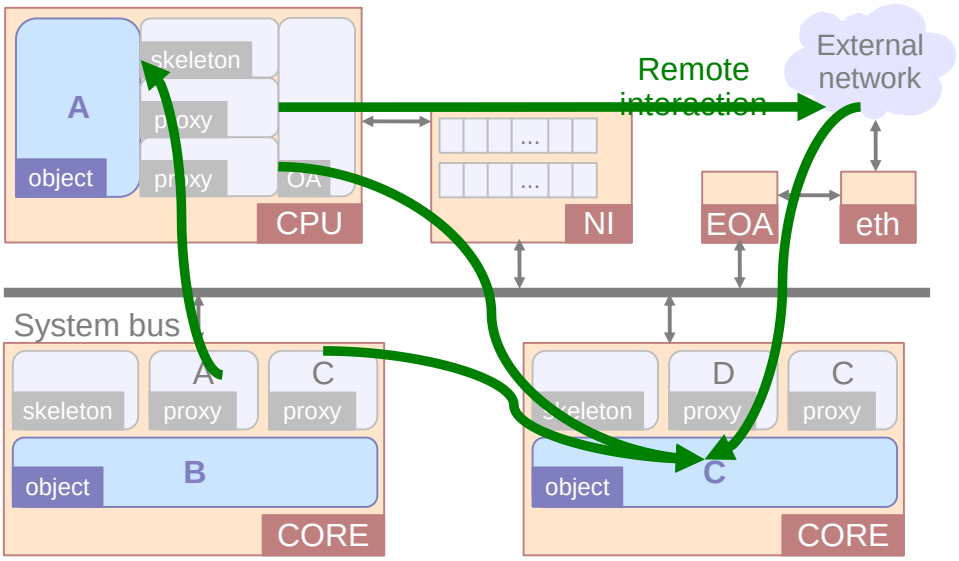
- Necesidad de mecanismos eficientes de comunicación entre cualquier tipo de componente
- El middleware se desplaza a la capa inmediatamente superior al Hw



Objetos HW

- Encapsulado mediante una entity
- Estado interno modificado con operaciones
- Interfaz bien conocido
- Pero el hardware es estático
  - No dynamic binding
  - No creación/destrucción dinámica (?)
- Cualquier componente con separación clara de estado y funcionalidad es un objeto

Arquitectura global de comunicaciones

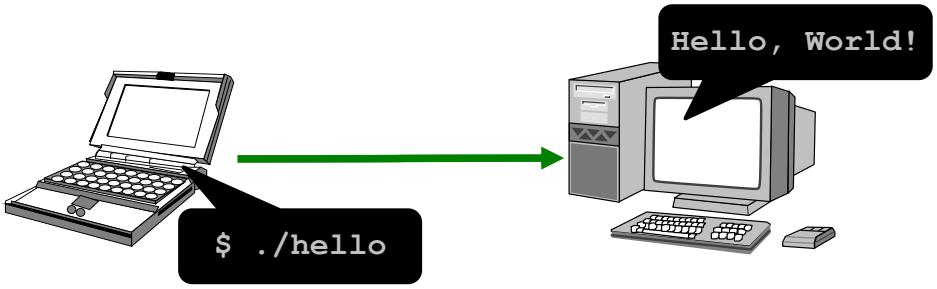


Aplicaciones de la Transparencia en SoCs

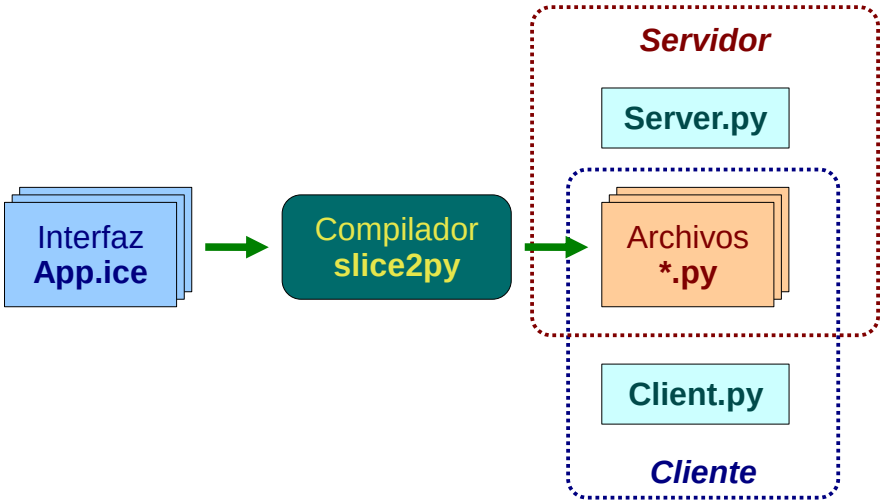
Tipo	Descripción	Aplicación
Acceso	Recursos locales y remotos son accecidos utilizando mecanismos idénticos	**HW/SW interfacing, codisñeo, modelo único de SoC, reutilización de modelos, integración/intercambio de IPs, estabilidad del diseño en caso de cambios en la implementación de los componentes, acceso remoto, etc.**
Localización	No es necesario conocer dónde residen los recursos	
Replicación.	Los programadores no son conscientes de múltiples instancias de un recurso	**Mejora de la fiabilidad y la productividad, mecanismos de balanceo de tráfico y carga de trabajo.**
Fallos	Usuarios y aplicaciones pueden completar sus tareas incluso si se producen fallos en el hardware, red o software.	**Reemplazo de componentes, incremento de la disponibilidad del sistema**
Migración	Usuarios y aplicaciones no se ven afectados por cambios en la localización o implementación de los clientes y servicios	**Implementación de mecanismos de calidad de servicio, gestión de la reconfiguración fácil, corrección de errores**

Un ejemplo en software

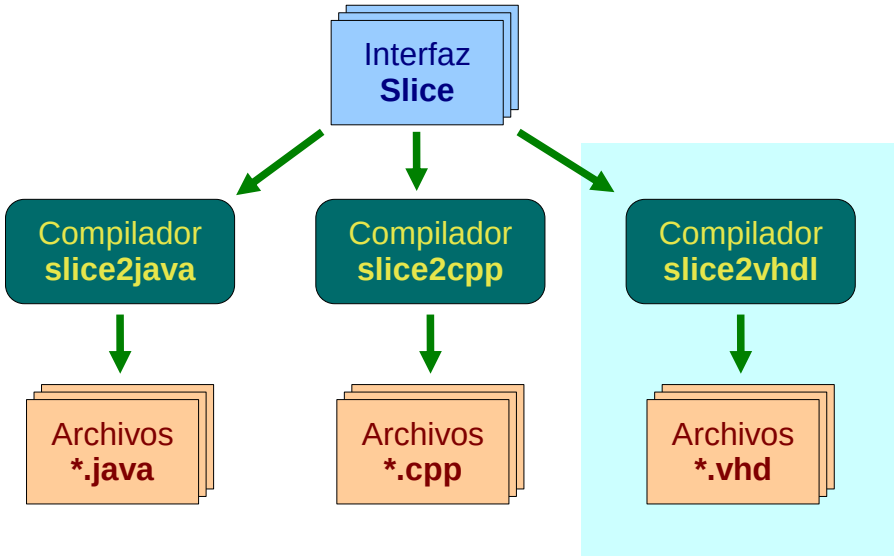
Hola Mundo distribuido



Secuencia de desarrollo



Desarrollo multi-lenguaje



Definir interfaz Slice

- Slice es un lenguaje de descripción de interfaces similar a Java y C++

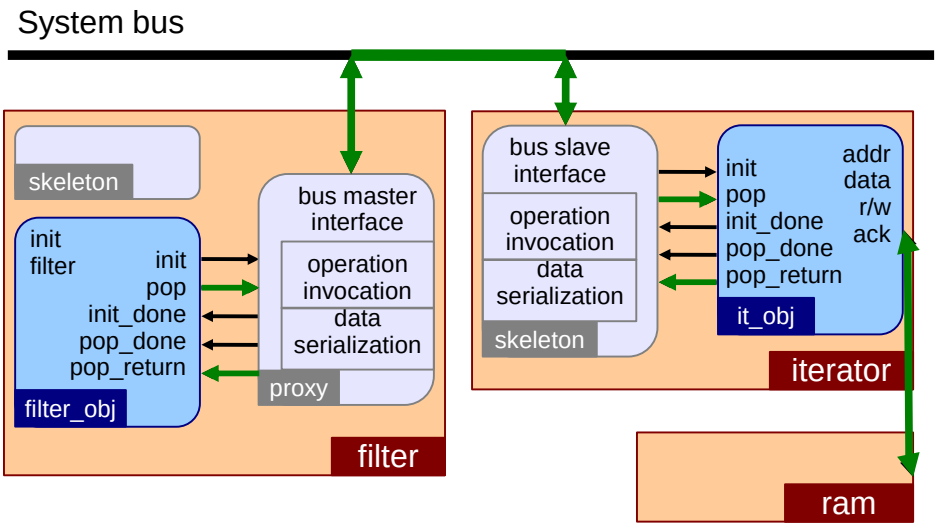
Hello.ice

```
module UCLM {
    interface Hello {
        void puts(string str);
    };
};
```

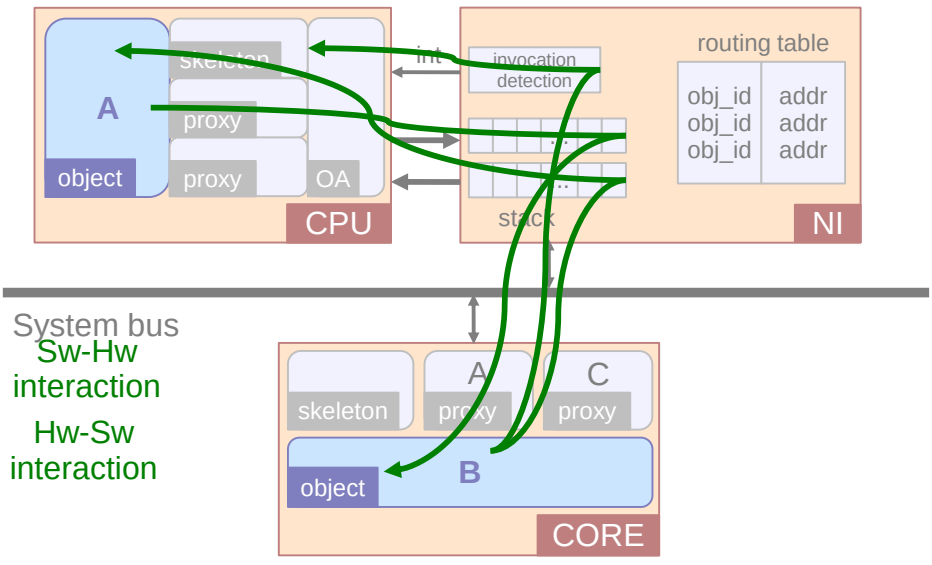
- Se compila con **slice2py**

```
$ slice2py Hello.ice
```

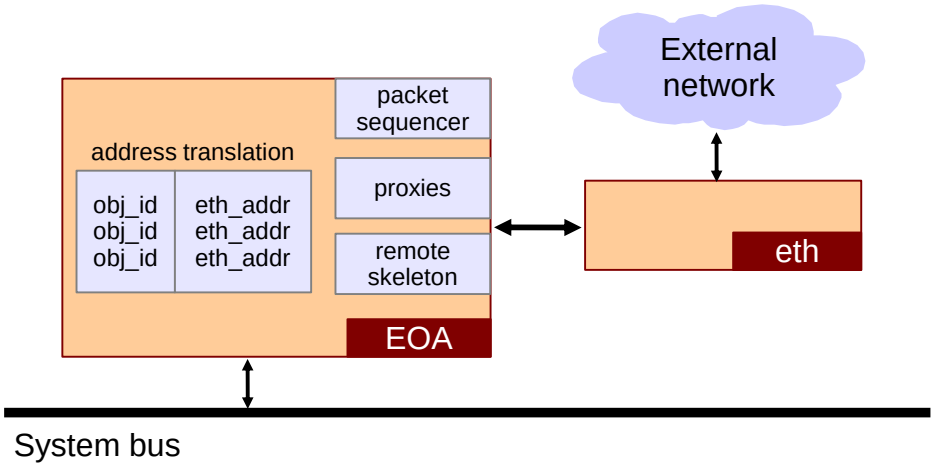
Comunicación Hw-Hw



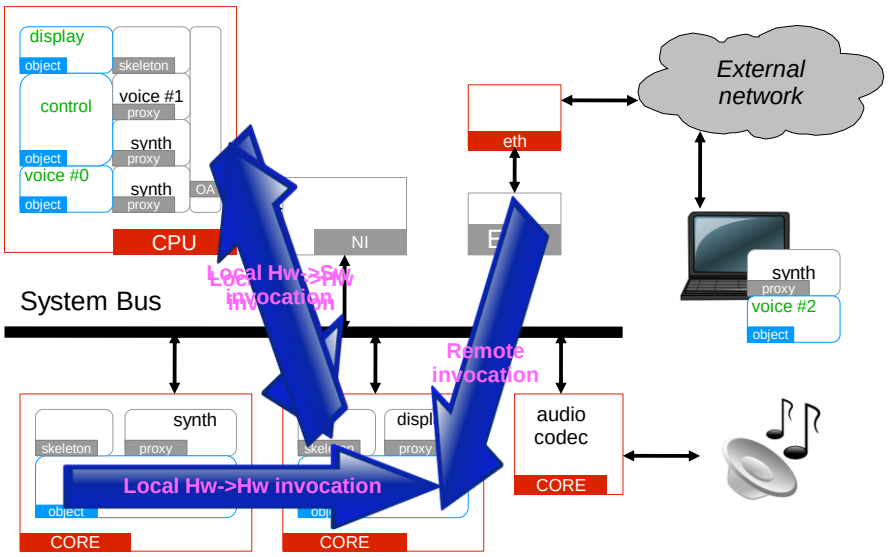
Comunicación Hw-Sw (2)



Comunicación remota

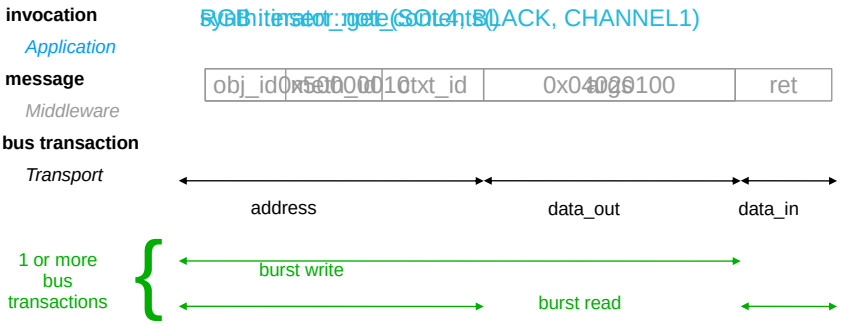


Middleware de sistema



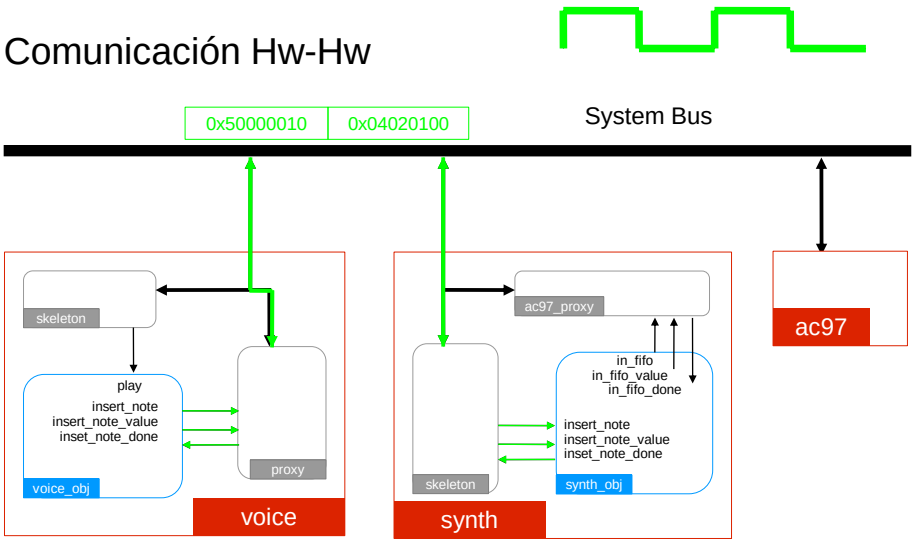
Mensajes

- Una de las claves de la transparencia radica en el uso de mensajes comunes
- Los mensajes se implementan eficientemente, a la medida de la infraestructura de transporte



Middleware de sistema

- Comunicación Hw-Hw

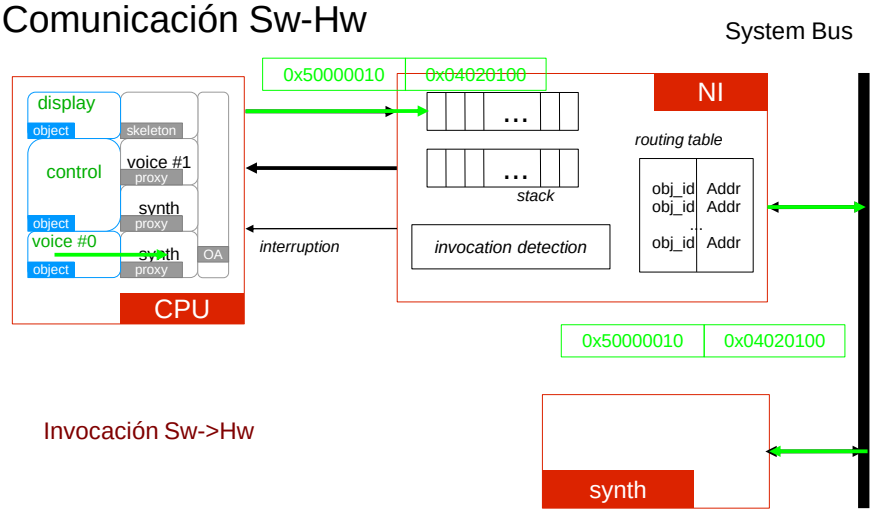


Comunicación HW-HW

- Transparencia de localización
  - El proxy encapsula la información de direccionamiento
  - Puede implementar indirección
- Transparencia de red
  - Los objetos no necesitan conocer la infraestructura de comunicaciones, sólo las interfaces

Middleware de sistema

- Comunicación Sw-Hw

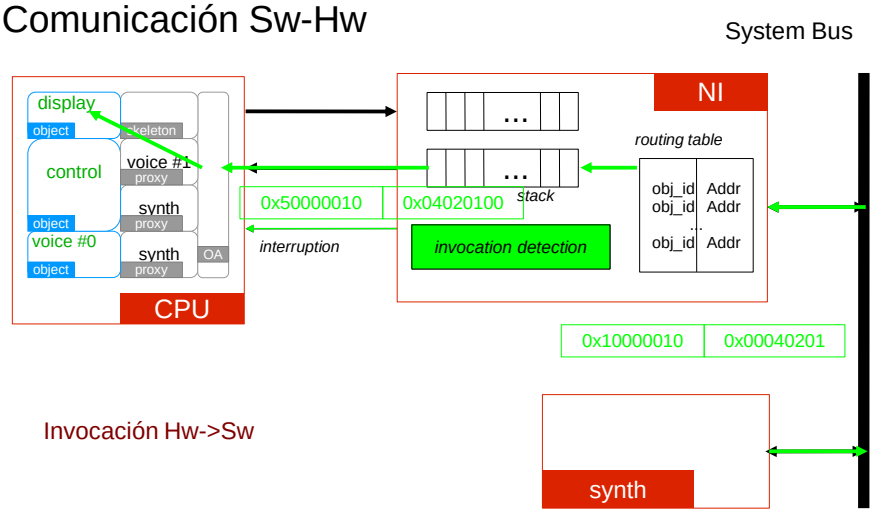


Comunicación Hw-Sw

- Transparencia de implementación
  - Interacción independiente de si son Hw o Sw
  - Mantenemos el formato de mensajes en cada red
- Los objetos Sw comparten un único punto de entrada: el bus de la CPU
  - Un adaptador de objetos implementa la demultiplexión
- Objetos Sw se acceden a través de un **interfaz de red**
  - **Envía los mensajes hacia la red o el bus**
  - **Recibe los mensajes a objetos Sw y notifica a la CPU**

Middleware de sistema

- Comunicación Sw-Hw



Invocación Hw->Sw

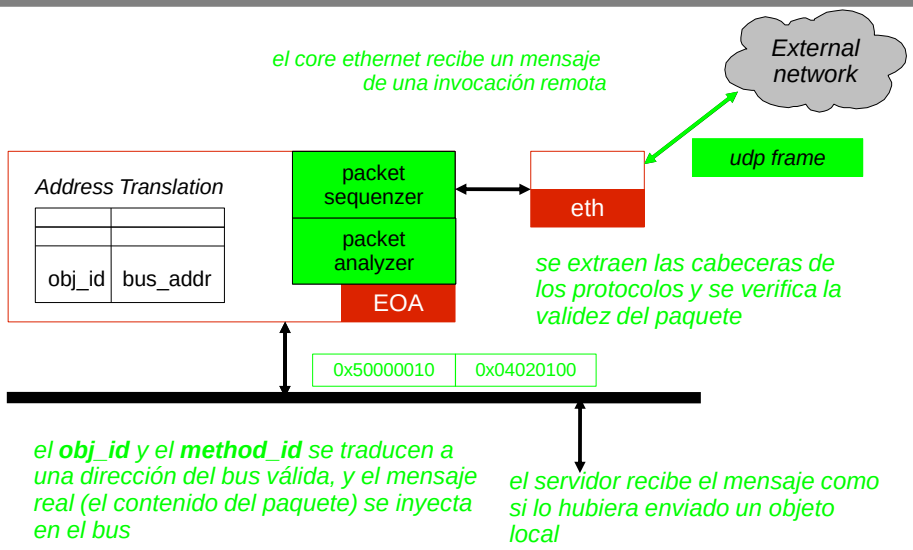
Interoperabilidad con sistemas remotos

- Es posible realizar invocaciones a/desde sistemas remotos
- Es un problema de paso de mensajes
  - Si el formato es el mismo la interoperabilidad es inmediata
  - Si no puede ser adaptado
- Adaptador de objetos remotos:
  - traduce los nombres de objetos y métodos a direcciones locales
  - ruta los mensajes a/desde el sistema de transporte interno desde/hacia la red

Comunicación remota

- Objeto de SoC invocado desde fuera del chip
  - Existe una entrada en la tabla de direcciones
  - Existe un proxy dentro del EOA
- Objeto externo invocable desde dentro del chip
  - Existe una entrada en la tabla de direcciones

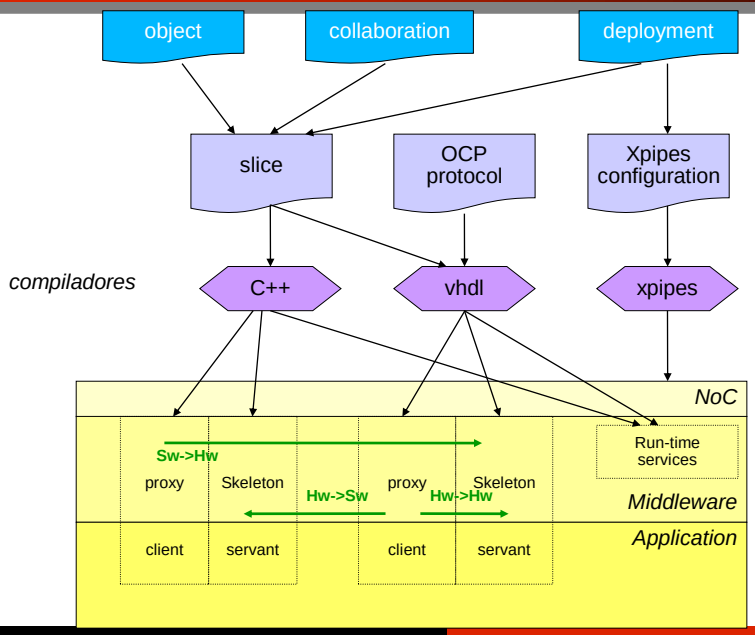
Interoperabilidad con sistemas remotos



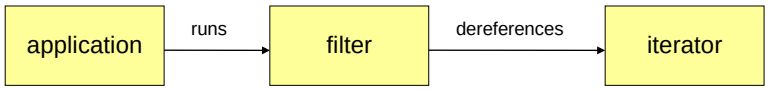
Características Middleware y SoC's

- Transparencia comunicaciones e implementación
  - *Integración e intercambio de IP's: mismo tipo de IP expone el mismo interfaz (posponer decisión)*
  - *Codiseño:* vista homogénea de HW y SW
  - *Reuso de modelos*: detalles de implementación se posponen y el modelo de sistema permanece invariante
- Transparencia de localización/implementación
  - *Migración HW-SW de componentes*: criterios de prestaciones o calidad de servicio
  - *Reemplazo de componentes:* tolerancia a fallos, reconfiguración
  - Instrumentación de parte del diseño para pruebas

Flujo de diseño unificado



Ejemplo de diseño



- 3 versiones del sistema
  - Sólo SW
  - Mixta, con el iterador en HW
  - Mixta, con el filtro y el iterador en HW

Ejemplo de diseño (y 2)

```
Iter::RGB operator*() {
  RGB rgb;
  RMI(ITEROBJ_ID, DEREF_ID, &rgb, null);
  return rgb;
}
```

Iterator proxy

Código SW en C++ para la implementaciones mixtas

```
filter::void run() {
  Iter it;
  ...
  pixel *it;
  ...
}
```

Filter object

```
filter::void run() {
  RMI(FILTEROBJ_ID, RUN_ID, void, null);
  return rgb;
}
```

Filter proxy

```
main() {
  filter f;
  ...
  f.run();
  ...
}
```

Application

```
main() {
  filter f;
  ...
  f.run();
  ...
}
```

Application

Resultados experimentales

Latencies (cycles)	Total	Per Message
Sw	170 + 338 * #pixels	-
Hw it / Bus	170 + 298 * #pixels	2
Hw filter / Bus	170 + 12 * #pixels	2
Hw it / NoC	170 + 328 * #pixels	32
Hw Filter / NoC	170 + 42 * #pixels	32

Tiempo de procesmiento por pixel para los tres tipos de implementaciones usando para las comunicaciones un Bus y una NoC

Prototipo sobre la placa Xilinx XUP-V2Pro

	Client		Servant	
	IPIF	Proxy	IPIF	Skeleton
Slices	208	5	59	5
FF	189	13	97	2
LUTs	346	8	18	13

Coste en recursos de FPGA para implementaciones equivalentes funcionalmente de adaptadores realizadas con el generador IPIF de Xilinx

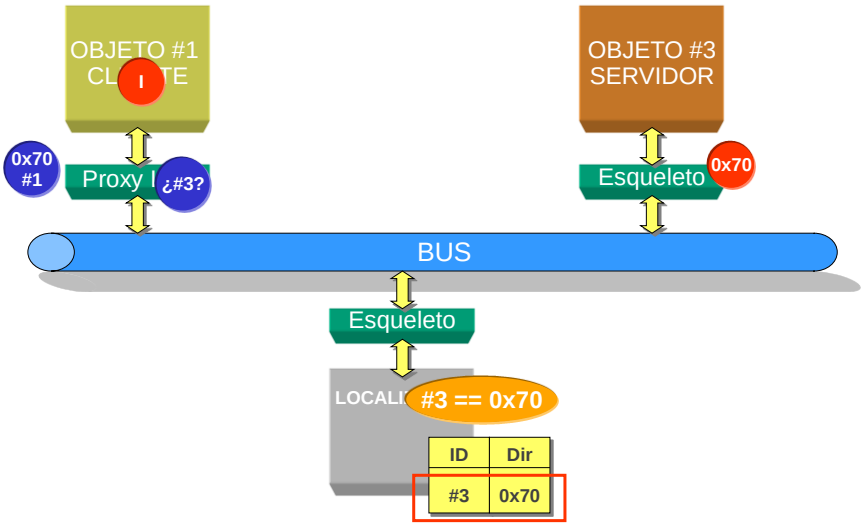
Aplicaciones

- Reconfiguración dinámica - Migración
- Replicación y Tolerancia a Fallos
- Testing remoto
- Sistema operativo distribuido
  - ...

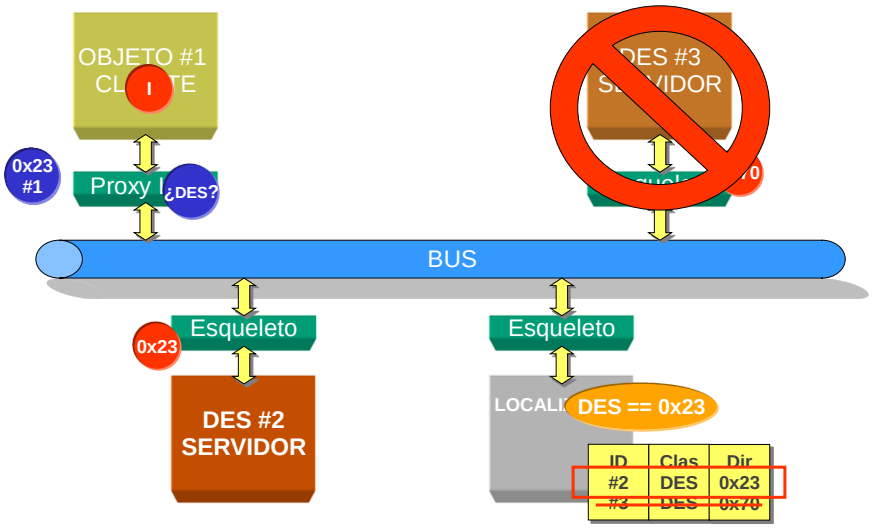
Comunicación indirecta

- No es necesario conocer el endpoint (parte de la dirección del componente en el sistema) del objeto servidor en tiempo de diseño.
- Un objeto *"servicio de localización"* proporciona la referencia en tiempo de ejecución.
- Aplicaciones:
  - Migración.
  - Tolerancia a fallos.
  - Calidad de servicio.
  - Sistemas dinámicos.

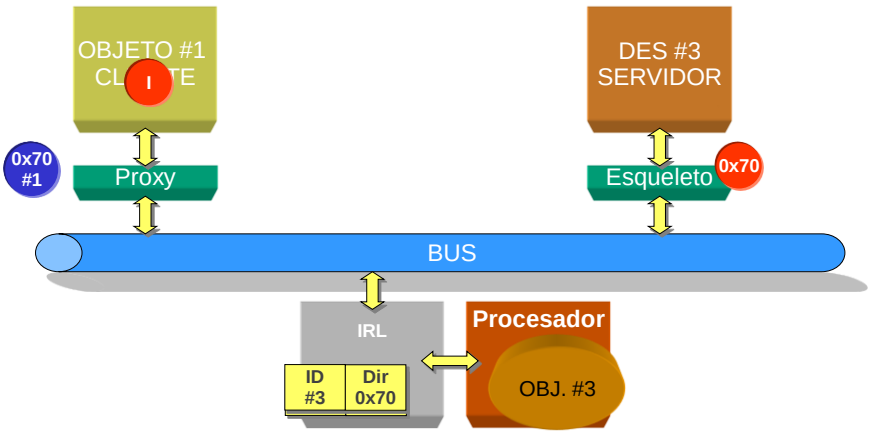
Servicio de localización



Aplicación: Tolerancia a fallos



Aplicación: Migración



El papel de los elementos de la arquitectura

Componente	Concepto de OOCE	Aplicación en el diseño de SoCs
▪ Proxy ▪ Esqueleto	▪ Transparencia de acceso al canal ▪ Transparencia de acceso al componente	▪ Reutilización de componentes ▪ Independencia del canal de comunicación ▪ Intercambio de IPs
▪ Adaptador de objeto local	▪ Transparencia de localización	▪ Integración HW/SW ▪ Migración
▪ Interfaz de red local	▪ Transparencia de acceso al canal	▪ Independencia del canal de comunicación ▪ Migración HW/SW
▪ Adaptador de objeto remoto	▪ Transparencia de localización	▪ Comunicación fuera del chip
▪ Servicio de localización	▪ Transparencia de replicación, fallos, escalado, migración.	▪ Calidad de servicio ▪ Replicación ▪ Balanceo de carga
▪ Proxy indirecto ▪ Esqueleto indirecto	▪ Transparencia de migración, fallos, escalado, migración.	▪ Tolerancia a fallos ▪ Mejora de los tiempos de respuesta

Conclusiones (I)

- El SoC puede tratarse como un sistema distribuido heterogéneo
- El paradigma de Objetos Distribuidos proporciona una visión unificada del sistema como un conjunto de objetos que se comunican
- Es posible establecer una intercomunicación transparente entre cualquier clase de objeto gracias al uso de un formato de mensaje común
- Se puede interoperar con middlewares SW comerciales
  - Objetos en el SoC pueden interactuar con entidades off-chip

Conclusiones (y II)

- La arquitectura de comunicaciones se puede generar completamente de forma automática a partir de un conjunto simple de descripciones
- Overhead de implementación mínimo
  - Adaptadores ad-hoc de "bajo peso"
  - Sólo se incurre en un coste extra cuando se requieren características avanzadas