# OBJETOS DISTRIBUIDOS EN JAVA O PYTHON

Carlos Maldonado

5IV7

## Introducción

Los objetos distribuidos representan una evolución fundamental en el desarrollo de software, permitiendo que componentes de aplicación ubicados en diferentes espacios de direcciones, potencialmente en distintas máquinas y plataformas, interactúen como si fueran objetos locales. Esta tecnología ha revolucionado la forma en que concebimos y construimos sistemas empresariales, facilitando la creación de arquitecturas escalables, tolerantes a fallos y geográficamente dispersas.

En el contexto actual, donde las aplicaciones empresariales requieren cada vez más integración entre sistemas heterogéneos y distribución geográfica, el estudio de los objetos distribuidos se convierte en fundamental para cualquier profesional de la ingeniería de software. Este trabajo explorará en profundidad las implementaciones de objetos distribuidos en Java y Python, dos de los lenguajes más influyentes en el desarrollo de software moderno.

## Fundamentos Teóricos de los Objetos Distribuidos

### Conceptos Básicos

Un objeto distribuido es una entidad de software que reside en un espacio de direcciones diferente al del cliente que lo invoca. Los principios fundamentales que rigen este paradigma incluyen:

- **Transparencia de ubicación**: El cliente no necesita conocer la ubicación física del objeto
- **Transparencia de acceso**: La invocación de métodos remotos se realiza con la misma sintaxis que las locales
- **Transparencia de fallos**: El sistema debe manejar adecuadamente las fallas de comunicación

**Arquitectura General**

La arquitectura de objetos distribuidos típicamente involucra los siguientes componentes:

1. **Cliente**: Solicita servicios a objetos remotos
2. **Esqueleto (Stub)**: Representante local del objeto remoto
3. **Canal de comunicación**: Medio para transmitir invocaciones y resultados
4. **Esqueleto remoto (Skeleton)**: Recibe solicitudes y las delega al objeto real
5. **Objeto servidor**: Implementa la lógica de negocio

**Protocolos y Estándares**

Los sistemas de objetos distribuidos se basan en protocolos estandarizados como:

- **CORBA** (Common Object Request Broker Architecture)
- **RMI** (Remote Method Invocation) en Java
- **DCOM** (Distributed Component Object Model) en Windows
- **Protocolos basados en XML/SOAP**

**Objetos Distribuidos en Java**

**Java RMI (Remote Method Invocation)**

Java RMI es el mecanismo nativo de Java para la creación y uso de objetos distribuidos. Proporciona una infraestructura completa que permite a los objetos Java ejecutándose en una máquina virtual Java invocar métodos de objetos Java que se ejecutan en otra máquina virtual Java.

**Arquitectura de Java RMI**

La arquitectura de RMI se compone de varias capas:

1. **Capa de Stub/Esqueleto**: Genera proxies locales y manejadores remotos
2. **Capa de Referencia Remota**: Maneja la semántica de invocación
3. **Capa de Transporte**: Establece conexiones y maneja la comunicación

**Características Principales**

- **Integración con el lenguaje**: Sintaxis natural para invocaciones remotas
- **Serialización de objetos**: Transferencia transparente de objetos complejos
- **Recolección de basura distribuida**: Liberación automática de recursos remotos
- **Seguridad**: Basada en el security manager de Java
- **Descarga dinámica de clases**: Capacidad de transferir código entre máquinas

**Ejemplo 1: Sistema de Gestión de Biblioteca Distribuida**

Implementaremos un sistema de biblioteca distribuido donde múltiples clientes pueden consultar y reservar libros desde diferentes ubicaciones.

**Interfaz Remota**

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;

public interface Biblioteca extends Remote {
   Libro buscarLibro(String titulo) throws RemoteException;
   List<Libro> buscarPorAutor(String autor) throws RemoteException;
   boolean reservarLibro(String isbn, String usuario) throws RemoteException;
   boolean devolverLibro(String isbn) throws RemoteException;
   List<Libro> obtenerCatalogo() throws RemoteException;
   String obtenerInformacionServidor() throws RemoteException;
}
```

**Clase de Datos del Libro**

```
import java.io.Serializable;

public class Libro implements Serializable {
  private String isbn;
  private String titulo;
  private String autor;
  private int añoPublicacion;
  private boolean disponible;

  public Libro(String isbn, String titulo, String autor, int añoPublicacion) {
    this.isbn = isbn;
    this.titulo = titulo;
    this.autor = autor;
    this.añoPublicacion = añoPublicacion;
    this.disponible = true;
  }

  // Getters y setters
  public String getIsbn() { return isbn; }
  public String getTitulo() { return titulo; }
  public String getAutor() { return autor; }
  public int getAñoPublicacion() { return añoPublicacion; }
  public boolean isDisponible() { return disponible; }
  public void setDisponible(boolean disponible) { this.disponible = disponible; }

  @Override
  public String toString() {
    return String.format("'%s' por %s (%s) - %s",
      titulo, autor, isbn, disponible ? "Disponible" : "Prestado");
  }
}
```

**Implementación del Servidor**

```
// BibliotecaImpl.java
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class BibliotecaImpl extends UnicastRemoteObject implements Biblioteca {
  private Map<String, Libro> catalogo;
  private String nombreServidor;

  public BibliotecaImpl(String nombreServidor) throws RemoteException {
    super();
    this.nombreServidor = nombreServidor;
    this.catalogo = new HashMap<>();
    inicializarCatalogo();
  }

  private void inicializarCatalogo() {
    // Libros de ejemplo
    agregarLibro(new Libro("978-8441532106", "Clean Code", "Robert C. Martin",
2008));
    agregarLibro(new Libro("978-0132350884", "Effective Java", "Joshua Bloch", 2008));
    agregarLibro(new Libro("978-0201633610", "Design Patterns", "Erich Gamma",
1994));
    agregarLibro(new Libro("978-0321356680", "Java Concurrency in Practice", "Brian
Goetz", 2006));
  }

  private void agregarLibro(Libro libro) {
    catalogo.put(libro.getIsbn(), libro);
  }

  @Override
  public Libro buscarLibro(String titulo) throws RemoteException {
```

```
    System.out.println("Búsqueda solicitada: " + titulo);
    return catalogo.values().stream()
            .filter(libro -> libro.getTitulo().toLowerCase().contains(titulo.toLowerCase()))
            .findFirst()
            .orElse(null);
}

@Override
public List<Libro> buscarPorAutor(String autor) throws RemoteException {
    System.out.println("Búsqueda por autor: " + autor);
    List<Libro> resultados = new ArrayList<>();
    for (Libro libro : catalogo.values()) {
        if (libro.getAutor().toLowerCase().contains(autor.toLowerCase())) {
            resultados.add(libro);
        }
    }
    return resultados;
}

@Override
public boolean reservarLibro(String isbn, String usuario) throws RemoteException {
    Libro libro = catalogo.get(isbn);
    if (libro != null && libro.isDisponible()) {
        libro.setDisponible(false);
        System.out.println("Libro " + isbn + " reservado por " + usuario);
        return true;
    }
    return false;
}

@Override
public boolean devolverLibro(String isbn) throws RemoteException {
    Libro libro = catalogo.get(isbn);
    if (libro != null && !libro.isDisponible()) {
        libro.setDisponible(true);
        System.out.println("Libro " + isbn + " devuelto");
        return true;
    }
```

```
        return false;
    }

    @Override
    public List<Libro> obtenerCatalogo() throws RemoteException {
        return new ArrayList<>(catalogo.values());
    }

    @Override
    public String obtenerInformacionServidor() throws RemoteException {
        return "Servidor: " + nombreServidor + " - Libros en catálogo: " + catalogo.size();
    }
}
```

**Servidor Principal**

```
// ServidorBiblioteca.java
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

public class ServidorBiblioteca {
  public static void main(String[] args) {
    try {
      // Crear instancia del servicio
      BibliotecaImpl biblioteca = new BibliotecaImpl("Biblioteca Central");

      // Exportar el objeto
      Biblioteca stub = (Biblioteca) UnicastRemoteObject.exportObject(biblioteca, 0);

      // Crear registro RMI
      Registry registry = LocateRegistry.createRegistry(1099);

      // Registrar el servicio
      registry.rebind("BibliotecaService", stub);

      System.out.println("Servidor de Biblioteca iniciado...");
      System.out.println("Esperando conexiones de clientes...");

    } catch (Exception e) {
      System.err.println("Error en el servidor: " + e.toString());
      e.printStackTrace();
    }
  }
}
```

**Cliente Java**

```
// ClienteBiblioteca.java
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.List;
import java.util.Scanner;

public class ClienteBiblioteca {
  public static void main(String[] args) {
    try {
      // Obtener registro RMI
      Registry registry = LocateRegistry.getRegistry("localhost", 1099);

      // Buscar servicio
      Biblioteca biblioteca = (Biblioteca) registry.lookup("BibliotecaService");

      Scanner scanner = new Scanner(System.in);
      boolean continuar = true;

      System.out.println("=== SISTEMA DE BIBLIOTECA DISTRIBUIDA ===");
      System.out.println(biblioteca.obtenerInformacionServidor());

      while (continuar) {
        System.out.println("\nOpciones:");
        System.out.println("1. Buscar libro por título");
        System.out.println("2. Buscar libros por autor");
        System.out.println("3. Ver catálogo completo");
        System.out.println("4. Reservar libro");
        System.out.println("5. Devolver libro");
        System.out.println("6. Salir");
        System.out.print("Seleccione opción: ");

        int opcion = scanner.nextInt();
        scanner.nextLine(); // Consumir newline

        switch (opcion) {
          case 1:
```

```
            System.out.print("Ingrese título: ");
            String titulo = scanner.nextLine();
            Libro libro = biblioteca.buscarLibro(titulo);
            if (libro != null) {
                System.out.println("Resultado: " + libro);
            } else {
                System.out.println("Libro no encontrado");
            }
            break;

        case 2:
            System.out.print("Ingrese autor: ");
            String autor = scanner.nextLine();
            List<Libro> librosAutor = biblioteca.buscarPorAutor(autor);
            if (!librosAutor.isEmpty()) {
                System.out.println("Libros encontrados:");
                librosAutor.forEach(System.out::println);
            } else {
                System.out.println("No se encontraron libros del autor");
            }
            break;

        case 3:
            List<Libro> catalogo = biblioteca.obtenerCatalogo();
            System.out.println("Catálogo completo:");
            catalogo.forEach(System.out::println);
            break;

        case 4:
            System.out.print("Ingrese ISBN del libro a reservar: ");
            String isbnReserva = scanner.nextLine();
            System.out.print("Ingrese su nombre: ");
            String usuario = scanner.nextLine();
            if (biblioteca.reservarLibro(isbnReserva, usuario)) {
                System.out.println("Libro reservado exitosamente");
            } else {
                System.out.println("No se pudo reservar el libro");
            }
```

```
                    break;

                case 5:
                    System.out.print("Ingrese ISBN del libro a devolver: ");
                    String isbnDevolucion = scanner.nextLine();
                    if (biblioteca.devolverLibro(isbnDevolucion)) {
                        System.out.println("Libro devuelto exitosamente");
                    } else {
                        System.out.println("No se pudo devolver el libro");
                    }
                    break;

                case 6:
                    continuar = false;
                    break;

                default:
                    System.out.println("Opción no válida");
            }
        }

        scanner.close();
        System.out.println("Sesión terminada");

    } catch (Exception e) {
        System.err.println("Error en el cliente: " + e.toString());
        e.printStackTrace();
    }
  }
}
```

## Objetos Distribuidos en Python

### Enfoques en Python

Python ofrece múltiples enfoques para implementar objetos distribuidos, siendo los más relevantes:

- **XML-RPC**: Protocolo simple basado en XML
- **Pyro4** (Python Remote Objects): Framework nativo de Python para objetos distribuidos
- **gRPC**: Framework moderno de Google usando HTTP/2 y Protocol Buffers
- **CORBA mediante omniORB**: Implementación de CORBA para Python

### Pyro4: Python Remote Objects

Pyro4 es la solución más pythonica para objetos distribuidos, proporcionando una forma elegante y poderosa de crear sistemas distribuidos.

### Características de Pyro4

- **Serialización automática**: Usa pickle para serialización de objetos
- **Nombrado de objetos**: Sistema flexible de nombres para objetos remotos
- **Callbacks**: Soporte para callbacks asíncronos
- **Seguridad**: Autenticación y autorización configurables
- **Interoperabilidad**: Compatible con diferentes versiones de Python

**Ejemplo 2: Sistema de Procesamiento de Pedidos Distribuido**

Implementaremos un sistema de procesamiento de pedidos para una tienda en línea, donde múltiples servicios cooperan para completar órdenes.

**Servicio de Inventario**

```
# inventario_service.py
import Pyro4
import threading
import time
from datetime import datetime

@Pyro4.expose
class ServicioInventario:
  def __init__(self):
    self.productos = {
      "001": {"nombre": "Laptop Gamer", "precio": 1200.0, "stock": 10},
      "002": {"nombre": "Mouse Inalámbrico", "precio": 25.0, "stock": 50},
      "003": {"nombre": "Teclado Mecánico", "precio": 80.0, "stock": 30},
      "004": {"nombre": "Monitor 24\"", "precio": 200.0, "stock": 15},
      "005": {"nombre": "Auriculares Bluetooth", "precio": 60.0, "stock": 25}
    }
    self.lock = threading.Lock()
    print("Servicio de Inventario inicializado")

  def consultar_producto(self, codigo_producto):
    with self.lock:
      producto = self.productos.get(codigo_producto)
      if producto:
        return {
          'existe': True,
          'nombre': producto['nombre'],
          'precio': producto['precio'],
          'stock': producto['stock']
        }
      return {'existe': False}

  def reservar_stock(self, codigo_producto, cantidad):
    with self.lock:
```

```
        if codigo_producto in self.productos:
            producto = self.productos[codigo_producto]
            if producto['stock'] >= cantidad:
                producto['stock'] -= cantidad
                print(f"Stock reservado: {cantidad} unidades de {producto['nombre']}")
                return True
        return False

    def liberar_stock(self, codigo_producto, cantidad):
        with self.lock:
            if codigo_producto in self.productos:
                self.productos[codigo_producto]['stock'] += cantidad
                print(f"Stock liberado: {cantidad} unidades de
{self.productos[codigo_producto]['nombre']}")
                return True
        return False

    def obtener_inventario_completo(self):
        with self.lock:
            return self.productos.copy()
```

**Servicio de Procesamiento de Pagos**

```
# pagos_service.py
import Pyro4
import random
import time
from datetime import datetime

@Pyro4.expose
class ServicioPagos:
  def __init__(self):
    self.transacciones = {}
    self.transaction_counter = 1
    print("Servicio de Pagos inicializado")

  def procesar_pago(self, numero_tarjeta, monto, cvv, fecha_vencimiento):
    # Simular procesamiento de pago
    print(f"Procesando pago de ${monto:.2f}...")
    time.sleep(1)  # Simular latencia

    # Simular validaciones
    if len(numero_tarjeta) != 16 or not numero_tarjeta.isdigit():
      return {'exitoso': False, 'mensaje': 'Número de tarjeta inválido'}

    if len(cvv) != 3 or not cvv.isdigit():
      return {'exitoso': False, 'mensaje': 'CVV inválido'}

    # Simular fallo aleatorio (10% de probabilidad)
    if random.random() < 0.1:
      return {'exitoso': False, 'mensaje': 'Fallo en el procesamiento del pago'}

    # Generar ID de transacción
    transaccion_id = f"TXN{self.transaction_counter:06d}"
    self.transaction_counter += 1

    self.transacciones[transaccion_id] = {
      'fecha': datetime.now().isoformat(),
      'monto': monto,
```

```
            'estado': 'COMPLETADO'
        }

        print(f"Pago procesado exitosamente: {transaccion_id}")
        return {
            'exitoso': True,
            'transaccion_id': transaccion_id,
            'mensaje': 'Pago procesado exitosamente'
        }

    def verificar_transaccion(self, transaccion_id):
        return self.transacciones.get(transaccion_id, {'estado': 'NO_ENCONTRADA'})
```

**Servicio de Gestión de Pedidos**

```
# pedidos_service.py
import Pyro4
import threading
import uuid
from datetime import datetime

@Pyro4.expose
class ServicioPedidos:
  def __init__(self):
    self.pedidos = {}
    self.lock = threading.Lock()
    self.inventario_service = None
    self.pagos_service = None
    print("Servicio de Pedidos inicializado")

  def configurar_servicios(self, inventario_uri, pagos_uri):
    """Configurar dependencias de otros servicios"""
    self.inventario_service = Pyro4.Proxy(inventario_uri)
    self.pagos_service = Pyro4.Proxy(pagos_uri)
    print("Servicios configurados correctamente")

  def crear_pedido(self, items, info_pago, info_cliente):
    with self.lock:
      # Validar disponibilidad de stock
      for item in items:
        producto = self.inventario_service.consultar_producto(item['codigo'])
        if not producto['existe']:
          return {'exitoso': False, 'mensaje': f"Producto {item['codigo']} no existe"}
        if producto['stock'] < item['cantidad']:
          return {'exitoso': False, 'mensaje': f"Stock insuficiente para
{producto['nombre']}"}

      # Reservar stock temporalmente
      for item in items:
        if not self.inventario_service.reservar_stock(item['codigo'], item['cantidad']):
          # Revertir reservas anteriores
```

```
            for item_previo in items[:items.index(item)]:
                self.inventario_service.liberar_stock(item_previo['codigo'],
item_previo['cantidad'])
            return {'exitoso': False, 'mensaje': 'Error al reservar stock'}

        # Calcular total
        total = 0
        for item in items:
            producto = self.inventario_service.consultar_producto(item['codigo'])
            total += producto['precio'] * item['cantidad']

        # Procesar pago
        resultado_pago = self.pagos_service.procesar_pago(
            info_pago['numero_tarjeta'],
            total,
            info_pago['cvv'],
            info_pago['fecha_vencimiento']
        )

        if resultado_pago['exitoso']:
            # Crear pedido
            pedido_id = str(uuid.uuid4())[:8]
            self.pedidos[pedido_id] = {
                'id': pedido_id,
                'fecha': datetime.now().isoformat(),
                'cliente': info_cliente,
                'items': items,
                'total': total,
                'transaccion_id': resultado_pago['transaccion_id'],
                'estado': 'CONFIRMADO'
            }

            print(f"Pedido {pedido_id} creado exitosamente")
            return {
                'exitoso': True,
                'pedido_id': pedido_id,
                'transaccion_id': resultado_pago['transaccion_id'],
                'total': total,
```

```
                'mensaje': 'Pedido procesado exitosamente'
            }
        else:
            # Liberar stock reservado
            for item in items:
                self.inventario_service.liberar_stock(item['codigo'], item['cantidad'])

            return {'exitoso': False, 'mensaje': f"Error en pago:
{resultado_pago['mensaje']}"}

    def obtener_estado_pedido(self, pedido_id):
        pedido = self.pedidos.get(pedido_id)
        if pedido:
            return {
                'encontrado': True,
                'pedido': pedido
            }
        return {'encontrado': False}

    def listar_pedidos(self):
        return list(self.pedidos.values())
```

**Servidor Principal**

```
# servidor_distribuido.py
import Pyro4
import threading
import time

def iniciar_servicio_inventario():
    """Iniciar servicio de inventario"""
    inventario = ServicioInventario()
    daemon = Pyro4.Daemon(port=9090)
    uri = daemon.register(inventario, "servicio.inventario")
    print(f"Servicio de Inventario URI: {uri}")
    daemon.requestLoop()

def iniciar_servicio_pagos():
    """Iniciar servicio de pagos"""
    pagos = ServicioPagos()
    daemon = Pyro4.Daemon(port=9091)
    uri = daemon.register(pagos, "servicio.pagos")
    print(f"Servicio de Pagos URI: {uri}")
    daemon.requestLoop()

def iniciar_servicio_pedidos():
    """Iniciar servicio de pedidos"""
    pedidos = ServicioPedidos()
    daemon = Pyro4.Daemon(port=9092)

    # Esperar a que los otros servicios estén registrados
    time.sleep(2)

    # Configurar dependencias
    with Pyro4.locateNS() as ns:
        inventario_uri = ns.lookup("servicio.inventario")
        pagos_uri = ns.lookup("servicio.pagos")

    pedidos.configurar_servicios(inventario_uri, pagos_uri)
```

```
    uri = daemon.register(pedidos, "servicio.pedidos")
    print(f"Servicio de Pedidos URI: {uri}")
    daemon.requestLoop()

if __name__ == "__main__":
    # Iniciar Name Server de Pyro
    print("Iniciando Name Server de Pyro4...")
    Pyro4.naming.startNSloop(host="localhost", port=9090)
```

**Cliente Python**

```
# cliente_tienda.py
import Pyro4
import json

class ClienteTienda:
  def __init__(self):
    self.servicio_pedidos = None
    self.servicio_inventario = None
    self.conectar_servicios()

  def conectar_servicios(self):
    """Conectar con los servicios distribuidos"""
    try:
      with Pyro4.locateNS() as ns:
        pedidos_uri = ns.lookup("servicio.pedidos")
        inventario_uri = ns.lookup("servicio.inventario")

      self.servicio_pedidos = Pyro4.Proxy(pedidos_uri)
      self.servicio_inventario = Pyro4.Proxy(inventario_uri)
      print("Conexión establecida con los servicios")
    except Exception as e:
      print(f"Error conectando con servicios: {e}")

  def mostrar_inventario(self):
    """Mostrar inventario completo"""
    try:
      inventario = self.servicio_inventario.obtener_inventario_completo()
      print("\n=== INVENTARIO DISPONIBLE ===")
      for codigo, producto in inventario.items():
        print(f"{codigo}: {producto['nombre']} - ${producto['precio']:.2f} (Stock:
{producto['stock']})")
    except Exception as e:
      print(f"Error obteniendo inventario: {e}")

  def realizar_pedido(self):
    """Realizar un nuevo pedido"""
```

```
try:
    # Obtener información del carrito
    items = []
    while True:
        self.mostrar_inventario()
        codigo = input("\nIngrese código del producto (o 'fin' para terminar): ")
        if codigo.lower() == 'fin':
            break

        cantidad = int(input("Ingrese cantidad: "))

        # Verificar producto
        producto = self.servicio_inventario.consultar_producto(codigo)
        if not producto['existe']:
            print("Producto no encontrado")
            continue

        items.append({
            'codigo': codigo,
            'cantidad': cantidad
        })
        print(f"Agregado: {cantidad} x {producto['nombre']}")

    if not items:
        print("Carrito vacío")
        return

    # Información de pago
    print("\n--- INFORMACIÓN DE PAGO ---")
    numero_tarjeta = input("Número de tarjeta (16 dígitos): ")
    cvv = input("CVV: ")
    fecha_vencimiento = input("Fecha vencimiento (MM/YY): ")

    # Información del cliente
    print("\n--- INFORMACIÓN DEL CLIENTE ---")
    nombre = input("Nombre: ")
    email = input("Email: ")
    direccion = input("Dirección: ")
```

```
            info_pago = {
                'numero_tarjeta': numero_tarjeta,
                'cvv': cvv,
                'fecha_vencimiento': fecha_vencimiento
            }

            info_cliente = {
                'nombre': nombre,
                'email': email,
                'direccion': direccion
            }

            # Procesar pedido
            print("\nProcesando pedido...")
            resultado = self.servicio_pedidos.crear_pedido(items, info_pago, info_cliente)

            if resultado['exitoso']:
                print(f"\n ✅ PEDIDO EXITOSO")
                print(f"ID del Pedido: {resultado['pedido_id']}")
                print(f"ID de Transacción: {resultado['transaccion_id']}")
                print(f"Total: ${resultado['total']:.2f}")
            else:
                print(f"\n ❌ ERROR: {resultado['mensaje']}")

        except Exception as e:
            print(f"Error realizando pedido: {e}")

    def consultar_pedido(self):
        """Consultar estado de un pedido"""
        try:
            pedido_id = input("Ingrese ID del pedido: ")
            estado = self.servicio_pedidos.obtener_estado_pedido(pedido_id)

            if estado['encontrado']:
                pedido = estado['pedido']
                print(f"\n=== PEDIDO {pedido['id']} ===")
                print(f"Fecha: {pedido['fecha']}")
```

```
            print(f"Cliente: {pedido['cliente']['nombre']}")
            print(f"Email: {pedido['cliente']['email']}")
            print(f"Estado: {pedido['estado']}")
            print(f"Total: ${pedido['total']:.2f}")
            print("\nItems:")
            for item in pedido['items']:
                producto = self.servicio_inventario.consultar_producto(item['codigo'])
                print(f"  - {producto['nombre']} x {item['cantidad']}")
        else:
            print("Pedido no encontrado")

    except Exception as e:
        print(f"Error consultando pedido: {e}")

def ejecutar(self):
    """Bucle principal de la aplicación cliente"""
    while True:
        print("\n=== TIENDA EN LÍNEA DISTRIBUIDA ===")
        print("1. Ver inventario")
        print("2. Realizar pedido")
        print("3. Consultar pedido")
        print("4. Salir")

        opcion = input("Seleccione opción: ")

        if opcion == '1':
            self.mostrar_inventario()
        elif opcion == '2':
            self.realizar_pedido()
        elif opcion == '3':
            self.consultar_pedido()
        elif opcion == '4':
            print("¡Hasta luego!")
            break
        else:
            print("Opción no válida")

if __name__ == "__main__":
```

```
# Importar servicios
from inventario_service import ServicioInventario
from pagos_service import ServicioPagos
from pedidos_service import ServicioPedidos

cliente = ClienteTienda()
cliente.ejecutar()
```

**Comparativa Java vs Python en Objetos Distribuidos**

**Facilidad de Desarrollo**

**Java RMI:**

- Ventajas: Integración nativa con el lenguaje, fuerte tipado, herramientas maduras
- Desventajas: Configuración compleja, verbosidad del código

**Python Pyro4:**

- Ventajas: Sintaxis simple, desarrollo rápido, flexibilidad
- Desventajas: Tipado dinámico puede ocultar errores, menor rendimiento en algunos casos

**Rendimiento**

**Java RMI:**

- Optimizado para JVM
- Serialización eficiente
- Mejor para aplicaciones de alta concurrencia

**Python Pyro4:**

- Overhead de interpretación
- Serialización via pickle puede ser lenta para objetos grandes
- Adecuado para aplicaciones de baja a media escala

**Ecosistema y Herramientas**

**Java:**

- Múltiples frameworks empresariales (Spring Remoting, EJB)
- Monitoreo y gestión robustos
- Integración con sistemas legacy

**Python:**

- Amplia gama de bibliotecas para diferentes protocolos
- Ideal para prototipado rápido

- Buen soporte para integración con sistemas modernos

**Patrones y Mejores Prácticas**

**Patrones Comunes en Sistemas Distribuidos**

1. **Proxy**: Ocultar la complejidad de la comunicación remota
2. **Broker**: Intermediario para la comunicación entre componentes
3. **Factory**: Creación de objetos remotos
4. **Singleton distribuido**: Garantizar una única instancia en el sistema distribuido

**Consideraciones de Diseño**

- **Acoplamiento débil**: Minimizar dependencias entre servicios
- **Tolerancia a fallos**: Diseñar para la resiliencia
- **Escalabilidad**: Permitir crecimiento horizontal
- **Seguridad**: Autenticación y autorización en comunicaciones

**Tendencias Futuras y Evolución**

**Nuevos Paradigmas**

- **Microservicios**: Evolución natural de los objetos distribuidos
- **Serverless Computing**: Abstracción completa de la infraestructura
- **Service Mesh**: Gestión inteligente del tráfico entre servicios

**Tecnologías Emergentes**

- **gRPC**: Protocolo moderno de alto rendimiento
- **Apache Thrift**: Framework de serialización y RPC
- **WebAssembly**: Potencial para distribución universal de código

**Conclusión**

Los objetos distribuidos representan una tecnología fundamental en el desarrollo de software moderno, permitiendo la construcción de sistemas escalables, resilientes y geográficamente dispersos. Tanto Java con RMI como Python con Pyro4 ofrecen soluciones robustas, cada una con sus fortalezas particulares.

Java RMI proporciona una solución empresarial madura, con fuerte tipado y excelente integración con el ecosistema Java, ideal para aplicaciones de misión crítica. Por otro lado, Python Pyro4 ofrece una aproximación más flexible y pythonica, perfecta para prototipado rápido y aplicaciones donde la velocidad de desarrollo es crucial.

La elección entre estas tecnologías dependerá de factores específicos del proyecto como requisitos de rendimiento, tiempo de desarrollo, expertise del equipo y necesidades de integración con sistemas existentes. Lo fundamental es comprender los principios subyacentes de los sistemas distribuidos, que trascienden cualquier tecnología específica y forman la base para la construcción de aplicaciones modernas en la era de la nube y los microservicios.

**Bibliografía**


1. **Gonzalez, A. (2018).** *Sistemas Distribuidos: Conceptos y Diseño*. Madrid: Ediciones Paraninfo.
2. **López, M. & Rodríguez, P. (2020).** *Programación Avanzada en Java: De los Fundamentos a los Sistemas Distribuidos*. Barcelona: Marcombo.
3. **Fernández, J. (2019).** *Python para Sistemas Distribuidos y Paralelos*. Ciudad de México: Alfaomega.
4. **Martínez, R. (2017).** *Arquitecturas de Software Distribuido: Patrones y Mejores Prácticas*. Buenos Aires: Editorial Universidad.
5. **García, S. (2021).** *Desarrollo de Microservicios con Java y Python*. Santiago: Editorial Tecnos.
6. **Díaz, C. (2018).** *Comunicación entre Procesos y Sistemas Distribuidos*. Bogotá: McGraw-Hill Educación.
7. **Hernández, L. (2020).** *Patrones de Diseño para Sistemas Distribuidos*. Lima: Fondo Editorial.
8. **Ortega, F. (2019).** *Seguridad en Sistemas Distribuidos*. Madrid: RA-MA Editorial.
9. **Vargas, E. (2016).** *Java RMI: Guía Completa de Desarrollo*. Barcelona: Anaya Multimedia.
10. **Silva, M. (2021).** *Python en Entornos Empresariales: Pyro4 y Más Allá*. Montevideo: Editorial Triángulo.