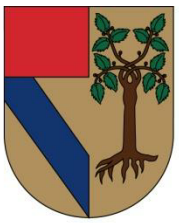

UNIVERSIDAD
PANAMERICANA

Cómputo Distribuido

“Proyecto Calculadora”

Nombres: Carlos Arias Cortina (TI) 0217787

Profesor: Carlos Pérez Leguízamo

Fecha: 02 de Diciembre de 2021

FACULTAD DE INGENIERÍA
UNIVERSIDAD PANAMERICANA

Proyecto Calculadora

***Resumen*— En el presente reporte se verán las mejoras que se le hicieron a la calculadora que habíamos desarrollado en el primer parcial, estas actualizaciones consisten en la implementación de la arquitectura SOA, con el fin de convertir nuestras operaciones en microservicios y de está forma mejorar la eficiencia de la misma.**

***Índice de Términos*— Microservicios, SOA, ADS y ADSOA**

I. Objetivo

Este proyecto está orientado a reforzar los temas vistos en clase a partir de crear un sistema distribuido de computo.

II. Desarrollo

La computación distribuida es una colección de computadoras que están separadas físicamente y conectadas entre sí usando WAN. Estas computadoras trabajan en una estrecha colaboración para la división del trabajo en pequeñas tareas individuales, por lo que cada ordenador recibe los datos necesarios para realizar la tarea asignada, una vez que la hacen, devuelven los datos obtenidos para unirlos en un resultado final.

Al estar físicamente separados los ordenadores, no se puede utilizar una memoria compartida por lo que se comparten mensajes y datos a través de una red. Esta comunicación entre ordenadores se puede realizar de forma local o a nivel nacional y mundial a través del internet. El transporte de los mensajes se lleva a cabo mediante los protocolos de internet como TCP/IP y UDP.

En la computación distribuida existe un principio de transparencia que es presentar al mundo exterior el sistema como una unidad funcional, en la cual se simplifica su funcionamiento a un nivel técnico.

Las aplicaciones distribuidas generalmente suelen utilizar una arquitectura relacional de cliente-servidor. En esta arquitectura el cliente actúa como la instancia de entrada e interfaz de usuario, por lo que recibe la solicitud del usuario y la prepara para que pueda pasar a un servidor. El servidor se hace cargo de la mayor parte de la funcionalidad de búsqueda y realiza estas búsquedas en la base de datos.También el servidor es el que le da formato al resultado de la búsqueda , y este se lo devuelve al cliente a través de la red. Por último, el resultado final se muestra en la pantalla del usuario.

Los servicios de middleware se utilizan a menudo en los procesos distribuidos.El middleware es software que se encuentra entre el sistema operativo y las aplicaciones que se ejecutan. Por lo general el middleware se encarga de las tareas de gestión de datos, servicios de aplicaciones, mensajería , autenticación y gestión de API. De alguna forma actúa como un hilo conductor entre las aplicaciones, los datos y los usuarios. También funciona como una capa de traducción oculta para permitir la comunicación y la administración de datos en aplicaciones distribuidas. El uso de esta herramienta le permite a los usuarios hacer solicitudes como el envío de formularios en un explorador web o que un servidor web pueda devolver páginas web dinámicas en función al usuario. Algunos de los ejemplos comunes de middleware serían los programas que suele proporcionar servicios de mensajería como el Protocolo simple de acceso a objetos “SOAP”, servicios web, transferencias de esto representacional “REST” y notación de objetos JavaScript “JSON”.

La arquitectura SOA (Arquitectura Orientada a Servicios) es un concepto de arquitectura de software que consiste en reutilizar sus elementos disponibles, por medio de una red que permite la comunicación entre los servicios. Los servicios se pueden clasificar como funciones autónomas en el software, diseñados para ejecutar tareas predeterminadas.

La arquitectura de SOA funciona a partir de la comunicación de los servicios por medio de un sistema sin una conexión directa. Por lo que se interconectan los servicios en una red, donde se pueda permitir el flujo de la información.

Unos de los principales principios de SOA son los siguientes:

- Los servicios deben de ocultar la lógica del negocio y exponer una interfaz en la que los consumidores puedan interactuar.
- Nivel bajo de acoplamiento y cohesión elevada entre los servicios para evitar una interdependencia de los mismos.
- Reutilización de los servicios con el fin de ensamblar unos con otros y de está forma componerse en niveles superiores para realizar funciones de menor granularidad.

Algunas de las ventajas de SOA son las siguientes:

- Las instalaciones de la SOA están disponibles para todos.

- SOA genera aplicaciones más confiables, ya que es más fácil depurar servicios pequeños que un código de gran volumen.
- Permite ejecutar los servicios en varios lenguajes de programación, servicios y plataformas, lo cual aumenta la escalabilidad de forma considerable. Además, utiliza un protocolo de comunicación estandarizado para qué las empresas puedan disminuir la interacción entre los clientes ´ y los servicios, lo cual permite ampliar las aplicaciones con menos presiones e inconvenientes
- Los servicios son autónomos e independientes, se puede modificar y actualizar cada uno cuando sea necesario, sin afectar al resto.
- Reducción de los costos gracias a una mayor agilidad y un desarrollo más eficiente.
- Uso de la infraestructura heredada en los mercados nuevos: la SOA permite que los desarrolladores tomen las funciones de una plataforma o un entorno y las ampl´ıen e implementen en otros
- Reutilizar los servicios agiliza y simplifica el proceso de creación de las aplicaciones. Los desarrolladores no tienen que empezar siempre desde cero, como en el caso de las aplicaciones monolíticas.

Las tres funciones principales de SOA:

- Ser un proveedor de servicios, ya sea siendo el encargado de crear servicios web, ofrecerlos a un registro de servicios disponibles y gestionar sus condiciones de uso.
- Ser un agente de servicios , el cual se encarga de brindar información acerca del servicio a quien lo solicite, y puede ser público o privado.
- Permitir que el usuario pueda buscar un servicio en el registro o el agente, y conectarlo con el proveedor para recibirlo.

Los microservicios son un estilo de arquitectura como un modo de programar software,donde las aplicaciones se dividen en elementos más pequeños e independientes entre sí. En comparación al enfoque tradicional, en el que todo se compila en una sola pieza, los microservicios son elementos independientes que funcionan en conjunto para llevar a cabo las mismas tareas. Cada uno de estos elementos o procesos son un microservicio. Este tipo de enfoque de desarrollo de software valora más el nivel de detalle, la sencillez y la capacidad para compartir un proceso similar en varias aplicaciones. Debido a esto es un elemento fundamental de la optimización del desarrollo de aplicaciones hacia un modelo nativo de la nube y sistemas distribuidos.

La arquitectura de los microservicios se basa en un sistema de persistencia que será el único para administrar y usar, en caso de que necesite hacer un estado persistente o almacenar datos externos. La arquitectura de microservicios elimina el acoplamiento entre servicios a través de esquemas de base de datos. Está partición del espacio de datos le permite a las empresas tener lo necesario para confrontar las actualizaciones de datos y evitar inconsistencias. Por lo que una solución es implementar un programa de gestión de datos maestros.

El sistema autónomo descentralizado (ADS) es un sistema módulos o componentes que funcionan de manera independiente y son capaces de interactuar entre sí para lograr el objetivo global del sistema. El sistema tiene la capacidad de seguir funcionando en el caso de que alguna falla se llegue a presentar en alguno de los componentes.

Las capacidades que cualquier sistema de ADS debe de tener son:

- Diagnosticar un fallo producido en el sistema, detectar al componente que lo está produciendo y realizar los cambios necesarios en el mismo. De esta manera se disminuyen de forma drástica el número de interrupciones que un proceso del sistema pueda tener.
- Adaptabilidad del sistema al momento de agregar nuevos requerimientos sin generar errores al realizar los cambios correspondientes.
- Dar continuidad en los servicios que se ofrecen, a pesar de que el sistema se encuentre atacado y una parte de este no se encuentre en funcionamiento.
- Repararse automáticamente al detectar un error en los datos introducidos por el usuario.

Los requerimientos necesarios para que un sistema de ADS funcione son:

- Tener un control sobre los subsistemas de modo que si uno llega a fallar,se encuentre en reparación o está siendo agregado, los demás subsistemas puedan seguir funcionando sin interrupciones.
- Llevar una coordinación de cada uno de los subsistemas para que puedan lograr los objetivos predeterminados.
- Cada subsistema debe de contar con una autonomía e inteligencia para administrarse por sí mismo, sin la necesidad de ser dirigido por otro subsistema.
- Cada subsistema debe ser capaz de administrarse sólo de forma que pueda coordinarse con los demás, con base en su información local.

La arquitectura ADSOA (Arquitectura Orientada a Servicios Autónoma y Descentralizada) surge de la combinación de las características de ADS y SOA. Cada Negocio con ADSOA está conformado por subsistemas, los cuales están construidos por un conjunto de entidades autónomas que tienen la característica de ser capaces de solicitar y ofrecer servicios mediante mensajes. También cada entidad puede

tener varias instancias independientes, con la característica de tener la misma funcionalidad que la entidad que representan, lo que permite tener una mayor tolerancia a fallos y una alta disponibilidad.

Al empezar a diseñar el sistema primero se determinó el número de puertos que serían asignados al cliente, servidor y middleware. Como se muestra en la Figura 1.

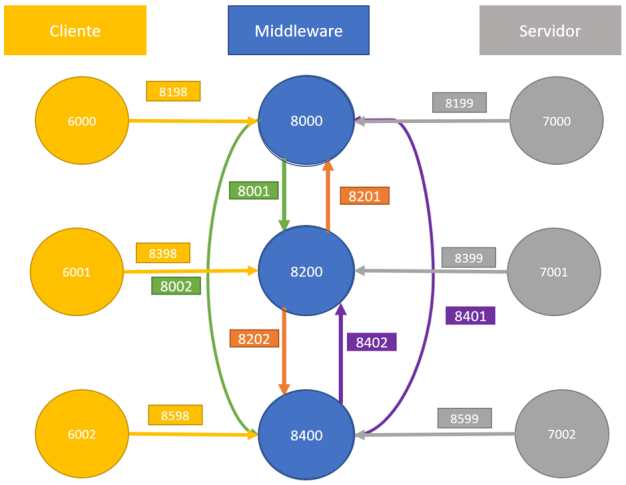

Fig. 1. Diseño del sistema distribuido

Para el cliente se le asignaron los puertos que van del 6000 a 6999 y para el servidor los puertos que van del 7000 a 7999. De está misma forma para el middleware los puertos que van del 8000 al 60,000.

En la Figura 2 se puede visualizar como se conecta el primer middleware en el puerto 8000.

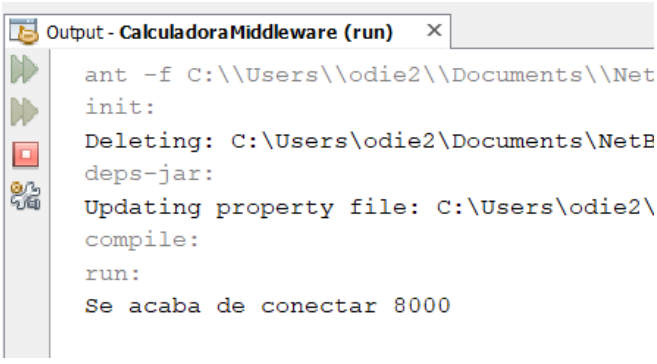

Fig. 2. Conexión del Middleware

Podemos visualizar en la Figura 3 la función encargada de crear middlewares hasta el el puerto 60,000 y son asignados a partir de sumarle 200 al puerto del último middleware.

```
static public ServerSocket create(int portInicial) throws IOException { //Se crean los Mi
    var flag = true;
    while(flag){
        if(portInicial < 60000){ //Su limite es de 60,000
            try {
                return new ServerSocket(portInicial);
            } catch (IOException ex) {
                Socket elsocket = new Socket("127.0.0.1",portInicial);
                DataInputStream in = new DataInputStream(elsocket.getInputStream());
                DataOutputStream out = new DataOutputStream(elsocket.getOutputStream());
                out.writeUTF("Contenido");
                String mensaje = in.readUTF();
                switch(mensaje)
                {
                    case "Nodo":
                        Nodos.add(portInicial);
                        break;
                    default:
                        System.out.println("Este puerto no sirve para nada");
                }
                elsocket.close();
                portInicial=portInicial+200; //Aumenta 200 para asignar un nuevo puerto
                continue;
            }
        }else
        {
            flag=false;
        }
    }
    return null;
}
```

Fig. 3. Código de creación de Middlewares

En la Figura 4 se puede visualizar como se conecta el servidor al puerto 7000 y se conecta al middleware por el puerto 8199.

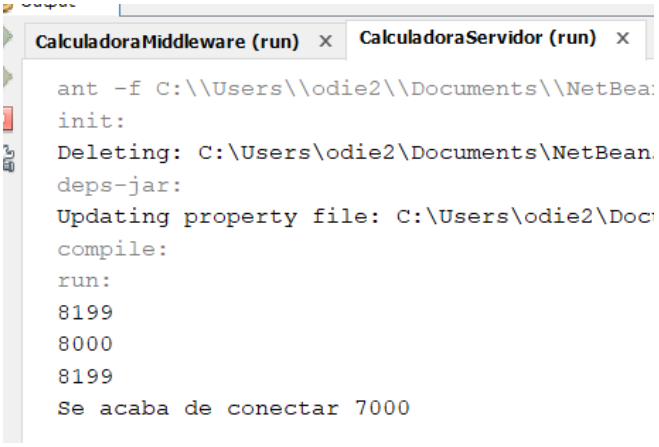

Fig. 4. Conexión del Servidor

En la Figura 5 se puede visualizar como son asignados los puertos de conexión del servidor con el middleware, a partir de sumarle 199 al puerto del middleware.

```
case "Servidor":
    if (servidor !=-5)
    {
       outn.writeUTF("0");//manda cero cuando no se pudo conectar
    }else
    {
        //Conexion con servidor
        servidor = Integer.parseInt(arrSplit[1]);
        System.out.println("Servidor "+ (mipuerto+199));
        outn.writeUTF(""+(mipuerto+199));// socket del servidor
        ControladorServidor Servidor = new ControladorServidor(mipuerto+199,mipuerto,Integer.parseInt(
        Thread tCl = new Thread(Servidor);
        servi.add(Servidor);
        tCl.start();
    }
    break;
```

Fig. 5. Código de creación de conexión de servidor con middleware

En la Figura 6 se puede visualizar como se conecta el cliente al puerto 6000 y se conecta al middleware por el puerto 8198.

I

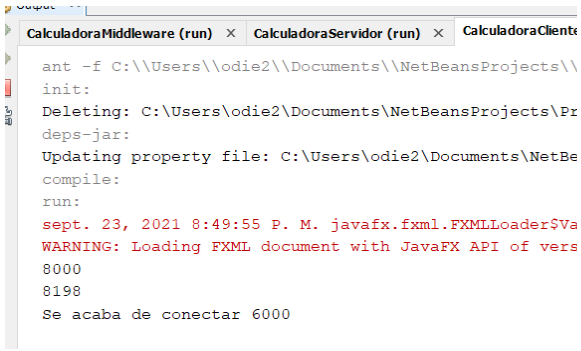

Fig. 6. Conexión del Cliente

En la Figura 7 se puede visualizar como son asignados los puertos de conexión del cliente con el middleware, a partir de sumarle 198 al puerto del middleware.

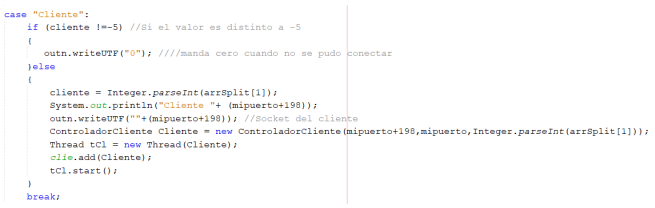

Fig. 7. Código de creación de conexión del cliente con middleware

Por último, en la Figura 8 se muestra la interfaz gráfica del cliente, donde va a poder interactuar con la calculadora y hacer las operaciones de suma, resta, multiplicación y división.

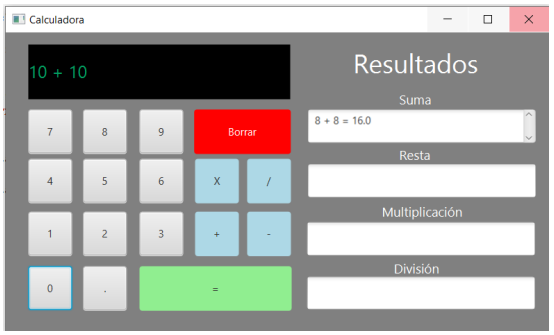

Fig. 8. Interfaz gráfica del Cliente

Lo primero que se realizó, fue identificar los puntos de comunicación con el servidor por lo que se descartaron las operaciones que se hacían al momento en el que el servidor recibe los datos del cliente. Por lo que esos datos se terminaron enviando al .jar respectivo a cada operación.Tal como se muestra en la Figura 9.

Fig.9. Llamado a ejecución de Microservicios

Para crear los archivos .jar, se escribió un programa en java para cada operación como el que se muestra en la Figura 10.

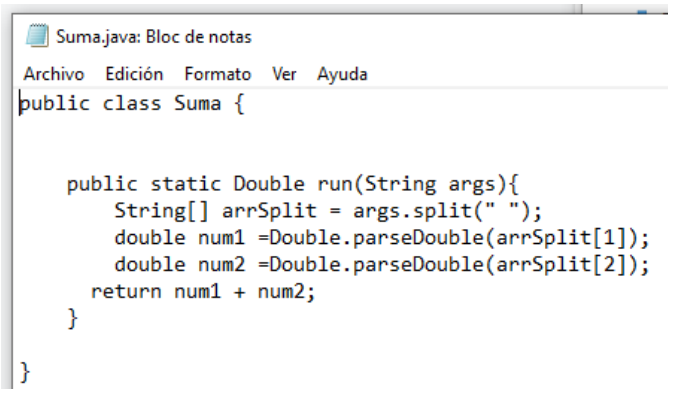

Fig.10. Codigo de Suma.java

Después desde Git Bash se ejecuta un comando para crear el archivo de Suma.class. Tal como se muestra en la Figura 11.

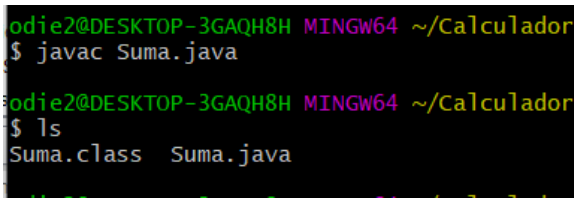

Fig.11. Creación de archivo Suma.class

De la misma forma se ejecuta otro comando desde Git Bash para generar el archivo de suma.jar. Tal como se muestra en la Figura 12.

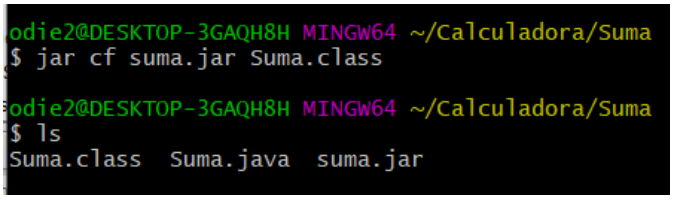

Fig.12. Creación de archivo suma.jar

Una vez generado el archivo .jar para cada microservicio se puede decir que el sistema ya está listo para realizar operaciones.

Algo que también se modificó fue el estado de los botones, por lo que se realizaron cambios en el archivo CalcuadoraController.java ubicado en la parte del cliente. Tal como se muestra en la Figura 13.

```
switch(operacion)
{
    case '1':
        suma.setDisable(true);
        Sumas.add(acum);
        sumat=sumat+"\n"+acum;
        Recibir.hilo.sumita = total;

        break;
    case '2':
        resta.setDisable(true);
        Restas.add(acum);
        restt=restt+"\n"+acum;
        Recibir.hilo.restita = total;

        break;
    case '3':
        multiplica.setDisable(true);
        Multiplicaciones.add(acum);
        multt=multt+"\n"+acum;
        Recibir.hilo.multita = total;

        break;
    case '4':
        divide.setDisable(true);
        Divisiones.add(acum);
        divit=divit+"\n"+acum;
        Recibir.hilo.divita = total;
```

Fig.13. Código para modificar el comportamiento de los botones

El código que se muestra en la Figura 13 es un ejemplo de los cambios que se hicieron para lograr la deshabilitación de los botones al momento en el que alguno de los servicios se caiga.

En la Figura 14 podemos ver como se ven los botones cuando el servicio no se encuentra disponible, en este caso el servicio de sumar se puede utilizar.

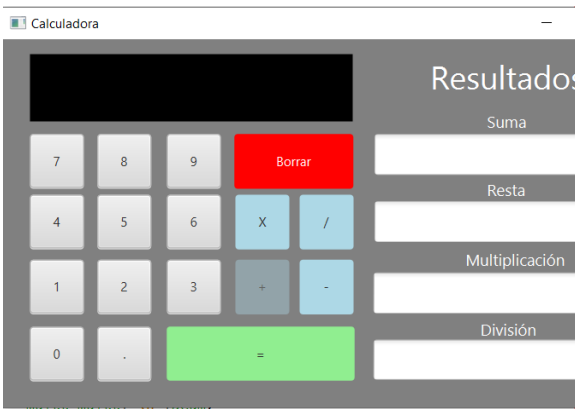

Fig.14. Deshabilitación del botón de sumar

También se le avisa al cliente de que el servicio está caído por medio de mensajes en la consola. Tal como se muestra en la Figura 15.

```
Servicio de Suma no disponible
5 5 6 0.0
Se acaba de conectar 6000
Servicio de Suma no disponible
5 5 6 0.0
Se acaba de conectar 6000
Servicio de Suma no disponible
5 5 6 0.0
```

Fig.15. Mensaje del estatus del servicio

Por último, en la Figura 16 se muestra como se restablece el sistema y se puede seguir haciendo operaciones.

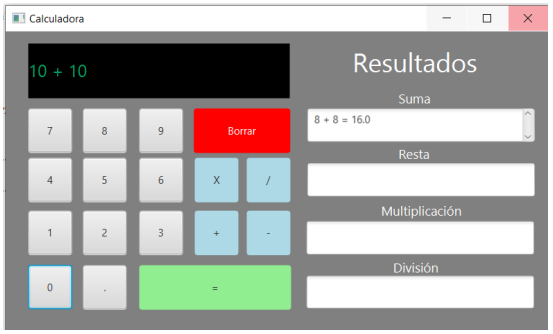

Fig.16. Restablecimiento del servicio de sumar

Se diseñó el sistema aplicando el protocolo de foliado, tal como se muestra en la Figura 17.
El límite mínimo de acuses en el servicio de suma será de dos y en los otros servicios de uno.

Los códigos de operación son los siguientes:

1) Para la operación de suma (+)
2) Para la operación de resta (-)
3) Para la operación de multiplicación (*)
4) Para la operación de división (/)
5) Para el resultado de suma (= +)
6) Para el resultado de resta (= -)
7) Para el resultado de multiplicación (= *)
8) Para el resultado de división (= /)
66) Para el ACK (acknowledge)

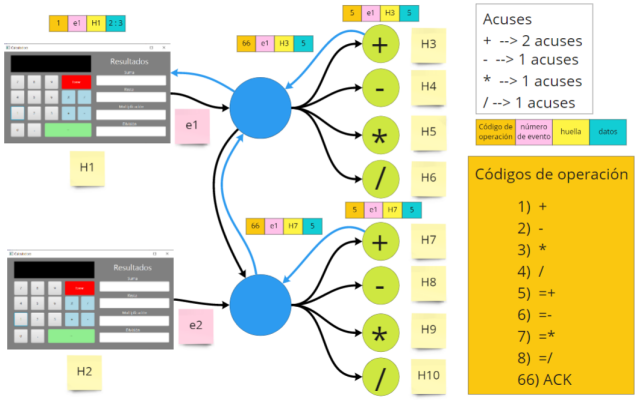

Fig.17. Diseño del sistema aplicando el protocolo de foliado

Para generar la huella se obtuvo el tiempo actual de la máquina por medio de la función de time stamp y se hasheo el mismo para producirla. Tal como se muestra en la Figura 18.

```
public static String GeneraHuella()
{
    String timeStamp = new SimpleDateFormat("yyyy.MM.dd.HH.mm.ss").format(new Date());
    String hashtext = "";
    try {
        MessageDigest m = MessageDigest.getInstance("MD5");
        m.reset();
        m.update(timeStamp.getBytes());
        byte[] digest = m.digest();
        BigInteger bigInt = new BigInteger(1,digest);
        hashtext = bigInt.toString(16);
    }catch(NoSuchAlgorithmException e){

    }

    return hashtext;
}
```

Fig.18. Código para generar la huella

El evento fue generado a partir de concatenar en un string los datos, más la huella y el código de operación. Se hasheo todo este mensaje para producir el evento. Tal como se muestra en la Figura 19.

```
public static String HashearMensaje(String total, String ope)
{
    String mensaje = total+huella +ope;
     String hashtext = "";
    try {
        MessageDigest m = MessageDigest.getInstance("MD5");
        m.reset();
        m.update(mensaje.getBytes());
        byte[] digest = m.digest();
        BigInteger bigInt = new BigInteger(1,digest);
        hashtext = bigInt.toString(16);
    }catch(NoSuchAlgorithmException e){

    }

    return hashtext;
}
```

Fig.19. Código para generar el evento

Después de haber generado la huella y el evento, se puede proceder a enviar el mensaje.

```
case '1':
    suma.setDisable(true);
    Sumas.add(acum);
    sumat=sumat+"\n"+acum;
    Recibir.hilo.e_sumita = HashearMensaje(total,"1");//"MD5(to
    eventos.put(Recibir.hilo.e_sumita, total);
    acuses.put(Recibir.hilo.e_sumita, new HashSet<String>());
    Recibir.hilo.sumita = total;

    break;
```

Fig.20. Código para generar el protocolo de foliado

El mensaje que se va enviar al servidor va a estar concatenado de la siguiente forma: código de operación + evento + huella +datos. Tal como se muestra en la Figura 21.

```
if(sumita != null)
{
    if(acuses.get(e_sumita).size() < 2)
    {
    conexion(("1 "+ e_sumita + " "+huella +" "+ sumita),'+');
    System.out.println("Servicio de Suma no disponible");
    }
    else
    {

        acuses.remove(e_sumita);
        suma.setDisable(false);
        this.sumita = null;
    }
}
```

Fig.21. Código para enviar el protocolo de foliado y determinar el límite mínimo de acuses

Realizaremos una prueba con la operación de multiplicación para comprobar el funcionamiento del protocolo. Se realizará la siguiente operación: 2 * 3. Tal como se muestra en la Figura 22.

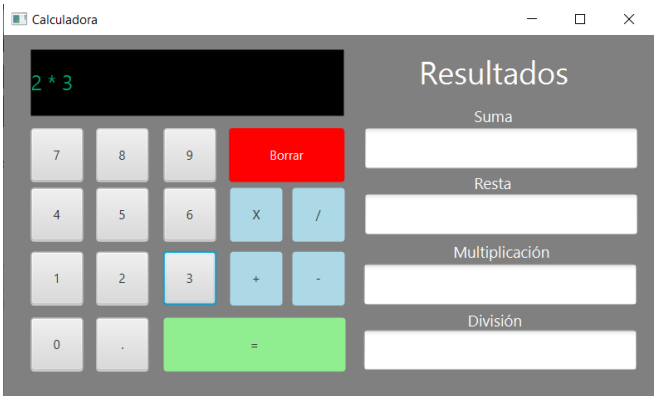

Fig.22. Solicitud del cliente para realizar la operación de 2*3

Como podemos ver en la Figura 23, el servidor recibe de forma correcta el protocolo y envió la respuesta de la operación.

```
Se acaba de conectar 7000
3 5a5bd00b551058c82002b85ce874b3cf 51fcb1b40a66be7d8a84a4be7354b339 2:3
6.0
66 5a5bd00b551058c82002b85ce874b3cf 926148d73450b76ea2274a16a865b85f 1
7 5a5bd00b551058c82002b85ce874b3cf 926148d73450b76ea2274a16a865b85f 6.0
```

Fig.23. Mensaje recibido por el servidor

Como podemos ver en la Figura 24, el cliente recibió la respuesta de servidor de forma correcta.

```
66 5a5bd00b551058c82002b85ce874b3cf 926148d73450b76ea2274a16a865b85f 1
Numero total de acuses :1
66 5a5bd00b551058c82002b85ce874b3cf 926148d73450b76ea2274a16a865b85f 1
7 5a5bd00b551058c82002b85ce874b3cf 926148d73450b76ea2274a16a865b85f 6.0
```

Fig.24. Mensaje recibido por el cliente con la respuesta de la multiplicación

Por último, podemos ver reflejado el resultado de la operación en la interfaz del cliente. Tal como se muestra en la Figura 25.

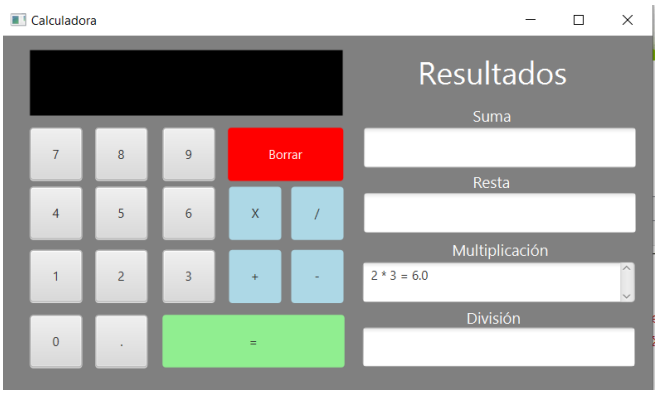

Fig.25. Impresión del resultado de la multiplicació

Realizaremos una prueba con la operación de suma para comprobar el funcionamiento del protocolo, en este caso vamos a ver lo que pasa cuando no se cumple con el mimo de acuses para operar Se realizará la siguiente operación: 4 + 8. Tal como se muestra en la Figura 26.

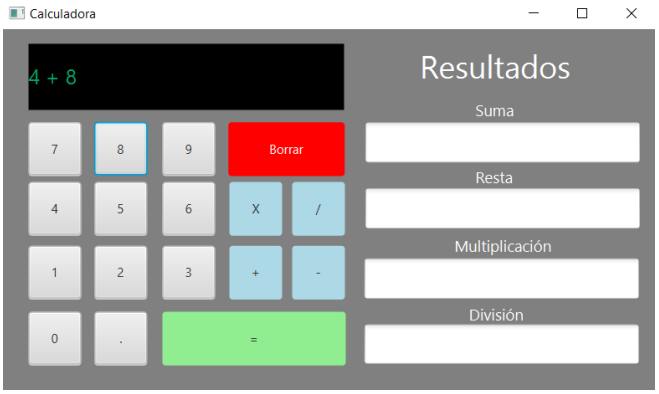

Fig.26. Solicitud del cliente para realizar la operación de 4 + 8

Como podemos ver en la Figura 27, el servidor recibe de forma correcta el protocolo y envió la respuespuesta de la operación.

```
Se acaba de conectar 7000
1 ccded9764c172bdac7a9acf276f6bf67 ddd5d5e640428bfa6d83b8e20d4f45a5 4:8
12.0
66 ccded9764c172bdac7a9acf276f6bf67 30c6649b715814f60697d64003bb1823 1
5 ccded9764c172bdac7a9acf276f6bf67 30c6649b715814f60697d64003bb1823 12.0
```

Fig.27. Mensaje recibido por el servidor

Como podemos ver en la Figura 28, el cliente recibió la respuesta del servidor y como no sé cumplio con el minimo de acuses por lo que el servicio deja de estar disponible.

```
66 ccded9764c172bdac7a9acf276f6bf67 30c6649b715814f60697d64003bb1823 1
Numero total de acuses :1
66 ccded9764c172bdac7a9acf276f6bf67 30c6649b715814f60697d64003bb1823 1
5 ccded9764c172bdac7a9acf276f6bf67 30c6649b715814f60697d64003bb1823 12.0
Se acaba de conectar 6000
Servicio de Suma no disponible
```

Fig.28. Mensaje recibido el cliente indicando que no se pudo conectar con el servicio, ya que no se cuenta con el número de acuses mínimo

Al no estar disponible el servicio de suma se deshabilita el botón de sumar. Tal como se muestra en la Figura 29.

Fig.29. Servicio de suma no se encuentra disponible

Sí corremos otro servidor se va a restablecer el servicio ya que se cumple con el mínimo de acuses que requiere la operación de sumar. Tal como se muestra en la Figura 30.

```
66 ccded9764c172bdac7a9acf276f6bf67 f98a564534e96666d96a954055e54fe1 1
Numero total de acuses :2
66 ccded9764c172bdac7a9acf276f6bf67 f98a564534e96666d96a954055e54fe1 1
5 ccded9764c172bdac7a9acf276f6bf67 f98a564534e96666d96a954055e54fe1 12.0
```

Fig.30. Restablecimiento del servicio de sumar debido a que se cuenta con el número de acuses mínimo

Por último, podemos ver reflejado el resultado de la operación en la interfaz del cliente y se va habilitar el botón de suma. Tal como se muestra en la Figura 31.

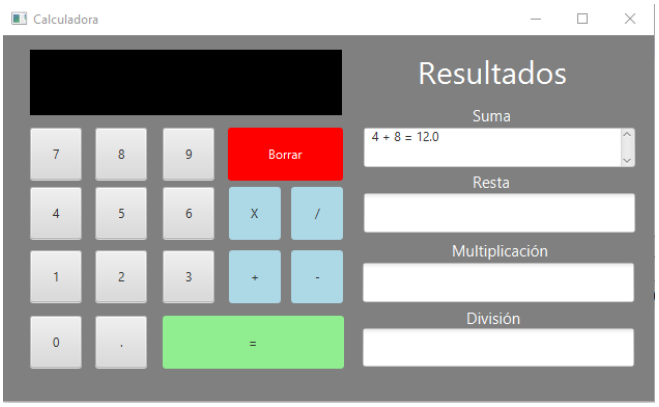

Fig.31. Servicio de suma restablecido y botón de sumar habilitado

Para empezar el protocolo de clonación se envía un mensaje concatenado de la siguiente forma: 99 + evento + huella +datos. Y se envía las veces necesarias para cumplir con el mínimo de acuses. Tal como se muestra en la Figura 32.

```
if(sumita != null)
{
    if(acuses.get(e_sumita).size() < 3)
    {
    conexion(("1 "+ e_sumita + " "+huella +" "+ sumita),'+');
    System.out.println("Servicio de Suma no disponible");
    try
    {
        Thread.sleep(2000);
    }
    catch(InterruptedException ex)
    {
        Thread.currentThread().interrupt();
    }
    conexion(("99 "+ e_sumita + " "+huella +" "+ 1),'+');
    }
    else
    {

        acuses.remove(e_sumita);
        suma.setDisable(false);
        this.sumita = null;
    }
}
```

Fig.32. Codigo para mandar un mensaje con la solicitud clonación

Al no estar disponible el servicio de suma se deshabilita el botón de sumar. Tal como se muestra en la Figura 34.

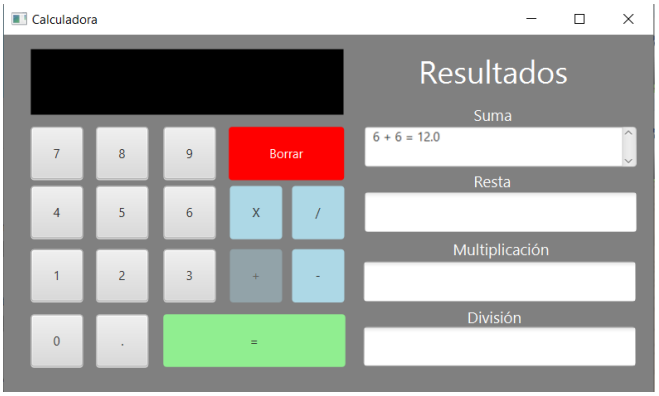

Fig.34. Servicio de suma no se encuentra disponible

Se manda a ejecutar el protocolo de clonación por medio del código que se muestra en la Figura 33.

```
case "99":
   if(acuses.containsKey(arrSplit[1]))
   {
       if(!acuses.get(arrSplit[1]).contains(arrSplit[2]))
       {
       acuses.get(arrSplit[1]).add(arrSplit[2]);
       Integer cuenta = acuses.get(arrSplit[1]).size();

        Runtime.getRuntime().exec("java -jar C:/SPB_Data/Calculadora/Servidor/CalculadoraServidor.jar");
       System.out.println("Clonando.. ");

       }
   }

   break;
```

Fig.33. Codigo para mandar a ejecutar el protocolo de clonación

Como podemos ver en la Figura 35, el cliente recibe a los nuevos servidores y lo podemos ver en la forma que cambia el número de acuses disponible.

```
Numero total de acuses :1
66 4c1b19d0f99fd3fa18b90093ade8ae18 380c7d3ad4396e838f6e7d5273745c49 1
5 4c1b19d0f99fd3fa18b90093ade8ae18 380c7d3ad4396e838f6e7d5273745c49 12.0
Se acaba de conectar 6000
99 4c1b19d0f99fd3fa18b90093ade8ae18 96ff3ef10160fa627db277b97176658 1
Clonando..
Se acaba de conectar 6000
Servicio de Suma no disponible
66 4c1b19d0f99fd3fa18b90093ade8ae18 380c7d3ad4396e838f6e7d5273745c49 1
5 4c1b19d0f99fd3fa18b90093ade8ae18 380c7d3ad4396e838f6e7d5273745c49 12.0
Se acaba de conectar 6000
66 4c1b19d0f99fd3fa18b90093ade8ae18 a158bd7ddc15e65e218f4244e98ea009 1
Numero total de acuses :3
```

Fig.35. Restablecimiento del servicio de sumar después de haberse ejecutado el protocolo de clonación

Por último, podemos ver como se restablece el servicio y se habilita el botón de suma. Tal como se muestra en la Figura 36.

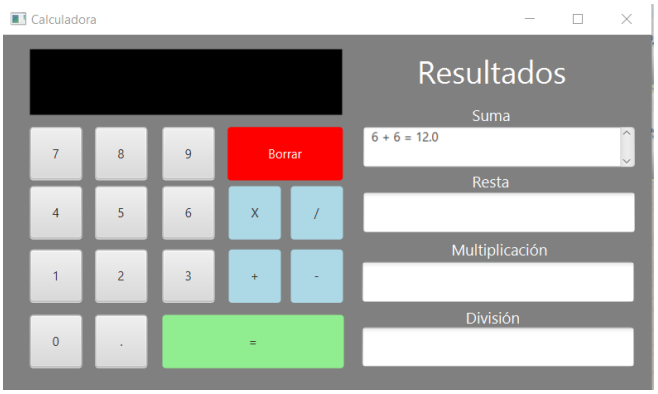

Fig.36. Servicio de suma restablecido y botón de sumar habilitado

III. Conclusiones

Se puede concluir que con este proyecto hemos logrado reforzar nuestros conocimientos previos sobre el cómputo distribuido y además pudimos crear un calculador funcional a partir de estos.

Se pudo utilizar los puertos de una forma eficiente para generar las conexiones requeridas por el sistema para generar una comunicación entre los clientes y los servidores en tiempo real.

También se aprendió a utilizar las aplicaciones de netbeans y SceneBuilder, que facilitaron la creación del proyecto, dado que cuentan con una interfaz amigable.
Algo que también pudimos lograr, fue el implementar la arquitectura SOA en el sistema actual y de está forma descentralizar aún más el cómputo. Por otro lado, se creó un sistema cada vez más inteligente y al dividir el servidor en microservicios, esto complicó un poco más el funcionamiento del sistema ya que lo hizo más complejo en la cuestión de conexiones.
Por último, no hay duda alguna sobre que este tipo de arquitecturas son y serán utilizadas en el futuro, porque garantizan una cierta estabilidad en los sistemas, a pesar de las fallas que se puedan ir presentando.

IV. Referencias bibliográficas

1. ¿Qué es la arquitectura orientada a los servicios? (redhat.com)
2. Cómo lograr una arquitectura de datos descentralizada con microservicios (chakray.com)
3. https://www.disrupciontecnologica.com/arquitectura-soa-y-composicion-de-servicios/
4. https://blog.infranetworking.com/modelo-cliente-servidor/
5. https://repositorio.tec.mx/bitstream/handle/11285/629082/33068001049731.pdf?sequence=1&isAllowed=y
6. https://www.disrupciontecnologica.com/arquitectura-soa-y-composicion-de-servicios/#:~:text=Un%20dise%C3%B1o%20arquitect%C3%B3nico%20SOA%20constituye,formada%20por%20proveedores%20y%20consumidores.
7. http://conogasi.org/articulos/computacion-en-la-nube-vs-computacion-distribuida/
8. https://www.ecured.cu/Computaci%C3%B3n_distribuida
9. https://www.ionos.mx/digitalguide/servidores/know-how/que-es-la-computacion-distribuida/
10. https://blog.infranetworking.com/modelo-cliente-servidor/
11. https://www.redhat.com/es/topics/middleware/what-is-middleware
12. https://azure.microsoft.com/es-mx/overview/what-is-middleware/