



UNIVERSIDAD TÉCNICA
FEDERICO SANTA MARÍA

Comunicación Middleware

INF-343 Sistemas Distribuidos

Prof. Jorge Díaz M.

April 2, 2024





Table of Contents

1 Comunicación entre procesos

► Comunicación entre procesos

► Invocación Remota

► Comunicación Indirecta

Comunicación entre procesos

1 Comunicación entre procesos

- El **paso de mensajes** está soportado por dos operaciones de comunicación definidas en términos de destinos y mensajes.
 - *Send/enviar* (una secuencia de bytes) a un destino.
 - *Receive/recibir* el mensaje.
- Se asocia una **cola** con cada destino de mensaje.
 - *Enviar*: agrega a la cola remota.
 - *Recibir* Recibir elimina de las colas locales.



Comunicación síncrona y asíncrona

1 Comunicación entre procesos

Comunicación síncrona

- Los procesos de envío y recepción se sincronizan en cada mensaje.
- Bloquean operaciones.
- Al *enviar* -> el proceso de envío se bloquea hasta que se emite la recepción.
- Cuando *recibe* -> se bloquea hasta que llega un mensaje.

Comunicación asíncrona

- *Enviar* y *recibir* no son operaciones bloqueantes.
- Al *enviar* -> el proceso de envío puede continuar tan pronto como el mensaje esté en el búfer local.
- Cuando se *recibe* -> el búfer se puede llenar en segundo plano (notificación posterior -> sondeo o interrupción)

Comunicación síncrona y asíncrona

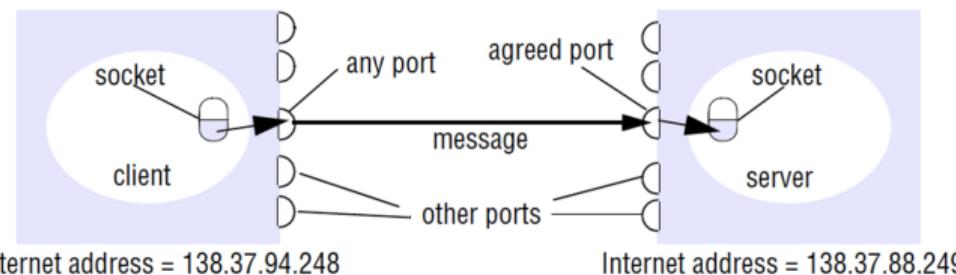
1 Comunicación entre procesos

- Al hacer subprocessos múltiples -> bloquear la recepción no tiene desventajas.
 - Otros hilos permanecen activos.
- El sincronismo es simple, lo cual es una ventaja sustancial.
- La comunicación sin bloqueo parece ser más eficiente
 - Pero implica una complejidad extra en el proceso de recepción.
 - Adquirir el mensaje fuera del flujo de control

Socket

1 Comunicación entre procesos

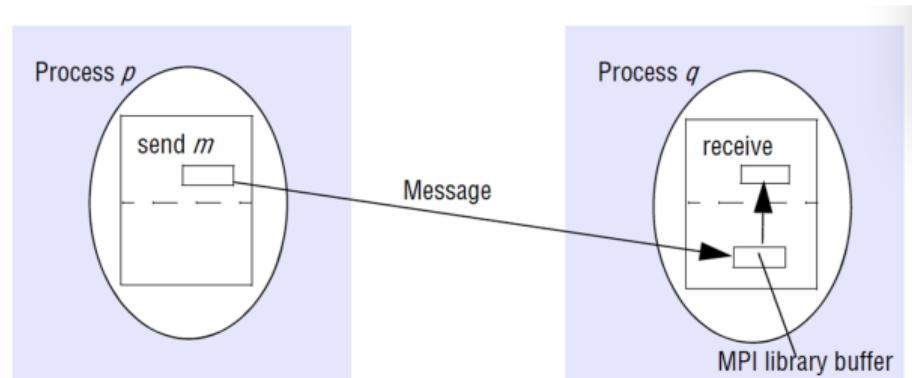
- Los sockets proporcionan comunicación de punto final entre procesos.
- La **comunicación entre procesos** consiste en transmitir un mensaje entre un socket en un proceso y un socket en otro proceso.
- El socket de recepción debe estar vinculado a un puerto local y una dirección de Internet.
- 2^{16} posibles números de puerto disponibles, no se pueden compartir entre procesos.



Estudio de caso: MPI Interfaz de paso de mensajes

1 Comunicación entre procesos

- MPI es una especificación, no una implementación.
- Intercambio síncrono de mensajes entre dos procesos mediante operaciones de envío y recepción.
- Centrarse en la informática de alto rendimiento
- Diseñado para ser simple, práctico, eficiente y portátil.
- Las aplicaciones usan MPI a través de la biblioteca de paso de mensajes disponible para una variedad de sistemas operativos y lenguajes de programación.





Estudio de caso: MPI Interfaz de paso de mensajes

1 Comunicación entre procesos

<i>Send operations</i>	<i>Blocking</i>	<i>Non-blocking</i>
<i>Generic</i>	<i>MPI_Send</i> : the sender blocks until it is safe to return – that is, until the message is in transit or delivered and the sender's application buffer can therefore be reused.	<i>MPI_Isend</i> : the call returns immediately and the programmer is given a communication request handle, which can then be used to check the progress of the call via <i>MPI_Wait</i> or <i>MPI_Test</i> .
<i>Synchronous</i>	<i>MPI_Ssend</i> : the sender and receiver synchronize and the call only returns when the message has been delivered at the receiving end.	<i>MPI_Issend</i> : as with <i>MPI_Isend</i> , but with <i>MPI_Wait</i> and <i>MPI_Test</i> indicating whether the message has been delivered at the receive end.
<i>Buffered</i>	<i>MPI_Bsend</i> : the sender explicitly allocates an MPI buffer library (using a separate <i>MPI_Buffer_attach</i> call) and the call returns when the data is successfully copied into this buffer.	<i>MPI_Ibsend</i> : as with <i>MPI_Isend</i> but with <i>MPI_Wait</i> and <i>MPI_Test</i> indicating whether the message has been copied into the sender's MPI buffer and hence is in transit.
<i>Ready</i>	<i>MPI_Rsend</i> : the call returns when the sender's application buffer can be reused (as with <i>MPI_Send</i>), but the programmer is also indicating to the library that the receiver is ready to receive the message, resulting in potential optimization of the underlying implementation.	<i>MPI_Irsend</i> : the effect is as with <i>MPI_Isend</i> , but as with <i>MPI_Rsend</i> , the programmer is indicating to the underlying implementation that the receiver is guaranteed to be ready to receive (resulting in the same optimizations),



Ejemplo básico MPI

1 Comunicación entre procesos

```
#include "mpi.h"
int main( int argc, char *argv[])
{
    char message[20];
    int myrank;
    MPI_Status status;
    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
    if (myrank == 0) /* code for process zero */
    {
        strcpy(message,"Hello, there");
        MPI_Send(message, strlen(message)+1, MPI_CHAR, 1, 99, MPI_COMM_WORLD);
    }
    else if (myrank == 1) /* code for process one */
    {
        MPI_Recv(message, 20, MPI_CHAR, 0, 99, MPI_COMM_WORLD, &status);
        printf("received :%s:\n", message);
    }
    MPI_Finalize();
    return 0;
}
```

Representación y marshalling de datos externos

1 Comunicación entre procesos

- La información en los programas se representa como **estructuras de datos** y la información en los mensajes consiste en **secuencias de bytes**.
- La estructura de datos debe aplanarse (convertirse) antes de la transmisión y reconstruirse al llegar.
- Problemas: orden little-endian o big-endian; Estándares ASCII o Unicode.
- Dos métodos para intercambiar valores de datos binarios:
 1. Los valores se convierten a un **formato externo acordado** y se convierten a la forma local al recibirlas.
 2. Los valores se transmiten en el **formato del remitente con una indicación** del formato utilizado.

Representación y marshalling de datos externos

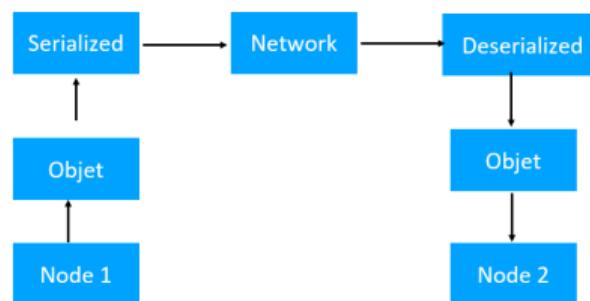
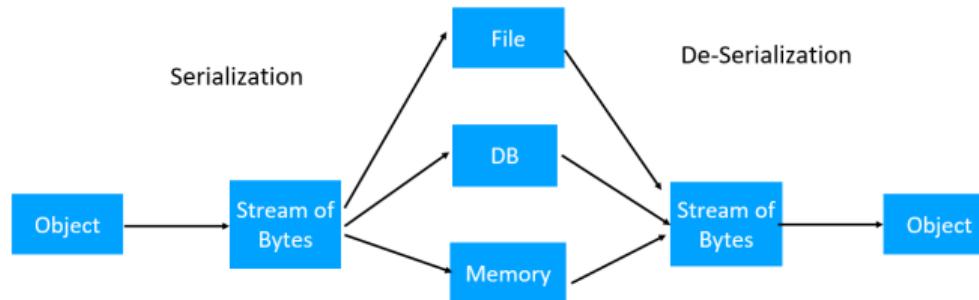
1 Comunicación entre procesos

- **Marshalling:** es el proceso de tomar una colección de elementos de datos y ensamblarlos en una forma adecuada para la transmisión en un mensaje.
- **Desarmado (Unmarshalling):** es el proceso de desensamblar los elementos a su llegada para producir una colección equivalente de elementos de datos en el destino.

- Serialización de objetos de Java: aplanamiento y representación de datos externos de cualquier objeto único o árbol de objetos.
- XML (Lenguaje de marcado extensible) define un formato textual para representar datos estructurados.
- JSON (Notación de objetos de JavaScript) para la representación de datos externos.
- Protocol buffers (la forma en que Google serializa datos estructurados)

Representación y marshalling de datos externos

1 Comunicación entre procesos



Serialización de objetos Java

1 Comunicación entre procesos

- La información sobre el nombre de la clase se incluye en el formulario serializado.
- Los objetos Java que contienen referencias a otros objetos se serializan juntos para garantizar una reconstrucción completa.
- Las referencias se serializan como identificadores: una referencia a un objeto dentro de un formulario serializado.

La interfaz serializable permite codificar sus instancias.

```
public class Person implements Serializable {  
    private String name;  
    private String place;  
    private int year;  
    public Person(String aName, String aPlace, int aYear) {  
        name = aName;  
        place = aPlace;  
        year = aYear;  
    }  
    //followed by methods for accessing the instance variables  
}
```

Serialización de objetos Java

1 Comunicación entre procesos

- Person p = new Person (“Smith” , “London” , 1984)
- Cree una instancia de la clase ObjectOutputStream e invoque el método writeObject, pasando el objeto Person como argumento.
- Para deserializar use ObjectInputStream y el método readObject.

Serialized values				Explanation
Person	8-byte version number		h0	class name, version number
3	int year	java.lang.String name	java.lang.String place	number, type and name of instance variables
1984	5 Smith	6 London	h1	values of instance variables

The true serialized form contains additional type markers; h0 and h1 are handles.

Serialización específica del lenguaje

1 Comunicación entre procesos

- Java -> java.io.Serializable
- Ruby -> Marshal
- Python -> pickle
- Cómodas y convenientes, no se agrega casi nada de código.
 - Problemas: Dependiente del lenguaje, leerlo desde un lenguaje diferente es difícil.
 - Problemas de seguridad.
 - Problemas de rendimiento.



Formatos Textuales

1 Comunicación entre procesos

XML

```
<empinfo>
  <employees>
    <employee>
      <name>James Kirk</name>
      <age>40</age>
    </employee>
    <employee>
      <name>Jean-Luc Picard</name>
      <age>45</age>
    </employee>
    <employee>
      <name>Wesley Crusher</name>
      <age>27</age>
    </employee>
  </employees>
</empinfo>
```

JSON

```
{
  "empinfo" : {
    "employees" : [
      {
        "name" : "James Kirk",
        "age" : 40,
      },
      {
        "name" : "Jean-Luc Picard",
        "age" : 45,
      },
      {
        "name" : "Wesley Crusher",
        "age" : 27,
      }
    ]
  }
}
```

JSON, XML y variantes Binarias

1 Comunicación entre procesos



- JSON, XML, CSV formatos textuales.
- Ambigüedad al codificar números y string que muestran números, o entre tipos de números como enteros y punto flotante.
- Se puede soportar esquema opcionalmente en JSON y XML.
- Suficientemente buenos para múltiples propósitos.



Serialización binaria

1 Comunicación entre procesos

- Para comunicación interna en una organización se puede usar un formato más compacto y rápido.
- Para datasets pequeños la ganancia no es mucha, pero si cuando se habla de terabytes de datos.
- Codificación binaria de JSON (MessagePack, BSON, BISON, Etc.)
- Ejemplo MessagePack bajar de 81 bytes a 66 bytes

La interfaz serializable permite codificar sus instancias.

MessagePack

Byte sequence (66 bytes):

83	a8	75	73	65	72	4e	61	6d	65	a6	4d	61	72	74	69	6e	ae	66	61
76	6f	72	69	74	65	4e	75	6d	62	65	72	cd	05	39	a9	69	6e	74	65
72	65	73	74	73	92	ab	64	61	79	64	72	65	61	6d	69	6e	67	a7	68
61	63	6b	69	6e	67														

Breakdown:

object (3 entries)	string (length 8)	u	s	e	r	N	a	m	e	string (length 6)	M	a	r	t	i	n	n	
83	a8	75	73	65	72	4e	61	6d	65	a6	4d	61	72	74	69	6e	ae	66
	ae	66	61	76	6f	72	69	74	65	4e	75	6d	62	65	72			
		66	61	79	64	72	65	61	6d	69	6e	67	a7	68				
array (2 entries)	uint16	1337								string (length 9)	i	n	t	e	r	e	s	
92	cd	05	39	a9	69	6e	74	65	72	65	73	74	73					
	ab	64	61	79	64	72	65	61	6d	69	6e	67						
	a7	68	61	63	6b	69	6e	67										

Protocol Buffers

1 Comunicación entre procesos



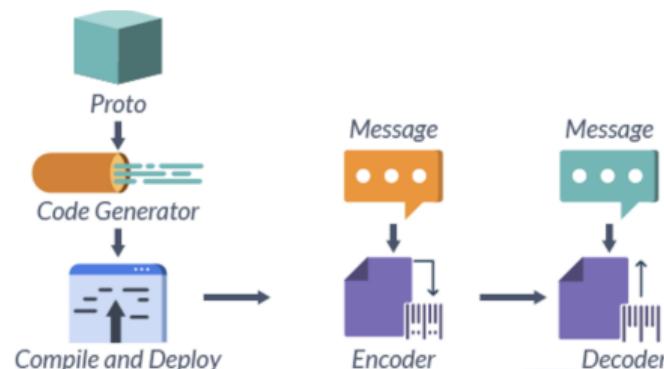
- Mecanismo para serializar datos estructurados, neutral al lenguaje y plataforma.
- Diseñado por Google en el 2001, todo lo que había en ese momento no era tan bueno.
- Se volvió open source en el 2008.
- Estable y confiable.
- Protocol buffers es el pegamento de todos los servicios de Google.
- Soporta C++, C#, Java, Python, JavaScript, Go, Kotlin, Dart, PHP, Ruby.
- Último release v3.21.5

Protocol Buffers

1 Comunicación entre procesos



- **Ventajas:** Formato binario es más rápido que formato de texto. Buen rendimiento.
- **Desventajas:** No es flexible al esquema





Protocol Buffers

1 Comunicación entre procesos

```
message Person {
    required string name = 1;
    required int32 id = 2;
    optional string email = 3;

enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
}

message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
}

repeated PhoneNumber phone = 4;
}
```

```
Person person;
person.set_name("John Doe");
person.set_id(1234);
person.set_email("jdoe@example.com");
fstream output("myfile", ios::out | ios::binary);
person.SerializeToOstream(&output);
```

```
fstream input("myfile", ios::in | ios::binary);
Person person;
person.ParseFromIstream(&input);
cout << "Name: " << person.name() << endl;
cout << "E-mail: " << person.email() << endl;
```

Ventajas sobre XML

- Más simple y menos ambiguo
- 3 a 10 veces más pequeño y 20 a 100 veces más rápido
- Genera clases de acceso a datos que sean más fáciles de usar mediante programación



Protocol Buffers

1 Comunicación entre procesos

JSON

```
{  
    "userName": "Martin",  
    "favouriteNumber": 1337,  
    "interests": ["daydreaming", "hacking"]  
}
```

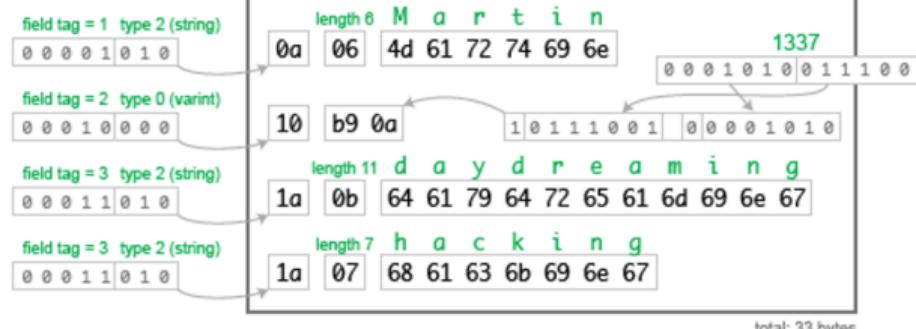
Encoded in 84 bytes

Protocol buffers

```
message Person {  
    required string user_name      = 1;  
    optional int64 favourite_number = 2;  
    repeated string interests     = 3;  
}
```

Encoded in 33 bytes

Protocol Buffers



Apache Thrift

1 Comunicación entre procesos



- Framework para el desarrollo de servicios entre diferentes lenguajes.
- Diseñado en el 2007 internamente en Facebook.
- Es open source, un proyecto Apache.
- Similar a Protocol buffers.
- Lenguajes soportados: C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, JavaScript, Node.js, Smalltalk, OCaml, Delphi, Dart, .NET, Go, etc.
- Incluye stack de RPC (próxima clase), no solo serialización.

Apache Thrift

1 Comunicación entre procesos

- Ventajas:
 - Soporta más lenguajes.
 - Soporta estructuras de datos más ricas (Map, Set)
- Desventajas:
 - Documentación no es tan buena como protocol buffers.

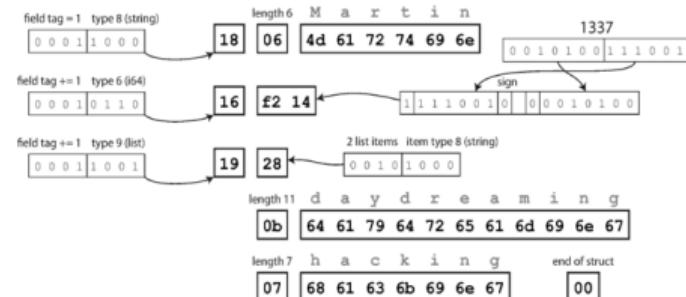
```
struct Person {  
    1: required string      userName,  
    2: optional i64          favoriteNumber,  
    3: optional list<string> interests  
}
```

Thrift CompactProtocol

Byte sequence (34 bytes):

18	06	4d	61	72	74	69	6e	16	f2	14	19	28	0b	64	61	79	64	72	65
61	6d	69	6e	67	07	68	61	63	6b	69	6e	67	00						

Breakdown:



Avro

1 Comunicación entre procesos



- Sistema de serialización de datos hecho para los servicios de Hadoop, para facilitar el intercambio de Big Data.
- Permite esquema dinámico. El esquema también se serializa, se define en JSON.
- Se usa para construir sistemas menos acoplados.
- No necesita compilar.
- Incluye RPC.
- Lenguajes soportados: Java, C, C++, C#, Python, Ruby, Perl, PHP.



Avro

1 Comunicación entre procesos

```
record Person {  
    string          userName;  
    union { null, long } favoriteNumber = null;  
    array<string>   interests;  
}
```

```
{  
    "type": "record",  
    "name": "Person",  
    "fields": [  
        {"name": "userName",      "type": "string"},  
        {"name": "favoriteNumber", "type": ["null", "long"], "default": null},  
        {"name": "interests",     "type": {"type": "array", "items": "string"}}  
    ]  
}
```

- No hay números, el esquema del lector puede ser diferente al escritor



Comparación rendimiento

1 Comunicación entre procesos

	Server CPU %	Avg. Client CPU %	Avg. Time
REST — XML	12.00%	80.75%	05:27.45
REST — JSON	20.00%	75.00%	04:44.83
RMI	16.00%	46.50%	02:14.54
Protocol Buffers	30.00%	37.75%	01:19.48
Thrift — TBinaryProtocol	33.00%	21.00%	01:13.65
Thrift — TCompactProtocol	30.00%	22.50%	01:05.12



UNIVERSIDAD TÉCNICA
FEDERICO SANTA MARÍA

Table of Contents

2 Invocación Remota

► Comunicación entre procesos

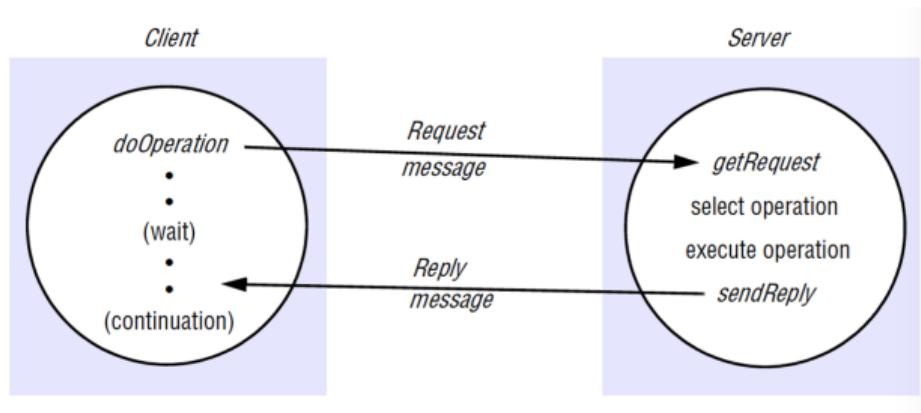
► Invocación Remota

► Comunicación Indirecta

Protocolos Request-reply

2 Invocación Remota

- Para interacciones típicas de cliente-servidor: Intercambio bidireccional de mensajes.
- Generalmente síncrono y confiable.
- Cuando se usa UDP, evitamos el overhead of TCP debido a:
 - ACK son redundantes, ya que las solicitudes son seguidas por respuestas en este modelo.
 - El establecimiento de una conexión implica dos pares de mensajes adicionales además del par necesario para una solicitud y una respuesta.





Comunicación síncrona y asíncrona

2 Invocación Remota

**Public byte[] doOperation(RemoteRef
s, int operationId, byte[] arguments)**

Envía un mensaje de solicitud al
servidor remoto y devuelve la
respuesta.

Public byte[] getRequest()

Adquiere una solicitud de cliente a
través del puerto del servidor. **Public**

**void sendReply(byte[] reply,
InetAddress clientHost, int clientPort)**

Envía el mensaje de respuesta al
cliente como sus direcciones de
Internet y puerto.

messageType	<i>int (0=Request, 1=Reply)</i>
requestId	<i>int</i>
remoteReference	<i>RemoteRef</i>
operationId	<i>int or Operation</i>
arguments	<i>// array of bytes</i>



Protocolos Request-reply

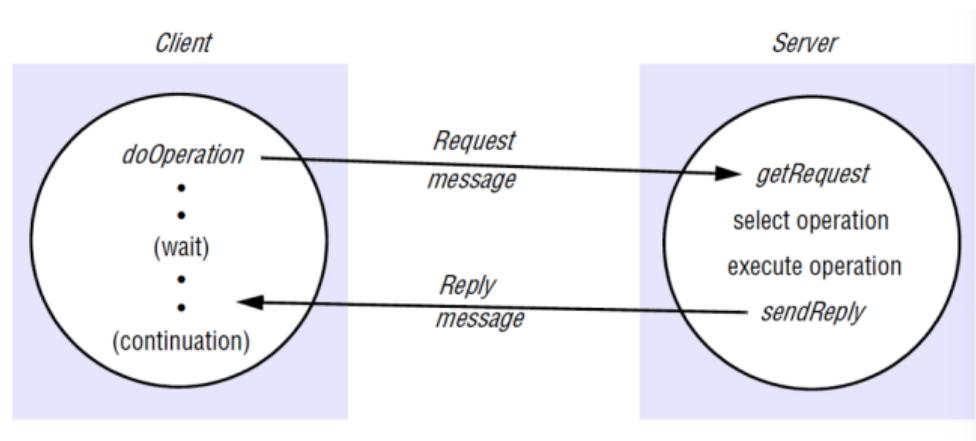
2 Invocación Remota

Identificador del
Mensaje

Uso de TimeOuts

Descartar peticiones
repetidas

Historial de mensajes de
respuesta perdidos



Estilos de protocolos de intercambio

2 Invocación Remota

- Si se utiliza TCP:
 - No hay necesidad de lidiar con retransmisiones de mensajes y filtrar duplicados o con historiales.
 - El control de flujo permite grandes argumentos.
 - Es de implementación simple
- Recordar las características de HTTP del curso de Redes de Computadores



Recordando HTTP

2 Invocación Remota

Está implementado sobre TCP

1. Conexión entre cliente y servidor
2. Cliente envía una petición
3. Servidor envía una respuesta
4. Se cierra la conexión

Las solicitudes y las respuestas se ordenan en cadenas de texto ASCII

Algunos métodos:

- GET: solicita el recurso cuya URL se da como argumento.
- POST: especifica la URL de un recurso que puede manejar los datos en el cuerpo de la solicitud.
- PUT: Solicitar que los datos suministrados en la solicitud se almacenen con una URL dada

Llamada a procedimiento remote (RPC)

2 Invocación Remota

- Extiende la abstracción de una llamada de procedimiento a entornos distribuidos.
- Los **procedimientos** se pueden llamar como si estuvieran en el **espacio de direcciones locales**.
- Oculta la codificación, decodificación de parámetros y resultados, el paso de mensajes.

Programando con interfaces

2 Invocación Remota

- **Service interface:** se refiere a la especificación de los procedimientos ofrecidos por un servidor, definiendo los tipos de argumentos.
- **Beneficios:** programación modular, abstracciones de los detalles de implementación, soporte natural para la evolución
- La interfaz de servicio no puede especificar el acceso directo a las variables (captador y definidor)
- Paso de parámetros como entrada o salida.
- Las direcciones no pueden pasarse como argumentos ni devolverse como resultado de llamadas a módulos remotos.

Las IDLs (Interface definition languages) están diseñadas para permitir que los procedimientos implementados en diferentes idiomas se invoquen entre sí.

Semántica y transparencia en RPC

2 Invocación Remota

Fault tolerance measures			Call semantics
<i>Retransmit request message</i>	<i>Duplicate filtering</i>	<i>Re-execute procedure or retransmit reply</i>	
No	Not applicable	Not applicable	<i>Maybe</i>
Yes	No	Re-execute procedure	<i>At-least-once</i>
Yes	Yes	Retransmit reply	<i>At-most-once</i>

Se aceptan llamadas fallidas ocasionales

Para operaciones idempotentes

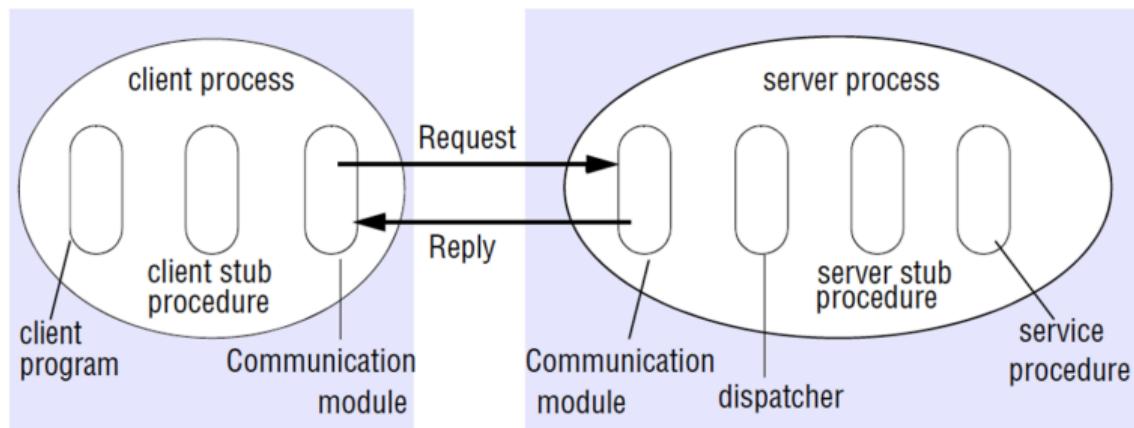
Para operaciones no idempotentes

- El fracaso es un problema mayor que en el espacio local.
- La latencia de RPC es varios órdenes de magnitud mayor que la de uno local.
- La persona que llama debería poder abortar la RPC.
- RPC no ofrece paso de parámetros por referencia.

Semántica y transparencia en RPC

2 Invocación Remota

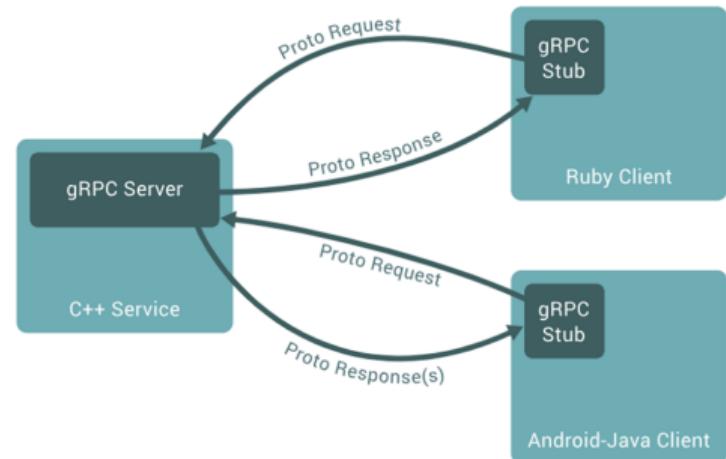
- El cliente contiene un **stub procedure** para cada procedimiento en la interfaz de servicio
- El procedimiento stub se comporta como un procedimiento local para el cliente, pero ordena el identificador y los argumentos del procedimiento en un mensaje de solicitud, que se envía a través del **módulo de comunicación**.
- El servidor contiene un **dispatcher** con un **stub server** y un procedimiento de servicio para cada service interface.
- Implementado como protocolo Request-Reply.



Ejemplo: gRPC

2 Invocación Remota

- Framework RPC de alto rendimiento que puede ejecutarse en cualquier entorno (10 lenguajes de programación).
- Conecte de manera eficiente los servicios en y entre los centros de datos.
- Compatibilidad conectable para equilibrio de carga, seguimiento, verificación de estado y autenticación.
- También para dispositivos móviles, navegadores y servicios backend.
- Se basa en el transporte HTTP/2.
- Uso de Protocol buffers como IDL y el formato de intercambio de mensajes subyacente.



Invocación de método remoto (RMI)

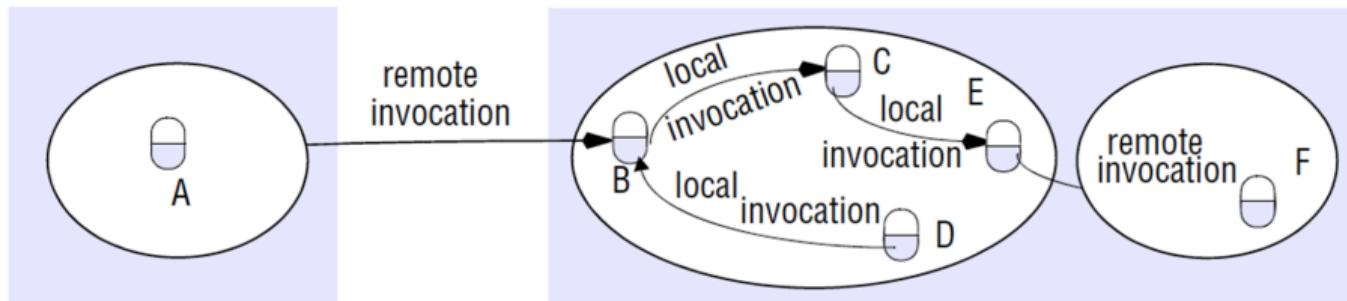
2 Invocación Remota

- Un objeto que llama puede invocar un método en un objeto potencialmente remoto.
- Cosas en común con RPC:
 - Soportan programación con interfaces.
 - Por lo general, se construyen sobre protocolos **request-reply** y pueden ofrecer una semántica similar.
 - Ofrecen un nivel similar de transparencia.
- Diferencias con RPC:
 - En RMI es posible utilizar metodologías de diseño más **orientadas a objetos**, uso de objetos, clases y herencia.
 - Todos los objetos en un sistema basado en RMI tienen referencias de objeto únicas (locales o remotas), se pueden pasar como parámetros.
 - RMI permite pasar parámetros por valor y referencia de objeto.

Modelo de Objetos Distribuidos

2 Invocación Remota

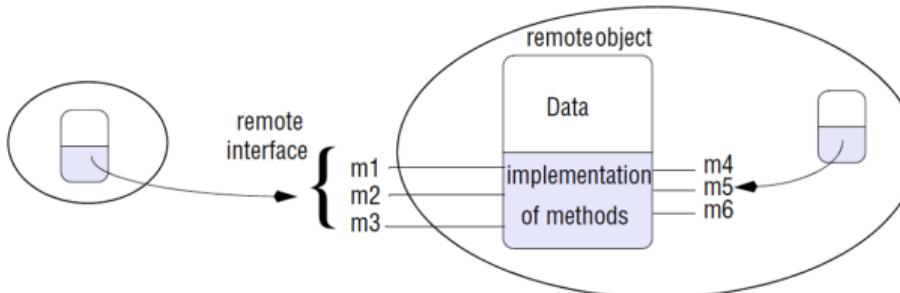
- Cada proceso contiene una colección de objetos, algunos de los cuales pueden recibir invocaciones tanto locales como remotas.
- **Objetos remotos:** Los objetos pueden recibir invocaciones remotas (B,F).
- **Referencia a objetos remotos:** La referencia a objetos remotos para B debe estar disponible para A.
- **Interfaces Remotas:** Cada objeto remoto tiene una interfaz remota que especifica cuál de sus métodos se puede invocar de forma remota.



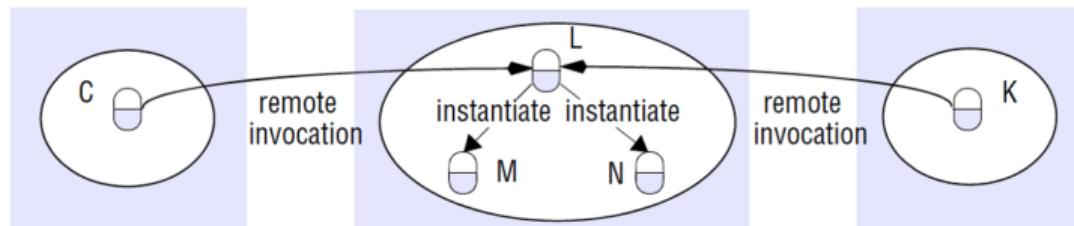
Modelo de Objetos Distribuidos

2 Invocación Remota

- Un objeto remoto y su interfaz remota



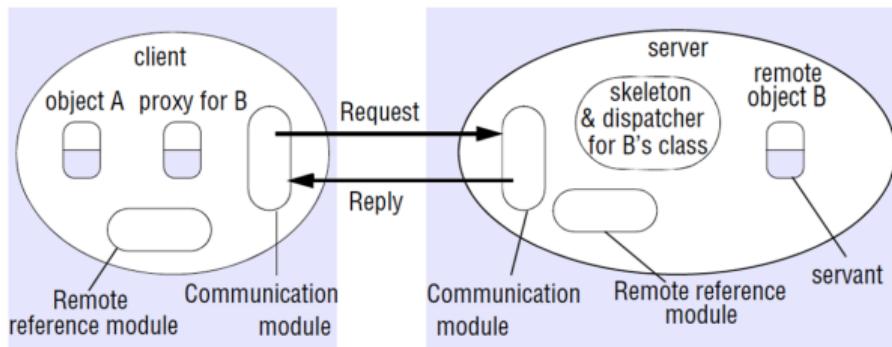
- Instanciación de objetos remotos



Modelo de Objetos Distribuidos

2 Invocación Remota

- **Modelo de Comunicación:** Llevar a cabo el protocolo request-reply protocol.
- El módulo de comunicación selecciona el despachador para la clase del objeto a invocar, pasando su referencia local, que obtiene del módulo de referencia remota a cambio del identificador de objeto remoto en el mensaje de solicitud.
- **Módulo de referencia remota:** responsable de traducir las referencias de objetos locales y remotos, y de crear referencias de objetos remotos. Contiene **una tabla de objetos remotos** que registra esta correspondencia.



Implementación de RMI

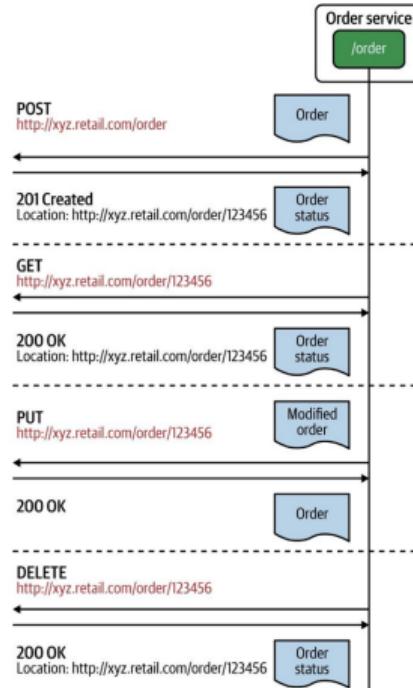
2 Invocación Remota

- **Servant:** instancia de una clase que proporciona el cuerpo de un objeto remoto. Maneja la solicitud remota transmitida por los esqueletos. Se crea cuando se instancian los objetos remotos.
- El software RMI: capa entre la aplicación y la comunicación.
- **Proxy:** comportarse como un objeto local para el invocador. Uno para cada objeto remoto. Ordena una referencia al objeto de destino, su propia identificación operativa y sus argumentos en un mensaje de solicitud y lo envía al destino.
- **Dispatcher:** Recibe mensajes de solicitud del módulo de comunicación y selecciona el método apropiado en el esqueleto, transmitiendo el mensaje de solicitud.
- **Skeleton:** Implementa los métodos en la interfaz remota. Ejecuta el unmarshalling de los argumentos en el mensaje de solicitud e invoca el método correspondiente en el sirviente.

Flujos de datos a través de servicios: REST - RPC - GraphQL

2 Invocación Remota

- Paradigmas de API Request-Response
- REST
 - No es un protocolo, sino que una filosofía de diseño.
 - Centrado en recursos identificados con URLs con operaciones CRUD (Create, Read, Update, Delete)
 - Típicamente JSON como respuesta de la API.
 - Para almacenar y rescatar información.
 - Sobre HTTP2 funciona bien pues se pasa a binario el JSON.
 - Predominante para API pública.
- RPC
- GraphQL



Flujos de datos a través de servicios: REST - RPC - GraphQL

2 Invocación Remota

- REST
- RPC
 - Nueva generación de frameworks gRPC, Avro, Thrift.
 - Realizan codificado binario.
 - Liviano y de alto rendimiento, se usa cuando la eficiencia y throughput es importante.
 - Usar cuando la comunicación es entre dos componentes de un sistema distribuido controlado por una misma organización, típicamente en un mismo datacenter.
 - Hay un acoplamiento en ambos lados de la comunicación.
- GraphQL

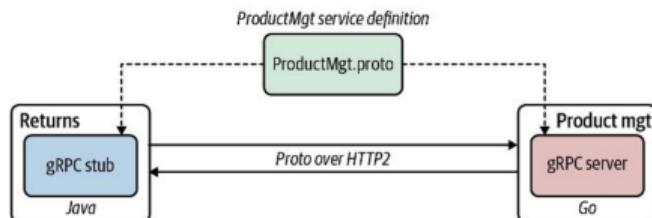
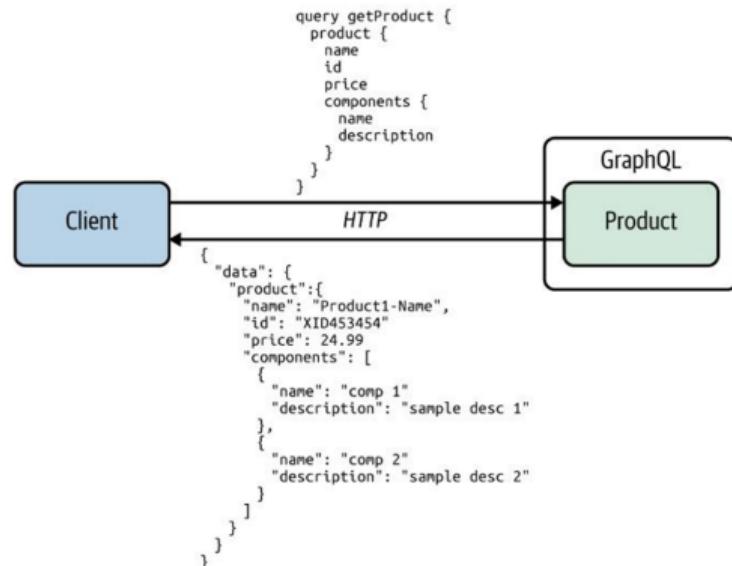


Figure 2-12. An example of microservices communication using gRPC

Flujos de datos a través de servicios: REST - RPC - GraphQL

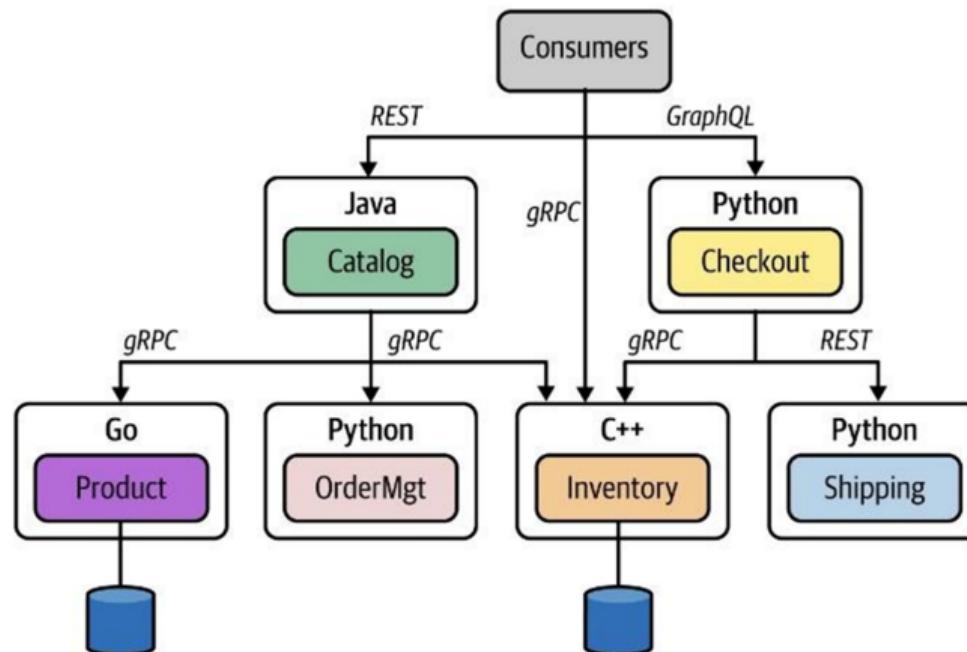
2 Invocación Remota

- REST
 - Casos con consultas/resuestas más complejas, por ejemplo altamente dimensionales en datasets grandes.
 - Se traspasa la cantidad de información exacta que se necesita mejorando costos de red y eficiencia.
 - Un solo Endpoint para múltiples tipos de consultas.
 - Uso cuando una REST API se hace muy compleja.
 - No ideales si la API es simple, dado que agregan complejidad.
- RPC
- GraphQL



Flujos de datos a través de servicios: REST - RPC - GraphQL

2 Invocación Remota





UNIVERSIDAD TÉCNICA
FEDERICO SANTA MARÍA

Table of Contents

3 Comunicación Indirecta

- ▶ Comunicación entre procesos
- ▶ Invocación Remota
- ▶ Comunicación Indirecta

Comunicación Indirecta

3 Comunicación Indirecta

Comunicación entre entidades en un sistema distribuido a través de un intermediario sin acoplamiento directo entre el emisor y el (los) receptor(es).

- Propiedades
 - **Desacoplamiento espacial:** el remitente no conoce la identidad del (los) receptor(es) y viceversa.
 - **Desacoplamiento de tiempo:** el remitente y el (los) receptor (es) pueden tener vidas independientes.
- Desventajas
 - Sobrecarga de rendimiento introducida por el nivel adicional de direccionamiento indirecto.
- Las dimensiones y el nivel de desacoplamiento varían de un sistema a otro.
- ¿Qué pasa con la comunicación asíncrona?

Comunicación Grupal

3 Comunicación Indirecta

- La comunicación se realiza a través de una abstracción grupal en la que el remitente desconoce la identidad de los destinatarios.
- Abstracción sobre comunicación multicast.
- Puede implementarse sobre red IP multicast o overlay, agregando valor a la pertenencia al grupo, detectando fallas y brindando confiabilidad y garantías de orden.

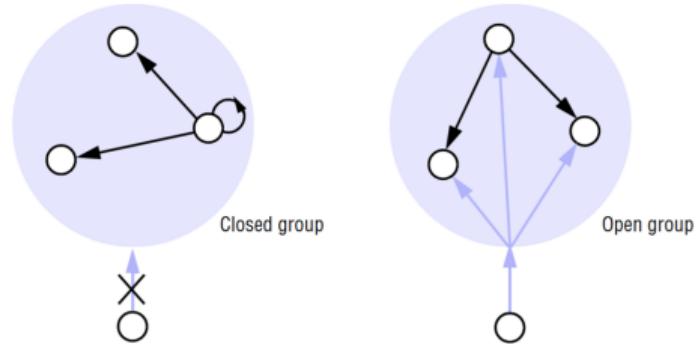
Principales Aplicaciones:

- **Difusión confiable** de información a un número potencialmente grande de clientes (sector financiero).
- Soporte para **aplicaciones colaborativas**, donde los eventos deben ser difundidos a múltiples usuarios para preservar una vista común (juegos multiusuario).
- **Estrategias tolerantes a fallas**: actualización constante de datos replicados o servidores de alta disponibilidad.
- **Supervisión y gestión del sistema** (estrategias de equilibrio de carga)

Modelo de programación

3 Comunicación Indirecta

- Concepto central GRUPO con membresía GRUPAL asociada.
- Los procesos pueden unirse o salir del grupo.
- Luego, los procesos pueden enviar un mensaje a este grupo y hacer que se propague a todos los miembros del grupo.
- Solo una operación de multidifusión para enviar un mensaje a cada uno de un grupo de procesos.
- Sistemas síncronos y asíncronos.



- **Grupo cerrado:** solo los miembros del grupo pueden multidifundir.
- **Grupo abierto:** los procesos fuera del grupo pueden multidifundirse.



Problemas de Implementación

3 Comunicación Indirecta

Confiabilidad

- Integridad: entregar el mensaje correctamente como máximo una vez.
- Vigencia: el mensaje enviado eventualmente será entregado.
- Acuerdo: si un mensaje se entrega a un proceso, se entrega a todos los procesos del grupo.

Orden

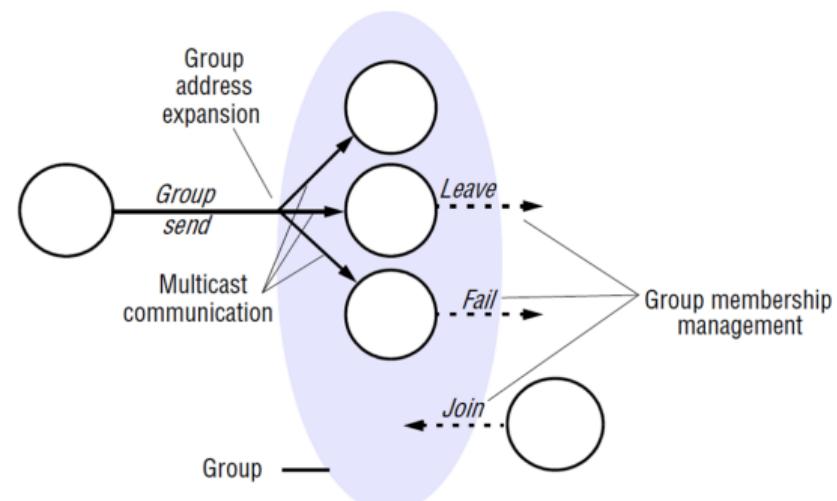
- FIFO: preservación del orden desde la perspectiva de un emisor.
- Ordenación causal: tiene en cuenta las relaciones causales entre los mensajes.
- Pedido total: Si se entrega un mensaje antes que otro en un proceso, entonces se conservará el mismo pedido en todos los procesos.

Problemas de Implementación

3 Comunicación Indirecta

Tareas de gestión de pertenencia a grupos:

- Proporcionar una interfaz para los cambios de pertenencia a grupos.
- Detección de fallas.
- Notificar a los miembros de los cambios de membresía del grupo.
- Realización de expansión de dirección de grupo.



Sistemas Publish-subscribe

3 Comunicación Indirecta

- Una familia de enfoques que comparten las características comunes de difundir eventos a múltiples destinatarios a través de un intermediario.
- Sistema donde los editores (publicadores) generan eventos estructurados en un servicio de eventos.
- Los suscriptores expresan interés en eventos particulares a través de suscripciones que pueden ser patrones arbitrarios sobre los eventos estructurados.
- El sistema compara las suscripciones con los eventos publicados y garantiza la entrega correcta de las notificaciones de eventos.

Aplicaciones

- Sistemas de información financiera.
- Feeds en vivo de datos en tiempo real.
- Trabajo cooperativo, donde algunos participantes necesitan ser informados de eventos de interés compartido.
- Computación ubicua.
- Aplicaciones de monitoreo.

Características de los sistemas publish-subscribe

3 Comunicación Indirecta

Heterogeneidad

- Conectar componentes heterogéneos.
- Los usuarios describen los eventos que ofrecen y los suscriptores seleccionan patrones de eventos.

Asincronía

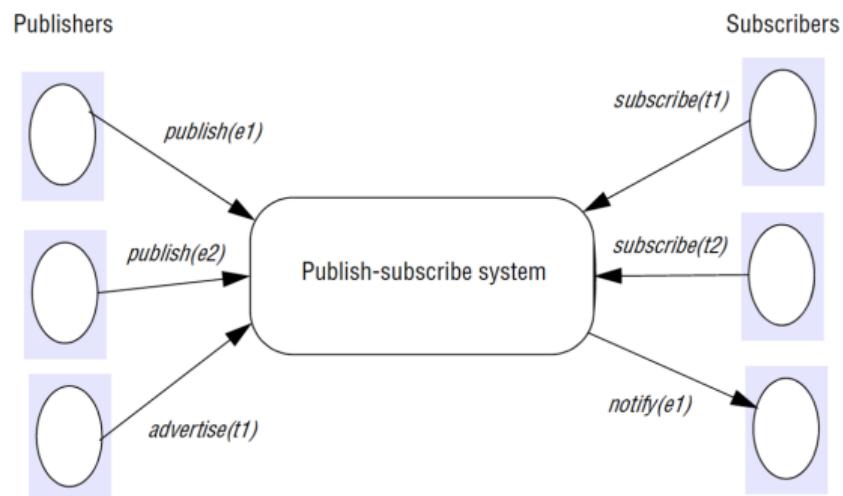
- Los editores generadores de eventos envían notificaciones de forma asíncrona a todos los suscriptores que expresaron interés.
- Los editores y los suscriptores están desacoplados.

- Las garantías de entrega dependen de la aplicación.
- Los requisitos en tiempo real dependen de las aplicaciones.

Modelo de programación

3 Comunicación Indirecta

- **Publicadores** diseminan un evento usando el operador $publish(e)$.
- **Suscriptores** expresan un interés en un conjunto de eventos a través de suscripciones -> $subscribe(f)$, donde f se refiere a un filtro (patrón).
- Puede revocar su interés con $unsubscribe(f)$.
- Los eventos llegan a los suscriptores usando el operador $notify(e)$.
- Complementar
 - $advertise(f)$ -> los publicadores declaran la naturaleza de los hechos. revocar con $unadvertise(f)$.



Modelo de programación

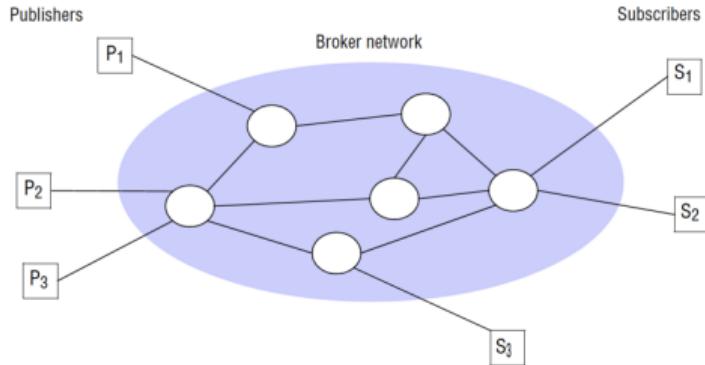
3 Comunicación Indirecta

Modelo de suscripción

- **Channel-based:** Los editores publican eventos en canales con nombre y se suscriben, luego se suscriben a uno de estos canales con nombre para recibir todos los eventos enviados a ese canal.
- **Topic-based:** Cada notificación etiquetada con un tema, las suscripciones definen los temas de interés. Puede utilizar la organización jerárquica de los temas.
- **Content-based:** expresión de suscripción sobre un rango de campos en una notificación de evento. La suscripción es una consulta sobre los valores de los atributos del evento.
- **Type-based:** en los enfoques basados en objetos, tienen un tipo específico. Los eventos tienen un tipo y una suscripción que coincide con los tipos o subtipos.

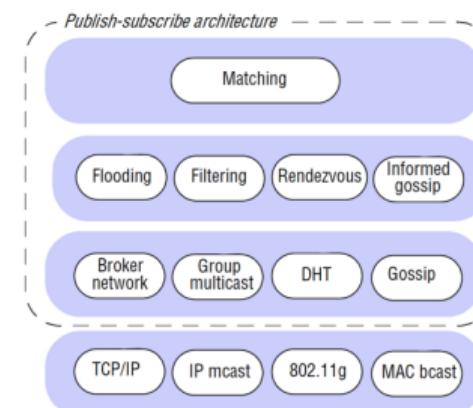
Problemas de Implementación

3 Comunicación Indirecta



- Arquitectura centralizada versus distribuida.

- Arquitectura de Sistema publish-subscribe



Sistemas de colas de mensajes

3 Comunicación Indirecta

Los mensajes se dirigen a la abstracción familiar de una cola con receptores que extraen mensajes para dichas colas.

- Los sistemas de colas de mensajes ofrecen capacidad de almacenamiento a medio plazo para los mensajes, sin necesidad de que el remitente o el receptor estén activos durante la transmisión.
- Garantías de inserción de mensaje en cola.
- No hay garantías sobre cuándo el destinatario leerá el mensaje.

Sistemas de colas de mensajes

3 Comunicación Indirecta

Modelo de programación

- **Producir** los procesos pueden enviar mensajes a una cola específica.
- **Consumir** los procesos pueden recibir mensajes de esta cola.

Estilos de recepción

- **Recepción Bloqueante**, que bloqueará hasta que el mensaje apropiado esté disponible.
- **Recepción no bloqueante** (polling), se verificará el estado de la cola y devolverá un mensaje si está disponible, o una indicación de no disponible en caso contrario.
- **Operación de notificación**, utilizará una notificación de evento cuando haya un mensaje disponible en la cola asociada.

Sistemas de colas de mensajes

3 Comunicación Indirecta

Interfaz básica de la cola

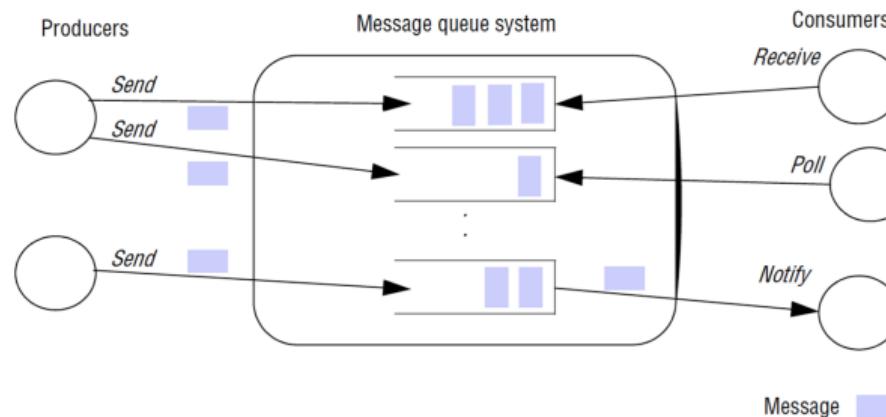
Operación	Descripción
put	Agregar un mensaje a una cola especificada
get	Bloquear hasta que la cola especificada no esté vacía y eliminar el primer mensaje
poll	Compruebe una cola específica de mensajes y elimine el primero. Nunca bloquee.
Notify	Instale un controlador para que se llame cuando se coloque un mensaje en la cola especificada.

Modelo de programación

3 Comunicación Indirecta

La política de colas es normalmente **first-in-first-out** (FIFO), pero la mayoría de las implementaciones de colas de mensajes también admiten "prioridad".

- **Prioridad:** la prioridad más alta se entrega primero.
- Los consumidores también pueden **seleccionar** mensajes de la cola según las propiedades de un mensaje.

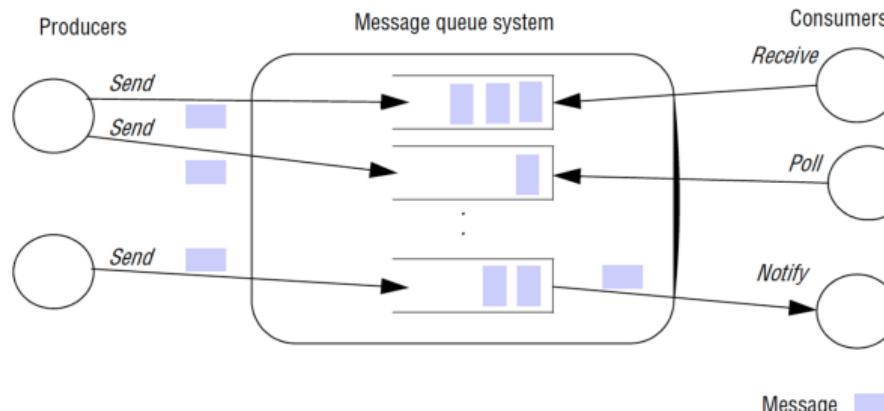


Modelo de programación

3 Comunicación Indirecta

Un mensaje consiste en:

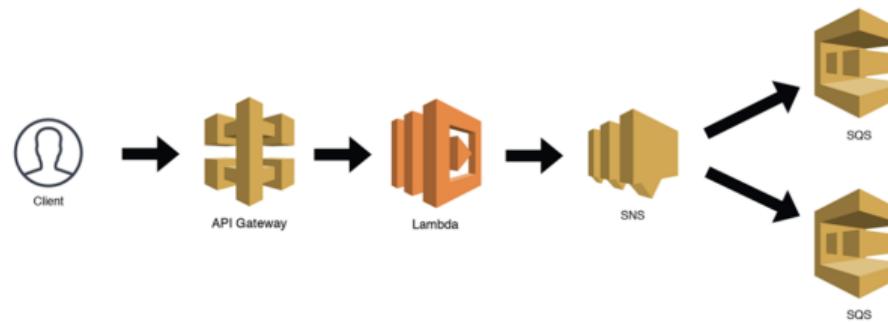
- Un **destino**: identificador que designa la cola de destino.
- **Metadata**: campos como prioridad, modo de entrega, cuerpo del mensaje.
- Los mensajes son persistentes: la cola almacenará el mensaje indefinidamente, hasta que se consuma.
- También confirme el mensaje en el disco para permitir una entrega confiable.



Ejemplos de comunicación indirecta

Patrón de cola distribuida

3 Comunicación Indirecta



Distributed Queue Pattern

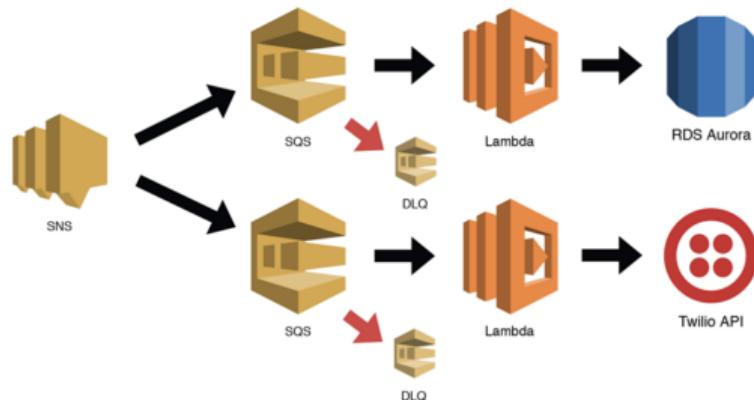
- Standard Queue: Unlimited throughput, at least once delivery, best effort ordering.
- FIFO Queue: 300 messages per second (batch may increase this two 3000), exactly once delivery, FIFO order



Ejemplos de comunicación indirecta

Cola de mensajes fallidos

3 Comunicación Indirecta



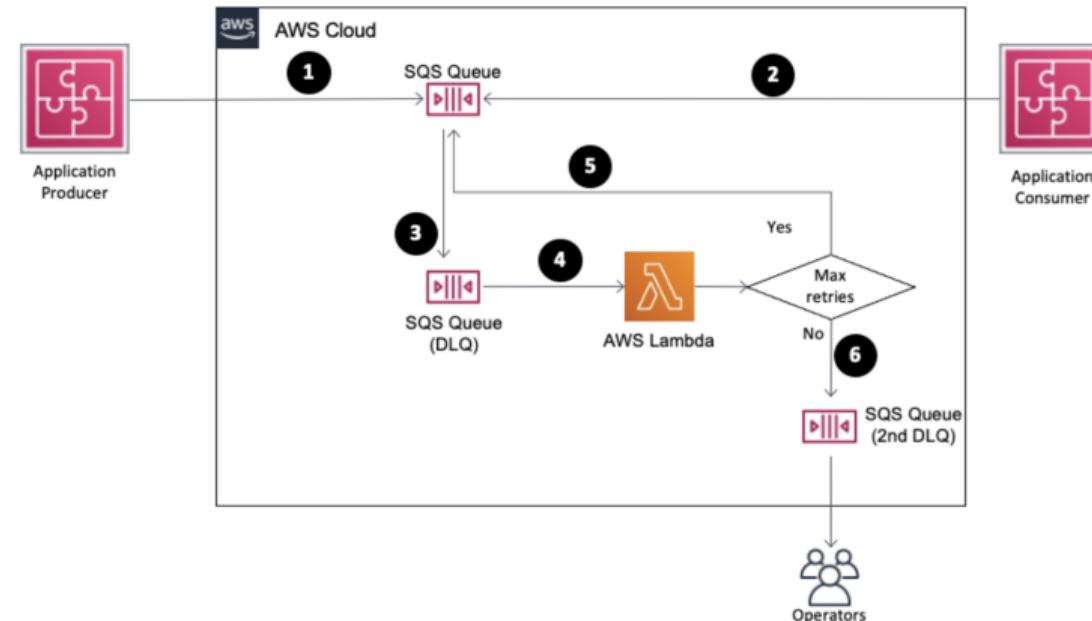
- Twilio API: API Rest to send SMS messages
- DLQ: Dead Letter Queue
- RDS Aurora: Rela



Ejemplos de comunicación indirecta

Segunda cola de mensajes fallidos

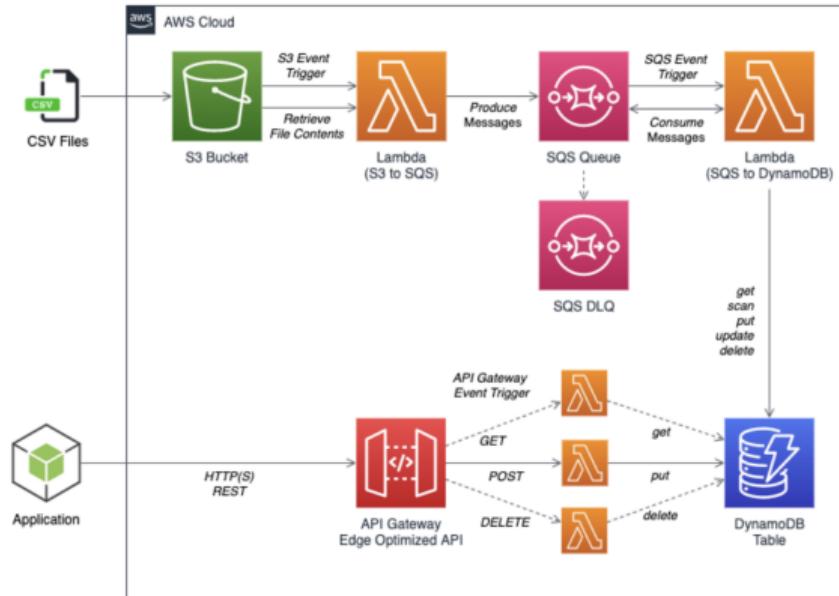
3 Comunicación Indirecta





Arquitectura sin servidor impulsada por eventos

3 Comunicación Indirecta

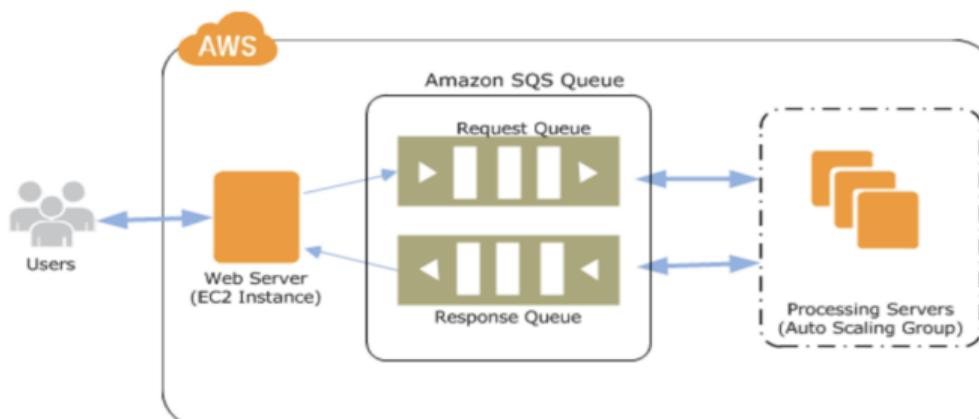




Ejemplos de comunicación indirecta

No solo sin servidor

3 Comunicación Indirecta



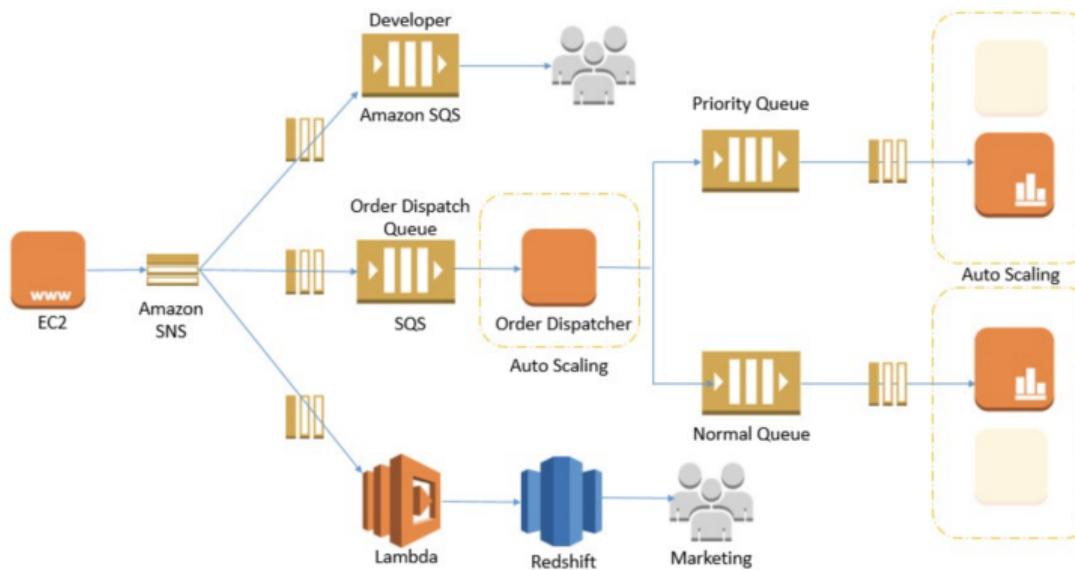
AWS SQS Architecture



Ejemplos de comunicación indirecta

No solo sin servidor

3 Comunicación Indirecta



Ejemplo de SNS: Reciba alertas de llamadas telefónicas para eventos de seguridad de la cuenta de AWS con Amazon Polly

3 Comunicación Indirecta





UNIVERSIDAD TÉCNICA
FEDERICO SANTA MARÍA

Comunicación Middleware

INF-343 Sistemas Distribuidos

Prof. Jorge Díaz M.

April 2, 2024

