Universidad de los Andes
Facultad de Ingeniería
Centro de Estudios en Microelectrónica y
Sistemas Distribuídos (CEMISID)

Técnicas de Localización de Objetos Móviles

Andrés Arcia – Leandro León

amoret@ula.ve - lrleon@ula.ve

Mérida, diciembre 2001.

Introducción

- ¿Qué se entiende por sistema distribuido a objetos?

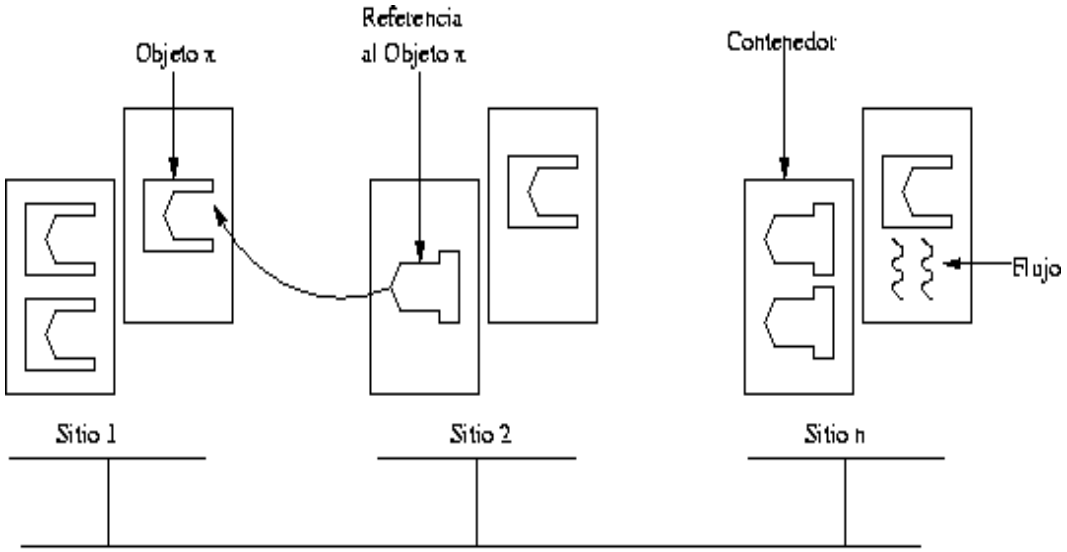
  Un sistema distribuído a objetos es un conjunto de máquinas autónomas, que interactúan a través de un medio de comunicación con el objetivo de compartir recursos representados como objetos.

- ¿De qué forma se lleva a cabo esta interacción?

  A través del pase de mensajes, invocación remota, etc.

Introducción

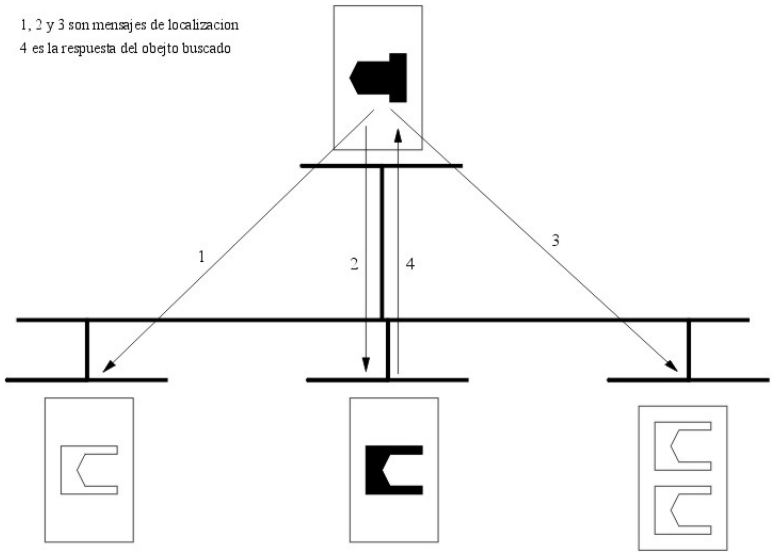
- La migración de objetos:



- Consecuencias:
  - •Referencias inválidas.
  - •Fracaso en la invocación.
  - •Induce a la recuperación de las referencias inválidas

Introducción

- Solución posible para la reposición de referencias invalidas: técnica de difusión.
  - Difusión luego de migrar.
  - Difusión antes de invocar.

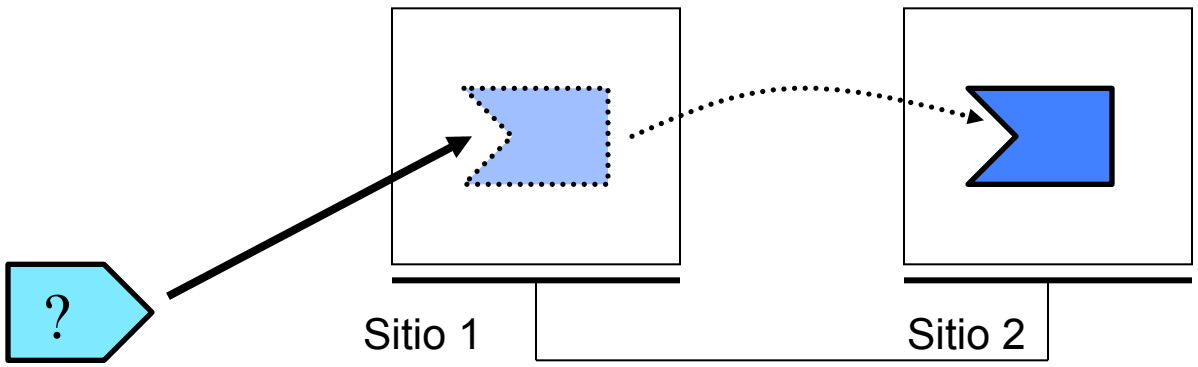


Prueba y actualización

Antes de que se efectúe la invocación se verifica si el servidor está en su espacio de direccionamiento. Es utilizado por los sistemas Emerald, SOS, Amber.

Inconveniente:

- Todas las invocaciones hacen el test.
- Buen desempeño en sistemas de baja escala.

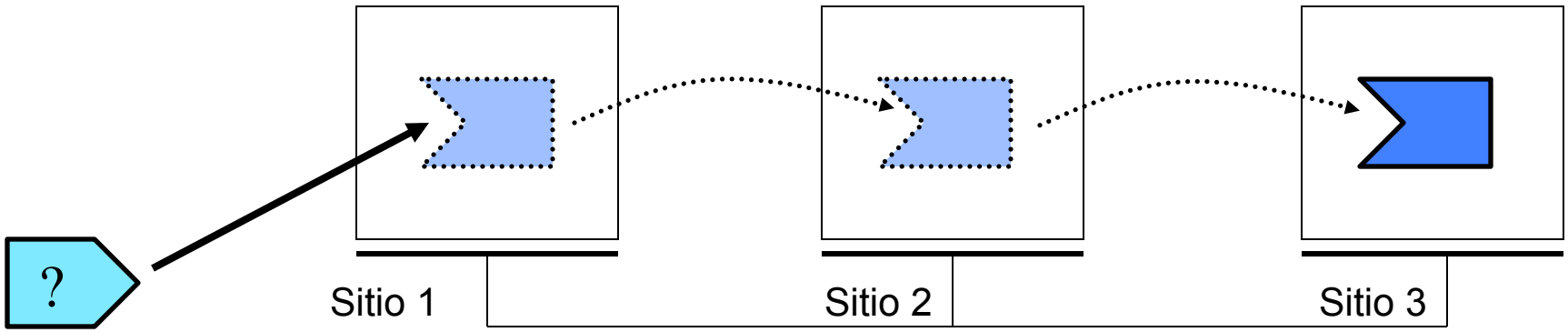


Lazos de persecución

Es una indicación dejada en el sitio origen de la migración y que apunta al sitio donde se encuentra el objeto. Esta técnica es usada por los sistemas Emerald, Guide 2, Demos/MP, Galaxy, DC++.

Inconvenientes:

- Dependencias residuales.
- Debe ser incorporado al protocolo de migración.



Problemas

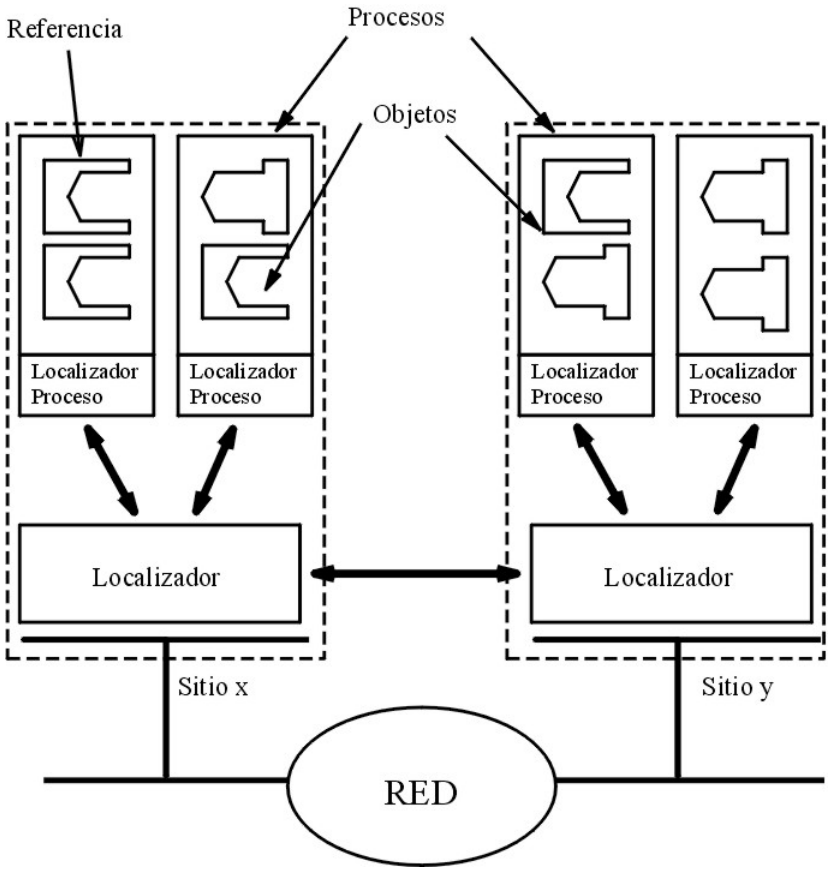
- Tolerancia a fallas
  - Dificultad para la recuperación de una falla.
  - Sensible a fallas.
  - Recolección de basura
- Desempeño
  - Mayor escala => menor desempeño.
- Escalabilidad

Objetivos y alcances

- *Versatilidad*: El sistema es parametrizable y permite escoger diferentes técnicas.
- *Portabilidad* a una amplia cantidad de plataformas materiales y s.o.
  - Protocolo sobre IP.
  - C++ estándar.
  - Conforme POSIX.
- Completamente *distribuido*.
- Tolerancia a fallas:
  - Protocolo de recuperación.
  - Técnicas redundantes de localización.

Arquitectura general del servicio

- Un localizador por sitio
- Una biblioteca run-time por proceso
- Visión de interfaz centralizada



Arquitectura general del servicio

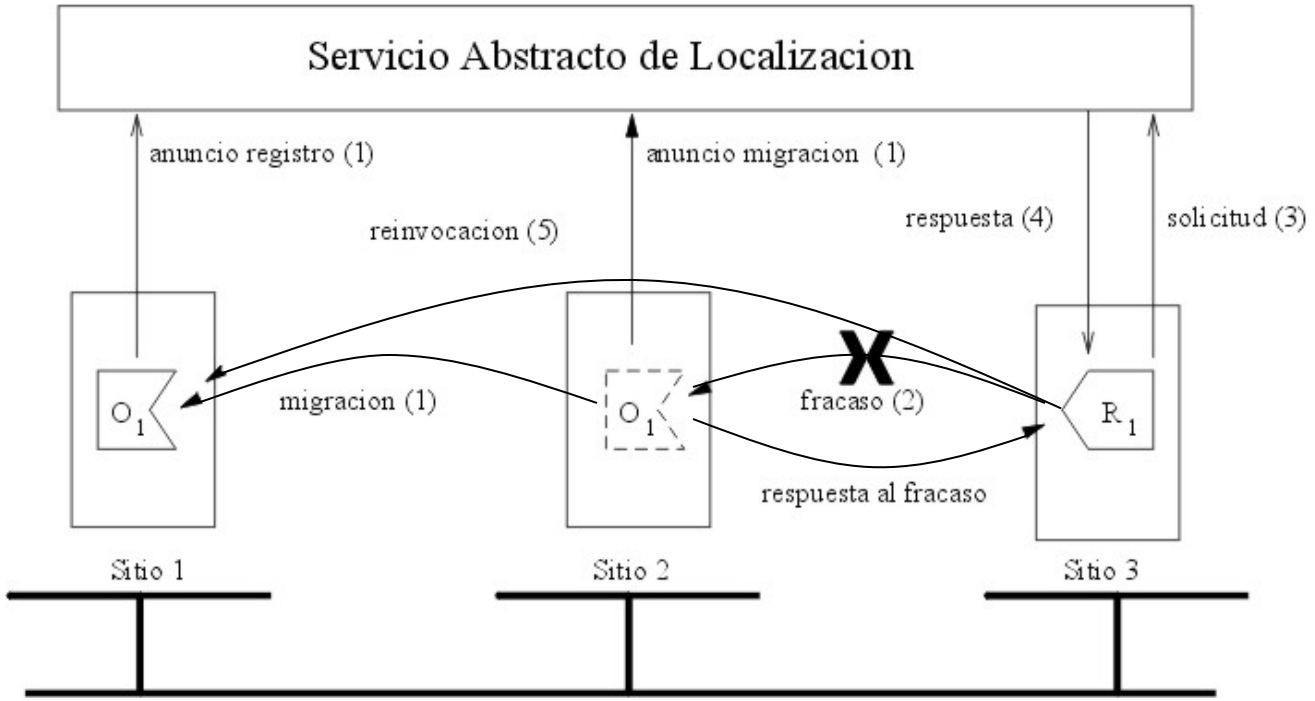
- Premisas del localizador:
  - Conoce las coordenadas de localización de objetos y procesos.
  - Mantiene información necesaria para la localización distribuida.
  - Gestiona localización local y distribuida.
  - Interviene las invocaciones entrantes y salientes.
  - Participa en el protocolo de reconstrucción.
- Premisas del localizador proceso:
  - Mantiene información de objetos y métodos.
  - Valida invocaciones.
  - Gestiona la migración de objetos.
  - Participa en el protocolo de reconstrucción

Técnicas propuestas

- Caching
- Prefetching
- Piggybacking
- Difusiones en cascada

Actualización por fracaso

- El objeto no fue encontrado
- Existe una referencia más reciente
- El objeto está eliminado



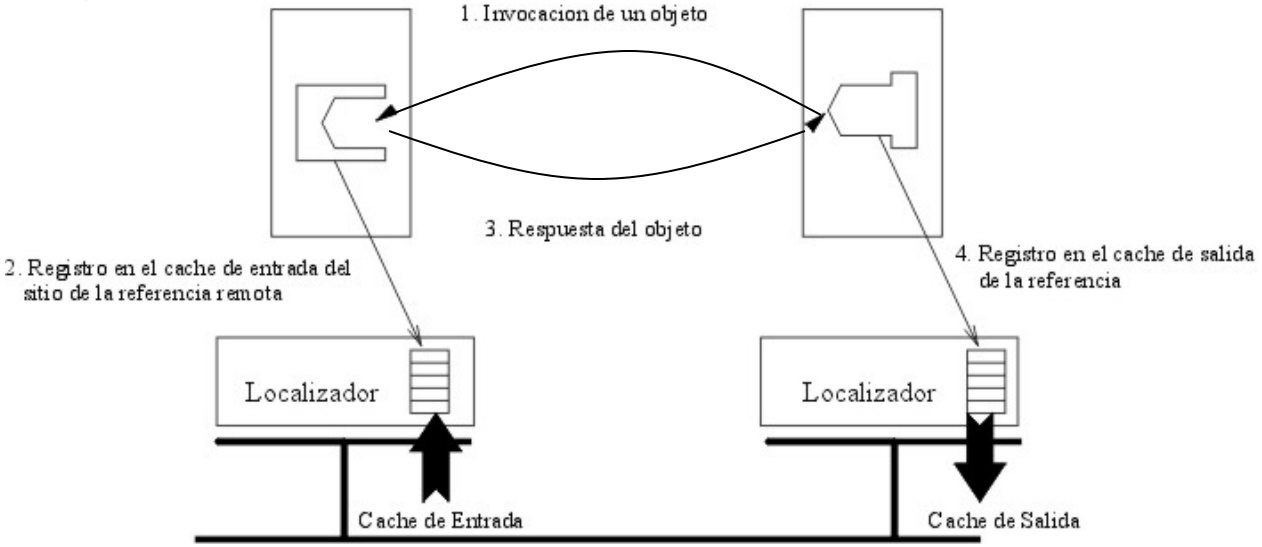
Caching I

El caching consiste en almacenar indicaciones referentes a la ubicación de objetos. Una indicación es obtenida a través de búsquedas previas o de mensajes de actualización.

- Tipos de caches:
  - ***Entrada***: Guarda información referente a las invocaciones entrantes.
  - ***Salida***: Guarda información sobre los objetos invocados.
  - ***Migración***: Mantiene información sobre migración de objetos.
  - ***Nuevas refencias***: Mantiene referencias para futuras invocaciones.
  - ***Eliminaciones***: Guarda objetos eliminados.

Caching II

- **Cache de entrada**: Un registro este cache contiene la tupla <O,s_o,t>. O es el identificador de objeto, s_o es el sitio donde existe la refencia, y t es el tiempo lógico.
- **Cache de salida**: Un registro de este cache contiene la tupla <O,s_d,t>. O y t son como en el cache de entrada, s_d es el sitio donde reside el objeto O.



Prefetching I

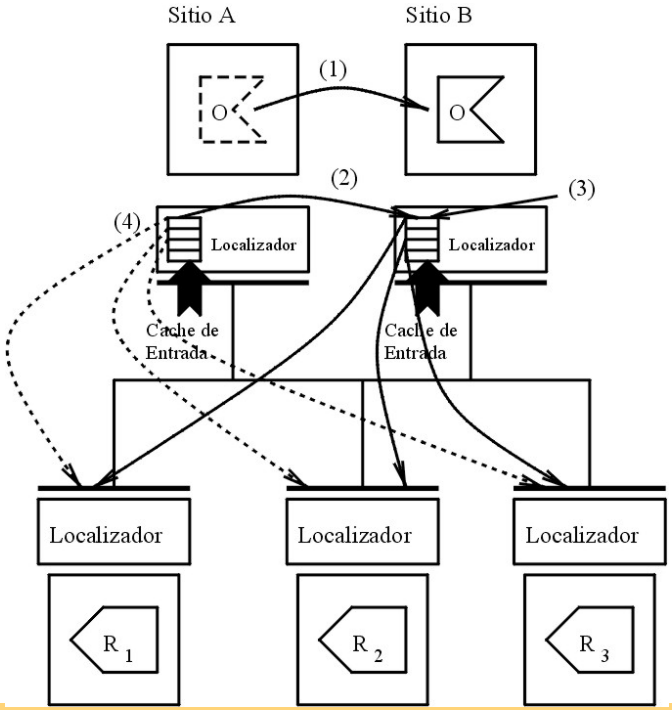
Es la acción de actualizar una referencia inválida antes de que ocurra una invocación. El sentido de esta técnica es anticiparse al fracaso en la invocación.

Tipos de prefetching:

– Prefetching con el sitio origen de la migración.
– Prefetching con el sitio destino de la migración.
– Prefetching con el cache de salida.

Prefetching II

- Prefetching desde el sitio origen de la migración:
  - ***Con el cache de entrada***: buscar registros del objeto migrante y notificar a los sitios origen de las invocaciones.
  - ***Con el cache de migración***: cuando llegue una invocación hacia el objeto migrado, responder con la nueva dirección almacenada en este cache.

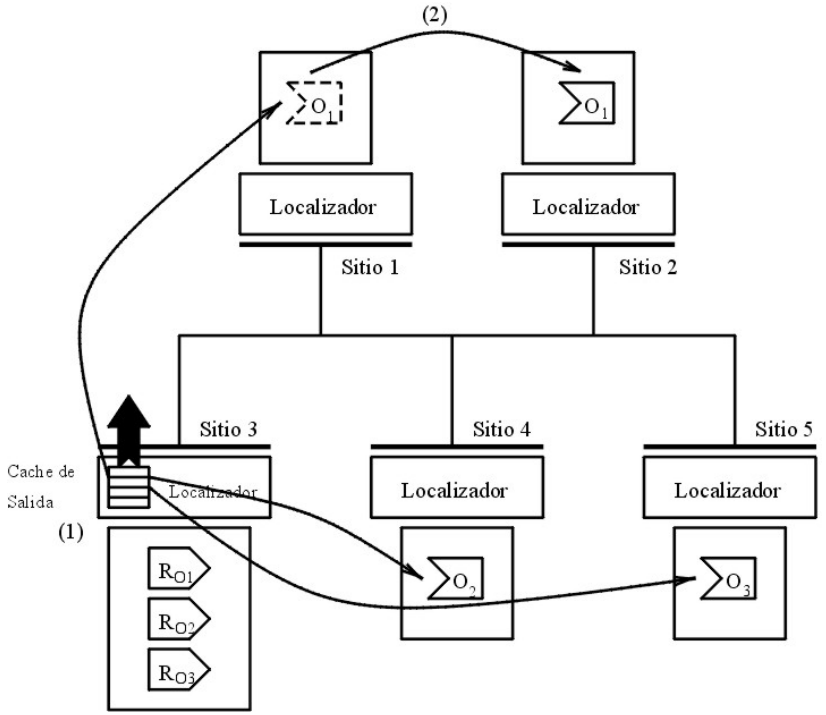


Prefetching III

- Prefetching desde el sitio destino de la migración:
  - ***Con el cache de entrada***: copiar los registros pertinentes al cache de entrada del sitio origen. Esto permitira volver a aplicar la técnica anterior.
  - ***Con el cache de salida***: Verificar las referencias en el cache para convertirlas a locales de ser necesario.

- Prefetching con el cache de salida:

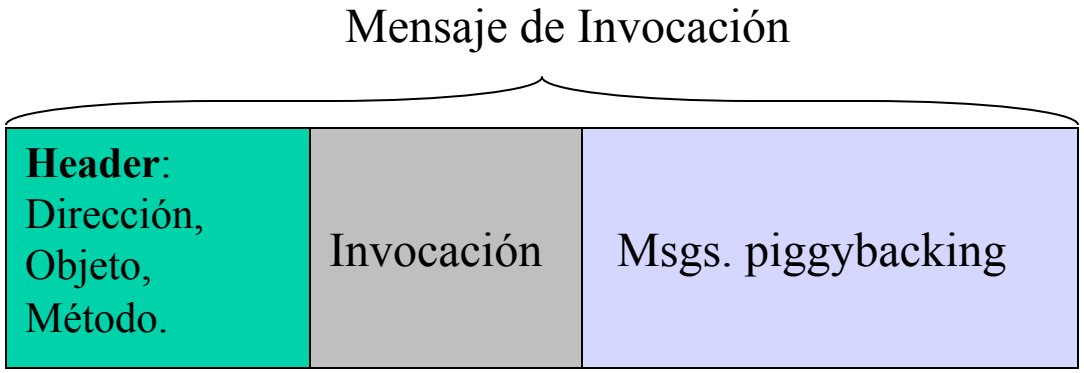
  -Consiste en verificar periodicamente los registros del cache de salida. Si una referencia es invalida, se emprenden acciones de búsqueda anticipada.



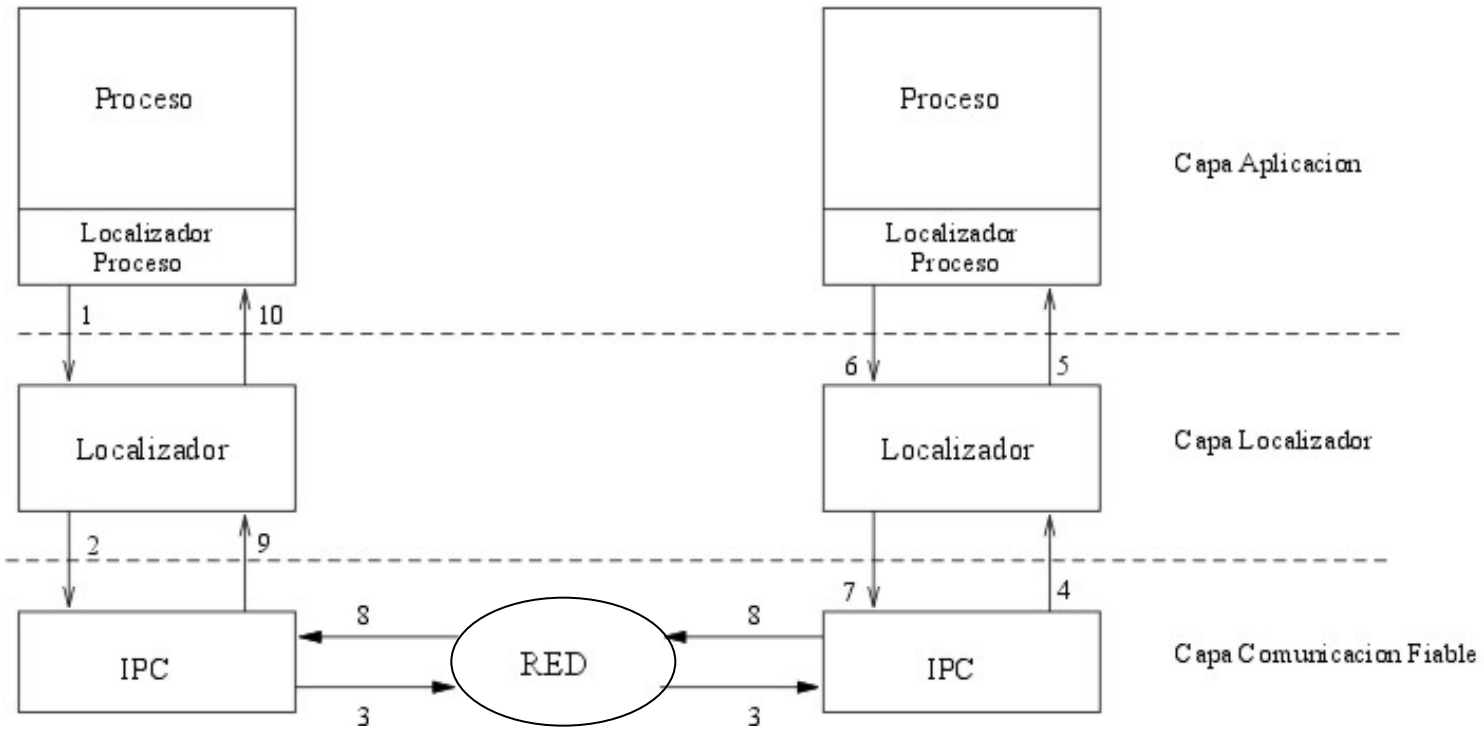
Piggybacking

Dado un mensaje explicito (invocaciones, localizaciones, etc):

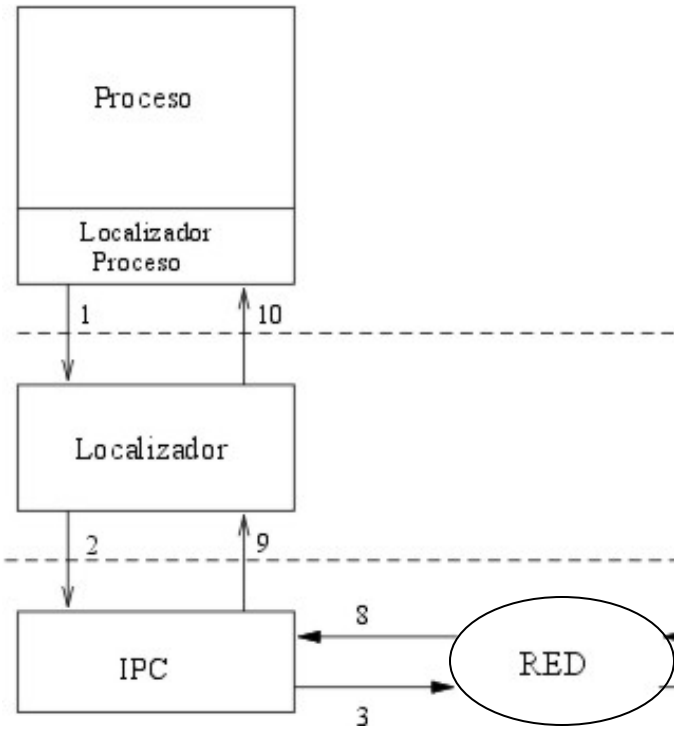
- Se añaden mensajes adicionales.
- Se completa el grano del MTU.



Flujo de una invocación I



Flujo de una invocación II



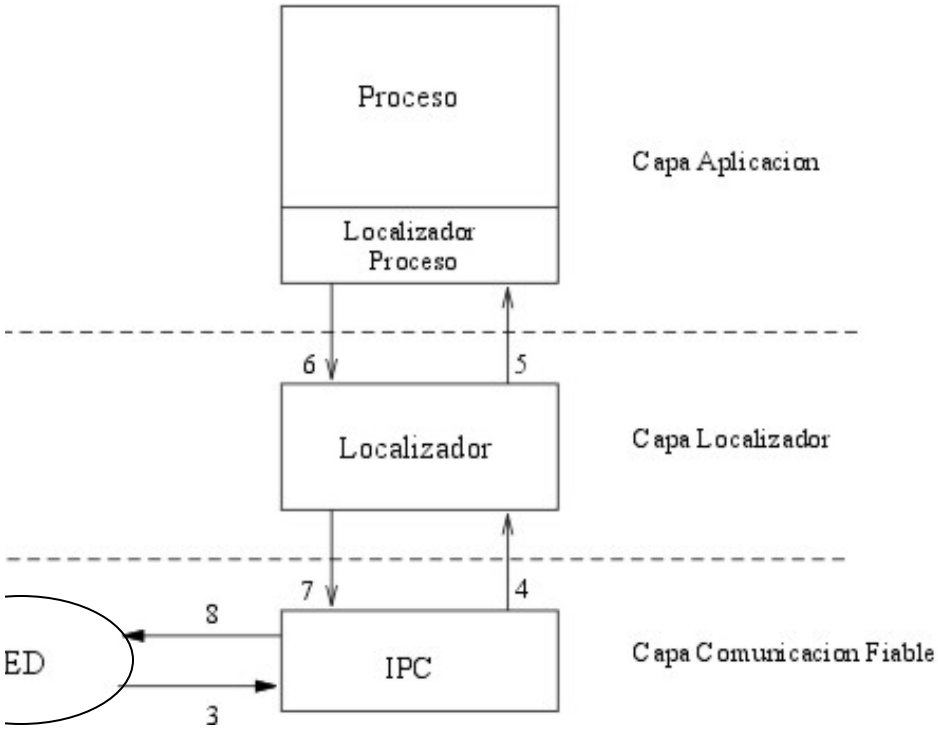
1. Llamada externa al localizador que indica la necesidad de efectuar una invocación.
2. Pase del mensaje explicito con la invocación al sistema IPC.
3. Comunicación fiable de una solicitud IPC.

...

8. Comunicación fiable de la respuesta al mensaje del paso 3.
9. Pase de la respuesta del IPC al localizador origen.
10. Llamada ascendende (*upcall*) que despierta al proceso bloqueado.

Flujo de una invocación III

3. Comunicación fiable de una solicitud IPC.
4. Pase del mensaje explicito con la invocación al localizador.
5. Llamada ascendente (*upcall*) con la invocación que despierta al proceso que contiene al objeto invocado.
6. Respuesta del objeto invocado.
7. Pase de la respuesta al IPC.
8. Comunicación fiable de la respuesta al mensaje del paso 3.



Mensajes piggybacking

1. ***Eliminación de objeto***: cuando un objeto es eliminado del sistema, su eliminación es anunciada este mensaje.
2. ***Solicitud de búsqueda***: indica la necesidad de búsqueda o actualización de una referencia.
3. ***Anuncio de referencia***: propaga un evento de actualización de una referencia. Esto, debido al nacimiento de un objeto ó a una migración.

Control de congestión

Piggybacks consumen recursos (memoria, cpu, red, etc.)

⇒ ***acotar cantidad de piggybacks***

⇒ ***acotar propagación de piggybacks***

Técnicas:

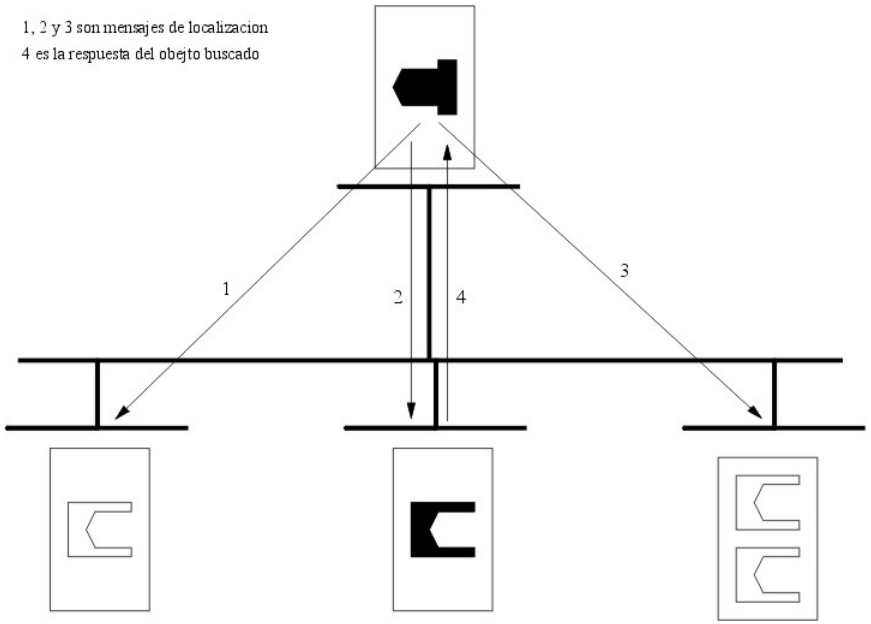
- *Prioridades*
- *Grafo de frecuencias*
- *Tiempo lógico*
- *Tiempo físico*
- *Máxima cantidad de mensajes*

Difusiones

Dos tipos de difusiones:

– Difusión débil: no confiable; poco costosa.

– Difusión fuerte: confiable, pero costosa.

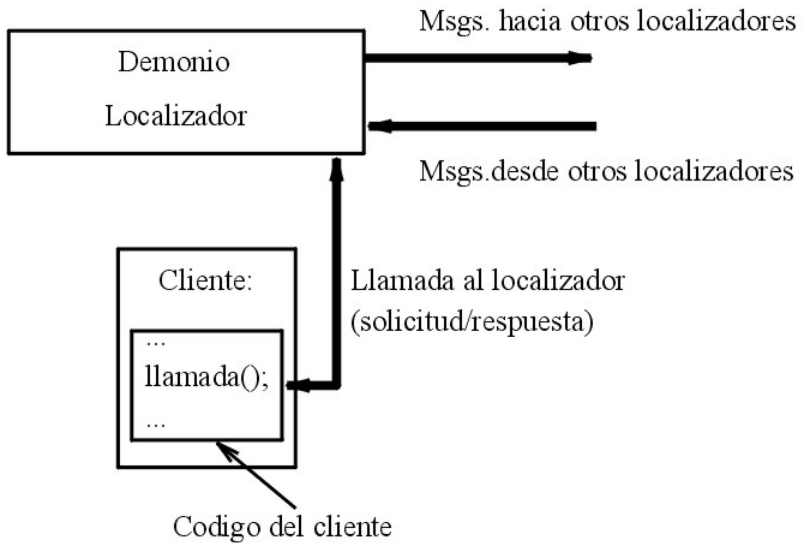


Protocolo de difusión

- Difusión débil
  1. Efectuar una difusión débil solicitando respuesta del localizador propietario
  2. Si la difusión anterior no logra encontrar al objeto, entonces efectuar una difusión débil ordenando a cada localizador recabar toda la información en los caches.
- Difusión fuerte
  1. Efectuar una difusión fuerte solicitando respuesta del localizador propietario.
  2. Si el paso anterior fracasa, difundir un mensaje ordenando la reconstrucción de la tabla propietaria.
  3. Si el paso anterior fracasa, declarar al objeto inexistente.

Interfaz

- Esta compuesta por un conjunto de funciones que denominaremos llamadas al localizador. Estas, permiten una visión centralizada del sistema (transparencia).
- Estas llamadas son invocadas desde los clientes del localizador.
- Los clientes del localizador deben enlazar una biblioteca.
- Utiliza el sistema de excepciones C++.



Llamadas de registro

- Registro de procesos

```
void register_prc(Process_Id &)
     throw(std::exception, Duplicated)
void unregister_prc(const Process_Id &)
     throw(std::exception, NotFound, RefusedService)
```

- Registro de objetos

```
void register_obj(Object_Id &, const Process_Id &)
    throw(std::exception, Duplicated, NotFound)
void unregister_obj(const Object_Id &)
     throw(std::exception, NotFound)
```

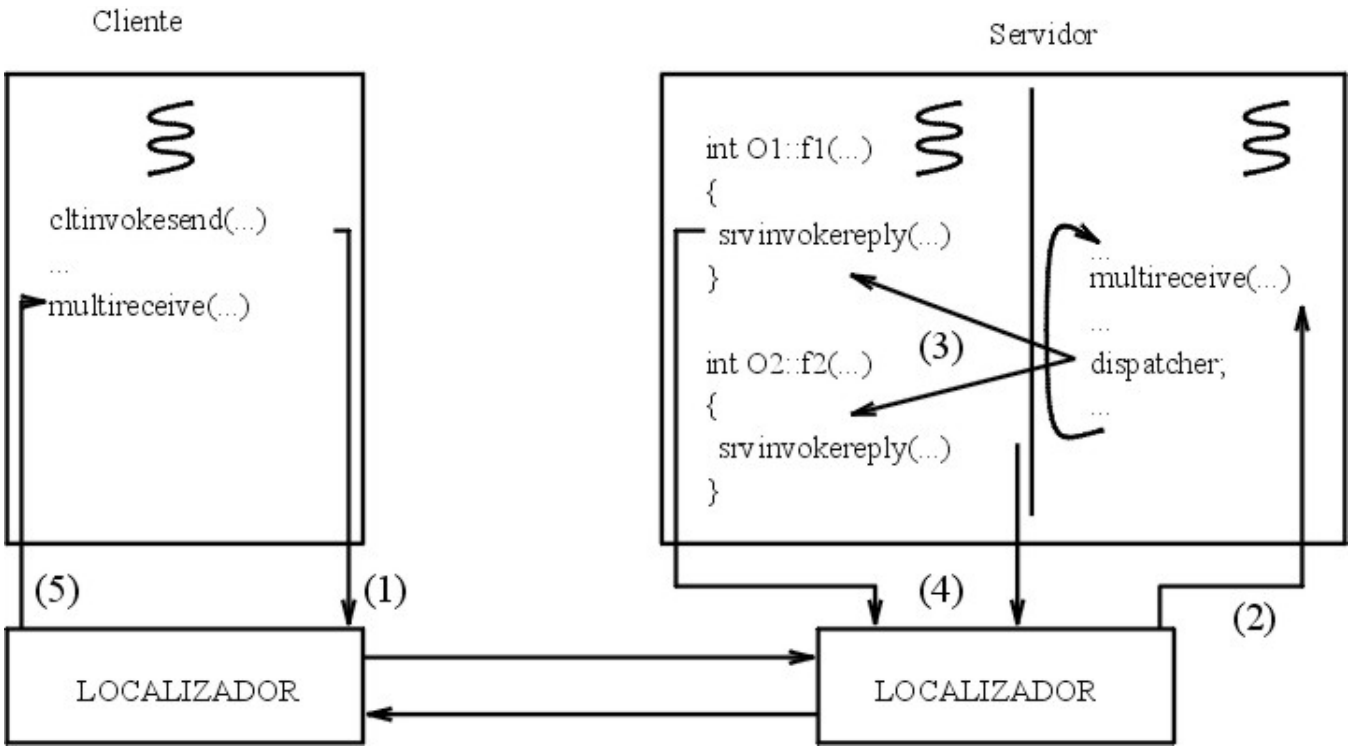
Llamadas de invocación

```
Message_Id multi_receive(Binding & binding,
                         const Process_Id & receiving_process_id,
                         void * data,
                         size_t & data_size,
                         Reception_Type & message_type)
throw (std::exception, NotFound, ObjectDead, RecentBinding)

Message_Id clt_invoke_send(Binding &, const void *, const size_t)
throw (std::exception, ObjectDead)

void srv_invoke_reply(const Message_Id &, const Binding &,
                      const Process_Id &, const void *, const size_t)
throw (std::exception)
```

Llamadas de invocación II



Llamadas de migración

– Migración de objetos

```
void src_unreg_mig_obj(const Object_Id &, const Site_Id &)
    throw(std::exception, NotFound, ObjectBusy)
void tgt_reg_mig_obj(const Object_Id &, const Process_Id &,
                     const Logical_Timestamp)
    throw(std::exception, NotFound, Duplicated)
```

– Migración de procesos

```
void src_unreg_mig_prc(const Process_Id &, const Site_Id &)
    throw(std::exception, NotFound)
void tgt_reg_mig_prc(const Process_Id &)
    throw(std::exception, Duplicated)
```

Llamadas de localización

```
Locator strong_locate(const Object_Id &)
     throw (std::exception, ObjectDead)

Locator weak_locate(const Object_Id &)
     throw (std::exception, NotFound, ObjectDead)

void implicit_locate(const Binding &)
     throw (std::exception)

void test_location(Locator &)
     throw (std::exception, NotFound, ObjectDead)

void ping(int number_of_entries, const Cache_Update_Policy policy)
     throw (std::exception, NotFound)
```

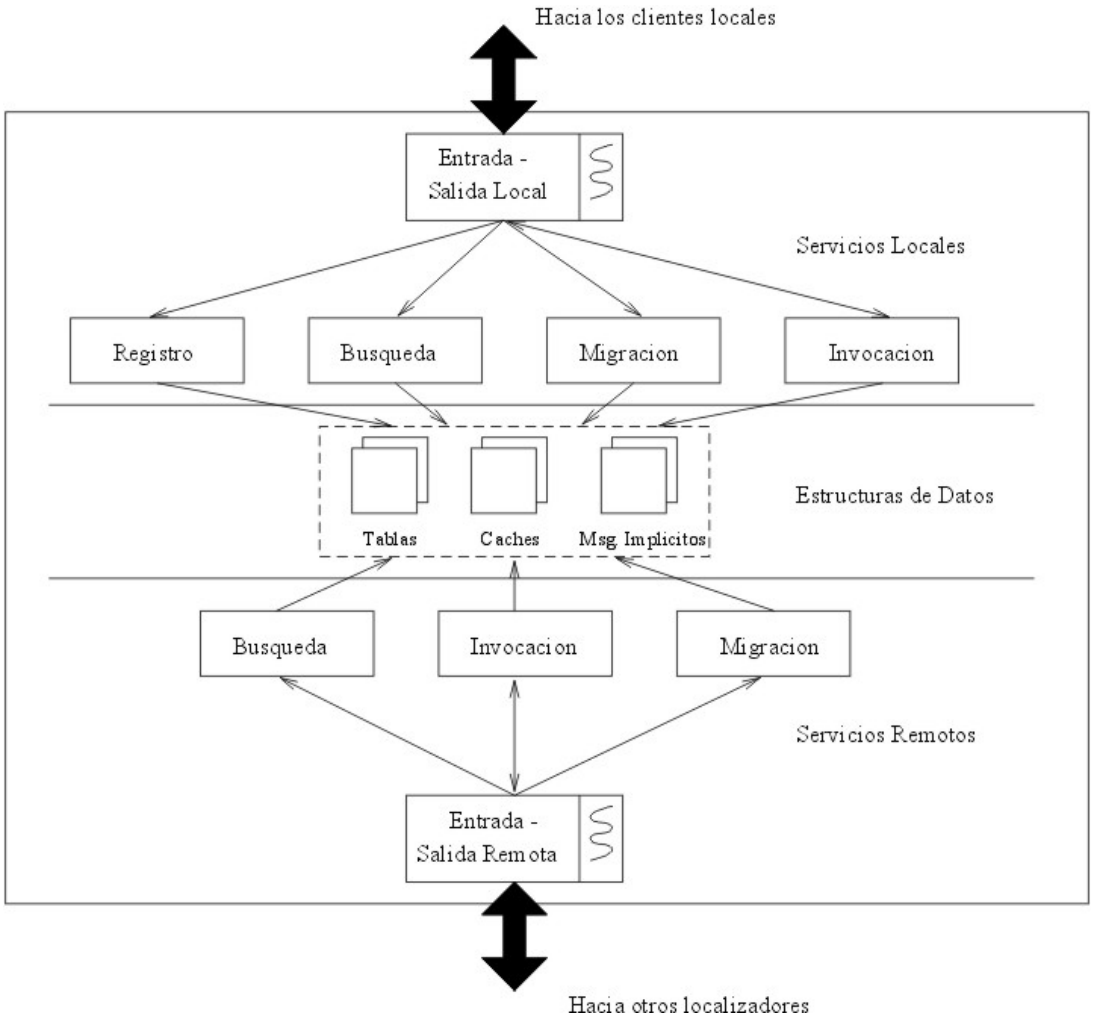
Ejemplo

```
# include "locator_calls.H"
// usage: server <number of iterations> <number of objects>
# define RECEPTION_BUFFER_SIZE 4096
int main(int argc, char ** argv)
{
  bootstrap_services();
  ...
  Site_Id   this_site(INVALID_SITE_ID);
  get_site_id(this_site);
  Object_Id this_object;
  Process_Id this_process;
  // REGISTRATION STAGE
  register_prc(this_process);
  register_obj(this_object, this_process);
  // BINDING FOR INVOCATIONS
  Binding binding;
  Message_Id msg_id;
  Reception_Type reception_type;
  size_t reception_size;
```

Ejemplo

```
for (int services = -1;
       (services < n_times - 1) || (n_times == -1);
       n_times==-1 ? n_times = -1 : services++)
    {
      reception_size = RECEPTION_BUFFER_SIZE;
      msg_id = multi_receive(binding, this_process,
                   reception_buffer, reception_size,
                   reception_type);
      // processing of the invocation
      srv_invoke_reply(msg_id, binding, this_process,
                   reception_buffer, reception_size);
    }
  unregister_prc(this_process);
return 0;
}
```

Implantación del servicio



Implantación del servicio

- Mensajes explícitos

  Un mensaje explicito tiene como función anunciar un evento de un localizador a otro, con la finalidad de ejecutar una invocación o una acción de localización.

  Tipos de mensajes explicitos:

  – Solicitud de invocación
  – Respuesta de invocación
  – Solicitud de busqueda
  – Anuncio de referencia
  – Anuncio de inexistencia
  – Indagación de existencia
  – Respuesta a la indagación
  – Registros del cache de entrada

Desempeño

- Los experimentos realizados son sintéticos, de baja escala y no representan una aplicación real.
- Permiten una primera evaluación de desempeño de las técnicas propuestas.
- 5 sitios, 2 procesos por sitio, 5 objetos por proceso y 5 referencias por proceso, lo que totaliza 50 objetos y 50 referencias en todo el sistema.
- Invocación según probabilidad p
- Migración según probabilidad q, donde $p+q=1$.

Desempeño

p	q	Exitos	Métodos
0.9	0.1	99.2%	Caching, Prefetching
0.7	0.3	94.38%	Caching, Prefetching
0.7	0.3	97.3%	Caching, Prefetching, Piggybacking ☺

Conclusiones

- Mediante una interfaz sencilla es posible llevar a cabo tareas de localización, migración e invocación.
- El sistema es portable entre las distintas versiones unix, pues utiliza codigo conforme a POSIX.
- El sistema de localización de objetos sirve como base para el desarrollo de aplicaciones distribuidas orientadas a objetos.
- Los objetos pueden moverse a voluntad. Esto tiene una amplia aplicación en el campo de balance de carga en los sistemas distribuidos.
- El sistema es escalable. Esta propiedad se hereda de la arquitectura del sistema y de las técnicas de actualización de referencias.
- Este es un sistema con un alto grado de cohesión y de mínimo acoplamiento, lo que implica que el sistema se pueda ampliar y mantener con mayor facilidad.

Perspectivas

- Lazos de persecución a nivel de transporte
- Captura de objetos rápidos
- Modelos analíticos
- Modelos de simulación
- Experimentos de mayor escala

Gracias por su atención.