Trabajo Práctico 2 - Buscador de Sinónimos Rústico

Integrantes

Integrante	Padrón	Mail
Botalla, Tomás		tbotalla@fi.uba.ar
Alvarez, Dylan		helllo5d123@gmail.com
Donato, Juan Pablo	100839	judonato@fi.uba.ar

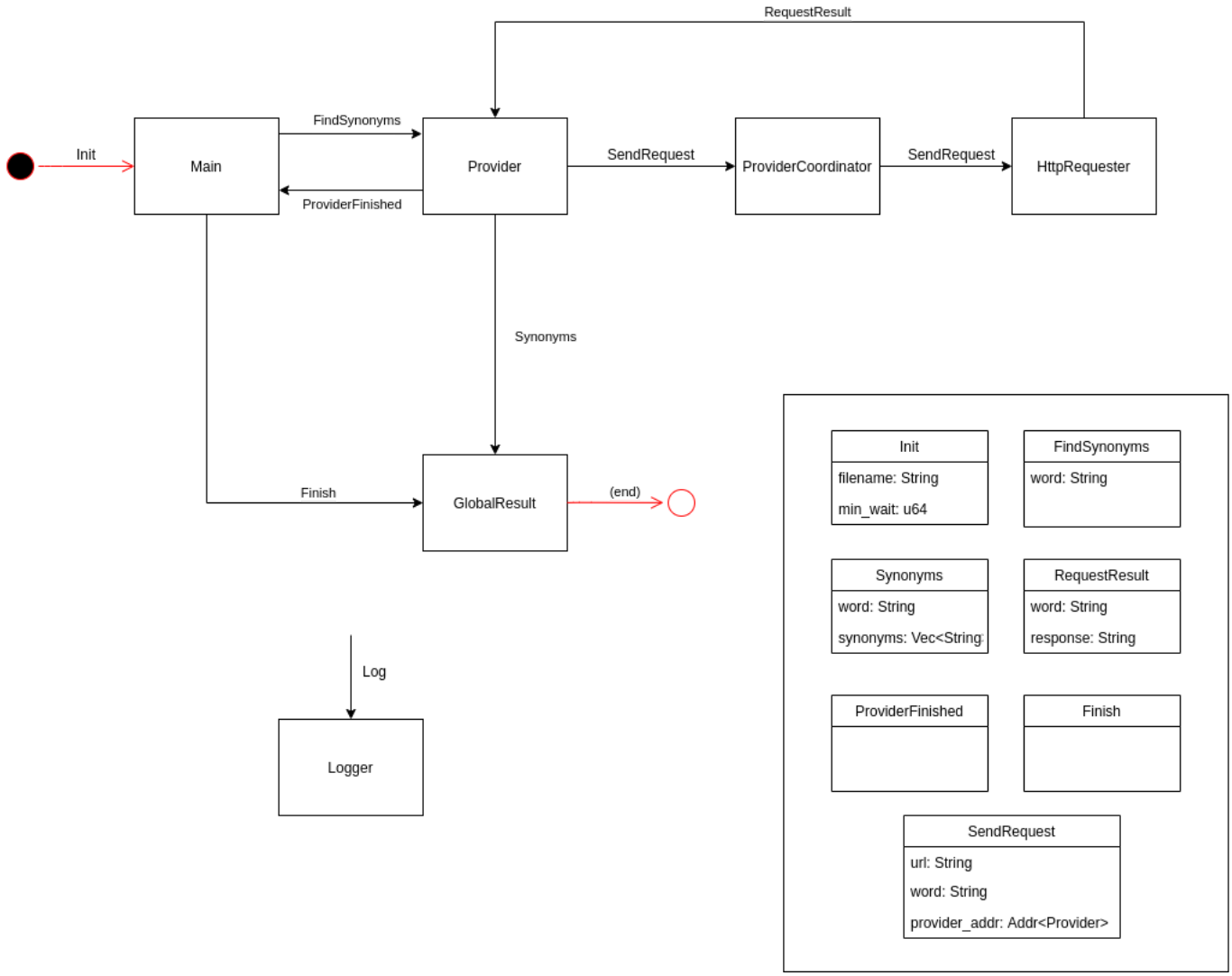
Introducción

El objetivo de este informe será presentar y detallar las soluciones implementadas por el grupo 1 para la construcción del trabajo practico 2, el cuál consistió en implementar un buscador de palabras utilizando distintos proveedores de información en la web, mediante el uso del lenguaje de Rust, sus distintas alternativas para manejar concurrencia y librerias.

Solución A - "Sin Actores"

Solución B - "Con Actores"

En esta sección explicaremos los detalles mas importantes de esta solución B, donde como herramienta principal de manejo de concurrencia utilizamos el modelo de **Actores**, provistos por la libreria *actix*. A continuación, un gráfico de soporte para entender la arquitectura básica y el modelo de actores implementados en esta sección:



En el esquema, podemos ver a los siguientes Actores:

Actor	Descripción
Main	Actor principal en la solución. Da comienzo a la ejecución del proceso y, cuando lo determine, tambien lo finaliza
Provider	Actor que representa a un proveedor (por ejemplo, Merriam-Webster). Son "tipados" segun el proveedor que les toca y contienen la logica de parseo, segun el sitio
Provider Coordinator	Actor que contiene la logica de coordinación y tiempo minimo entre requests de proveedores del mismo sitio.
HttpRequester	Tiene la logica basica de realizar el request Http al sitio y con la palabra que se le indique. Devuelve el body del request
Global Result	Actor que centraliza el resultado global de la ejecución del programa. Dada una palabra y una lista de sinonimos, los agrupa y lleva el conteo de las repeticiones segun resultados.
Logger	Actor que tiene la tarea de logear sucesos indicados en el archivo de output

Y por otra parte tambien podemos ver la lista de Mensajes implementados. A continuación, el detalle:

Mensaje	Descripción
Init	Mensaje de inicio del programa. Contiene la ruta del archivo a parsear con las palabras a buscar y el tiempo de espera minimo en milisegundos
FindSynonyms	Mensaje de petición a un provider para buscar los sinonimos de determinada palabra
SendRequest	Mensaje de petición de realización de request Http contra un sitio en particular, para buscar los sinonimos de una palabra. Contiene ademas la dirección de la casilla del proveedor que espera recibir el resultado.
RequestResult	Una vez obtenido el resultado del request (producto del mensaje de arriba), se lo envia utilizando este mensaje
Synonyms	Luego de realizar la tarea de parseo que le corresponda a cada proveedor, mediante este mensaje se envian la palabra y la lista de sinonimos encontrados
ProviderFinished	Mensaje que envia un proveedor para indicar que su tarea ha finalizado
Finish	Mensaje final para indicar que todos los proveedores han finalizado y se espera obtener el resultado final
Log	Mensaje que contiene la cadena a escribir en el archivo de output del proceso

Luego del detalle de los principales componentes, pasamos a describir las caracteristicas del flujo principal del programa para realizar la busqueda de los sinonimos y obtener el resultado final:

- En la función `start_actors` que se llama para comenzar el programa se inicializan los actores **Main**, **GlobalResult**, **Logger** y **HttpRequester**, y se envia el mensaje *Init* al **Main** para comenzar la ejecución.
    - La caracteristica escencial en este paso es la forma en como se inicializa el actor **HttpRequester**, la cual es clave para la solución de la cantidad maxima de requests concurrentes permitida. En lugar de inicializarlo utilizando la forma mas convencional (`Actor.start()`), se inicializa utilizando la función `SyncArbitrer::start` provista por la libreria de actix, que permite indicarle que cantidad de threads tiene que levantar para inicializar el actor que le pasamos por parámetro. De esta forma, por ejemplo, si quisieramos realizar como máximo 3 requests de forma concurrente, entonces pasandole el número indicado a esta función entonces podriamos "instanciar" de alguna manera tres instancias del **HttpRequester** que permitirán procesar tres mensajes *SendRequest* como máximo de forma concurrente.
- Luego de recibir el mensaje *Init*, el actor **Main** instancia un **Provider** de cada tipo, junto con el **ProviderCoordinator** que le corresponda. De esta forma, cada **Provider** de cada tipo tiene su propio **ProviderCoordinator**.
- Luego de instanciar los proveedores, parsea el archivo de palabras y envia un mensaje *FindSynonyms* a cada proveedor que tenga para indicarle que debe buscar los sinonimos de esa palabra.
- Para comenzar, cada **Provider** necesita del material para parsear. Entonces, envia el mensaje *SendRequest* al **HttpProvider**, utilizando a su **ProviderCoordinator** como "middleware" en la

operación.
    - La tarea del coodirnador es, entonces, hacer de mediador entre el provider y el requester para coordinar los tiempos minimos entre requests para cada proveedor.
    - La forma de hacerlo es frenando al Coordinator por un tiempo `MIN_TIME_BEWTWEEN_REQUESTS` de recibir nuevos eventos, y simplemente luego reenviar el mensaje al **HttpRequester**.
- Al recibir el mensaje *SendRequest*, el requester efectua la petición al proveedor indicado, y devuelve la respuesta a la dirección del provider que lo haya solicitado.
- Cuando el provider reciba la respuesta, parsea el HTML del contenido (segun las reglas establecidas para cada proveedor), y envia al **GlobalResult** la lista de sinonimos encontrados. Por último, envia el mensaje *ProviderFinished* al actor **Main** para indicarle que ha finalizado su tarea.
- El actor **Main** lleva el conteo de la cantidad de **Providers** que hayan finalizado su tarea de busqueda de sinonimos. Cuando todos hayan terminado de buscar los sinonimos de todas las palabras, envia el mensaje final *Finish* al **GlobalResult** para que devuelva como output el resultado final.