24 de mayo de 2018

**INFORME DEL FRAMEWORK OCTOPUS**

El presente informe tiene como objetivo dar a conocer los puntos más importantes sobre la investigación del Framework Octopus, el cual nos llevó al trabajo de sverhoeven llamado octopus-job-ws alojado en un repositorio en GitHub (<https://github.com/sverhoeven/octopus-job-ws>), donde nos indica lo siguiente:

Servicio web para enviar trabajos a través de un programador compatible con Octopus.

Requisitos

* JDK 7 ( http://www.java.com )
* Maven 3 ( http://maven.apache.org )
* Octopus ( https://github.com/NLeSC/octopus )

Instalación

1. Construir pulpo

git clone https://github.com/NLeSC/octopus.git

cd octopus

construcción de hormigas -Dversión = -1.0

1. Agregue el contenedor Octopus al repositorio maven local, para que pueda usarse como dependencias maven

mvn install:install-file -Dfile=dist/octopus-1.0.jar -DartifactId=octopus -Dversion=1.0 -DgroupId=nl.esciencecenter.octopus -Dpackaging=jar -DgeneratePom=true

1. Haga una copia de 'joblauncher.yml-dist' a 'joblauncher.yml' 3.1 Configure el programador de Octopus 3.2 Configure el directorio raíz de la caja de arena Octopus 3.3 Configure la clave / id de MAC opcional 4. Cree uber-jar o ejecútelo desde maven. 4.1. Uber-jar, para iniciar en otra máquina, deben copiarse los archivos octopus-job-ws-2.0.jar y joblauncher.yml.

mvn package

java -jar target/octopus-job-ws-2.0.jar server joblauncher.yml

1. Ejecutar desde maven

mvn compile exec: java

El framework envía trabajo al servicio web utilizando una solicitud HTTP POST. Informa el estado del trabajo al remitente utilizando una URL de devolución de llamada.

Al momento de armar el ambiente de trabajo para poder implementar este trabajo, se obtuvo el siguiente inconveniente para obtener uno de los requisitos:

El código Octopus alojado en la siguiente dirección <https://github.com/NLeSC/octopus> ya no se encuentra disponible, en lugar de ello te redirige a otra enlace donde se encuentra Una biblioteca de abstracción de middleware que proporciona una interfaz de programación simple para varios recursos de computación y almacenamiento, llamada Xenon. En consecuencia, se procedió a realizar una búsqueda exhaustiva por conseguir el código en algún otro medio para poder llevar acabo la implementación sin tener resultados positivos.

En dicha investigación se pudo corroborar en el artículo Octopus to the Rescue: the Fascinating World of Inter-App Communications at Uber Engineering By Apple Chow and Bian Jiangque, que el código pertenece a la empresa Uber y no está abierta al público, además nos presenta lo siguiente:

**Octopus: nuestra solución Uber-Useful**

Utilizando el ejemplo del piloto, Octopus inicia dos simuladores / emuladores. Lanza la aplicación del controlador y ejecuta la prueba del controlador en uno, al mismo tiempo que ejecuta la aplicación del conductor y ejecuta la prueba del conductor por el otro. Además, maneja la comunicación entre procesos entre las dos pruebas.

El significado de una señal viene determinado por la lógica dentro del caso de prueba. Cuando cualquier instancia del simulador / emulador escribe una señal, especifica el destino donde se debe enviar la señal. Octopus es responsable de entregar la señal al destino especificado, por ejemplo, de un objetivo de prueba a otro objetivo de prueba.

Octopus proporciona dos métodos relacionados con la señalización:

readSignal():string

Lee la siguiente señal del canal de señal actual. Cada caso de prueba está vinculado a un canal de señal (una cadena como nombre de señal) desde el que puede leer señales.

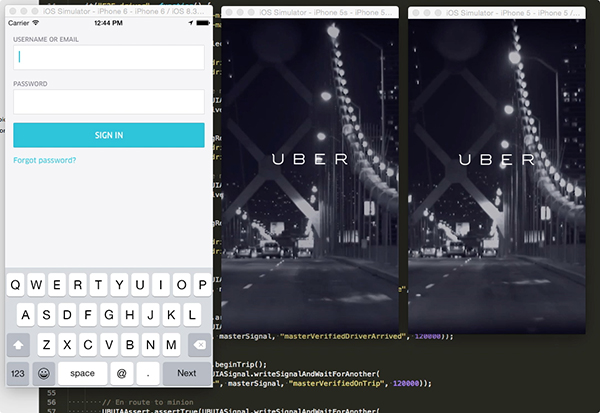
writeSignal(signalContent:string, destination:string):boolean

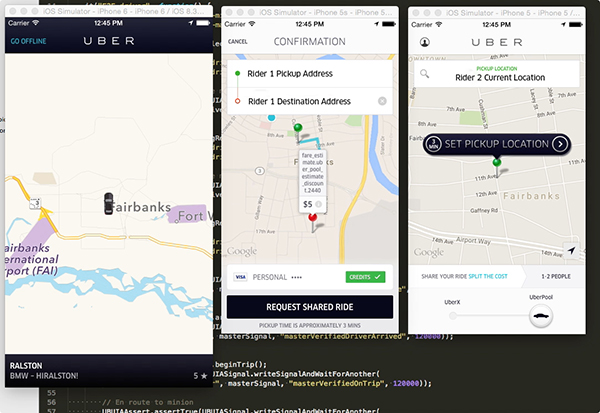
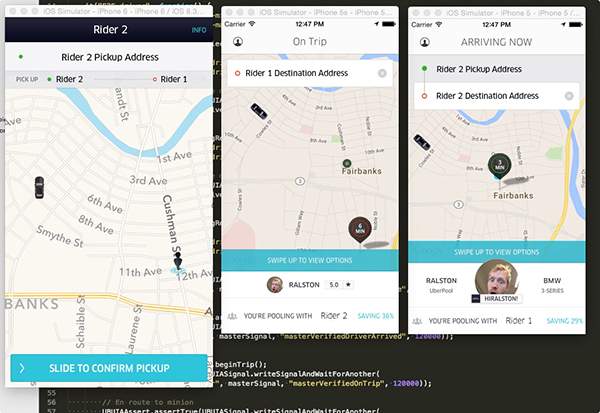
Escribe la señal en un canal de señal para que el objetivo de prueba vinculado a ese canal de señal pueda obtener la señal usando readSignal

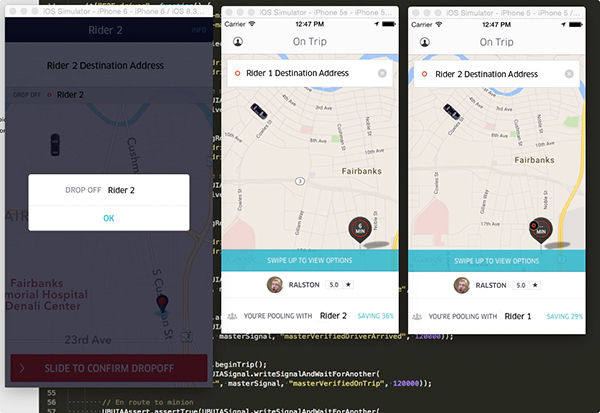
Otra forma de entender esto, imagina cómo cada simulador / emulador tiene una bandeja de entrada y el buzón de salida. El buzón de salida contiene mensajes para ser entregados a otro simulador. Buzón de salida controlador Ej dice driver\_online rider1. Pulpo copiará driver\_online en el rider1 bandeja de entrada del simulador. Entonces rider1 puede comenzar a solicitar el viaje.

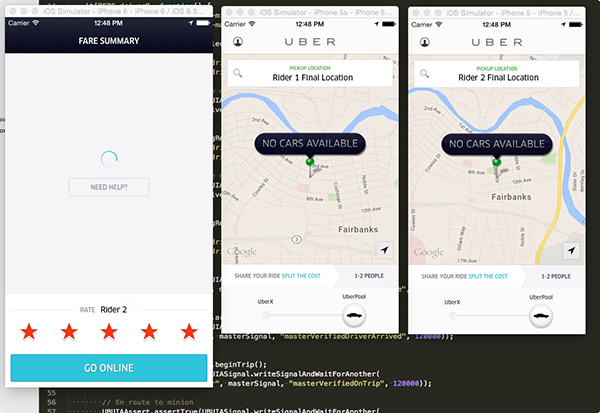
Las implementaciones de readSignal y writeSignal son totalmente diferentes en Android e iOS.

**Octopus en acción**

En un mundo de prueba con un vehículo Uber y dos jinetes, ¿a qué se parece un viaje simulado de uberPOOL? Siga con la presentación de diapositivas a continuación para ver a Octopus en acción en nuestro ejemplo de uberPOOL, desde la etapa de inicio de sesión en la primera diapositiva hasta el primer drop-off en la última diapositiva. El simulador de la extrema izquierda es la aplicación del controlador; el medio y la derecha son las aplicaciones del piloto:







En la simulación, ves que Rider 1 (centro) solicita un paseo. Debido a que este es un entorno sintético, una vez que el conductor (izquierda) acepta la solicitud, el Rider 2 (derecha) no ve automóviles disponibles. Sin embargo, estamos mostrando un uberPOOL, así que una vez que se inicia el viaje, el conductor vuelve a estar disponible para un viaje coincidente. La ubicación y el destino de Rider 2 se ajustan a los criterios, por lo que el Rider 2 ahora puede solicitar el mismo vehículo. El conductor recoge a los pasajeros y los dejan en el orden que tiene sentido para su viaje. Finalmente, el conductor califica a los pasajeros individuales al final del viaje.

Doy por culminado este informe para el uso que se estime conveniente.

Atentamente,

Jordy Andrés Castro López.

Practicante comunitarias Proyecto FCI