**ROTACIÓN DE IMÁGENES**

1. **REQUERIMIENTOS:**

**R1:** Poder rotar la imagen de un subespacio dado en unos grados dados.

**R2:** Leer un archivo de texto con la secuencia de operaciones. El archivo consiste en varias líneas, donde cada línea tiene una letra indicando la instrucción (**i**nput directory, **o**utput directory, **r**otate), separado de su parámetro por un espacio (en el caso de **i**, el directorio donde están las imágenes a procesar; en el caso de **o**, el directorio donde deben quedar las imágenes procesadas; en el caso de **r**, los grados por los que se van a rotar).

**R3:** Del directorio de entrada, cargarlas en orden alfabético y realizarles el proceso indicado por el archivo de texto.

**R4:** Guardar el resultado del proceso de cada imagen del directorio de entrada en el directorio de salida.

**R5:** El servicio debe de ser escalable: se debe de poder procesar tanto imágenes livianas como grandes en tiempos coherentes.

**R6:** El servicio debe de ser configurable: El tamaño de la imagen puede variar y se debe poder cambiar el total de nodos de procesamiento activos, además del total de componentes funcionando en cada nodo.

**R7:** Hacer uso del lenguaje de programación Java

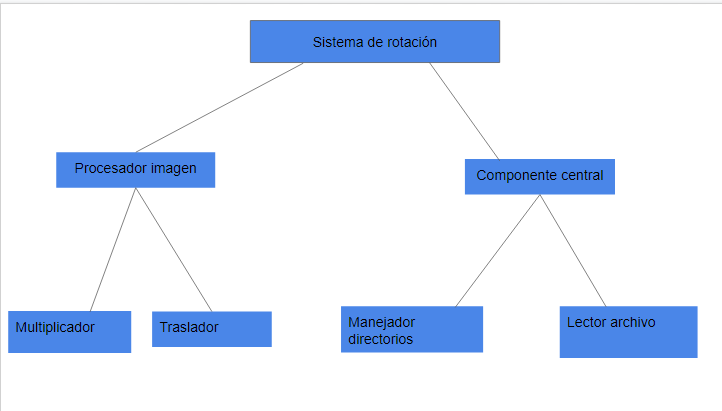
**R8:** Hacer uso del middleware FraSCAti

**R9:** El desarrollo debe de ser evolutivo, comenzando con el componente de rotación

1. **CLASIFICACIÓN:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Clasificación** | | | |  | **RAS: Orígen** | | | | |
| **Requerimientos del sistema** | **Fun** | **NFun** | **Prc** | **Prd** |  | **Vol Fun** | **Flia Prd** | **Op& Dep** | **Atr Cal** | **Det Tec** |
| **R1** | **x** |  |  | **x** |  | **x** |  |  |  |  |
| **R2** | **x** |  |  | **x** |  |  |  |  |  |  |
| **R3** | **X** |  |  | **X** |  |  |  |  |  |  |
| **R4** | **x** |  |  | **x** |  |  |  |  |  |  |
| **R5** |  | **x** |  | **x** |  |  |  |  | **x** |  |
| **R6** |  | **x** |  | **x** |  |  |  | **x** | **x** |  |
| **R7** |  | **x** |  | **x** |  |  |  |  |  |  |
| **R8** |  | **x** |  | **x** |  |  |  |  |  | **x** |
| **R9** |  |  | **x** |  |  |  |  |  |  |  |

1. **PARTICIONAMIENTO:**



**D) Modelo de procesamiento**

El modelo de procesamiento escogido para el desarrollo de la solución es SIMD (Single Instruction Multiple Data), ya que el procesamiento principal es una única instrucción: rotar una matriz en un ángulo dado, pero esa instrucción se aplica sobre múltiples datos (ya sea las diferentes fotos en una carpeta, o inclusive las diferentes secciones de una misma foto que son procesadas al mismo tiempo).

**E) Modelo de almacenamiento**

Como modelo de almacenamiento, se decidió de utilizar el modelo NORMA (NO Remote Memory Access). Dicha decisión fue tomada debido a que no era necesario que los diferentes componentes de rotación tuvieran acceso a los datos completos de las imágenes (ya sea de forma uniforme o no uniforme), lo que podría causar mayor latencia o riesgo de perder información al distribuir los datos. Así, como el tratamiento de un pedazo de la imagen es independiente del resto, es un componente central el que se encarga de enviar estructuras de datos que representan información de un pedazo de la imagen a cada procesador, para que estos apliquen el proceso.

**F) Patrones / Estrategias**

Para este caso, con el fin principalmente de cumplir con los requerimientos no funcionales, principalmente el de escalabilidad, se decidió hacer uso de un Proxy remoto. Así pues, el componente de ManejadorArchivos se encarga de cargar las imágenes y empezar a mandar cada una a rotarla como ella quiere: esto lo hace llamando el método de rotarImagen del coordinador (Implementa de RotadorImagenes), que en verdad es un proxy. Este proxy se encarga de partir la imagen en n pedazos (siendo n el total de nodos de rotación disponibles), y le envía un pedazo a cada uno.

En el Device de procesamiento de imágenes, el que recibe los pedazos de imagen es en realidad un controlador (Que también implementa RotadorImagenes), que a su vez puede verse como un Proxy cuyo cliente es el coordinador. El controlador se encarga nuevamente de dividir la imagen para repartirla entre los componentes de rotación que estén configurados dentro del Device (Para los casos del enunciado de este proyecto, se va a tener ya sea 1 o 4 componentes).

**Aclaración sobre el coordinador:** De cierto modo, podría llegar a dar la impresión que el coordinador se trata más de un Broker y no de un Proxy debido al hecho que conoce la dirección de varios “servidores” (Procesadores de imagen en esta caso) y se encarga de redireccionar las imágenes a dichos servidores. Sin embargo, el objetivo de este componente no es de seleccionar el “servidor” en donde se atenderá la solicitud para distribuir las cargas, sino que para cada solicitud le envía un pedazo con la misa carga a cada uno de los procesadores. Su objetivo es más bien evitar que el Manejador de imágenes tenga la responsabilidad de conocer cada uno de los dispositivos que hace la rotación y mandarselos de forma distribuida; más bien, simplemente se encarga de leer las imágenes e indicarle a un componente que se las devuelva rotadas con un ángulo dado (el proxy en este caso, quien se encarga de lo descrito anteriormente).

**G) Diagrama final de Deployment**

