Calculadora construida con una Arquitectura Autónoma Descentralizada Orientada a los Servicios

Ramón Alejandro Ruiz Pérez
Facultad de Ingeniería
Universidad Panamericana
0223276@up.edu.mx

***Resumen*—El presente documento tiene la finalidad de presentar el desarrollo de una calculadora con la implementación de una *Arquitectura Autónoma Descentralizada Orientada a los Servicios* (ADSOA por sus siglas en inglés), donde se toman consideraciones de arquitecturas anteriores como lo es *Servicio autónomo descentralizado* (ADS por sus siglas en inglés) y *Arquitectura Orientada a los Servicios* (SOA por sus siglas en inglés) y de su aplicación en el lenguaje de programación JAVA. Así mismo se describe la importancia del cómputo distribuido, el cambio necesario a este modelo de cómputo y las propuestas que se han dada en el área.**

1. Introducción

El constante avance de la tecnología y los pasos agigantados a los que lo realiza, han traído consigo la búsqueda de nuevas formas de hacer cómputo, de lograr implementar sistemas donde además de la resistencia a fallos sea posible la interconexión. A razón de esto, se presenta el cómputo distribuido, que busca descentralizar los procesos y los recursos físicos como el hardware, así como un acceso constante, escalabilidad, entre otras. De acuerdo con IBM(2021) en su sitio de documentación, un sistema de cómputo distribuido está formado por múltiples componentes de software que se encuentran en distintas computadoras, pero son ejecutadas como un único sistema. Con esta definición resulta más sencillo comprender la importancia de este tipo de cómputo y por qué las necesidades actuales tienden a esto, ya que con el simple hecho de observar a nuestro alrededor es posible notar que nos encontramos conectados y que en su mayoría los sistemas que hoy empleamos no se basan en cómputo centralizado pues hay aplicaciones donde si un servidor falla ni siquiera es notado o que el procesamiento se ha visto mejorado debido a la implementación de la distribución.

1.1. Arquitectura Orientada a Servicios

Para lograr este cometido existen múltiples arquitecturas que se encargan de tener como principal objetivo no centralizar información. Dentro de estas existe la *Arquitectura Orientada a Servicios* (SOA por sus siglas en inglés) que es un tipo de diseño de software que permite reutilizar sus elementos gracias a las interfaces de servicios que se comunican a través de una red con un lenguaje común [1]. Si bien la definición puede resultar teórica y técnica, hace

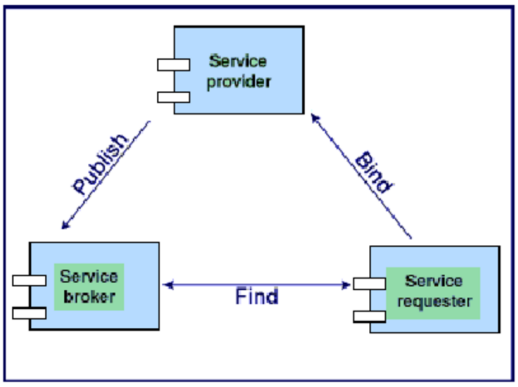

Figura 1. Arquitectura simple de SOA

uso de una palabra muy importante y que se pretende ampliar en el transcurso de este escrito, *reutilizar*. Con esto en mente, es posible plantear un acercamiento a la definición con otra perspectiva, el separar sus elementos de tal forma que puedan ser llamados de acuerdo al momento en que sean necesitados. Esta independencia y autonomía permite escalabilidad y crecimiento sin mermar el desempeño del sistema donde se esté implementado.
Antes del uso de esta arquitectura, resultaba complicado el agregar nuevas características, o realizar la conexión entre una aplicación y los servicios de otro. Este modelo centralizado y conocido como ”monolítico”limitaba la implementación de funcionalidades nuevas, la actualización o corrección de los existentes, pues si algo no funcionaba de manera adecuada, se debía *apagar* por completo el sistema, se solucionaba el error y se debía reiniciar como una versión nueva. Esto parece impensable en la actualidad, los sistemas que hoy conocemos como *críticos*, siendo estos aquellos que deben ser altamente fiables y mantener esta fiabilidad a medida que el sistema evoluciona sin incurrir en costes prohibitivos [2], han crecido de manera significativa y su importancia es tal que se perciben en fuentes de generación de energía como las nucleares donde un pequeño error en el mismo podría generar una catástrofe, como también en medios de transporte que antes se consideraban *utópicos*

como los tren bala, así como las propuestas de la compañía Hyperloop que busca realizar traslados de grandes distancias en un tiempo muy disminuido. Todos estos casos presentan la misma problemática, lo vulnerables que son a un error, por mínimo que este sea.

2. Problema

De manera general, el problema planteado se encuentra en la realización de una calculadora que implemente una *Arquitectura Autónoma Descentralizada Orientada a los Servicios* (ADSOA por sus siglas en inglés). Si bien puede parecer una implementación excesiva debido al uso de un servidor que procese la operación, la realidad es que resulta sorprendentemente acercado a la realidad. El problema entonces puede ser descrito como se menciona a continuación.

2.1. Primer problemática

El personal de una institución se encuentra en una quincena y para poder determinar el pago a cada uno de los trabajadores es necesario hacer un cálculo por medio de una calculadora que provee la empresa donde es posible realizar operaciones de adición, sustracción, multiplicación y división. Lo que en un principio se ve como una tarea para un sistema centralizado no considera los problemas que se mencionan a continuación, pues si se toma en cuenta que en este caso a medida que los trabajadores aumenten y el acceso a la calculadora sea por un solo medio, el tiempo de espera será mayor, sin mencionar que si dicho dispositivo falla, no habrá forma de realizar el pago en ese momento y la incertidumbre será entonces una preocupación. Por otro lado, si se implementa un sistema donde el cálculo no se realice en el mismo lugar de donde se introduce la información, sino en un servidor separado el problema ahora se ve segmentado, pues se hace más sencillo el determinar si el fallo ocurre del lado del cliente o del servidor.
Sin embargo, a pesar de encontrarse segmentado, la realidad es que sí existe un problema, se verá detenido el servicio. Es por ello que la adición de más servidores, así como clientes permitiría cierta interconexión, no obstante no es suficiente pues dichos dispositivos cuentan con un límite de puertos por los cuales es posible conectarse; de ahí que la implementación de un intermediario o middleware[1] resulte más que fundamental, pues este permitirá además de un mayor número de conexiones, una comunicación correcta entre ambas partes (Cliente y servidor).

1. Software que permite uno o más tipos de comunicación o conectividad entre 2 o más aplicaciones o componentes de aplicaciones en una red distribuida [4]

2.2. Segunda problemática

Ahora bien, con este primer acercamiento solo se desplaza el problema previamente planteado al middleware. A pesar de que lo descrito en el punto anterior es relativamente menos susceptible a fallos y caídas debido a que existen más de un servidor y más de un cliente que puede realizar la operación, si el intermediario falla o cae, generará un problema con la interconexión y de poco servirá que exista redundancia con los servidores y los clientes pues los mensajes no serán pasados entre sus elementos. Teniendo esta premisa en mente, no resulta desacertado considerar la inclusión de múltiples nodos que busquen evitar el error previamente mencionado y generar redudancia en las conexiones. En conjunto con esta premisa recien planteada, se agrega la segmentación por importancia de las operaciones. Esta importancia o criticidad de cada uno de las operaciones que se pueden realizar se determina por medio de la cantidad de servidores que deben responder para que la operación se considere como exitosa, esto se plantea teniendo en consideración que en una implementación real existen servicios que requieren mayor prioridad o mayor redundancia de acuerdo a lo que realizan, por lo que resulta importante la consideración de .ªcusesçuya función es indicar quien ha respondido y cuánto es el mínimo requerido.

2.3. Tercer problemática

Si se considera la redundacia como parte fundamental del sistema que se está planteando entonces resulta evidente que la intervención humana para el levantamiento de un nuevo servidor y cumplir con los acuses necesarios puede significar un error o la no respuesta de acuerdo a la criticidad de dicha operación. A partir de esto entra una nueva problemática, la autorecuperación. En palabras sencillas la autorecuperación indica la capacidad que tendrá una célula del tipo servidor para poder clonarse a sí misma y generar una nueva célula con la información que la progenitora ya poseía como lo es el procesado de información o el proceso de comunicación con algún nodo. En conjunto a esto, si un nodo al cual se conecta uno o más clientes sufriera algún fallo y su servicio se viera interrumpido la conexión de esos clientes se vería mermada y sus solicitudes no podrían ser procesadas; por lo que esto exige de manera clara la existencia de una forma en la que dicho cliente o clientes puedan reconectarse a un nodo que posea disponibilidad y se continúe con el proceso de información. La facilidad que otorga la estructura de microservicios se va puede ver mitigada o limitada si estos se colocan de manera manual en cada uno de los servidores, por lo que la existencia de una célula que se encargue de realizar un proceso de manejo e inyección, esto permitirá un mejor funcionamiento y actualización de servicios.
En conjunto, la generación de un sistema distribuido resistente a fallos y con autoregeneración. Un sistema independiente y automanejable y sostenible.

3. Desarrollo

De acuerdo al problema planteado, se consideraron los siguientes elementos con sus respectivas funciones:

1. **Cliente (Calculadora).** Introducción y lectura de operaciones en una interfaz gráfica que sea amigable al usuario. Para fines prácticos en este proyecto se consideraron cuatro operaciones suma, resta, multiplicación y división, las cuales recibirán dos parámetros cada una, el operando uno y el operando dos.
2. **Nodo (Middleware).** Recibimiento de solicitudes y respuestas. Para el primer acercamiento al problema se consideró únicamente realizar un proceso de broadcast[2] a los elementos conectados a él, pues esto permitiría a los elementos conectados conocer la información que se comunica y realizar un proceso de conexión correcto.
3. **Servidor.** El propósito del servidor es el recibir solicitudes de operaciones y procesarlas, de tal manera que se genere el resultado y este sea enviado a los nodos.
4. **Manager** El manager es la célula encargada de proveer los servicios que tendrá disponible cada uno de los servidores. En el caso de que se busque agregar una nueva operación o se modifique el funcionamiento de una, el administrador podrá *inyectar* el servicio en cuestión.

3.1. Primer entregable y solución propuesta

Este primer planteamiento permite conocer de manera general qué es lo que se espera realice el sistema a desarrollar, sin embargo todavía resulta ambiguo el proceso de comunicación. Dicho proceso se representa gráficamente en el siguiente diagrama, donde los elementos hasta la izquierda representan al cliente, el de medio al middleware y los de la derecha un servidor.

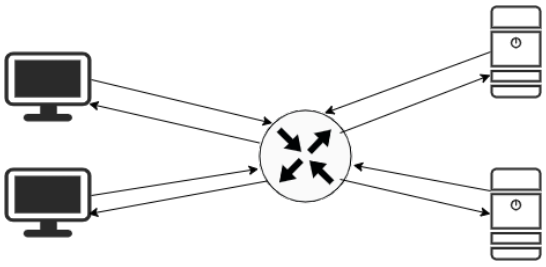

Figura 2. Modelo de conexión

De tal manera que teniendo ambas cuestiones en consideración y de acuerdo a lo esperado, se emplea el protocolo TCP. No obstante, para su implementación se requiere el uso de hilos, concepto que permite la construcción de múltiples procesos en una ejecución, por lo que permite realizar procesos paralelos, así como la apertura de un socket de comunicación para interconectar los distintos programas que simulan el sistema previamente descrito. Con los conceptos anteriores se conforma una solución. Donde el primer módulo en ser encendido es el nodo, pues es el elemento que estará continuamente escuchando en espera de nuevos clientes o servidores, posterior a él, es posible iniciar una conexión desde un cliente o servidor, que abrirá un socket y permitirá la conexión con el nodo. Sin embargo para su correcto funcionamiento es necesario que exista al menos un cliente y un servidor conectados al nodo, por lo que sin ambos existirá comunicación pero no será procesada la operación. Para que la conexión sea funcional, cada mensaje posee un código de contenido que se encuentra al inicio de cada uno. A continuación se muestran los códigos permitidos y en qué dispositivo son admitidos:

Tabla 1. TABLA DE CÓDIGOS DE CONTENIDO SOLICITUDES

Código de contenido	Operación
1	Suma
2	Resta
3	Multiplicación
4	División
11	Clonación
100	Administrador

Tabla 2. TABLA DE CÓDIGOS DE CONTENIDO GENERADOS POR EL SERVIDOR

Código de contenido	Operación
5	Resultado de suma
6	Resultado de resta
7	Resultado de multiplicación
8	Resultado de división

Como consecuencia de ello, los mensajes que no posean ninguno de dichos códigos no serán procesados ni generados por el servidor y el cliente.

3.2. Segundo entregable y solución propuesta

El camino que se toma continúa con el acercamiento a la problemática del segundo caso, es decir la redundancia de los nodos y el evitar que el servicio se vea afectado por ello. Este procedimiento resulta fundamental debido a que como se mencionó en la introducción, esta redundancia y servicio escalable permite que si existe error alguno no se vea un problema y por lo tanto se eviten accidentes. En este contexto en particular esta solución de redundancia y la generación de una malla de nodos permite que si uno se cae la operación pueda ser procesada y sea procesada sin mayor complicación. El diseño del modelo de conexión e inclusión de la malla se presenta a continuación, mantiene la

2. Modo de transmisión de información (difusión) donde un nodo emisor envía información a una multitud de nodos. [3]

estructura original de la primer entrega y solo agrega el uso de una cantidad n de nodos. Sin embargo una implementa-

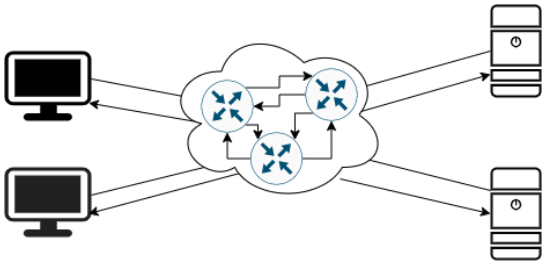

Figura 3. Modelo de conexión ADSOA

ción de esta naturaleza donde no se tiene conocimiento de a quien se le envía el mensaje genera el problema de que el envío de mensajes se puede ciclar y por lo tanto generar un sistema que no resulte eficiente.
En esta nueva entrega se cambia la consideración que se tiene para clientes, servidores y nodos. Los clientes y los servidores se generalizan en la palabra *célula* y el nodo. La generación de este impedimento se puede satisfacer si solo se conoce la asignación de quien envía el puerto y si es célula o nodo. Con esta premisa se evalúa si quien envía es un puerto que corresponde a una célula, el envío se realiza a todos los puertos conectados, es decir tanto a nodos como a células; pero si por el contrario el puerto que envía corresponde a un nodo entonces para evitar que se haga un ciclo infinito de envíos el mensaje será únicamente mandado por las células. A continuación se muestra el pseudocodigo del algoritmo aplicado:

```
Se recibe un mensaje proveniente de un elemento de la red.;
Se identifica el puerto del emisor y se asigna con célula o nodo.;
if Emisor corresponde a un nodo then
    Enviar mensaje a las células conectadas
else
    Se envia tanto a nodos como a células conectadas
end
```

El siguiente requerimiento consistió en el procesado de las operaciones de acuerdo a la célula cliente que lo envía y la consideración de que cada operación sea identificable, a raíz de esto se consideró la siguiente estructura de mensaje que cumpla de manera satisfactoria con lo requerido:

Tabla 3. ESTRUCTURA DEL MENSAJE ENVIADO

ID célula	ID evento	Código contenido	N1	N2

Donde **N1** y **N2** corresponden al primer operando y al segundo operando.
Con esta información es posible establecer entonces un sistema que reconozca que servidor está contestando y por lo tanto determine si es válida la entrada que se dio de acuerdo a la criticidad del evento en particular. Esto funciona si un proceso requiere del funcionamiento de n servidores y de esta manera se válida y se vigila que se mantenga constante el desempeño.
El tercer requerimiento correspondió a la implementación de la arquitectura SOA, es decir separar las operaciones que se realizan en microservicios, trasladando estos procesos al disco duro y así evitando el uso de la memoria RAM de manera continua. Esto se logró al generar los siguientes ejecutables:

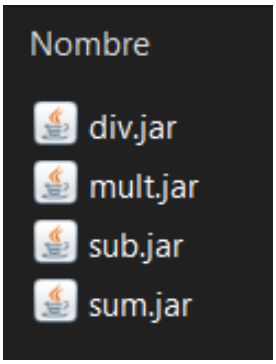

Figura 4. Archivos ejecutables generados

3.3. Tercer entregable y solución propuesta

Para el último entregable se retoma la importancia de los acuses y la redundancia establecida en el entregable pasado. Sin embargo en este caso se corta la interacción humana y se genera un protocolo que permita la autorecuperación de los servidores. La adición de esta funcionalidad permite que exista una reacción más rápida cuando suceda un fallo que limite la cantidad de servidores que respondan, sin embargo si el proceso de autorecuperación no se regula se puede caer en un problema de exceso de clonación de células que significaría en nuestro sistema un cáncer. ¿A qué se debe que un servidor pueda clonarse más de una vez? De manera general esto se resume a que se recibirán la misma cantidad de solicitudes que de clientes haya, por lo que si el mensaje se envía constantemente esperando la cantidad pertinente de acuses que la operación requiera, se podría clonar una cantidad de veces que podría colapsar el sistema. A partir de esta problemática entonces ¿Qué se puede realizar para evitar este problema? La solución propuesta para evitar dicho problema fue realizar un mensaje con la siguiente información:

Tabla 4. ESTRUCTURA DE LA SOLICITUD DE DUPLICACIÓN

ID servidor	Acuses requeridos	Código contenido

El propósito de la tabla 4 es permitir que se envíe la solicitud y sea procesada únicamente por un servidor. Se considerará el primero que se encuentre entre los acuses que respondieron y se enviará la solicitud a dicho servidor.

Si bien este método presenta la complicación de que para n clientes puede no ser el primero en cuestión, se tiene como ventaja que esta solicitud solo existirá de acuerdo al cliente que solicita por lo que es posible evitar múltiples solicitudes no deseadas debido a que se respeta que solicitud-respuesta sea para un cliente de acuerdo a su identificador único.

Además de lo previamente mencionado, la redundancia en la arquitectura es fundamental, por lo que si un cliente al intentar comunicarse no logra establecer conexión con el nodo que ya la tenía, se reintenta nuevamente en caso de que la falla haya sido momentánea; no obstante si este no fue el caso, se busca un nuevo nodo al cual conectarse y se notifica que es una célula.

Por último pero no menos importante, se estableció una nueva célula, el administrador cuya función se describió con anterioridad. Ahora bien este no recibe mensajes por lo que el proceso de comunicación es unidireccional con el nodo que encuentre disponible; al establecer esto asegura que no deberá realizar el procesado de los mensajes cliente-servidor y podrá cumplir su función con mayor eficiencia, es decir la adición de servicios.

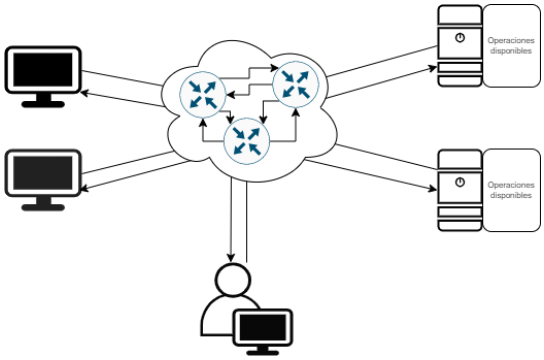

Figura 5. ADSOA con el administrador

La adición de servicios funciona de la siguiente forma, en un directorio se almacenan los microservicios que conoce el administrador, a partir de estos es posible inyectarlos en los servidores. Esta función permite que si uno de los microservicios que poseen los servidores falla, se establezca una nueva versión de la operación

4. Solución codificada propuesta

4.1. Interfaz de la calculadora

La interfaz que se diseñó para la calculadora fue la siguiente:

Considerando 2 apartados, por un lado la introducción de las operaciones por operando y por otro el historial de operaciones donde se colocan las solicitudes que se realizan, en conjunto con sus eventos.

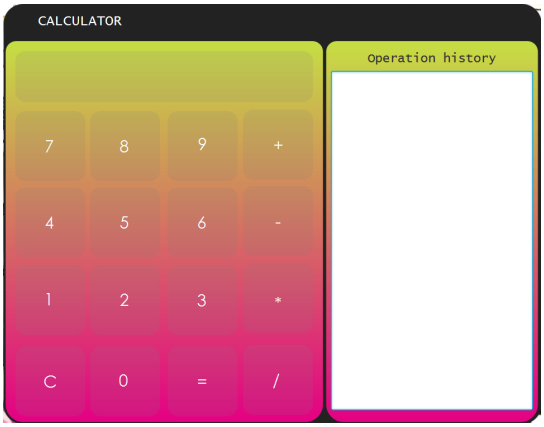

Figura 6. Interfaz de la calculadora

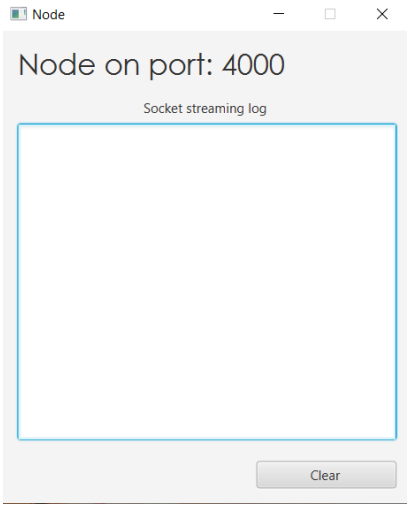

Figura 7. Interfaz del nodo

4.2. Interfaz del nodo

El nodo funciona como una consola de registro donde mencionan las operaciones que se realizaron, si es una comunicación dada por una célula o si está dada por el nodo.

4.3. Interfaz del servidor

De manera semejante al nodo, el servidor busca realizar un registro de las operaciones que se realizaron

4.4. Interfaz del administrador

La interfaz del servidor se encarga de implementar la solución a la tercer problemática, es decir el poder realizar el inyectado de servicios de manera adecuada. Tiene a su vez el propósito de mostrar los servidores que se encuentren conectados como dato de referencia y conocer a quienes se les inyectaran los respectivos servicios.

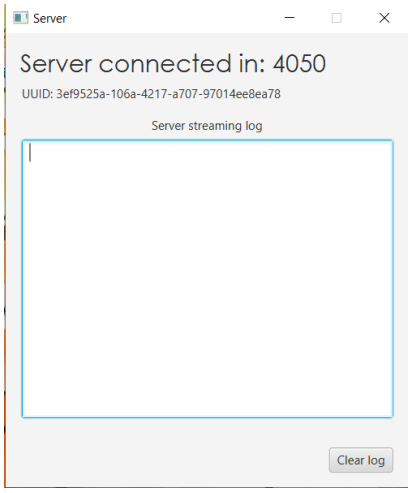

Figura 8. Interfaz del servidor

En paralelo a esta función, se incluye el poder modificar la cantidad de acuses que requieren los clientes, tomando como mínimo un valor de 1 y como máximo 10. Así mismo coloca como valores iniciales los que tiene el cliente al momento de encenderse. La inclusión de esta funcionalidad busca crear un entorno más real donde la prioridad de las funciones puede modificarse y que este cambio se vea reflejado en cada uno de los respectivos clientes.

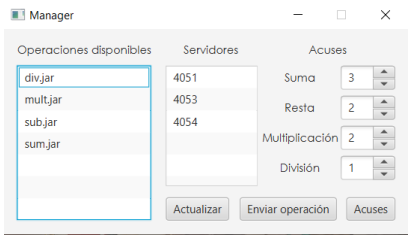

Figura 9. Interfaz del administrador

4.5. Estructura de los directorios

La estructura en la que cada servidor mantiene la independencia de servicios con los otros, así como la carpeta que corresponde a los microservicios en donde se tomará la base para inyectar, se muestra de la siguiente forma:

En la figura 10 el directorio de **manager** corresponde a la carpeta donde se encontrarán los microservicios que podrán ser inyectados, mismo donde podrán ser modificados y actualizadas las versiones en caso de que algún microservicio se vea afectado o su funcionalidad quiera ser cambiada o actualizada. Por otro lado, en el directorio de **microservices** se encuentran todos los servidores que se encuentren activos y que se crearán de manera independiente para almacenar los servicios que el administrador decida inyectar.

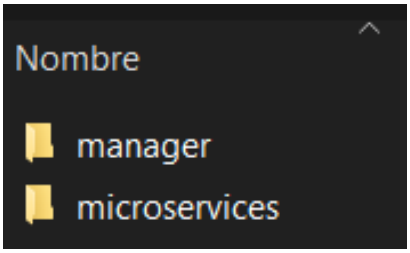

Figura 10. Estructura de la carpeta

5. Conclusion

Con la realización de esta nueva solución se exploraron nuevos campos y arquitecturas para lograr que se logre el cometido. Considero que si bien hubo impedimentos en cierto momento debido al conocimiento previo sobre el lenguaje de JAVA, el punto central de estas entregas, así como el análisis del problema y el planteamiento de una estrategia no se encuentra directamente relacionado con el lenguaje per se, sino en reflexionar y analizar los métodos posibles con los que será posible lograr los puntos que se planteen. Creo además que la dificultad de la segunda entrega se encontró principalmente en el procesado de los acuses y por lo tanto de las respuestas provistas por los servidores, pues el requerimiento de que el mensaje se esté enviando continuamente y sea posible escuchar respuestas de otros canales dificultó el procedimiento de solución. Por último pero no menos importante, la entrega final adicionó el protocolo de recuperación, lo que implicó comprender y tratar el asunto de la clonación con precaución y cautela, esto con el fin de evitar el incremento desmedido de uso de memoria por la creación de nuevas instancias.

Referencias

[1] "¿Qué es la arquitectura orientada a los servicios?", *Redhat.com.* [En línea]. Disponible en: https://www.redhat.com/es/topics/cloud-native-apps/what-is-service-oriented-architecture. [Consultado: 24-oct-2022].

[2] M. Hinchey and L. Coyle, "Evolving critical systems: a research agenda for computer-based systems.". University of Limerick, 01-Jan-2010 [En línea]. Disponible en: https://hdl.handle.net/10344/2085. [Consultado: 24-Oct-2022]

[3] "Broadcast", *Ecured.cu.* [En línea]. Disponible en: https://www.ecured.cu/Broadcast. [Consultado: 15-sept-2022].

[4] IBM Cloud Education, "What is middleware?", *Ibm.com*, 05-mar-2021. [En línea]. Disponible en: https://www.ibm.com/cloud/learn/middleware. [Consultado: 15-sept-2022].

[5] "IBM documentation", *Ibm.com*, 19-abr-2021. [En línea]. Disponible en: https://www.ibm.com/docs/en/txseries/8.2?topic=overview-what-is-distributed-computing. [Consultado: 15-sept-2022].