# Múltiples IPC en invocaciones a objetos móviles

**Leandro R. León (lrleon@cemisid.ing.ula.ve)**
Centro de Estudios en Microelectrónica y Sistemas Distribuidos (CEMISID)
Centro Nacional de Cálculo Científico (CECALCULA)
Universidad de Los Andes – Mérida - Venezuela

**Resumen**

Movilidad de objetos implica un cambio de las localidades de los entes involucrados en una comunicación. Esto plantea un problema de desempeño al mecanismo subyacente de comunicación (IPC), pues los mecanismos IPC tradicionales han sido diseñados para trabajar en condiciones estáticas en las cuales las localidades de los entes involucrados no cambian.

Este artículo versa acerca cómo involucrar eficazmente diversos mecanismos de IPC según la localización de los entes de comunicación. En primer lugar, argüimos que en presencia de movilidad, el desempeño del IPC está estrechamente vinculado al mecanismo de enlace de puntos de comunicación. En contraposición con mecanismos clásicos de enlace dinámico, tales como los lazos de persecución, abogamos por un método de enlace llamado "enlace bajo excepción". Este método permite encapsular la selección dinámica del IPC según las localidades de los entes de comunicación. Esta selección es realizada de forma tal que los troncos de comunicación son siempre los mismos, independientemente del método de invocación.

**Palabras clave: Sistemas Distribuidos, IPC, RPC, LRPC, Pase de mensajes, Sistemas Distribuidos a objetos, Migración**

## 1 Introducción

El diseño de un mecanismo de comunicación entre procesos (IPC) considera las circunstancias de localización de los entes comunicantes. Un IPC distribuido se concentra en diseñar un protocolo de transmisión de datos eficiente en función de su capa de red. Empero, a menudo, un IPC distribuido exhibe un rendimiento pobre en comparación a un IPC especialmente diseñado para comunicar dos procesos que residan en el mismo sitio.

Por la razón anterior, los sistemas operativos ofrecen diversos mecanismos IPC que son exportados a las aplicaciones bajo interfaces diferentes, adaptadas a la tecnología del mecanismo. Cada mecanismo IPC ofrece promesas de desempeño en función de parámetros tales como longitud de mensajes, frecuencia de comunicación y localización de los entes de comunicación. A menudo, estas promesas involucran compromisos entre la latencia, el ancho de banda y el consumo global de recursos (CPU y memoria, particularmente). Los usuarios de aquellos mecanismos transan estos compromisos según sus necesidades y circunstancias.

### 1.1 El problema

En los sistemas distribuidos a objetos, toda la comunicación es englobada bajo una sola abstracción: la invocación de método. La estructuración por objetos hace que los sistemas distribuidos y las aplicaciones construidas bajo este paradigma tengan muchos más entes de comunicación. Las características y circunstancias de comunicación de estos entes son sumamente diversas. No hay una situación de comunicación, aislada o dominante, a la cual consagrar los principales esfuerzos de optimización. Cualquier caso de comunicación es frecuente en el mundo de los objetos distribuidos.

La unicidad de la invocación contrasta con las diferentes situaciones de comunicación de los sistemas distribuidos a objetos. Aunado a este problema, se presenta el carácter móvil de los objetos,

pues la movilidad de los objetos puede cambiar abruptamente las circunstancias de comunicación, lo que hace que los inconvenientes planteados por la unicidad de la invocación devengan más críticos. Por esta razón, la inmensa mayoría de los sistemas a objetos existentes adoptan un solo mecanismo de comunicación distribuido. Esta adopción a acarrea costes innecesarios cuando cambian las circunstancias de comunicación.

Los sistemas operativos ofrecen diversos mecanismos de comunicación. En aras de ser general, la interfaz de un sistema de comunicación trata de cubrir varias situaciones posibles de comunicación. Sin embargo, internamente, el mecanismo es diseñado para optimizar una situación particular de comunicación. Por ejemplo, un sistema de comunicación remota, tal como la interfaz Sockets, considera la comunicación local, pero el sistema está diseñado para optimizar la comunicación en red. Los casos locales, cuales podrían ser comunes en el mundo de los objetos, pagarían un sobrecosto en desempeño respecto al uso de un solo mecanismo IPC especialmente diseñado para la comunicación local. Pues, usar un sólo IPC acarrea costes para la invocación local que no serían pagados si se tuviese un IPC especialmente diseñado para el caso local.

Por otra parte, los sistemas operativos ofrecen mecanismos de comunicación especializados para la comunicación local, por ejemplo, colas de mensajes POSIX y memoria compartida. Empero, en la mayoría de los casos, las interfaces de estos mecanismos son radicalmente diferentes a las interfaces ofrecidas por los mecanismos de comunicación remota.

Los diseñadores de sistemas generales de invocaciones a objetos se encuentran con el problema de combinar sistemas de comunicación que exhiben interfaces muy diferentes. Por esta razón, los sistemas de invocación a objetos usan un solo mecanismo de comunicación, general, que cubra la comunicación remota y local, en detrimento del desempeño de la comunicación local. Esto es un problema serio en los sistemas a objetos móviles donde las localidades de los entes de comunicación cambian dinámicamente. Cuando se realiza una migración de objeto para permitir una invocación local, esta invocación no aprovecha al máximo las posibilidades de comunicación

### 1.2 Los objetivos y motivaciones de este trabajo

Para disponer de un mecanismo eficiente de invocación de objetos, es necesario poder combinar diferentes mecanismos de IPC y usarlos según las localidades de la referencia y del objeto. Esto plantea el desafío de acoplar dos mecanismos conceptualmente diferentes: la invocación de objeto y la actualización de una referencia a objeto que devino inválida porque el objeto migró.

Este artículo versa sobre una manera de integrar eficazmente diversos mecanismos IPC y seleccionar dinámicamente el mecanismo IPC en función de diversas circunstancias, en particular, cambios dinámicos de las localidades de los entes de comunicación. Todo esto es logrado mediante una técnica que combina la invocación con el enlace cliente-objeto que es requerido luego de que un objeto es desplazado. El enlace cliente-objeto consiste en actualizar la información de localización de objeto (puertos, capacidades, direcciones memoria, etc.), cual deviene inválida luego de que el objeto ha sido migrado.

Nuestra técnica exhibe las siguientes características:

- **Desempeño**: El mecanismo de comunicación es seleccionado según el mejor mecanismo posible en función de las circunstancias de localización de los entes de comunicación.
- **Transparencia**: La selección es realizada sin ninguna mediación del programador.

- **Portátil**: El esquema permite combinar cualquier mecanismo de comunicación.
- **General**: Nuestra técnica no se restringe a los sistemas a objetos, ella puede utilizarse, sin ninguna modificación conceptual, a cualquier esquema general de comunicación, por ejemplo, la interfaz Sockets.

Una de las razones que motivan la migración es el deseo de mejorar el desempeño de la invocación. Por ejemplo, con el propósito de aumentar el desempeño de la invocación, una política de equilibrio de carga podría migrar un objeto al sitio de referencia. Del mismo modo, aquella política podría migrar el objeto al mismo proceso de referencia. Como es de esperar, ésta política presumiría que la invocación local inter-proceso es más rápida que la remota y, a su vez, que la invocación intra-proceso es más rápida que la inter-proceso local.

### 1.3 Las situaciones de comunicación

Por simplicidad, consideraremos a la noción clásica de proceso como una unidad de estructuración en la cual pueden residir objetos. Asumiremos que varios objetos pueden residir en un mismo proceso. Para poder invocar un objeto, es necesario poseer una referencia al objeto. Un objeto puede tener cualquier cantidad de referencias cuales pueden estar situadas en cualquier otro proceso, inclusive aquél de residencia del objeto. Los objetos pueden desplazarse de proceso en proceso y todas las referencias deben ser actualizadas transparentemente. Por transparencia, entendemos que el propietario de la referencia, quien puede ser visto como el usuario, nunca se percata de que ha ocurrido una migración de objeto. El poseer la referencia le garantiza el derecho a invocación y a su servicio.

Con éste modelo simple en mente, se pueden distinguir tres situaciones de comunicación según la localidad de los entes de comunicación (la referencia y el objeto):

- Intra-proceso: la referencia y el objeto se encuentran en el mismo proceso.
- Inter-proceso local: la referencia y el objeto se encuentran en procesos diferentes situados en la misma máquina
- Inter-proceso remoto: la referencia y el objeto se encuentran en procesos diferentes situados en diferentes máquinas.

## 2 Combinación de la invocación con la actualización

En esta sección, presentamos un método que, basado en el acoplamiento de la invocación con el enlace, permite seleccionar dinámicamente y sin costes el mecanismo de IPC.

Nuestro método está sesgado por la norma de estandarización CORBA [OMG-95]. Este fue un requerimiento del sistema objeto de esta investigación (COOL-LDM [León 98]) El lenguaje usado para implantar el sistema de invocación, así como para especificar implantaciones de objeto es el C++.

### 2.1 Consideraciones sobre el enlace referencia-objeto

Cuando un objeto migra, todas sus referencias devienen inválidas en el sentido de que aquellas ya no apuntan a la localidad actual donde se encuentra el objeto. Así pues, es necesario disponer de un mecanismo de actualización que garantice la transparencia; es decir, los usuarios de las referencias no deben percatarse de que ha ocurrido una migración. Además de la localidad del objeto, este proceso

debe cambiar el mecanismo de IPC en caso de que las circunstancias de comunicación hayan cambiado a causa de la migración.

La invocación de un objeto requiere que cierta maquinaria IPC esté permanentemente preparada sobre los procesos fuente y destino: estructuras de datos, capacidades, etc. Actualizar inmediatamente esta maquinaria puede acarrear costes prohibitivos, pues ello implica una difusión global hacia todas las referencias y, además, requiere disponer de flujos (threads) activos y exclusivos en cada sitio donde se encuentre una referencia. En añadidura, la estructura de aquella maquinaria puede cambiar radicalmente si cambia el tipo de IPC.

Diversas técnicas de actualización pueden ser utilizadas. Empero, sólo dos permiten combinar fácilmente el enlace con la invocación: los lazos de persecución (forward addresses) y la recuperación de mensajes perdidos.

En los lazos de persecución, la nueva dirección del objeto es dejada sobre su antiguo sitio de residencia. Cuando una invocación es realizada hacia un objeto que ha migrado, ésta es redireccionada hacia el sitio apuntado por el lazo. Cuando el objeto es finalmente alcanzado, la nueva dirección es enviada con la respuesta y la referencia es definitivamente actualizada.

La recuperación de mensajes perdidos consiste en no hacer nada al momento de la migración. Cuando una invocación es realizada hacia un objeto que ha migrado, esta fracasa a causa de la inconsistencia entre la antigua dirección almacenada en la referencia y la nueva localidad del objeto. Este fracaso señala el hecho de que el objeto migró y que es necesario iniciar un proceso de localización.

- Ambos enfoques son perezosos, lo que exhibe las bondades siguientes:
- Bajo costo involucrado a la actualización, pues solo es actualizada la referencia involucrada a la invocación.

La actualización de la referencia y, eventualmente, del mecanismo IPC, puede ser realizada por el mismo flujo que inicia la invocación. Esta característica es la clave para combinar naturalmente el mecanismo de invocación con el mecanismo de enlace.

### 2.2 Enlace bajo excepción

El uso de lazos de persecución es la técnica de actualización más popular. Aunque ella permite integrar fácilmente mecanismos de IPC diferentes[1], su utilización acarrea costes adicionales sobre el desempeño de la invocación que son proporcionales a la longitud de la cadena de los lazos y al tamaño de los mensajes. Además, liberar los recursos consumidos por los lazos que ya no se utilicen implica recolección distribuida de basura (garbage collector), una actividad que también consume recursos de cálculo.

Nosotros optamos por una variante de recuperación de mensajes perdidos denominada “enlace bajo excepción”. Esta técnica se divide en varias partes como sigue:

- Cuando un objeto migra, él notifica su nueva dirección a un servidor de enlace.
- Cuando una invocación intenta acceder un objeto que ha migrado, ocurre una excepción de comunicación. Bajo esta excepción, se solicita al servidor de enlace la nueva localidad del objeto.

[1] Ningún sistema en la literatura ha reportado combinar diversos mecanismos de IPC en el sentido aquí expuesto.

- Cuando el servidor de enlace retorna la nueva localidad, la referencia deviene válida y la invocación es realizada de nuevo.

La excepción es el evento que señaliza la migración del objeto e indica que debe activarse el proceso de localización. La condición esencial para que éste método de actualización sea eficiente es que una excepción de comunicación sea rápidamente detectada. La rápida detección de excepción por puerto inválido es una actividad intrínsecamente realizada por cualquier sistema IPC, local o distribuido. Consecuentemente, la detección de esta situación es realizada rápidamente. Detectar éste tipo de excepción es mucho rápido que la expiración de retardos de transmisión. A lo sumo, la detección es hecha al arribo del primer paquete de comunicación. Además, las solicitudes al servidor de migración siempre son de tamaños fijos y muy pequeñas. Esto contrasta con los lazos de persecución, cuales pueden arrastrar mensajes muy grandes a lo largo de toda la cadena de lazos.

## 2.3 Diseño general del sistema de invocación

CORBA OMG exige que la jerarquía de clases que relaciona la referencia con el objeto derive de una clase una llamada `CORBA_Object`. Este requerimiento facilita enormemente la selección del mecanismo de invocación, pues múltiples IPC pueden ser implantados por las clases derivadas. Nuestra jerarquía está expresada por la Ilustración 1.

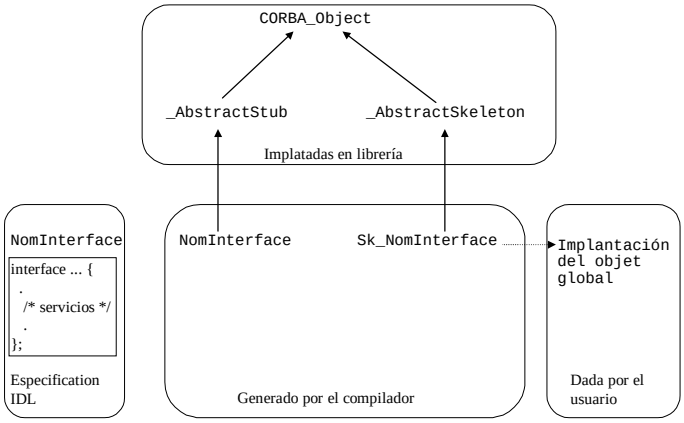


***Ilustración 1: Jeraquía de clases utilizadas***

La clase `CORBA_Object` encapsula información común a las referencias y al objeto. Las clases `_AbstractStub` y `_AbstractSkeleton` mantienen información y servicios exclusivos según el lado de la comunicación: una referencia o un objeto. Las clases `NomInterface` y `Sk_NomInterface` son clases generadas por un compilador IDL a partir de la especificación de interfaz del objeto. `NomInterface` funge de tronco del lado de la referencia. `Sk_NomInterface` conecta el objeto de implantación surtido por el usuario. Esta clase es responsable de despachar el servicio invocado desde `NomInterface`.

La verificación de tipos de la interfaz IDL está implantada por las firmas de los métodos de la clase `NomInterface`.

### 2.3.1 Invocación intra-proceso

Las clases generadas por el compilador IDL contienen firmas idénticas para cada uno de los métodos dados en la especificación IDL. Este aspecto permite resolver el caso intra-proceso con desempeño equivalente a simples llamadas a procedimiento. Conceptualmente, `Sk_NomInterface` es una especialización de `CORBA_Object`, pues, al tener firmas idénticas en sus métodos, la tabla virtual de `NomInterface` puede ser escrita para que apunte a la tabla virtual de `Sk_NomInterface`. Cada una de las firmas de `Sk_NomInterface` es un envoltorio (wrapper) al método de implantación. Esto permite que método de la implantación sea alcanzado mediante una indirección

El enlace bajo excepción puede ser usado para efectuar la actualización. No obstante, sólo en este caso, transamos por aprovechar el flujo de ejecución que coloca el objeto entrante y actualizar directamente las referencias que apunten a él dentro del proceso donde llega el objeto. Esto es una actividad muy rápida porque se realiza dentro de un solo espacio de direccionamiento. La actualización por excepción requeriría una conmutación de contexto en el caso local, o un mensaje síncrono en el caso distribuido, cuales son, respectiva y aproximadamente, 2 y 4 órdenes de magnitud más costosos que una llamada directa a procedimiento. En añadidura, en realidad una sola referencia física está presente sobre cada proceso. Otras referencias lógicas son unificadas en una sola mediante un mecanismo recolector de basura de tipo contador de referencias. Puesto que éste recolector es local, sus costes son ínfimos: solo incrementar y decrementar. Además, la presencia de éste recolector es menos costosa que poseer varias instancias de la misma referencia.

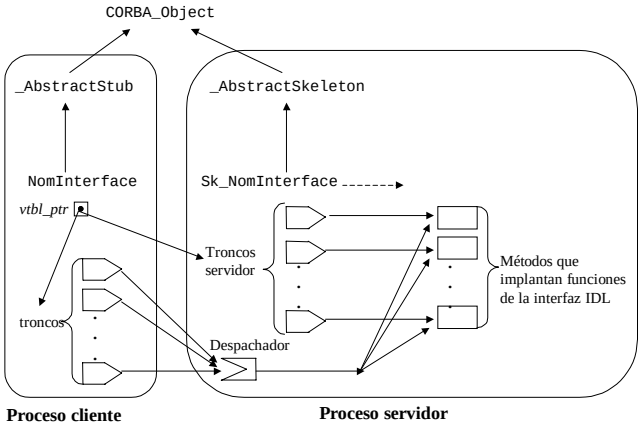


***Ilustración 2: Esquema de encadenamiento de las clases de la Ilustración 1.***

La Ilustración 2 detalla cómo difieren los enlaces de la comunicación intra-proceso de la inter-proceso. En resumen, cuando un objeto entra a un proceso, se busca si existe una referencia hacia él. Si ese es el caso, la tabla virtual de la referencia es modificada para apuntar a la clase `Sk_NomInterface`. Una invocación a través de ésta referencia es de tipo intra-proceso. Cuando ésta es realizada, el método envoltorio de la clase `Sk_NomInterface` es llamado en lugar del método tronco contenido en `NomInterface`. A su vez, el envoltorio llama al método que implanta la interfaz. Todo este proceso ocurre sin copia de parámetros: ellos siempre son pasados por referencia.

Cuando la invocación es inter-proceso, se llama al correspondiente método tronco de la clase `NomInterface`.

El código de actualización de la tabla virtual podría parecer poco común. Sin embargo, éste es conforme al estándar de C++. Esto garantiza portatibilidad a través de diversas plataformas.

### 2.3.2 Invocación inter-proceso

Una vez asumido el enlace por excepción, la invocación es muy simple. La clase `NomInterface` contiene métodos de librería que emiten las solicitudes de enlace al servidor, efectúan la actualización y repiten la invocación. Estos métodos son usados por los talones cuando éstos obtienen una excepción de comunicación.

La combinación de métodos IPC es llevada a cabo por especialización. Una clase abstracta ofrece primitivas básicas y genéricas de RPC a objetos tales como: manejo de tapones (buffers), embalaje/desembalaje (marshaling/unmarshaling) de parámetros, etc. El envío y recepción de mensajes es implantado con métodos virtuales que son llamados desde los troncos del lado cliente (`NomInterface`) y desde los esqueletos del lado servidor. Los servicios de comunicación son implantados en clases derivadas que especializan el tipo de comunicación (local o remota). Tres clases derivadas que implntan tres mecanismos diferentes de IPC fueron adoptadas. En cada una de éstas implantaciones se contempló el uso de parámetros por referencias directas al tapón del mensaje recibido en el servidor. Las combinaciones mencionadas de los diferentes IPC son resumidas en la Tabla 1.

El método LRPC [Bershad et al 90], se fundamenta en el compartimiento una zona de memoria común a los dos procesos. El mecanismo de invocación está realizado de manera tal que el embalaje y desembalaje de parámetros sobre los tapones de comunicación pueda ser el mismo.

Cuando ocurre una invocación inter-proceso, el método tronco de `NomInterface` es llamado. Este método embala los parámetros e invoca el servicio de envío de mensaje. Puesto que este servicio es polimórfico, la especialización, según la localidad del objeto, es automáticamente seleccionada.

Si ha ocurrido una migración, una excepción será detectada y la actividad de enlace será iniciada. Cuando la nueva dirección del objeto es obtenida desde del servidor de enlace, la localidad del objeto es comparada con la localidad de la referencia. Si ocurre algún cambio de circunstancias, por ejemplo, de inter-proceso remoto a inter-proceso local, el mecanismo de IPC es actualizado.

| Tipo de IPC | Descripción |
|---|---|
| Msg pop | Mensajes sistema; desembala parámetros; nuevo flujo en destino |
| Msg ref | Mensajes sistema; parámetros de entrada por referencia en servidor; nuevo flujo en destino |
| Lrpc ref | RPC ligero; parámetros de entrada por referencia en servidor |
| Lrpc pop | RPC ligero; desembala de parámetros |
| Pool pop | Múltiples flujos en servidor; desembala parámetros |
| Pool ref | Múltiples flujos en servidor; parámetros de entrada por referencia en servidor |

***Tabla 1: tipos de IPC adoptados.***

Si la invocación transcurre sin excepción, el mecanismo de comunicación hace que el flujo en el servidor llame al método de despacho (ver Ilustración 2). Este despachador observa el método cuya

invocación es solicitada, desembala los parámetros y llama al método de implantación. Terminado el servicio, el flujo es retomado por el despachador quien embala los resultados y emite la respuesta.

Tanto la actualización de las referencias, como la selección dinámica del mecanismo IPC, son realizadas transparentemente. El carácter perezoso de la actualización por excepción permite usar el mismo flujo invocante para también actualizar el mecanismo de comunicación. En todos los casos, siempre son usados los mismos talones y esqueletos generados por el compilador IDL. El compilador IDL siempre genera un solo método talón y un solo método esqueleto por cada método definido en una interfaz.

# 3 Desempeño

Para evaluar el desempeño, se usó la especificación de interfaz IDL mostrada en la Ilustración 3. Los experimentos fueron realizados sobre una red, aislada, tipo **Ethernet** de 10 Mbytest/sec con tarjetas de red **3com** 3c501 de buses internos de 8 bits. Se utilizaron dos máquinas modelos **Compaq Deskpro 386/25e** con procesadores Intel 386, una frecuencia de reloj de 25 Mhz, 16 Mbytes de memoria, 32 Kbytes de cache y un desempeño medido por **Dhrystones** de 3,17 VAX MIPS equivalentes a 1,49 MIPS. El sistema operativo utilizado fue el micro núcleo ***Chorus* r2.3** [Rozi-90] versión de núcleo 3.5.

El compilador C++ utilizado fue el ***ATT-cfront*** versión 3.0.3 cual genera código cruzado hacia lenguaje C. Las salidas C de ***cfront*** fueron compiladas con el compilador ***gcc*** *–O2* versión 2.6.3.

```
typedef sequence<octet> ByteStream;

interface Foo {

 void round_trip();

 void send(in ByteStream stream);
};
```

***Ilustración 3: especificación de interfaz IDL usada para los experimientos.***

Un total de once (11) referencias a la implantación de Foo, cada una ligada a un tipo de IPC diferente, y distribuidas entre tres procesos sobre las dos máquinas, cohabitaron simultáneamente durante los experimentos. Esto corroboró el carácter múltiple del tipo de IPC. Simulaciones de migración fueron realizadas para cada uno de los métodos de invocación. Esto corroboró el funcionamiento del mecanismo de enlace.

La invocación intra-proceso exhibió un desempeño de 3,3 μsecs y 3,6 μsecs para los métodos `round_trip` y `send` respectivamente. El desempeño del segundo método es independiente del tamaño del flujo de bytes, pues en el caso intra-proceso el pase de parámetros siempre es por referencia.

Las curvas de desempeño de las invocaciones inter-proceso son mostradas en las ilustraciones 4 y 5 respectivamente.

Las curvas de la invocación local inter-proceso merecen dos observaciones. La primera concierne al uso de referencias a los tapones del servidor. Las referencias al tapón servidor arrojan una ganancia, aproximada, en velocidad y espacio, de 200% respecto a la copia de parámetros. La segunda

observación es que las curvas de LRPC exhiben mejor desempeño que el esquema más optimizado para pasar mensajes. Además, la implantación del LRPC tiene mucho margen de optimización; de hecho, ella no usa un soporte de llamadas ascendentes (upcalls).

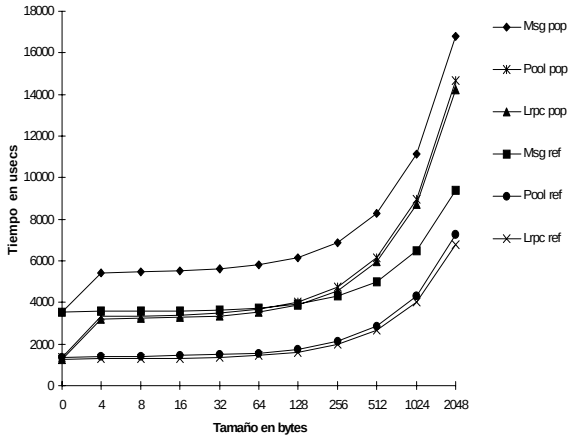


*Ilustración 4: Desempeño inter-proceso local*

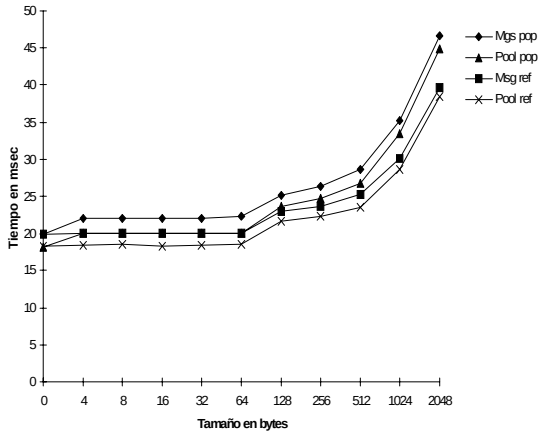


*Ilustración 5: Desempeño inter-proceso remota*

# 4 Antecedentes

Esta discusión abordará brevemente los antecedentes de investigación en los mecanismos de enlace e invocación para sistemas a objetos móviles y los mecanismos de IPC locales.

## 4.1 Técnicas de enlace para sistemas a objetos móviles

La preocupación por el enlace de objetos se remonta al mismo inicio de los sistemas distribuidos. En un principio, variantes de difusión global de la nueva localidad fueron investigadas. La difusión fue de desempeño muy pobre y no escalable. Ejemplos parciales de éste enfoque pueden encontrarse en las versiones preliminares de los sistemas **V** [Theimer et al 85], y **Charlotte** [Artsy et Finkel 89].

El uso de los lazos de persecución es la técnica de enlace dinámico más popular. Ella ha sido adoptada por una gran cantidad de sistemas. Por ejemplo, **Demos/MP** [Powel et Miller 83], **Galaxy** [Sinha et al 91], ambos sistemas orientados a procesos. Todos los sistemas a objetos revisados adoptaron lazos de persecución: **Guide 2** [Hagimot et al 95], **DC++** [Schill et Mock 93], **Soul** [Shapiro et al 92], **Amber** [Chase et al 89] y **SOS** [Shapiro et al 89] y **Emerald** [Jul et al 88]. De todos estos sistemas, solo **Emerald** distingue 2 tipos diferentes de comunicación equivalentes a nuestros casos intra-proceso e inter-proceso remoto. En **Emerald**, todos los objetos de un sitio residían sobre un mismo proceso. Esto restringía las aplicaciones a ser de vida corta. A la diferencia de aquél sistema, nuestro enfoque permite colocar objetos sobre un mismo sitio separados en contextos diferentes. El resto de los sistemas usaban un solo mecanismo de IPC para todos los casos (TCP, inclusive).

La migración de procesos en **Amoeba** [Steketee et al 94] es uno de los muy pocos sistemas donde una técnica parecida a la nuestra fue utilizada. Aquella técnica estaba basada en recuperación de mensajes perdidos detectados por expiración de retardos. Nuestro enfoque, aparte de ser orientado a

objetos y no a procesos, define la noción de excepción y explica como ésta puede ser usada para señalizar la migración.

Las experiencias adquiridas en algunas implantaciones de migración recomiendan evitar las dependencias residuales ocasionadas por los lazos de persecución [Douglis y Ousterhout 87], [Goscinski 91], [Singhal y Shivaratri 94] y [Sinha 96]. Los argumentos son la degradación de desempeño, la necesidad de un mecanismo de recolección de lazos perdidos, y la sensibilidad a fallas.

### 4.2 Mecanismos de IPC local

[Bershad et al 90] evidenciaron el alto costo del RPC cuando aquél es llamado localmente. Sobre tres sistemas (V, Taos y UNIX-NFS), se observó que la frecuencia de llamadas remotas era muy baja y que las llamadas restantes eran ejecutadas localmente sobre una misma máquina. Sus medidas constataron frecuencias de 3% para el sistema V, 5,3% para el Taos y 0,6% para UNIX-NFS, un sistema ampliamente usado hoy en día. Aunque aun no existen estudios que evidencien directamente esta suposición, se espera que un estudio similar al de [Bershad et al 90] arroje resultados similares sobre un sistema a objetos. Puesto que los objetos pueden ser entidades de grano más fino que el de un servidor, un sistema a objetos tiene mucho más enlaces y mucha más comunicación que un sistema cliente-servidor.

Los estudios de [Bershad et al 90] motivaron la concepción de mecanismos RPC especiales para la comunicación local, tales como LRPC [Bershad et al 90], URPC [Bershad et al 90] y ARPC [Yarvin et al]. Cualquiera de estos mecanismos puede perfectamente ser usado en nuestra implantación.

## 5 Conclusión

Para poder cambiar el mecanismo de IPC luego de una migración, es necesario usar un enfoque perezoso de actualización de referencia devenida inválida por migración. La técnica de actualización adoptada en éste trabajo es denominada actualización por excepción. Esta técnica se resume en: el objeto migrado notifica la nueva dirección a un servidor de enlace, una referencia inválida causa una solicitud de localización a éste servidor y, repetición de la invocación luego de haber actualizado la referencia.

En la combinación invocación con actualización vale la pena destacar los aspectos siguientes:

La actualización por excepción aprovecha el flujo de ejecución que realiza una invocación para solicitar al servidor de enlace y realizar de nuevo la invocación. La actualización del mecanismo IPC es realizada por aquél flujo cuando ya se conoce la nueva localidad y antes de realizar de nuevo la invocación.

Puesto que no se crea un adicional para realizar la actualización, la implantación de nuestra técnica es sumamente fácil y no exige sincronización. Basta con añadir en librería el código de actualización, o generarlo a través del compilador IDL, o una combinación de los dos. En nuestro caso optamos por implantarlo enteramente en librería.

La consecuencia de éstos dos aspectos es que el proceso de actualización de referencia y mecanismo IPC es completamente transparente.

El tiempo que lleva la actualización de referencia domina considerablemente al tiempo de actualización del mecanismo IPC[2]. El costo en desempeño de actualizar el IPC es despreciable comparado a los costes involucrados en actualizar la nueva localización.

En todos los casos, la selección dinámica del mecanismo de IPC es realizada sin duplicar el código de los talones y los esqueletos. El compilador IDL siempre genera un solo método talón y un solo método esqueleto por cada método definido en una interfaz.

# 6 Bibliografía


[Artsy et Finkel 89] Artsy Y., Finkel R. « Designing a process migration facility - the Charlotte experience », IEEE Computer, 22(9), 1989, pp 47-56.

[Bershad et al 90] Bershad, Brian, Anderson Thomas, Lazowska, Edward et Levy, Henry. « Lightweight Remote Procedure Call » ACM Transactions on Computer Systèmes, 8(1), pp. 37-55 (1990).

[Bershad et al 91] Bershad, Brian, Anderson Thomas, Lazowska, Edward et Levy, Henry. « User-Level Interprocess Communication » ACM Transactions on Computer Systèmes, 9(2), pp. 37-55 (1991).

[Chasse et al. 89] Chasse Jeffrey S., Amador Franz G., Lazowska Edward D., Levy Henry M. et Litlefield Richard J. « The amber system: Parallel programming on a network of multiprocessors ». 12th ACM symposium on Operating Systems Principles, pp 147-158, December 1989.

[Douglis y Ousterhout 87] Douglis Frederick et Ousterhout John K. « Process Migration in the Sprite Operating System ». 7th Internation Conference on Distributed Computer Systèmes, September 1987.

[Jul et al 88] Jul, Eric, Levy, Henry, Hutchinson, Norman et Black, Andrew. « Fine-grained Mobility in the Emerald System ». ACM Transactions on Computer Systems 6(1) February 1988, pp109-113.

[Goscinski 91] Goscinski A. « Distributed Operating Systèmes The Logical Design ». Addison-Wesley. ISBN 0-201-41704-9.

[Hagimot et al 95] Hagimot Daniel, Mossière Jacques, Rousset de Pina Xavier et Chevalier Pierre-Yves. « Le système réparti à objets Guide ». Technique et Science Informatiques, numéro spécial sur les Systèmes à Objets, vol. 15 (6), pp. 801-830, 1996.

[León 98] León Leandro. « Une architecture pour les systèmes à objets mobiles ». Thèse du Doctorat de l'Université Paris VI. Février, 1999. 250 pages.

[Singhal y Shivaratri 94] Singhal, Mukesh y Shivaratri, Niranjan. "Advanced concepts in operating systems" McGraw-Hi;; ISBN 0-07-113668-1.

[OMG 95] Object Management Group. « The Common Object Broker: Architecture and Specification » July 1995.

[Powel et Miller 83] Powel M., y Miller, B. « Process Migration in DEMOS/MP ». 9th Symposium on Operating Systems Principles, October 1983, pp 110-119.

[Rozier 90] Rozier, Marc. « Technologie des Noyaux de Systèmes. Exemples de Chorus et Mach ». Ecole d'été INRIA 1990, Septembre 1990. pp 345-384

[Schill et Mock 93] Schill, Alexander et Mock, Markus. « DC++: distributed object-oriented system support on top of OSF DCE ». Distributed Systems Engineering, 1(2) December 1993, pp 112-125.

[Shapiro et al 89-2] Shapiro Marc, Gourhent Yvon, Habert Sabine, Mosseri Laurence, Ruffin Michel et Valot Céline. « SOS: An Objet-Oriented Operating System - Assessment and Perspectives » Computing Systems 2(4) Fall 1989, pp 287-337

[Sinha 96] Sinha, Pradeep. « Distributed Operating Systems ». IEEE Press. ISBN 0-7803-1119-1.


[2] Esto es deducible y no amerita medidas de desempeño.

[Steketee et al 94] Steketee C., Zhu W. et Moseley P. « Implementation of Process Migration in Amoeba ». 14th International Conference on Distributed Computer Systems, pp 194-203.

[Theimer et al 85] Theimer Marvin M., Lantz Keith A., Cheriton David R. « Preemptable Remote Execution Facilites for the V-System ». 10th ACM symposium on Operating Systems Principles, pp 2-14, December 1985.

[Yarvin et al 93] Yarvin, Curtis, Bukowski, Richard et Anderson, Thomas. « Anonymous RPC: Low-Latency Protection in a 64-Bit Address Space » USENIX Conference June 1993. Cincinati Ohio, pp 175-186