_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MEJORA DEL BLOQUE DE FLUÍDICA DE LA PLATAFORMA AWS F20. EVOLUCIÓN AL PRODUCTO AWS F30.

Carlos Molina Rosa

Tutor: Yolanda Jiménez Jiménez

Primer Cotutor: José Vicente García Narbón

Segundo Cotutor: Román Fernández Díaz

Trabajo Fin de Máster presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Máster en Ingeniería de Telecomunicación

Curso 2015-16

Valencia, 13 de septiembre de 2016

Resumen

Este Trabajo Fin de Máster (TFM) aborda dos retos actuales de las tecnologías acústicas basadas en sensores de cuarzo. El primero de ellos consiste en la mejora de la usabilidad de la instrumentación empleada en la caracterización y manejo de estos sensores. Desde los usuarios de este tipo de instrumentación se demandan aplicaciones software que simplifiquen su manejo. En este sentido, en este TFM se ha desarrollado una aplicación que permite la programación de experimentos. Dicha aplicación se ha integrado en el software AWS Suite comercializado por una empresa cuya principal línea de producto es el desarrollo de instrumentación para la caracterización de sensores acústicos (AWSensors SL). La aplicación desarrollada incrementa el valor añadido de la plataforma AWS F20 comercializada por AWSensors, ya que, desde un entorno muy intuitivo, permite al usuario de la misma reducir el tiempo que debe estar pendiente exclusivamente del manejo del instrumento. Por otro lado, la aplicación desarrollada también facilita enormemente el post-procesado de la respuesta proporcionada por el sensor. El segundo de los retos abordados en este TFM está relacionado con el incremento de la fiabilidad de la medida de la respuesta entregada por el sensor. La optimización del set up experimental es clave para conseguir este reto. En particular, en este TFM se ha contribuido a la optimización del sistema de fluídica evaluando un sistema de bombeo alternativo (mzr-7240) al empleado actualmente en la plataforma AWS F20 (Hamilton PSD4 5495-30). Tras la puesta en marcha y adaptación del sistema de bombeo a evaluar se ha concluido que el nuevo sistema no mejoraba las prestaciones del instrumento, por lo que se desaconseja sustituirlo por el ya existente. Este estudio ha servido para concluir que una bomba del tipo micro annular gear no es adecuada para integrarla en la evolución del producto AWS F20.

Resum

Este Treball Fi de Màster (TFM) aborda dos reptes actuals de les tecnologies acústiques basades en sensors de quars. El primer d'ells consistix en la millora de la usabilidad de la instrumentació empleada en la caracterització i maneig d'estos sensors. Des dels usuaris d'este tipus d'instrumentació es demanden aplicacions programari que simplifiquen el seu maneig. En este sentit, en este TFM s'ha desenrotllat una aplicació que permet la programació d'experiments. La dita aplicació s'ha integrat en el programari AWS Suite comercialitzat per una empresa la principal línia de producte de la qual és el desenrotllament d'instrumentació per a la caracterització de sensors acústics (AWSensors SL). L'aplicació desenrotllada incrementa el valor afegit de la plataforma AWS F20 comercialitzada per AWSensors, ja que, des d'un entorn molt intuïtiu, permet a l'usuari de la mateixa reduir el temps que ha d'estar pendent exclusivament del maneig de l'instrument. D'altra banda, l'aplicació desenrotllada també facilita enormement el post-processat de la resposta proporcionada pel sensor. El segon dels reptes abordats en este TFM està relacionat amb l'increment de la fiabilitat de la mesura de la resposta entregada pel sensor. L'optimització del set up experimental és clau per a aconseguir este repte. En particular, en este TFM s'ha contribuït a l'optimització del sistema de fluídica avaluant un sistema de bombament alternatiu (mzr-7240) a l'empleat actualment en la plataforma AWS F20 (Hamilton PSD4 5495-30). Després de la posada en marxa i adaptació del sistema de bombament a avaluar s'ha conclòs que el nou sistema no millorava les prestacions de l'instrument, per la qual cosa es desaconsella substituir-ho pel ja existent. Este estudi ha servit per a concloure que una bomba del tipus micro annular gear no és adequada per a integrar-la en l'evolució del producte AWS F20.

Abstract

This Master's Thesis (TFM) deals with two current challenges of acoustic technologies based on quartz sensors. The first challenge consists of improve the usability of the instrumentation used in the characterization and management of these sensors. The users of this type of instrumentation demand that software applications simplify their management. In this way, in this TFM has been developed an application that allows the experiments programmation. This has been integrated into the AWS Suite software marketed by a company whose main product line is the development of instrumentation for the characterization of acoustic sensors (AWSensors SL). The developed application increases the added value of the AWS F20 platform marketed by AWSensors because, from a very intuitive environment, allows to the users of the platform reduce the time to watch for exclusively handling the instrument. On the other hand, the developed application also greatly facilitates post-processing of the response provided by the sensor. The second challenge addressed in this TFM is related to the increased reliability of the measure of the response given by the sensor. The optimization of the experimental set up is the key for get this challenge. In particular, this TFM has contributed to the fluidic system optimization, with the evaluation of an alternative pumping system (mzr-7240), instead of the employee currently in the AWS F20 platform (Hamilton PSD4 5495-30). After the start-up and adaptation of the new pumping system, it has been concluded that the new system did not improve the performance, so it is inadvisable replace the existing one. This study has served for conclude that a new pump of micro annular gear pump technology is not adequate for its integration in AWS F20 product evolution.

Agradecimientos

Es posible que los agradecimientos sea la parta más difícil de escribir ya que es algo que no se tiene preparado o estudiado, sin embargo en esta ocasión es bastante sencillo. En primer lugar agradecer a Yolanda por haber podido volver a realizar 2 años después mi Trabajo Fin de Máster en AWSensors, donde tan grato recuerdo guardo después de haber realizado el TFG en su momento y ser una magnífica tutora. Agradecer también al resto del equipo de AWSensors por otros 6 meses fantásticos donde he aprendido incluso más que en mi anterior estancia, y de sentirme mucho más preparado para lo que me voy a encontrar a partir de ahora. En especial agradecer a Román, José Vicente, Paco y Miguel todas las horas que me han tenido que aguantar con las dudas y las peleas con Java, que por suerte o desgracia, no han sido pocas.

No me puedo olvidar de mis amigos, los cuales no me han aguantado solo durante el Máster, sino que me han aguantado y apoyado desde que entre en la escuela, y que seguramente esto hubiera sido mucho más duro sin ellos.

Por último a mi familia, pero sobre todo a mis padres por haberme dado la oportunidad y haber hecho el esfuerzo de que yo estudiara esta carrera y estudiarla fuera de mi ciudad. Sin duda, a este proyecto también han contribuido ellos.

Gracias a todos.

Índice

1. Marco en el que se desarrolla el Trabajo Fin de Máster

1.1 La empresa Advanced Wave Sensors S.L

AWSensors S.L es una empresa de alta tecnología que desarrolla, produce y comercializa instrumentación de alta precisión basada en sistemas electrónicos de caracterización de sensores acústicos fundamentados en diversos sistemas patentados. Impulsada por un grupo de investigadores del Departamento de Ingeniería Electrónica de la Universitat Politècnica de València (UPV), surge en el año 2009 a partir de una patente desarrollada en la UPV con el objetivo fundamental de cubrir la necesidad de disponer de técnicas de prevención y diagnóstico rápido en el área de salud. Las técnicas de análisis en tiempo real y de alta sensibilidad demandadas desde este sector pueden ser proporcionadas por la tecnología de AWSensors, que permite alcanzar resoluciones de varios órdenes de magnitud superiores a las alcanzadas por otras tecnologías ya establecidas.

AWSensors ha desarrollado una línea de sensores de alta sensibilidad cuyo uso puede destinarse a aplicaciones biotecnológicas en ámbitos muy diferentes, desde aplicaciones de diagnóstico médico (por ejemplo para la detección de patógenos como bacterias y virus, o biomarcadores para la detección del cáncer), aplicaciones de calidad y seguridad alimentaria, detección de polución en el medio ambiente y caracterización de nuevos materiales en aplicaciones del sector de la energía.

Fig.1 Logos de los organismos financiadores del Trabajo Fin de Máster

Este Trabajo Fin de Máster se ha desarrollado en la empresa AWSensors S.L. Spin-off UPV, y ha sido financiado por el Ministerio de Economía y Competitividad a través del proyecto AGL2013-48646-R.

1.2 Descripción de la tecnología desarrollada por AWSensors

Los sensores desarrollados por AWSensors están basados en diversas tecnologías acústicas fabricadas con materiales piezoeléctricos. AWSensors también diseña y comercializa celdas de flujo que sirven de soporte al sensor para las distintas aplicaciones, así como la instrumentación electrónica necesaria para extraer del sensor la información requerida en cada aplicación. En este apartado se describen brevemente los fundamentos de los sensores desarrollados en AWSensors, algunas aplicaciones en las que puede ser empleada la tecnología proporcionada por esta empresa, y por último, las características básicas de la instrumentación desarrollada (Plataforma AWS-A20 y AWS-F20).

1.2.1 Fundamentos de los sensores desarrollados en AWSensors

El cristal de cuarzo es el material piezoeléctrico más utilizado para realizar el control de frecuencia en sistemas electrónicos de comunicaciones, y desde hace unas décadas también es uno de los más empleados para determinar propiedades geométricas y físicas de diferentes materiales.

El término piezoelectricidad define el fenómeno que describe la aparición de cargas positivas y negativas en las superficies de un cristal poseedor de esta propiedad al comprimirlo. Esa compresión provoca la deformación de su estructura cristalina lo que resulta en la aparición de pequeños dipolos, es decir, el cristal queda polarizado apareciendo un campo eléctrico. Si se colocan sobre ambas caras del cristal unos electrodos unidos por un hilo conductor, el campo eléctrico interno da lugar a un flujo de cargas libres en dicho hilo. Al dejar de ejercer la presión sobre el cristal, la polarización desaparece y el flujo de cargas cesa. Así, el cristal de cuarzo se comporta como un transductor puesto que la energía mecánica empleada para deformar el material se transforma en energía eléctrica. Del mismo modo, la aplicación de un campo eléctrico sobre el material a través de dos electrodos colocados sobre sus superficies provoca la deformación del cristal, si dicho campo se aplica de forma periódica el cristal se deforma también periódicamente. De entre todas las frecuencias de excitación eléctrica del cristal, existe una que hace que su deformación sea la máxima. Esta frecuencia se conoce como frecuencia de resonancia, de ahí que también se conozca a estos materiales con el nombre de resonadores piezoeléctricos.

La aplicación más antigua del cristal de cuarzo como sensor es su empleo como microbalanza (Quartz Crystal Microbalance - QCM). En sus primeros usos, los cristales de cuarzo se utilizaban como patrones de referencia de frecuencia en los osciladores que generaban las portadoras de las emisoras de radio. Las frecuencias de resonancia de estos cristales se ajustaban mediante la deposición de masas muy pequeñas sobre el cristal hasta lograr la frecuencia buscada. Esta aplicación estaba basada en los estudios de Lord Raylegh, quien en 1945 demostró que pequeños cambios en la inercia de un sistema de vibración mecánico modificaba su frecuencia de resonancia; pero no fue hasta años más tarde cuando empezó a utilizarse como sensor [1]. En 1959 Sauerbrey demostró empíricamente que la variación en la frecuencia de resonancia de un cristal de cuarzo era proporcional a la masa añadida sobre el mismo, siempre y cuando dicha masa formara una capa muy fina y uniforme [2]. Estudios posteriores demostraron que el sensor de cuarzo era capaz de medir masas del orden de picogramos. Esta gran sensibilidad es debida a la tremenda aceleración a la que están sometidas las partículas depositadas sobre la superficie del cristal, la cual es proporcional al cuadrado de su frecuencia de vibración. Para un cristal vibrando a 10MHz, la aceleración resultante es del orden de 10^7 g, siendo g la aceleración de la gravedad. Esto supone que una partícula pesa 10^7 veces más sobre una balanza de cristal de cuarzo que sobre una balanza convencional.

En los años 80, fue demostrada la estabilidad de un oscilador controlado por cristal de cuarzo estando en contacto con un medio líquido (hasta el momento se creía que al depositar un fluido sobre el sensor éste dejaría de oscilar) [3]. Ello daba la oportunidad de utilizar el cristal de cuarzo para realizar procesos de detección que se producían mejor en líquido que en gas, este es el caso de los biosensores.

En AWSensors se trabaja con tres tipos de sensores fabricados con materiales piezoeléctricos (Ver Fig.2):

- Quartz Crystal Microbalance (QCM)
- High Fundamental Frequency QCM (HFF-QCM)
- Surface Acoustic Wave Love (SAW-Love)

QCM HFF-QCM SAW-Love

Fig.2 Detalle de los tres tipos de sensores comercializados por AWSensors

Según la aplicación a realizar será conveniente emplear uno u otro. En este trabajo fin de máster se utilizarán cristales QCM y HFF-QCM. Los sensores QCM y HFF-QCM son sensores fabricados a partir de cuarzo en corte AT. El corte realizado condiciona el modo de vibración del cristal, además de otras características como los modos de vibración no deseados o la influencia de la temperatura sobre la frecuencia de vibración. Los sensores QCM tradicionales se fabrican depositando unos electrodos de oro sobre cada cara de una lámina de cuarzo AT. Al aplicar una diferencia de potencial entre los electrodos, lo cual genera un campo eléctrico en la dirección del espesor, se produce en el cristal una deformación en forma de cizalla. Cuando el potencial aplicado es alterno, esta deformación se produce a la misma frecuencia de variación de dicho potencial, generándose una onda acústica que se propaga en dirección del espesor. Esta onda acústica se refleja en las superficies del cristal generándose ondas incidentes y reflejadas. Para una cierta relación entre la frecuencia de excitación del potencial eléctrico y el espesor del cristal (ver Ec. (1)) la velocidad y la amplitud de dichas ondas incidentes y reflejadas se igualan, generándose una onda estacionaria, y por tanto, la resonancia del cristal.

$$f = \frac{N}{h_q} n \quad con\ n = 1{,}3{,}5, \ldots \tag{1}$$

En la Ec.(1) $N = 1664kHz \cdot mm$ para un cuarzo AT y h_q es el espesor del cristal.

Cuando el sensor se recubre de algún material del que se quieren estudiar sus propiedades físicas, la onda acústica que se propaga por el cristal penetra en dicho material deformándolo. Al entrar en contacto con un medio de propiedades mecánicas diferentes a las del cuarzo, la onda acústica modifica su velocidad y amplitud. Debido al acoplamiento electromecánico del cuarzo, los cambios en las propiedades mecánicas del material por el que se propaga la onda se traducen en cambios en las propiedades eléctricas del resonador, los cuales pueden ser medidos en términos de variaciones en el espectro de impedancia eléctrica.

Como se ha mencionado, el uso del sensor de cuarzo como microbalanza es una de las aplicaciones más antiguas que se le pueden atribuir. La gran sensibilidad de estos sensores lo convierte en un elemento ideal para la monitorización de espesores de película delgada en sistemas de evaporación de metales al vacío; también para la detección de diversos compuestos contaminantes en el ambiente. Para ello se recubre el cristal con sustratos sensibles a determinados gases, la absorción del gas por parte del sustrato se traduce en un aumento de la densidad de masa superficial que, como ya se ha comentado en apartados anteriores, puede ser cuantificada a partir de la frecuencia de resonancia (2). Los sensores acústicos también pueden utilizarse para detectar cambios en las propiedades viscosas y elásticas de los materiales depositados sobre su superficie. En este tipo de aplicaciones, la medida de la frecuencia de resonancia del cristal no es suficiente para caracterizar al sensor, siendo necesaria la medida de otros parámetros eléctricos como su resistencia o incluso su impedancia eléctrica a diferentes frecuencias. Esto es lo que ocurre en aplicaciones como la caracterización de propiedades de aceites o la detección de propiedades físicas y químicas de materiales poliméricos [4]. De entre todas las aplicaciones de los sensores acústicos en medio líquido, una destaca sobre el resto: el empleo del cristal de cuarzo como biosensor. Al recubrir el cristal con un polímero o al modificar su superficie bioquímicamente (por ejemplo pegando antígenos o anticuerpos), se

obtiene una interfaz biológica útil para ser expuesta a complejos biomoleculares en solución acuosa. De esta manera, un biosensor permite la realización de medida de biofluidos in situ como son la detección de reacciones inmunológicas o bioelectroquímicas de enzimas redox, así como la detección de pesticidas y antibióticos en alimentos (zumos de frutas, mieles o agua de consumo humano) [5]. Otra aplicación en alza del cristal de cuarzo como sensor es la microbalanza eletroquímica, muy útil en la caracterización de materiales para baterías o células solares [6].

Los sensores QCM tradicionales trabajan a frecuencias de resonancia entre 5 y 10 MHz, frecuencias que corresponden a espesores del cristal entre 332,8μm y 166,4μm (ver Ec.(1)). La sensibilidad teórica del dispositivo, es decir, la variación de frecuencia que es capaz de medir el sensor frente a una variación en la masa de la capa depositada, queda fijada por la frecuencia de resonancia fundamental del cristal de cuarzo. Aumentar esa sensibilidad es uno de los objetivos fundamentales que se ha perseguido históricamente para satisfacer la demanda que presenta el mercado de niveles de detección más pequeños [7]. La solución a ese reto es el uso de los sensores HFF-QCM, los cuales se basan en el mismo principio físico que los QCM pero con frecuencias de resonancia fundamentales superiores, entre 50 y 150MHz. El aumento de la frecuencia fundamental de resonancia de estos dispositivos se consigue disminuyendo el espesor de los cristales, lo que da lugar a la tecnología Inverted Mesa (ver Fig. 3a). En esta tecnología se realiza un "comido" de la parte central del sensor, quedando a su alrededor un marco. El principal problema que se presenta cuando se trabaja a estas frecuencias es la fragilidad del sensor, para 150MHz el espesor del cristal se reduce a unas 11μm. Para hacerlos manejables, AWSensors ha desarrollado unos marcos de fijación realizados en Poly-Phenylene Sulphide (PPS) al que se le hace un agujero cónico que deja accesible la superficie activa del electrodo del oro (ver Fig. 3b).

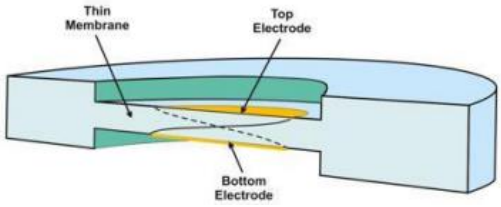


Fig. 3a Tecnología Inverted Mesa

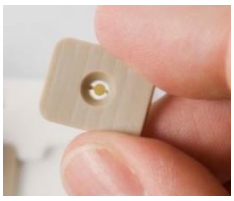


Fig. 3b Sensor HFF-QCM montado sobre el marco de PPS

Las tecnologías de sensores acústicos descritas anteriormente, necesitan celdas de soporte que permitan poner en contacto la superficie del sensor con los diferentes fluidos o materiales a caracterizar y, además, que permitan conectar el sensor al instrumento de caracterización. En la Fig.4 se muestra un detalle de una de las celdas fabricadas por AWSensors.

Fig.4 Celda fabricada por AWSensors

1.2.2 Plataformas AWS A20RP y AWS F20

Respecto a la línea de desarrollo de instrumentación, AWSensors comercializa dos plataformas que pueden trabajar independientemente o conectadas entre sí: AWS A20 y AWS F20 (Ver Fig.5a y Fig.5b, respectivamente). Ambas plataformas se controlan a través del software AWS Suite, el cual complementa el servicio prestado a sus clientes.

Fig.5a Plataforma AWS A20

Fig.5b Plataforma AWS F20

La plataforma AWS A20 RP es una plataforma universal orientada a la investigación, diseñada principalmente para la caracterización de biosensores basados en tecnologías acústicas. Cuenta con 4 canales de medida que soportan señales de frecuencia de hasta 180 MHz, y que permiten la caracterización y seguimiento simultáneo de 4 sensores acústicos de cualquiera de las tecnologías comercializadas por AWSensors (QCM, HFF-QCM y SAW-LOVE).

La plataforma AWS A20 RP permite el control remoto y el seguimiento de los experimentos a través de un ordenador utilizando el software AWS Suite. La plataforma AWS A20 RP incorpora internamente: 1) un subsistema de termostatización, el cual permite que los experimentos puedan realizarse a una temperatura controlada, 2) un subsistema de control remoto y seguimiento a través de Internet y 3) un sistema electrónico de caracterización del sensor que permite la medida de los parámetros eléctricos del mismo.

La plataforma AWS A20 RP tiene varios modos de funcionamiento, los cuales son seleccionables en función del experimento que se vaya a realizar: Modo Sweep, Modo High Resolution y Modo Tracking.

En el **modo Sweep** (Ver Fig.6) la plataforma trabaja de forma similar a como lo hace un analizador de impedancias. En este modo el usuario puede visualizar en un determinado rango de frecuencias, la impedancia del sensor.

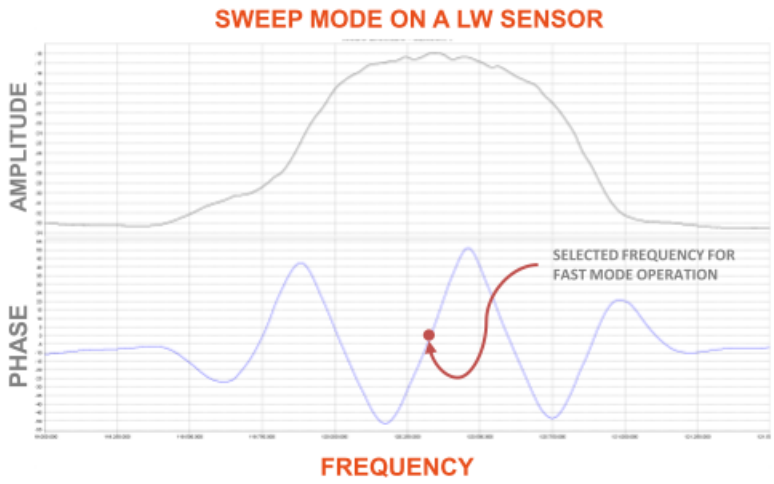


Fig.6 Registro adquirido por el software AWS suite para un sensor SAW-LOVE con la plataforma trabajando en modo Sweep.

Para aplicaciones donde se esperan obtener variaciones muy pequeñas en la respuesta del sensor, y en las que se necesite una gran resolución en las medidas, este es el caso de aplicaciones con biosensores, se utiliza el modo **High Resolution** (Ver Fig.7). En este modo de operación se selecciona y se fija una frecuencia de excitación óptima del sensor. Una vez fijada esta frecuencia de excitación el sistema monitoriza los cambios en la fase y en la amplitud de la respuesta eléctrica del sensor en función del tiempo. Cambios que son debidos a alteraciones en las propiedades físicas del material que se desea caracterizar. Cuando estas alteraciones en la propiedades físicas del material se traducen en variaciones grandes en la respuesta del sensor, el modo High Resolution no es adecuado, en estos casos es conveniente trabajar en el modo **Tracking**. En este modo de operación, la plataforma proporciona de forma continua a lo largo del tiempo datos de la frecuencia a la que el sensor presenta la máxima conductancia, así como la amortiguación de la respuesta eléctrica a dicha frecuencia.

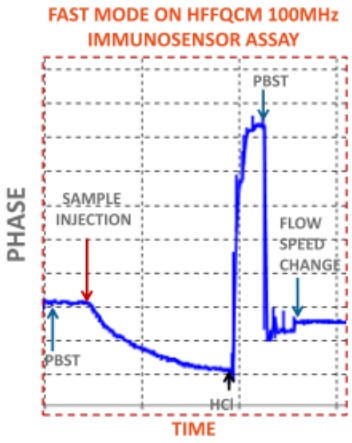


Fig.7 Detalle de un registro adquirido por el software AWS Suite para un sensor HFF-QCM con la plataforma trabajando en modo High Resolution y con una frecuencia de excitación fija de 100 MHz.

La plataforma AWS F20 RP es una plataforma completamente integrada con la AWS A20, que controla el flujo de fluidos sobre el sensor. Dispone de 4 canales, lo que implica poder realizar 4 experimentos de manera simultánea en medio fluido. Cada canal está constituido por una jeringa y una válvula de distribución para establecer el flujo continuo, una bomba solenoidal para cargar la muestra y una válvula de inyección para inyectarla en el canal de flujo principal (Ver Fig.8). En el sistema de flujo mostrado en la Fig.8 se ha implementado una versión de montaje de sistema de flujo muy utilizado en experimentos con biosensensores. En este tipo de experimentos se pasa un flujo continuo por la superficie del sensor (circuito azul) del buffer en el que se produce el reconocimiento biológioco, y en instantes controlados se inyecta la muestra que se desea analizar a través del circuito dibujado en rojo. El software AWS suite permite controlar cada canal de flujo de forma sincronizada con las medidas de la AWS A20.

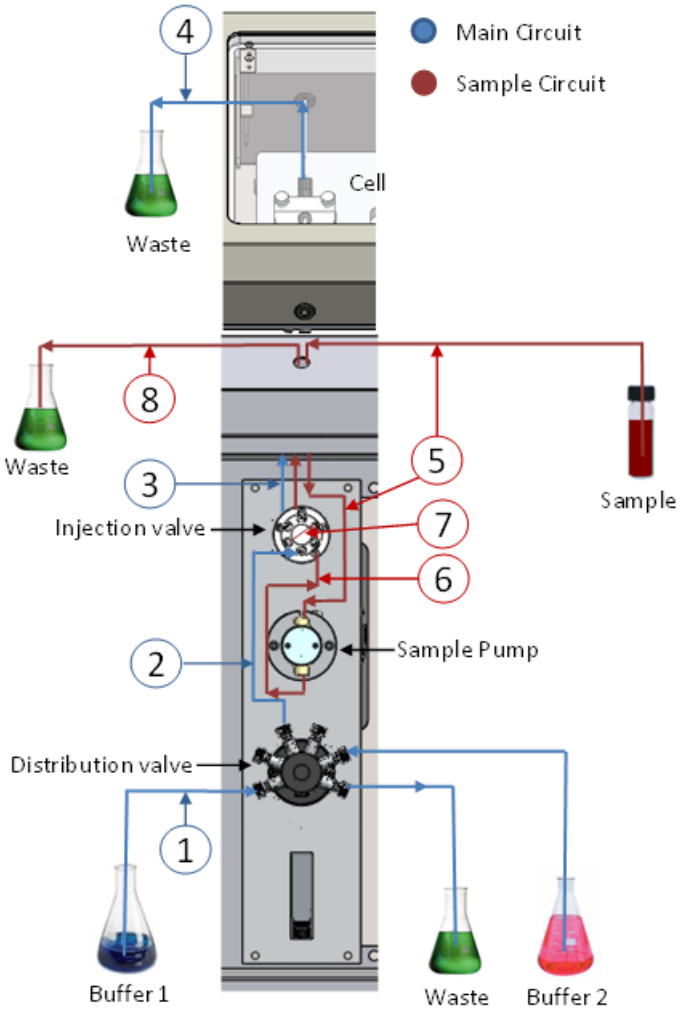


Fig.8 Esquema del circuito principal (azul) y del circuito de muestra (rojo) en la plataforma AWS F20.

A modo de ejemplo, y para entender la necesidad de un sistema de flujo continuo como el que proporciona la AWS F20 RP, en la Fig.9 se muestra el ejemplo de un ciclo de ensayo típico de una aplicación biosensora para la detección del pesticida Carbaryl en zumos de frutas. El esquema de conexiones de la fluídica es el que se muestra en la Fig.8. En la Fig.9 se muestra un registro adquirido con la plataforma AWS A20 RP funcionando en modo High Resolution para un sensor HFF-QCM. En el eje de las ordenadas aparece representado el parámetro eléctrico proporcionado por el sistema electrónico de caracterización y en el eje de las abscisas el tiempo. Las diferentes flechas que aparecen en el registro indican los instantes de tiempo en los que se van inyectando diferentes compuestos sobre el dispositivo sensor: Durante un determinado intervalo de tiempo, se hace pasar sobre el sensor el medio líquido en el que se desarrollará la interacción antígeno-anticuerpo (PBST 20µL/min), posteriormente se inyecta la muestra que se desea analizar (sample injection). El decremento observado en la señal indica la presencia del pesticida en el zumo, la magnitud de dicho decremento es un indicativo de la concentración de pesticida en el zumo. Una vez realizada la medida se aplica ácido clorhídrico (HCl) para regenerar el sensor y dejarlo preparado para el siguiente ciclo de medida, tras un periodo de lavado (PBST 250µL/min) se vuelve a pasar el buffer por el sensor [8,9].

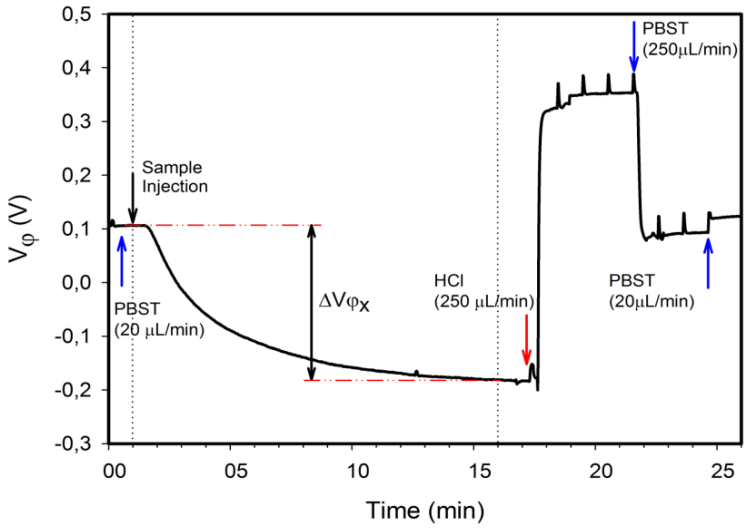


Fig.9 Ejemplo de ciclo de ensayo de una aplicación biosensora para la detección de Carbayl.

Las características que tiene actualmente la plataforma AWS F20 introducen ciertas limitaciones en su uso, por ello, entre las líneas de acción de la compañía AWSensors se encuentra el llevar a cabo la evolución de este producto hacia una versión que se denominará AWS F30. Los principales inconvenientes que presenta la versión actual son los siguientes:

1) El precio del sistema de módulos PSD (jeringa+válvula de distribución) es elevado, por lo que el coste de la plataforma también lo es.
2) Su tamaño es excesivamente elevado, esto genera problemas en la termostatización de la cámara en la que están los fluidos.
3) Con frecuencia aparecen burbujas debido fundamentalmente a tres factores: Cambios térmicos, sistema de bombeo para la carga de la muestra (sample pump en Fig.8) y sistema de conducción del fluido sobre la superficie del sensor mediante la celda que sirve de soporte. Las burbujas distorsionan la medida de la respuesta del sensor.
4) Desplazamientos de la línea base debidos al movimiento de las jeringas.
5) Sistema de control totalmente manual. La mayoría de los experimentos tiene protocolos perfectamente establecidos que si pudieran automatizarse permitiría obtener repeticiones de cada experimento perfectamente sincronizadas (inyecciones de muestra en los mismos instantes, cambios de velocidades perfectamente sincronizados…). Esta automatización introduciría dos grandes ventajas:
    I. El post procesado estadístico y la comparación de los resultados de las repeticiones de un experimento se simplificaría y los resultados serían más valiosos.
    II. Permitiría al usuario estar realizando otras tareas relacionadas con el experimento sin necesidad de estar pendiente de los instantes en los que se debe realizar un cambio de velocidad, inyección de la muestra…
6) Inclusión de un autosampler que no obligue al usuario a estar pendiente del instrumento cada vez que sea necesario realizar un cambio en la muestra.

1.3 Retos de las tecnologías acústicas

Los retos más importantes a los que se enfrentan actualmente las tecnologías basadas en dispositivos acústicos son los siguientes:

1) Detección multianalito mediante el desarrollo de sistemas de caracterización basados en Arrays. Este reto es muy importante para competir con otras técnicas analíticas químicas que permiten realizar múltiples medidas de forma simultánea.
2) Incrementar la fiabilidad de la medida de la respuesta del sensor. Esto se consigue fundamentalmente a través de la mejora del set up experimental para evitar que efectos no deseados (cambios de temperatura, burbujas, vibraciones, cambios en la línea base…) se transfieran a la respuesta del sensor.
3) Mejora de la usabilidad de la instrumentación, fundamentalmente desarrollando aplicaciones software que le confieran más valor añadido. En particular, actualmente existe un interés creciente en el desarrollo de aplicaciones que permitan automatizar el control del sistema de fluídica y en aplicaciones que permitan realizar un post-procesado de la señal registrada para extraer la información física de los medios depositados sobre el sensor.

En este Trabajo Fin de Máster (TFM) se aborda parte de los retos 2) y 3). En particular, se ha trabajado en la mejora del set up asociado al sistema de fluídica y en el desarrollo de una aplicación que permita automatizar el control de dicho sistema. Los objetivos de partida que se han planteado para abordar estos retos se describen en el siguiente apartado.

2. Objetivos del Trabajo Fin De Máster

El objetivo fundamental de este Trabajo Fin de Máster (TFM) es contribuir a la evolución del actual producto AWS F20 hacia el nuevo producto AWSF30. Esta contribución se realizará abordando parte de los retos 2 y 3 descritos en el apartado anterior con una serie de objetivos particulares que se detallan más abajo. La aportación fundamental de este trabajo se ha realizado sobre el reto 3, al que se ha dedicado aproximadamente el 80% del tiempo total de este TFM.

Los objetivos de este trabajo son los siguientes:

1. Desarrollo de una aplicación software que permita a un usuario de la actual plataforma AWS F20 la programación de los experimentos que realice con ella. La aplicación software desarrollada se integrará en el software AWS suite comercializado por la empresa AWSensors.
2. Estudiar la viabilidad de la inclusión de un nuevo sistema de bombeo que permita abaratar el sistema actual, reducir su tamaño y eliminar burbujas y otras señales espurias producidas por las bombas actuales (picos de señal, desplazamientos de la línea base…).

3. Aportaciones realizadas con el Trabajo Fin de Máster

3.1 Desarrollo de una aplicación software para la automatización de experimentos

Para trabajar en la consecución del primer objetivo de este TFM se definieron las siguientes actividades:

1) Puesto que la aplicación desarrollada deberá ser integrada en el software AWS suite comercializado por la compañía, la primera actividad que se realizó consistió en adquirir los conocimientos necesarios sobre la plataforma empleada para el desarrollo del software AWS suite: NetBeans platform, así como sobre la estructura de clases y objetos empleados en AWS Suite.
2) Realización de una propuesta de la filosofía de trabajo de la aplicación diseñada.
3) Definición de la arquitectura de la aplicación: clases, métodos y objetos.
4) Programación de la aplicación.
5) Validación de la aplicación.

En los diferentes puntos que forman este apéndice se irán desarrollando las actividades enumeradas.

3.1.1 Introducción a NetBeans Platform

Como se ha descrito en el epígrafe 2 de este trabajo, el objetivo principal de este TFM es el desarrollo de una aplicación que permita al usuario realizar la programación de experimentos, dicha aplicación se integrará dentro del software AWS suite.

El software AWS suite se ha desarrollado sobre la plataforma NetBeans, por tanto, en este TFM se utilizará la misma. Sobre esta plataforma se desarrollan actualmente gran cantidad de aplicaciones. La plataforma NetBeans es un entorno de desarrollo con una amplia variedad de APIs (Application Programming Interface) que resuelven y facilitan gran cantidad de tareas con las que se encuentran los programadores a la hora de desarrollar aplicaciones que incluyan una interfaz gráfica y representación de resultados. Sobre esta plataforma se construye, entre otras aplicaciones, NetBeans IDE (Integrated Development Environment).

La filosofía de NetBeans Platform es modular (Ver Fig.10). Una gran ventaja de la construcción modular es que permite crear una aplicación conformada por módulos diferentes, en los que se implementan diferentes acciones. En función del rol de la persona que la va a utilizar para programar, solo se cargan en la aplicación los módulos que permiten cumplir con su tarea (Ver Fig.11). Adicionalmente, la plataforma ofrece implementados los mecanismos de descubrimiento de nuevos módulos (y de actualizaciones de los existentes) desde repositorios remotos, resolución de dependencias, activación/desactivación de módulos en caliente, comunicación entre los mismos, etc. Lo cual facilita el desarrollo de la aplicación y permite extender su funcionalidad a posteriori de una forma sencilla.

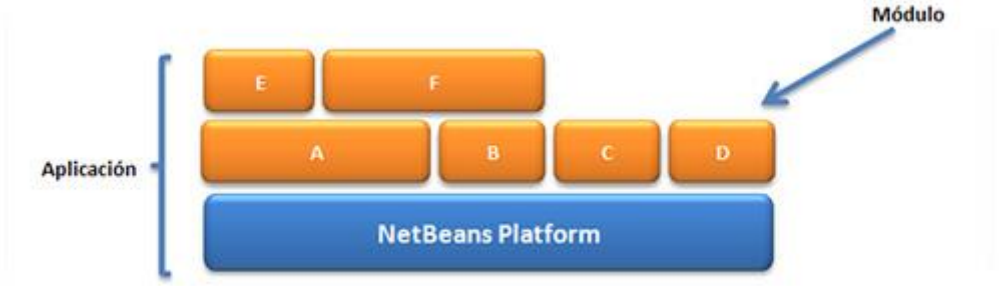


Fig.10 Filosofía modular de NetBeans Platform.

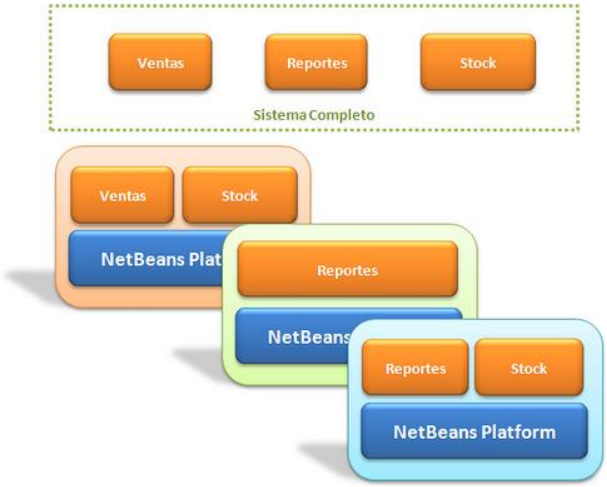


Fig.11 Carga distinta de módulos en función de los requerimientos en NetBeans Platform.

Otras características que hacen interesante la elección de NetBeans Platform como plataforma para el desarrollo de una aplicación son las siguientes:

- Sistema de ventanas práctico para desarrollar las interfaces de usuario.
- Sistema de ventanas práctico para desarrollar las interfaces de usuario (Framework).
- Framework para la creación de asistentes (Wizards).
- Sistema de datos que permite obtener información de diferentes orígenes de datos(FTP, CVS, XML o de una base de datos).
- Su licencia permite construir tanto aplicaciones open source como comerciales.
- Compatibilidad con Java Web Start.
- Soporte completo para desarrollar desde NetBeans IDE, por lo que no necesitaremos otra herramienta adicional para el desarrollo.

3.1.2 Propuesta de diseño del módulo de programación de experimentos

Actualmente, en el software AWS suite el control sobre las acciones que realiza la plataforma AWS F20 se realizan de forma manual, es decir, cada vez que el usuario desea realizar una acción como la carga de una muestra, seleccionar el flow rate que se desea pasar por el sensor, etc. debe realizarlo, acción a acción, a través del panel de control que se muestra en la Fig.12, el cuál controla todas las acciones que puede realizar la plataforma AWS F20. Como consecuencia de ello, el usuario debe estar pendiente exclusivamente de la activación de cada una de las acciones que incluye un experimento, y realizarla a través del panel del control de fluídica del software AWS suite. En un experimento como el de un biosensor, la distancia temporal entre

diferentes eventos es de unos pocos minutos. Adicionalmente, el manejo manual de las acciones sobre la fluídica del experimento dificulta el post-procesado estadístico de los resultados, ya que, las diferentes repeticiones del experimento no están sincronizadas porque las acciones no se realizan exactamente en el mismo instante.

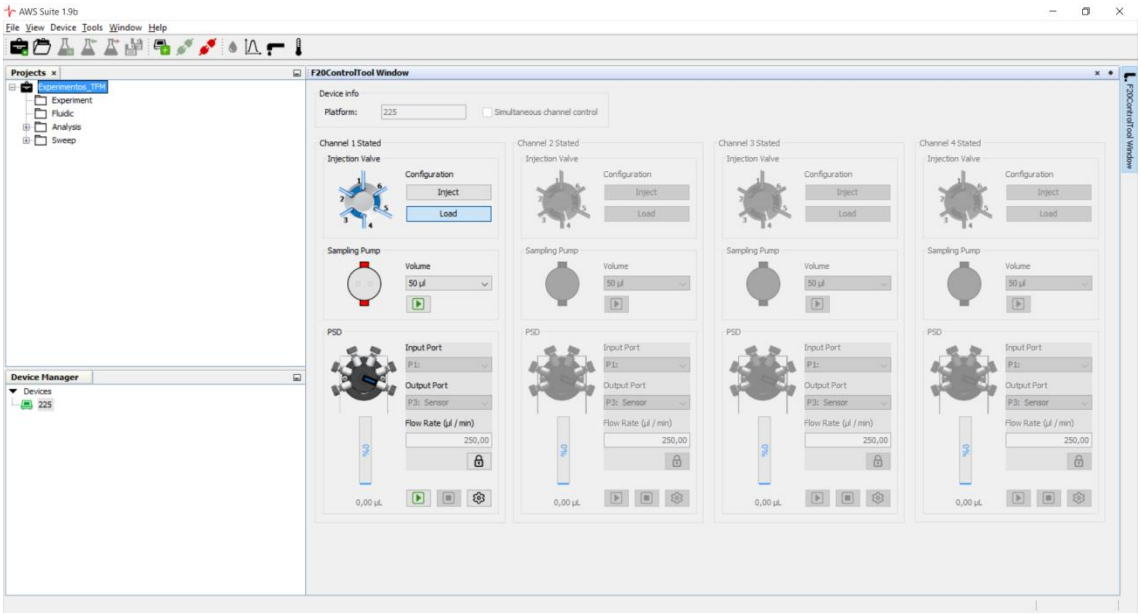


Fig.12 Panel de control de la fluídica de la plataforma F20 dentro del software.

Para resolver los inconvenientes que actualmente presenta el manejo de la plataforma AWS F20, desde AWSensors se propuso el desarrollo de una nueva aplicación dentro del software AWS suite que permita la realización automática de los experimentos.

Inicialmente, se propusieron dos planteamientos para el desarrollo de la aplicación. En el primer planteamiento se utilizan tantos flujos temporales como elementos hay en el sistema de fluídica de la plataforma AWS F20. Este planteamiento presenta el inconveniente de que cualquier cambio en los elementos que forman el sistema de fluídica (eliminación de un elemento, incorporación de uno nuevo…) implicaría tener que realizar cambios sustanciales en el diseño de la interfaz de la aplicación. Sin embargo, este planteamiento ofrece una vista más intuitiva de la evolución de los experimentos con el tiempo. En la Fig.13 se describe la filosofía del primer planteamiento para un sistema de fluídica formado por dos elementos: Una válvula de inyección y una jeringa. Como se muestra en dicha figura, los eventos realizados sobre cada elemento se registran sobre su propio eje temporal. La aplicación mostraría al usuario un gráfico similar al que se muestra en la Fig.13. De este modo, para el ejemplo que en ella se muestra se ha programado la siguiente secuencia: En el instante t_1 se coloca la válvula de inyección en la posición adecuada para cargar la muestra, y en el mismo instante, a una velocidad de 250µl/min, la jeringa hace circular el medio buffer, el cual se carga por el puerto 3 de la válvula de distribución y se dispensa por el puerto 5 de dicha válvula. En el instante t_2, la válvula de inyección pasa a la posición inject y la jeringa, a una velocidad de 20µl/min, carga buffer del puerto 3 de la válvula de distribución y lo dispensa por el puerto 2.

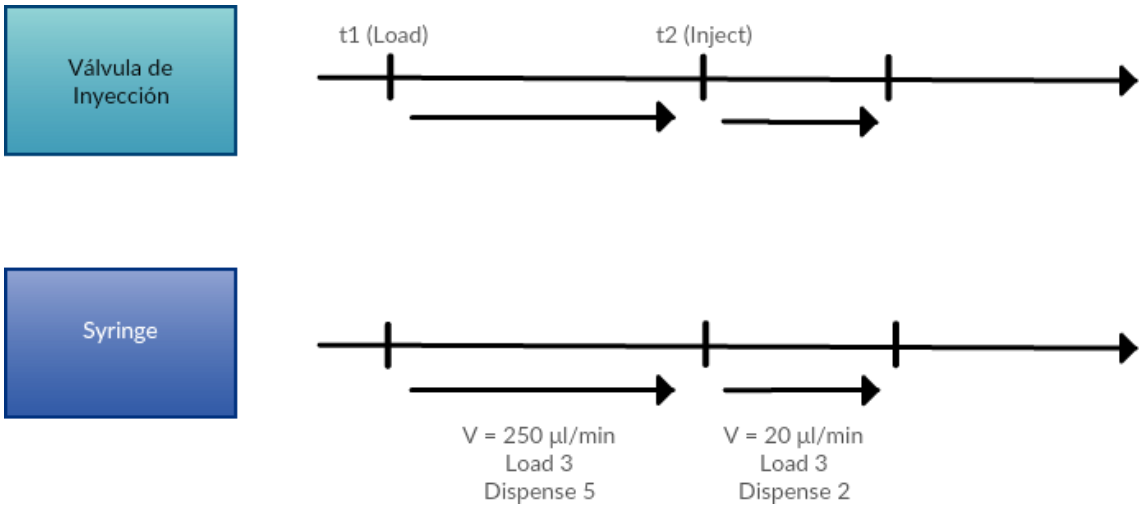


Fig.13 Propuesta de diseño del módulo de programación de experimentos. Flujos temporales.

El segundo planteamiento que se propuso está inspirado en la filosofía de funcionamiento de Labview (Fig.14a). El usuario dispone de una paleta de comandos en la que se incluye una representación gráfica de los diferentes elementos que puede ir agregando para realizar la programación del experimento (Ver Fig.14b). De este modo, el usuario puede arrastrar el bloque correspondiente a la ventana de diseño, y posteriormente interconectarlo con otros bloques (Ver Fig.15) para generar la programación del experimento.

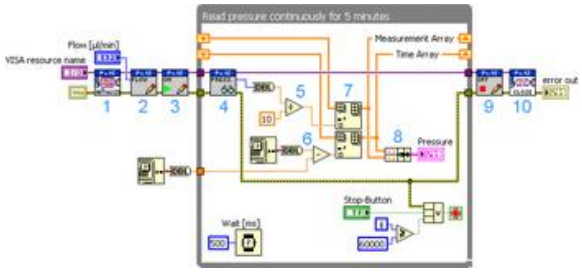


Fig.14a Conexión de bloques en Labview

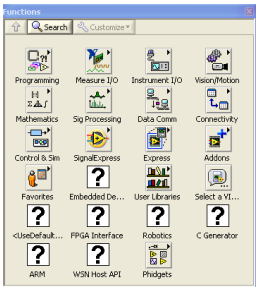


Fig.14b Paleta de comandos en Labview

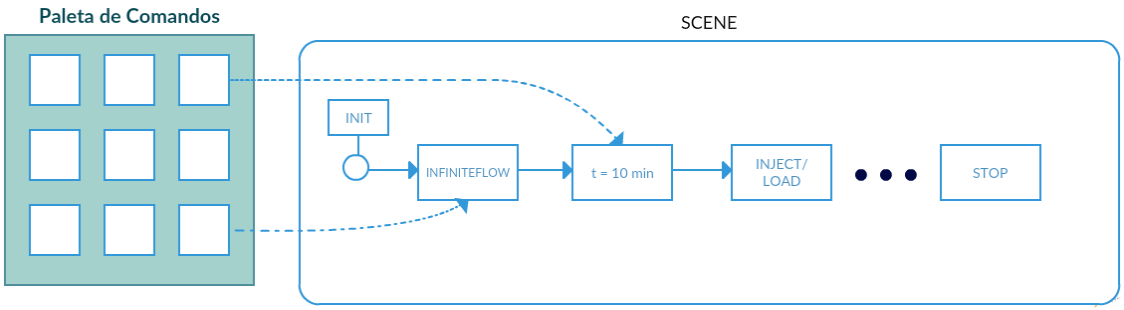


Fig.15 Propuesta de diseño de la aplicación de programación de experimentos. Planteamiento inspirado en Labview.

En este segundo planteamiento sólo se tiene un eje temporal. Con este tipo de representación la visualización del instante en el que se produce cada evento en relación con otro no es tan directa como en el planteamiento anterior, sin embargo, tiene la ventaja de que cualquier cambio en los elementos de la fluídica (eliminación o incorporación de uno nuevo) sólo repercute en la eliminación o adición de un bloque en la paleta, permaneciendo inalterado el resto del interfaz.

Puesto que parte de la evolución de la plataforma AWS F20 incluye la búsqueda de nuevos elementos a incorporar al sistema de fluídica, y la retirada de aquéllos que generan más problemas, se decidió seguir el planteamiento 2, y dejar como futura línea de este trabajo una representación gráfica (sin posibilidad de interactuar con ella) del experimento en varios ejes temporales, esto le proporcionaría al usuario información más clara de cuándo se producen los diferentes eventos en el experimento. Además, las mismas representaciones podrían utilizarse para imprimir información del estado de los elementos del sistema de fluídica sobre la gráfica de resultados.

Una vez fijado el planteamiento se definieron los elementos (y sus propiedades) de la paleta de comandos. Los elementos de la paleta deben ser fundamentalmente los mismos que aparecen en el panel del control del software AWS suite (ver Fig.12), con algunos elementos añadidos que permitan mejorar la funcionalidad de la aplicación desarrollada. A continuación se enumeran los elementos definidos en la paleta (Ver Fig.16):

1. Start experiment: Elemento que activa el inicio del experimento programado
2. Injection Valve: Representa la válvula de inyección. Su función es cagar la muestra, que se encuentra en un reservorio, en el circuito principal de flujo (circuito azul en Fig.8). Tiene dos posiciones: Load/Inject, para cada una de estas dos posiciones se define un elemento en la paleta.
3. Sampling Pump: Esta bomba es la encargada de succionar la muestra almacenada en el reservorio. En su atributo *volumen* se define el volumen de muestra a cargar, debe definirse en múltiplos de 50µl con un tope de 1ml.
4. Wait: Este elemento define un tiempo de espera entre dos eventos. Su atributo *delay* es donde se define dicho tiempo.
5. Flow: Este elemento mantiene un flujo continuo entre dos puntos del sistema de fluídica a una velocidad definida por el usuario, sus atributos son:
    a. Flow Rate: Velocidad del flujo
    b. Load Port: Puerto de entrada de la válvula de distribución (por él se produce la carga de un líquido en el circuito de fluídica).
    c. Dispense Port: Puerto de salida de la válvula de distribución (por él se produce la dispensación de un líquido en el circuito de fluídica)
    d. Time: Tiempo que se mantiene dicho flujo continuo.
6. Stop experiment: Elemento que marca la finalización del experimento programado

Como funcionalidad añadida se incluyó como especificación del diseño la inclusión de un cuadro que permitiera repetir todos los eventos incluidos en él un número de veces programable por el usuario (elemento *loop*). Este elemento no aparecerá directamente en la paleta al no tratarse de un evento, sino que aparece como herramienta a utilizar por el usuario.

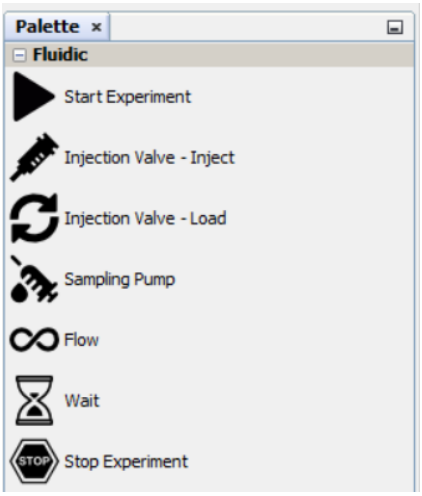


Fig.16 Paleta de la aplicación desarrollada

En la Fig.17 se muestra el aspecto del interfaz de la aplicación desarrollada. En ella se observa la paleta, situada a la derecha de la imagen. En el área central es donde el usuario diseña la programación del experimento, esto lo hace arrastrando los elementos de la paleta que corresponden a los eventos que quiere que se produzcan en su sistema de fluídica. Con la barra de herramientas que aparece en la parte superior derecha de dicha área de diseño puede crear loops, eliminarlos, mover elementos e interconectarlos. A los elementos que se emplazan en la ventana de diseño para programar el experimento se les denominará *nodos* y las flechas que los unen *conexiones*. A lo largo de los sucesivos apartados que forman este epígrafe se describirá la programación llevada a cabo para la implementación de la aplicación y se explicará con mayor detalle el interfaz desarrollado.

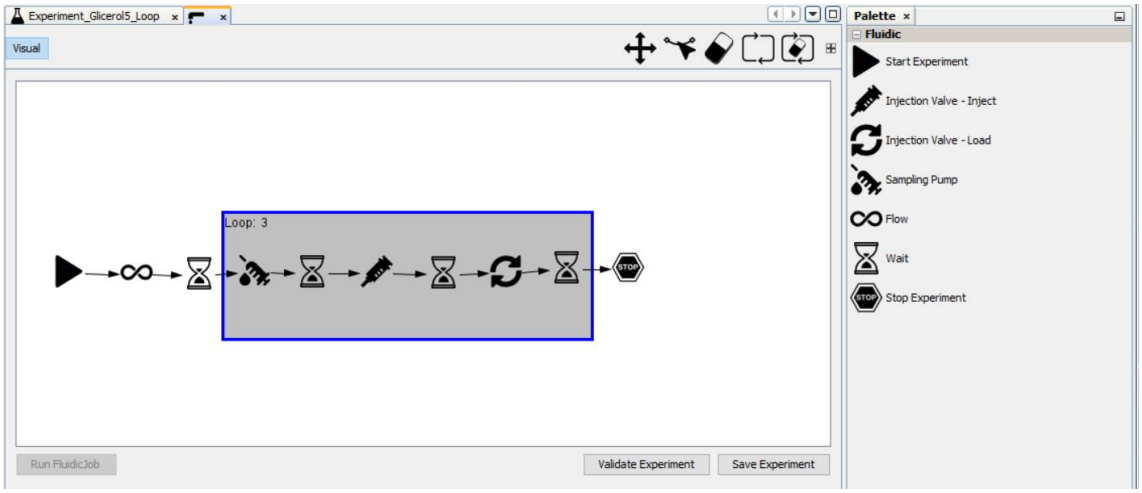


Fig.17 Interfaz de la aplicación desarrollada

3.1.3 Descripción de la programación del módulo

La aplicación desarrollada está formada por dos módulos de NetBeans Platform, denominados **ItemFluidicJob** e **ItemFluidicJobAPI**. NetBeans Platform permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

A continuación se incluye una breve descripción de la función de los módulos **ItemFluidicJob** e **ItemFluidicJobAPI**:

- **ItemFluidicJob**: En este módulo se define tanto el aspecto como la funcionalidad de la aplicación desarrollada. De este modo, en este módulo se programa la interfaz de usuario (GUI), la paleta de comandos, así como las acciones que realizan los elementos contenidos en la paleta o envío de comandos a la plataforma AWS F20. Más adelante se explicará este módulo con más detalle.

- **ItemFluidicJobAPI**: Este módulo es el encargado de comunicarse con el resto de módulos de la aplicación. Con esta filosofía de funcionamiento, cualquier cambio en un método del módulo **ItemFluidicJob** es transparente al resto de la aplicación. De este modo, otro módulo de la aplicación no puede llamar directamente a un método que se encuentre en el módulo **ItemFluidicJob** sino que debe hacerlo a través de **ItemFluidicJobAPI** (Ver Fig.18).

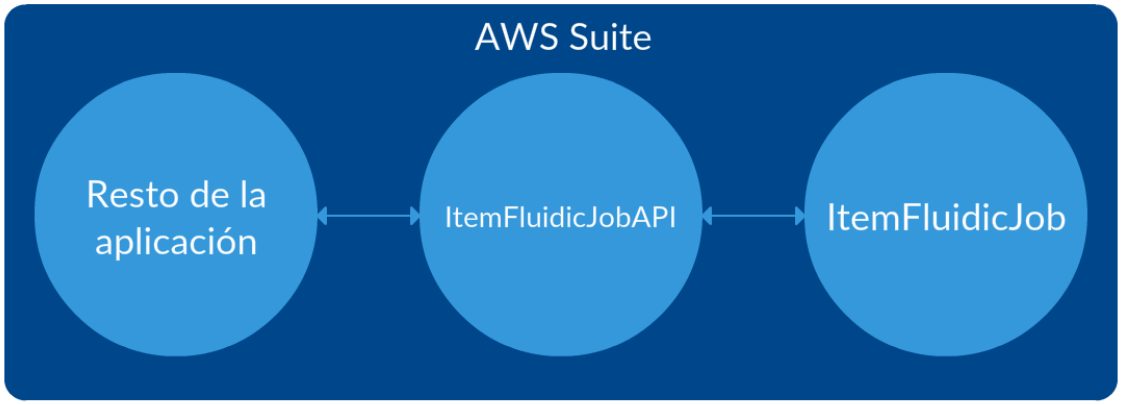


Fig.18 Comunicación entre los módulos desarrollados y el resto de la aplicación

Para el desarrollo del módulo **ItemFluidicJob** se han utilizado diversos APIs de Java, de entre todos los empleados cabe destacar los siguientes:

- Visual Library API: Esta API proporciona un conjunto de elementos visuales (widgets) reutilizables. El objetivo es construir un árbol jerárquico con estos widgets, la raíz de dicho árbol se representa por una clase Scene (que será añadida al TopComponent), la cual contiene todos los datos visuales de la escena. Más adelante se realiza una descripción más detallada de esta API.

- Common Palette API: Esta API proporciona los métodos necesarios para la creación de una paleta personalizada. El contenido de dicha paleta se mostrará en el TopComponent (Scene).

La Visual Library API es la API más importante dentro del módulo **ItemFluidicJob**. Los elementos principales que la componen son los siguientes:

- Widget: Es un elemento visual primitivo, similar a lo que un JComponent es para Swing (biblioteca gráfica para Java). Contiene información sobre su localización, tamaños máximos/mínimos, layout, bordes, fuentes, cursores, tooltips… Es el elemento sobre el cuál se basa prácticamente toda la Visual Library.
- Scene: Como se ha comentado, los componentes de la Visual Library (widgets) se organizan en una estructura de árbol jerárquico. Esto significa que un widget puede contener a otro widget. La clase Scene, que es un widget, representa el contenedor para el resto de elementos, y por lo tanto, es el elemento raíz del árbol jerárquico. Gráficamente, un Scene es representado por una vista, el cual es una instancia de un JComponent. Normalmente la Scene es añadido a un JScrollPane (ver Fig.19). Una aplicación que haga uso de la Visual Library, siempre debe comenzar con la creación de la Scene y posteriormente, dependiendo del propósito de la aplicación, añadir más widgets al árbol.

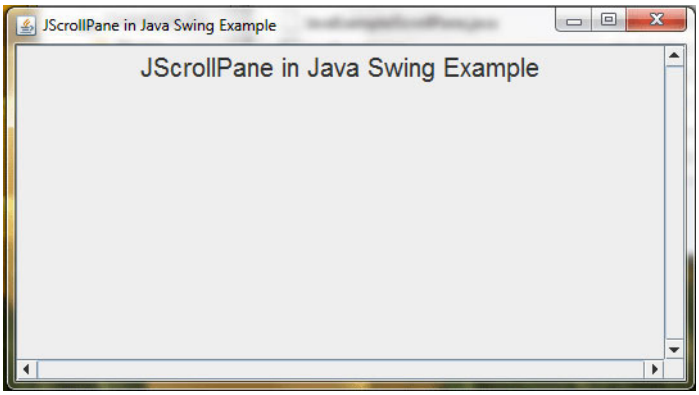


Fig.19 JScrollPane en Java.

- GraphScene: Se trata de una clase abstracta cuya única función es la gestión de los modelos de datos y widgets. Dependiendo del modelo de datos utilizado, los widgets son creados por unas subclases u otras. Los modelos de datos son definidos por *templates* y varían dependiendo del tipo de widget que se desee definir (nodos, conexiones o pines).

- LayerWidget: Este widget representa un panel transparente, similar a lo que un JGlassPane es para Swing. Los LayerWidget son utilizados en el módulo como capas de la Scene. En la clase GraphScene hay definidas 4 LayerWidget:

    - mainLayer: para los widget principales.
    - backgroundLayer: para acciones temporales (por ejemplo, acción que permite la selección de widgets a través de la creación de un rectángulo al arrastrar el ratón)
    - connectionLayer: para la creación de conexiones
    - interactionLayer: para acciones interactivas (por ejemplo, mover un widget de un sitio a otro de la Scene).

A modo de ejemplo, para definir los nodos y las conexiones que aparecen en la programación de un experimento como el que se muestra en la Fig.17, se deben definir dos capas (LayerWidget), una para los nodos y otra para las conexiones, y añadir cada una de ellas a la Scene.

El módulo **ItemFluidicJob** está organizado en paquetes de Java. Un paquete en Java es un contenedor de clases que permite agrupar las distintas partes de un programa en función del papel que desarrollen dichas clases, definiendo la ubicación de estas clases en un directorio de estructura jerárquica. En la Fig.20 se muestran todos los paquetes de Java que conforman el módulo **ItemFluidicJob**. En los siguientes epígrafes se realiza una descripción de las clases que forman cada paquete.

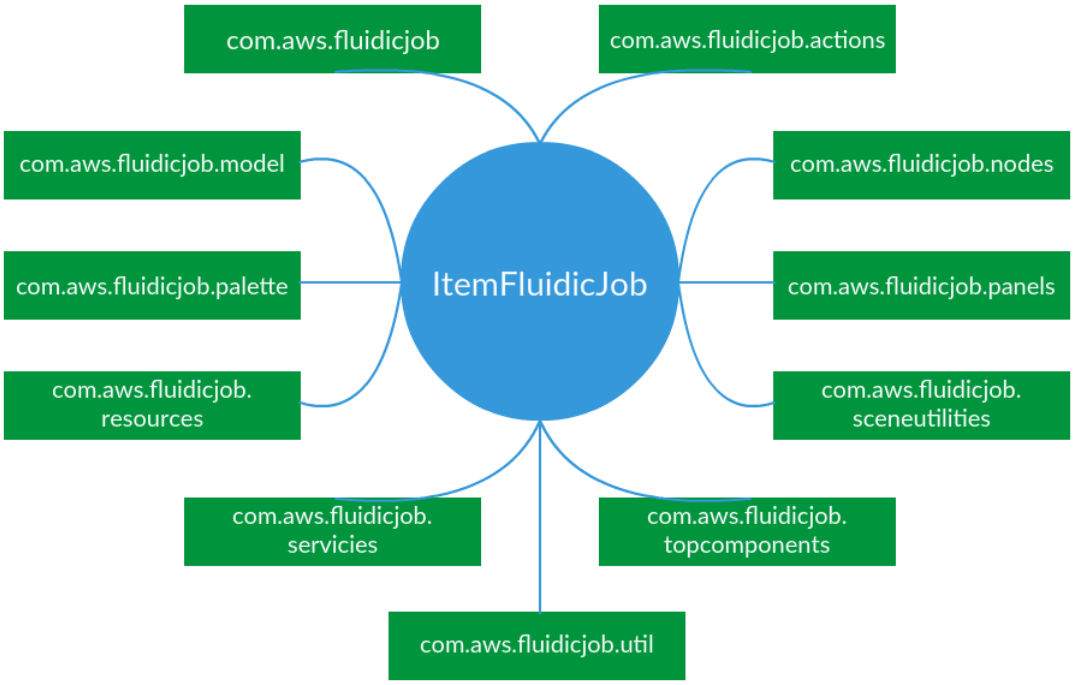


Fig.20 Paquetes Java que conformar el módulo ItemFluidicJob.

3.1.3.1 Paquete com.aws.fluidicjob

El paquete *com.aws.fluidicjob* contiene la clase FluidicDataObject, la cual es la encargada de la creación de un Data Object. Un Data Object es una representación de una estructura de datos, y tiene 3 atributos asociados: 1) un objeto de tipo FluidicJob, 2) el lookup asociado, y 3) un InstanceContent asociado.

Un lookup es un mapa que contiene todos los datos asociados a un data object. Una característica de los lookup es que se pueden registrar los cambios realizados en estos datos, cuando el programador desee. Por último, el InstanceContent se utiliza para poder añadir y quitar datos al Lookup de forma dinámica. Dentro de la clase FluidicDataObject se añaden los datos asociados a los elementos de la paleta de comandos, esto se hace utilizando el atributo InstanceContent.

3.1.3.2 Paquete com.aws.fluidicjob.model

El paquete *com.aws.fluidicjob.model* contiene las clases encargadas de almacenar la información asociada al experimento creado por un usuario (en adelante FluidicJob), e información necesaria para la reconstrucción del mismo a partir de un archivo generado al pulsar el botón de *Save Experiment* que aparece en el interfaz de la aplicación (Ver Fig.17). Las clases de este paquete se clasifican en 4 bloques: 1) clases encargadas de guardar información de los nodos, 2) clase encargada de guardar información de las conexiones, 3) clase encargada de guardar información de los loops (bucles), y 4) clases que contienen los atributos utilizados para reconstruir la Scene y envío de comandos a la plataforma para lanzar experimentos.

Clases encargadas de guardar la información de los nodos

En la Fig.21 se muestran todos los elementos que un usuario tiene disponibles en la paleta, para cada uno de ellos se indica el nombre de la clase que tiene asociada. Las clases encargadas de guardar la información de los elementos son: AwsNode, AwsInfiniteFlowNode, AwsInjectNode, AwsLoadNode, AwsSamplingPumpNode, AwsStartNode, AwsStopNode y AwsWaitingTimeNode. Cada clase está asociada a un elemento de la paleta de comandos. Por ejemplo, si se arrastra un elemento de *Start Experiement* a la ventana de diseño del experimento

(Scene), se creará un objeto de la clase AwsStartNode, y se tendrá el primer nodo de un experimento programado (FluidicJob).

La única clase que no está asociada a ningún elemento de la paleta de comandos es la clase AwsNode, se trata de la "clase padre", el resto de clases son "clases hijas" que descienden de la clase AwsNode. Las "clases hijas" poseerán los atributos de la "clase padre", y podrán incluir otros nuevos dependiendo de cuál sea su función. En Java esta propiedad se denomina herencia. La herencia es una propiedad esencial de la programación orientada a objetos, que consiste en la creación de nuevas clases a partir de otras ya existentes. Cada objeto de la clase AwsNode o que descienda de ella, posee los siguientes atributos:

- id: tipo *String*, contiene la ID del nodo. La ID de cada nodo se define como "Node"+nodeIDCounter, donde nodeIDCounter es un contador que se incrementa cada vez que se añade un nodo a la Scene.
- command: tipo *String*, contiene el comando asociado al nodo creado
- name: tipo *String*, contiene el *label* que el usuario puede añadir al nodo en la Scene.
- icon: tipo *Image*, contiene la imagen asociada al nodo.
- fluidicCommand: este atributo es un objeto de tipo FluidicJobCommand. Los objetos de este tipo se utilizan para gestionar la información que se enviará en forma de comandos a la plataforma. Se explicará más adelante, cuando se describa el módulo encargado del envío de comandos.
- x: tipo *int*, contiene la coordenada X del nodo dentro de la Scene.
- y: tipo *int*, contiene la coordenada Y del nodo dentro de la Scene.

La clase AwsNode es de tipo "clase Serializable". Un programa Java puede convertir un objeto en un conjunto bytes si la clase es "serializable". El objeto convertido en bytes puede ser almacenado en un fichero y posteriormente recuperado. Esta propiedad es muy útil para poder salvar un experimento programado definido por un usuario y posteriormente volver a abrirlo. Las clases descendientes de AWSNode también serán "serializables".

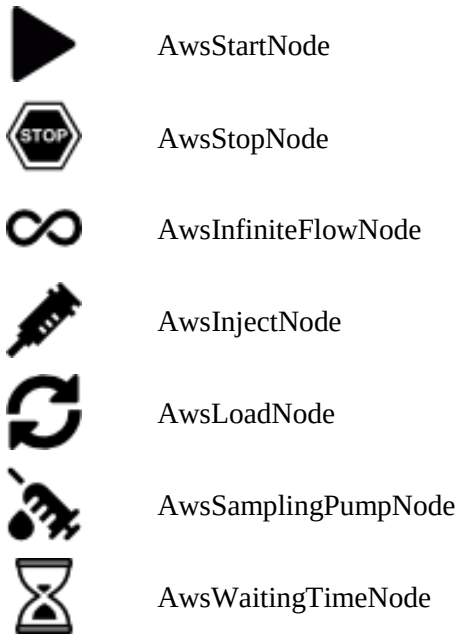


Fig.21 Clases Java que extienden de AwsNode.

La clase AwsInfiniteFlowNode desciende de la clase AwsNode y añade los atributos inPort, outPort y flowRate. El atributo inPort y outPort son del tipo *int* e indican el puerto de entrada y salida a utilizar en la PSD (jeringa+válvula de distribución), respectivamente. El atributo flowRate es de tipo *double* e indica la velocidad del flujo (en µl/min) que tiene que generar la PSD. Por último, esta clase posee el método *getConfig()*. Este método se invoca cuando el usuario hace doble click sobre un nodo de la clase AwsInfiniteFlowNode, abriendo la ventana de propiedades correspondiente a este tipo de nodo. Cuando el usuario introduce las propiedades del nodo y pulsa sobre el botón de *OK*, éstas se guardan en el objeto creado.

La clase AwsSamplingPumpNode desciende de la clase AwsNode y únicamente añade el atributo volume. El atributo volume es del tipo *int* e indica el volumen de líquido o muestra, que se carga por medio de la bomba solenoidal. Esta clase también posee el método *getConfig()*.

La clase AwsWaitingTimeNode desciende de la clase AwsNode y añade los atributos minutes y seconds. Ambos son del tipo *int* e indican el tiempo de espera entre la ejecución de un comando y otro. Esta clase posee el método *getConfig()*.

La clase AwsInjectNode y AwsLoadNode descienden de la clase AwsNode y no añaden ningún atributo adicional. Sí que añade el método *getConfig()* para abrir su correspondiente ventana de propiedades.

Por último, las clases AwsStartNode y AwsStopNode descienden de la clase AwsNode y no añaden ningún atributo adicional. Estas dos clases no poseen el método *getConfig()* ya que no tienen asociada una ventana de propiedades.

A modo de resumen, en la Fig.22 se muestran las clases creadas para almacenar información de los nodos en la Scene, así como los atributos de cada una.

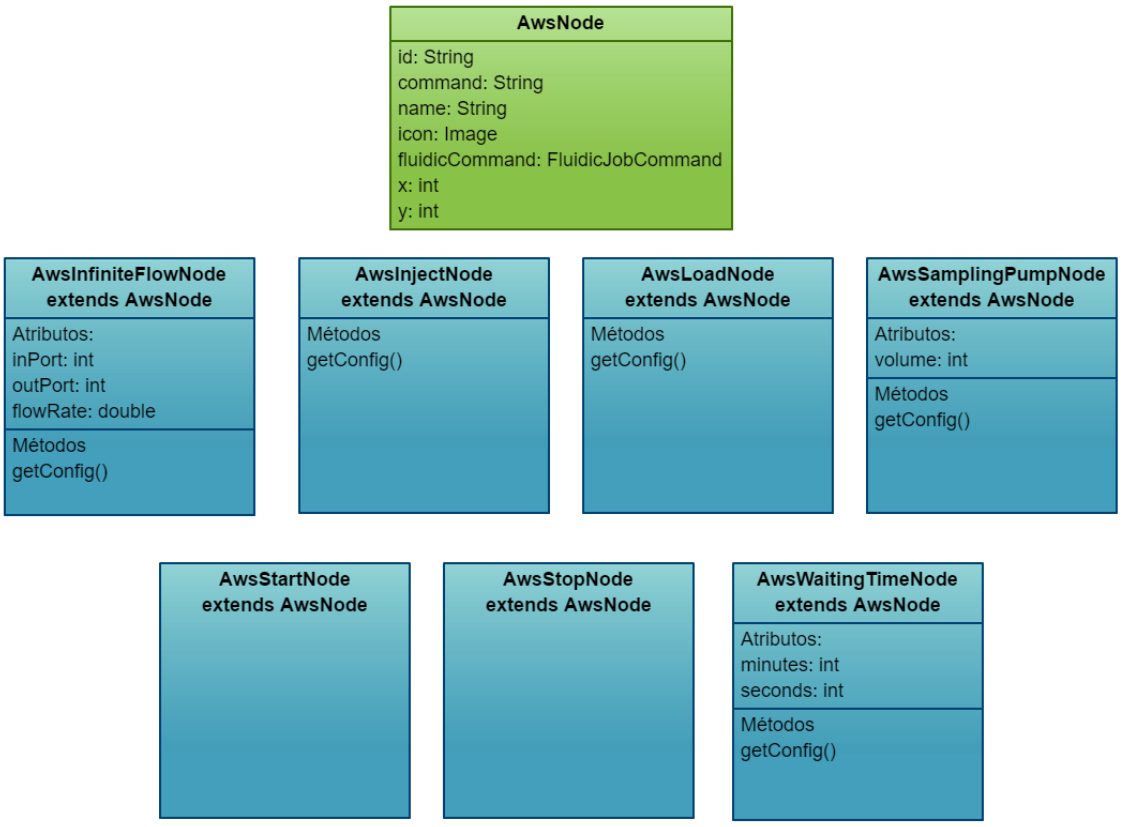


Fig.22 Clases Java encargadas de guardar información de los nodos.

Clase encargada de guardar la información de las conexiones

La clase utilizada para guardar información de las conexiones realizadas entre dos nodos, es la clase AwsConnection. Los objetos de esta clase son creados cada vez que se crea una conexión y poseen 3 atributos: 1) id, 2) sourceID y 3) targetID. El atributo id es de tipo *String* y se define como "Edge"+edgeIDCounter, donde edgeIDCounter es un contador que se va incrementando cada vez que se crea una conexión entre dos nodos. El atributo sourceID es de tipo *String* y contiene la ID del nodo en el que tiene origen la conexión. Por último, el atributo targetID, es de tipo *String* y contiene la ID del nodo en el cual finaliza la conexión. Esta clase es del tipo "clase Serializable", para poder almacenar su información en un fichero. En las Fig.23a y Fig.23b se muestra un ejemplo de dos conexiones entre nodos y el resumen de los atributos de la clase AwsConnection, respectivamente.

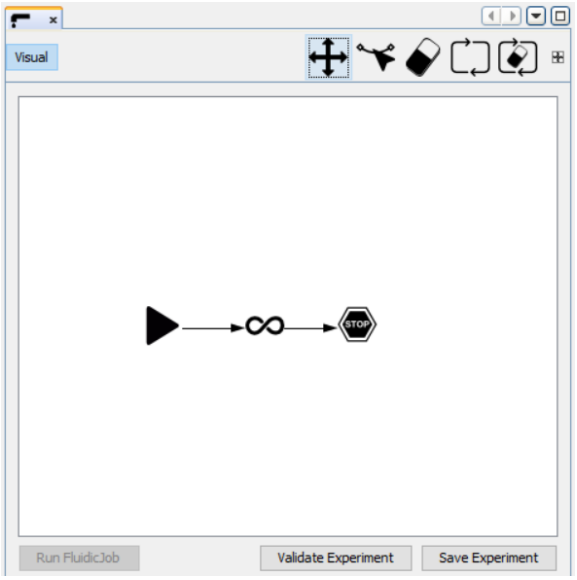


Fig.23a Ejemplo de conexión entre nodos.

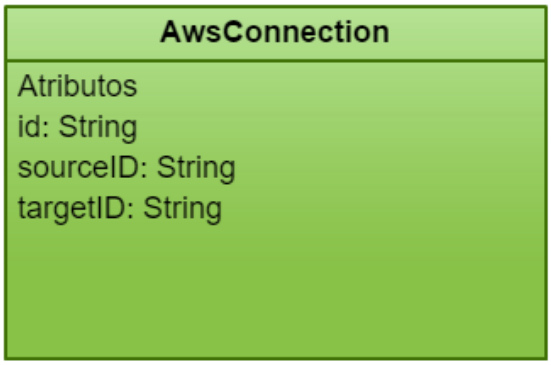
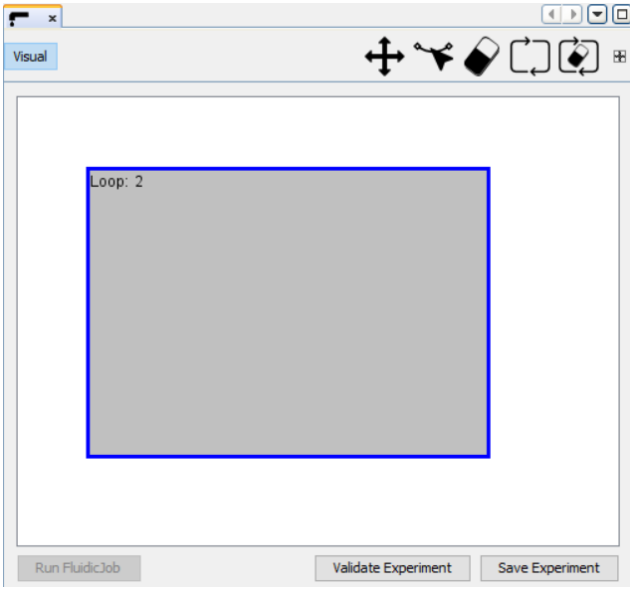


Fig.23b Atributos de la clase AwsConnection

Clase encargada de guardar la información del loop

La clase utilizada para guardar la información de un loop creado en la Scene, es la clase AwsLoop. Un loop permite repetir un conjunto de acciones un número *x* de veces. Gráficamente tiene forma rectangular (Ver Fig.24a). En el ejemplo que se muestra en la Fig.25 las acciones start y flujo continuo se repiten do veces (el número de repeticiones aparece tras la etiqueta *loop:*). Los atributos del loop se muestran en la Fig.24b. Los objetos de esta clase tienen 3 atributos. El atributo numberOfRep (tipo *int*) contiene el número de veces que se repiten las acciones que engloba el loop. Para la reconstrucción del loop, son necesarios dos atributos: initialPoint y finalPoint. Ambos son del tipo *Point* (coordenada x,y) y representan las coordenadas de la esquina superior izquierda y esquina inferior derecha respectivamente. Esta clase es del tipo "clase Serializable", para poder guardar su información en un fichero.



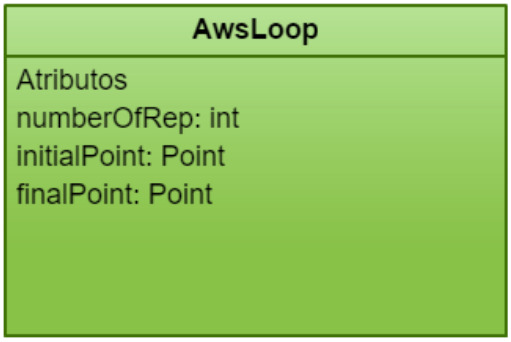


Fig.24a Ejemplo de loop en la Scene.

Fig.24b Atributos de la clase AwsLoop

Clase encargada de la reconstrucción de la Scene y de implementar el envío de comandos

La última clase dentro de este paquete es la clase FluidicJob.java. Esta es una de las clases más importantes del módulo desarrollado por las siguientes razones: 1) Contiene los campos necesarios para la reconstrucción de la Scene, 2) contiene el método *RunFluidicJob()*, encargado de mandar los comandos a la plataforma, y 3) contiene el método *getVisualRepresentation()*, encargado de la creación de la Scene, tanto cuando se crea un nuevo experimento programado como cuando se crea a partir de la información almacenada en un fichero xml. Este método devuelve un objeto de tipo JComponent y su diagrama de flujo se muestra en la Fig.27.

Para realizar la reconstrucción de la Scene, la información que se requiere es el valor de los contadores nodeIDCounter y edgeIDCounter y 3 ArrayList. El primer ArrayList es de tipo AwsNode, y contiene todos los nodos que se encuentran en la Scene en el momento que el usuario decide salvarla. El segundo ArrayList es de tipo AwsConnection, y almacena la información de las conexiones creadas en la Scene en ese momento. Por último, el tercer ArrayList es de tipo AwsLoop y contiene información de los loops creados. La información que hay en cada ArrayList es la descrita anteriormente en las clases AwsNode y sus hijos, AwsConnection y AwsLoop. A modo de ejemplo, se muestra el contenido de un archivo xml (Ver Fig.26) en el que se guarda la información necesaria para la reconstrucción de la scene que aparece en la Fig.25. Los objetos nodes, connections y loop que se muestran en el diagrama de flujo de la Fig.25 son ArrayList de tipo AwsNode, AwsConnection y AwsLoop respectivamente. El método *createView()*) se utiliza cuando se abre la aplicación y se diseña un nuevo experimento programado, o cuando se abre por primera vez un experimento programado cuyos datos están guardados en un fichero .xml. Por lo tanto, únicamente se utilizará el método *getView()*, cuando se cierre un FluidicJob y sin cerrar la aplicación, se vuelva a abrir.

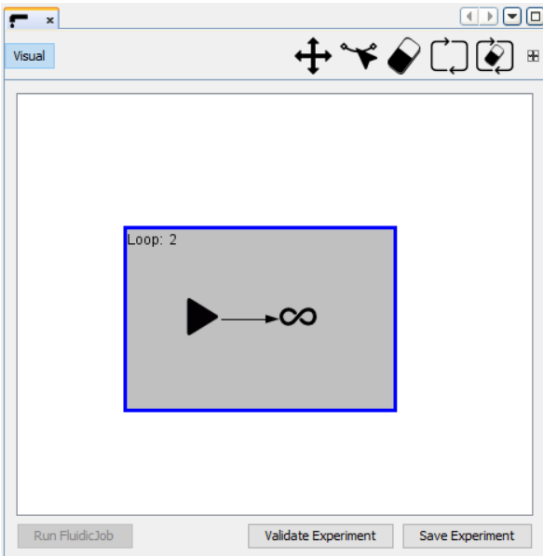


Fig.25 Ejemplo de Scene.

```
<ns2:fluidic xmlns:ns2="fluidic">
  <id>1850743084</id>
  <name>Prueba</name>
  <type>FLUIDIC_JOB</type>
  <nodes>
    <com.aws.fluidicjob.model.AwsStartNode>
      <id>Node1</id>
      <command>START</command>
      <x>143</x>
      <y>175</y>
    </com.aws.fluidicjob.model.AwsStartNode>
    <com.aws.fluidicjob.model.AwsInfiniteFlowNode>
      <id>Node2</id>
      <command>INFINITEFLOW</command>
      <name></name>
      <x>225</x>
      <y>175</y>
      <inPort>1</inPort>
      <outPort>3</outPort>
      <flowRate>250.0</flowRate>
    </com.aws.fluidicjob.model.AwsInfiniteFlowNode>
  </nodes>
  <connections>
    <com.aws.fluidicjob.model.AwsConnection>
      <id>Edge0</id>
      <sourceID>Node1</sourceID>
      <targetID>Node2</targetID>
    </com.aws.fluidicjob.model.AwsConnection>
  </connections>
  <loop>
    <com.aws.fluidicjob.model.AwsLoop>
      <numberOfRep>2</numberOfRep>
      <initialPoint>
        <x>59</x>
        <y>118</y>
      </initialPoint>
      <finalPoint>
        <x>294</x>
        <y>278</y>
      </finalPoint>
    </com.aws.fluidicjob.model.AwsLoop>
  </loop>
  <nodeIDCounter>2</nodeIDCounter>
  <edgeIDCounter>1</edgeIDCounter>
</ns2:fluidic>
```

Fig.26 Información que contiene el fichero .xml.

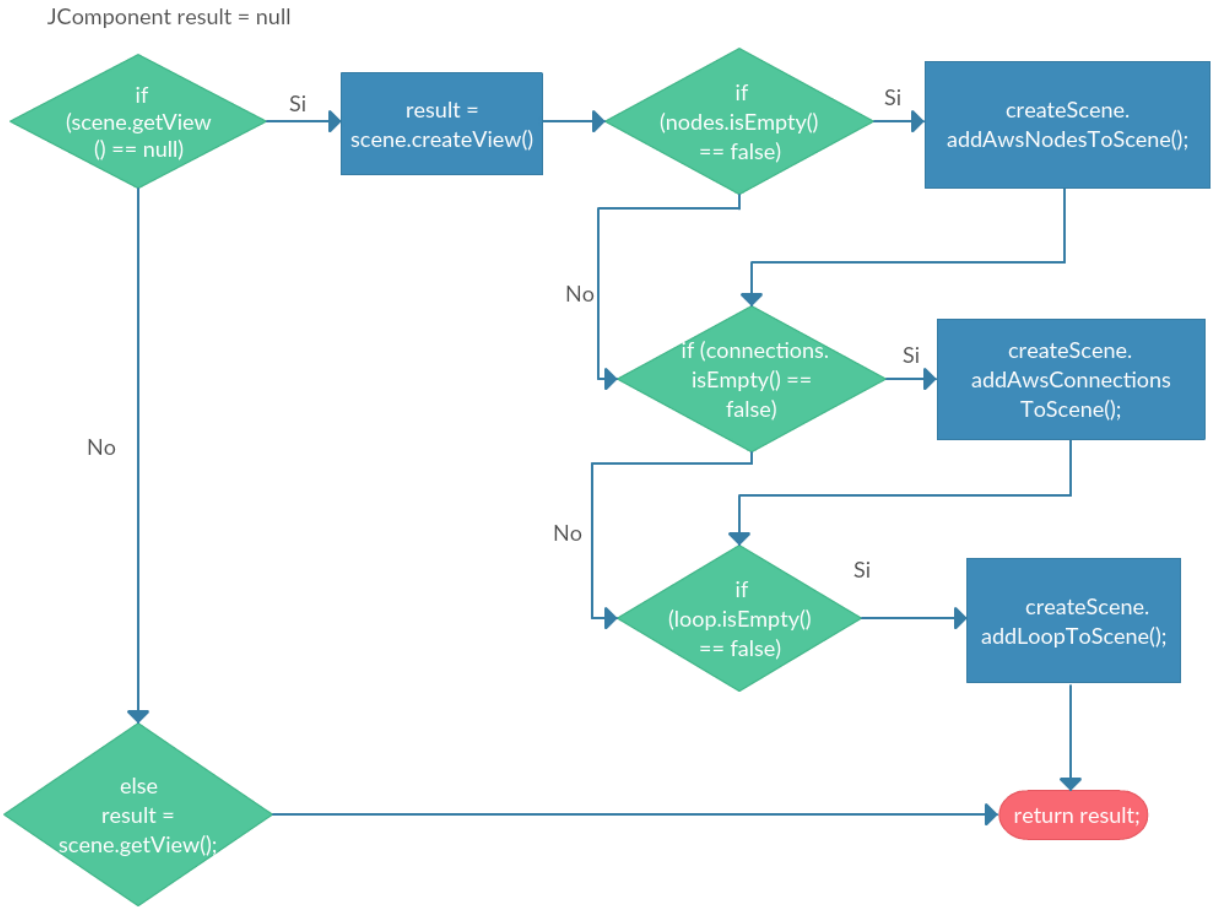


Fig.27 Diagrama de flujo del funcionamiento del método getVisualRepresentation().

El método *RunFluidicJob()* es el encargado de generar las acciones programadas en la aplicación a través de la paleta, y de actuar sobre el firmware que controla la plataforma AWSF20 para activar los elementos de la fluídica que se programen en el experimento (Jeringas, vávulas y bombas). Tiene como parámetros de entrada: un parámetro de tipo *DeviceA20Service* que indica el identificador de la plataforma AWSA20 con la que se van a realizar los experimentos, y un parámetro de tipo *int*, que indica el canal de la plataforma F20 que se va a controlar en el experimento programado. Cuando se ejecuta este método, se invocan dos métodos de la clase SceneUtilities del paquete com.aws.fluidicjob.sceneutilities. El primero obtiene el orden en el que están los nodos conectados y el segundo obtiene el tiempo en el que deben ejecutarse los comandos asociados a dichos nodos (estos nodos y tiempos se almacenan en dos ArrayList). Dentro de este método también se crea un timer (los timers son una herramienta muy útil para lanzar para arrancar acciones en instantes controlados). El timer tiene asociado el método *schedule* que obtiene el tiempo en el cuál debe ejecutarse cada comando (para ello hace uso de la tarea FluidicJobTimerTask que se explicará más adelante).

3.1.3.3 Paquete com.aws.fluidicjob.sceneutilities

Dentro del paquete com.aws.fluidicjob.sceneutilities se encuentran las clases encargadas de crear la Scene, de definir el comportamiento y acciones de los nodos ubicados en ella y de crear menús.

La clase GraphSceneImpl es una de las más importantes dentro del módulo ItemFluidicJob debido a que define las acciones que ocurren en la Scene cuando un elemento es arrastrado hasta

la Scene desde la paleta. En ella también se definen, las propiedades gráficas de los nodos, conexiones, loops…

Como se ha comentado anteriormente, el objeto Scene es el que ocupa la posición de elemento raíz dentro de la estructura de árbol jerárquico. La inicialización del objeto se lleva a cabo en el constructor. En Java, el constructor es un método especial dentro de una clase, quel cuale se llama automáticamente cada vez que se crea un objeto de esa clase. Posee el mismo nombre de la clase a la cual pertenece y no proporciona ningún valor. Por lo tanto, al crear la Scene (objeto de tipo GraphSceneImpl) automáticamente se ejecutará el constructor, en el cual se definen: 1) las capas que contendrá la Scene, 2) las acciones que puede ejecutar el usuario en la Scene, y 3) eventos que se producen cuando un nodo es arrastrado desde la paleta de comandos a la Scene.

Siguiendo la estructura de árbol jerárquico de la Visual Library, lo primero que se realiza en el constructor es añadir las capas como hijas de la Scene (elemento raíz). Para ello utiliza el método *addChild(widget)* el cual añade el elemento widget como hijo de la Scene. El resultado es el árbol jerárquico que se muestra en la Fig.28.

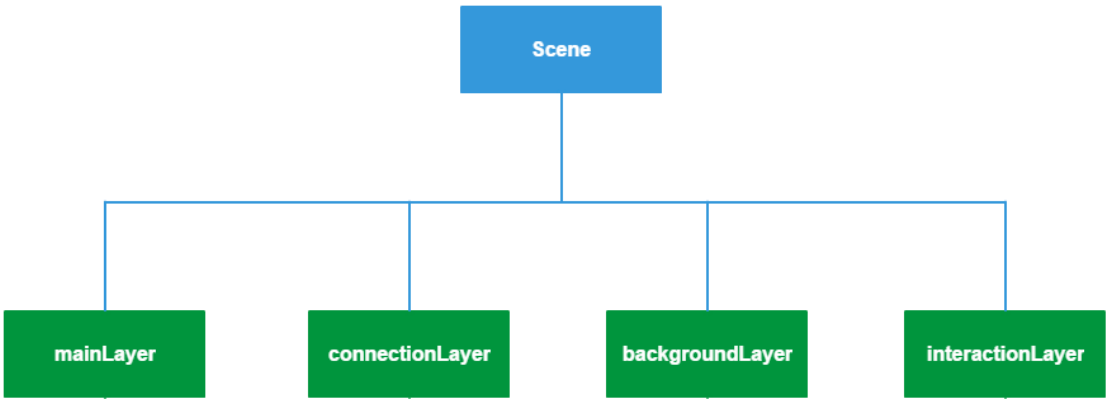


Fig.28 Estructura de árbol jerárquico definido en la clase GraphSceneImpl.

A continuación, se definen las acciones que el usuario puede realizar al interactuar con la Scene o al pulsar los botones definidos en la barra de herramientas de la Scene. Las acciones se obtienen de la factoría ActionFactory. Una factoría es un objeto que maneja la creación de otros objetos. Las factorías se utilizan cuando la creación de un objeto implica algo más que una simple instanciación. Todas las acciones que se han definido en la Scene usan esta factoría o descienden de ella. A continuación se incluye una breve descripción de las mismas:

- moveAction: utiliza el método *ActionFactory.createMoveAction(null, new MultiMoveProvider()). MultiMoveProvider()* es un método definido en esta misma clase, que permite mover varios widgets a la vez, con un único movimiento.
- connectAction: utiliza el método *ActionFactory.createConnectAction(interactionLayer, new SceneConnectProvider())*. SceneConnectProvider() es la clase que permite la creación de las conexiones, las conexiones cradas se añadiraán a la capa interactionLayer.
- loopAction: hace uso del método *ActionFactory.createRectangularSelectAction(this, backgroundLayer)* y a partir de él se crea el objeto AwsLoop.
- zoomAction: usa el método *ActionFactory.createMouseCenteredZoomAction(1.1)*, donde 1.1 es el factor de zoom que se utiliza al realizar dicha acción.
- rectangularSelectAction: La acción de pulsar y arrastrar con el ratón crea un rectángulo de selección.
- sceneMenu: a partir del método ActionFactory.createPopupMenuAction, se crea un menú al hacer click derecho sobre cualquier punto de la Scene.

Por último en el constructor de la clase GraphSceneImpl, se usa la acción *ActionFactory,createAcceptAction()*. Este método permite gestionar las acciones de arrastrar y soltar (Drag and Drop), que se producen cuando se quiere añadir un nodo desde la paleta de comandos a la Scene. Está acción contiene a su vez dos métodos: 1) *AcceptProvider.isAcceptable*, y 2) *AcceptProvider.accept*. El primero se invoca para decidir qué se hace cuando ocurre una acción de arrastrar y soltar, mientras que el segundo se utiliza para indicar qué pasos seguir, cuando la acción de arrastrar y soltar es aceptada. Es decir, si se realiza una acción de arrastrar y soltar, se invocará al método *AcceptProvider.isAcceptable* y si este método se ejecuta sin problemas (es aceptada), invocará al método *AcceptProvider.accept*.

El método *AcceptProvider.isAcceptable* obtiene la imagen del método *getImageFromTransferable()*. Como la Scene y la paleta de comandos son dos elementos distintos, se utiliza este método como "puente" para transferir la imagen del nodo arrastrado desde la paleta a la Scene. Se "dibuja" la imagen en el lugar donde el usuario arrastra el nodo y se llama al método *AcceptProvider.accept()*. Este método obtiene el atributo *command* del nodo que es arrastrado, a través del método *getCommandFromTransferable()*. Este método devuelve un *String*, que es utilizado en un estamento *switch*, para decidir qué tipo de nodo es creado (AwsStartNode, AwsInfiniteFlowNode…) y se invoca al método *addNode()* para añadirlo definitivamente a la Scene. Cada vez que un nodo es creado, se incrementa un contador (nodeIDCounter) en una unidad, para que cada nodo tenga un ID único.

Cuando un nodo es añadido a la Scene utilizando el método *addNode()*, antes se invoca automáticamente al método *attachNodeWidget()*, para obtener un widget con una serie de propiedades, que representará al nodo en la Scene. Estas propiedades pueden ser las acciones que puede realizar, si posee algún menú asociado, detalles gráficos (bordes,opacidad, etc.)…

Del mismo modo, cada vez que una conexión es creada utilizando el método *addEdge()*, se invoca previamente al método *attachEdgeWidget()*, que define las propiedades y las acciones que puede realizar una conexión.

Otros métodos que se encuentran dentro de la clase GraphSceneImpl son: 1) *widgetAction()*, 2) *createLoopShape()*, y 3) *nodeInformation()*.

El método *widgetAction()* determina la acción que el usuario puede realizar en la Scene, en función del botón pulsado en la barra de herramientas. Estas acciones pueden ser mover nodos, crear conexiones o crear loops. Para la creación del loop (método *createLoopShape()*) se parte de la *ActionFactory.createRectangularSelectAction()*. Si el usuario selecciona la herramienta de crear bucle, en el momento que el usuario realiza click con el ratón dentro de la Scene, se registra el punto donde se ha realizado dicho click. Cuando suelta el ratón se vuelve a registrar el punto donde se realiza dicha acción y se crea el bucle (gráficamente). Para poder obtener dichos puntos se hace uso de las clases MouseClicked y MouseReleased. Estas clases detectan cuando el usuario pulsa y suelta el ratón. Dentro del método *createLoopShape()* se definen las acciones que puede realizar el usuario interactuando con el loop(cambiar el número de repeticiones, moverlo y modificar su tamaño), así como las propiedades gráficas de él.

Por último el método *nodeInformation()* recibe información de los paneles de propiedades de los nodos, acerca de si los checkbox de las propiedades están activados, y si es así muestra debajo del nodo en la Scene, las propiedades correspondientes, como se muestra en la Fig.29a y Fig.29b respectivamente.

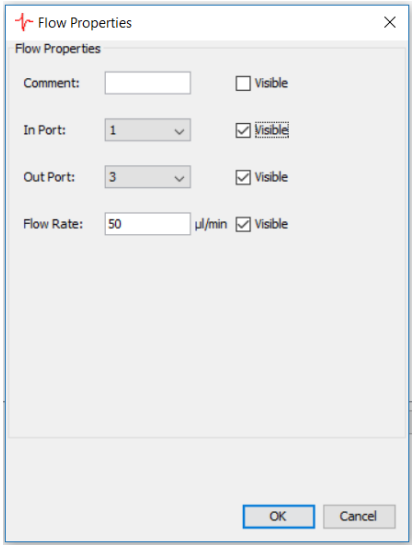


Fig.29a Ventana de propiedades de un nodo AwsInfiniteFlowNode

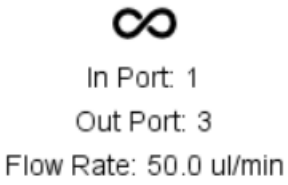


Fig.29b Nodo de tipo AwsInfiniteFlowNode

A modo de resumen, el árbol jerárquico definido en la Fig.28 se completa con el conjunto de acciones realizadas por el constructor quedando de la siguiente manera (Fig.30).

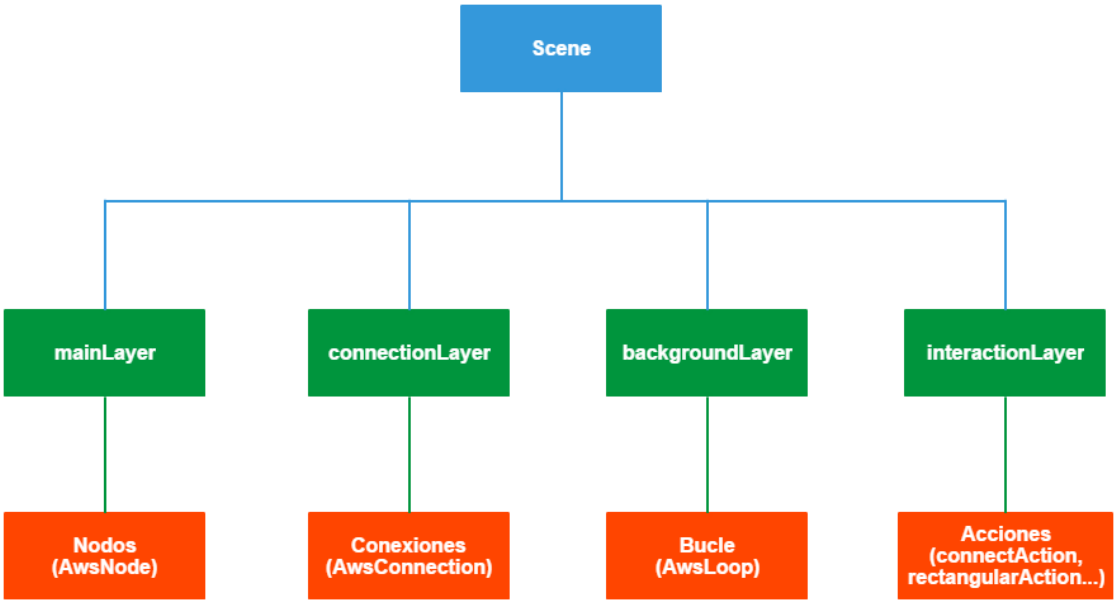


Fig.30 Estructura de árbol jerárquico definido en la clase GraphSceneImpl.

La clase SceneUtilities es otra de las clases más importantes dentro del módulo ItemFluidicJob, ya que los métodos que lo forman recopilan la información necesaria para que los experimentos sean lanzados de forma correcta, recopilan la información que se requiere para guardar el experimento en el archivo .xml y posee métodos que interactúan con la Scene.

El método *getSequence()* guarda en un ArrayList los nodos que hay en la Scene y el orden en que están conectados. Este método es fundamental que cada acción se realice en el instante programado por el usuario. El algoritmo diseñado para obtener esta secuencia se muestra en la Fig.31.

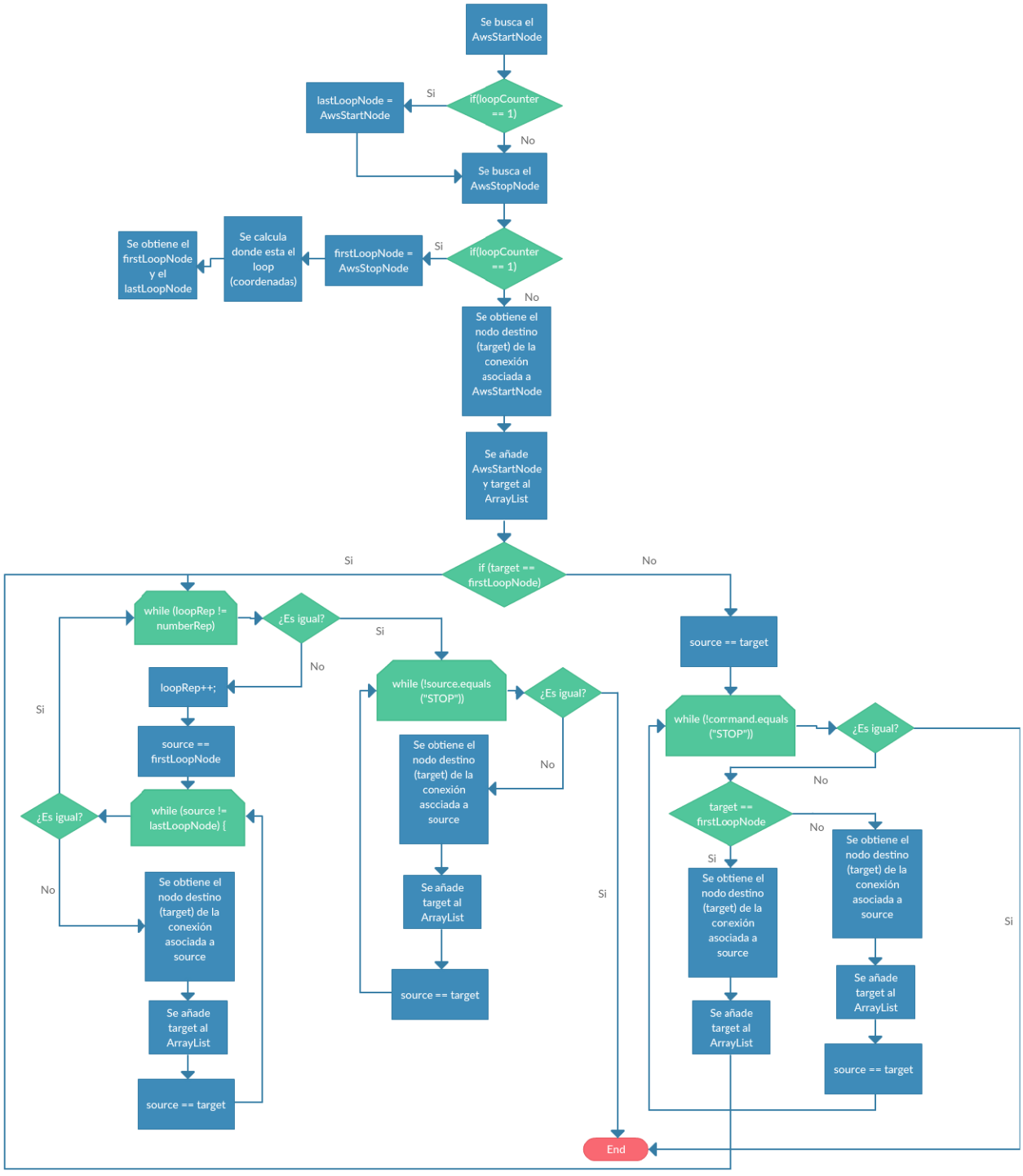


Fig.31 Algoritmo diseñado dentro del método getSequence().

El método *addTimesToArrayList()* recibe uno a uno, en el orden correspondiente, todos los nodos de tipo AwsWaitingTimeNode. Obtiene los atributos minutes y seconds, y convierte todo a milisegundos, ya que el timer utilizado en la clase FluidicJob trabaja en esta unidad. Por último se añade el tiempo obtenido en un ArrayList para su posterior procesado.

El método *getExecutableNodes()* invoca al método *getSequence()* y obtiene un ArrayList con los nodos de la Scene en el orden que tendrán que ejecutarse. Una vez se dispone de este ArrayList, se guarda en otro ArrayList (executableNodes) los nodos que van asociados a comandos. Los nodos de tipo AwsWaitingTimeNode que no van asociados a un comando no se guardan en este ArrayList.

El método *validateExperiment()* realiza una serie de validaciones para comprobar que el experimento generado por el usuario puede lanzarse. Es invocado cuando el usuario pulsa el botón de Validate Experiment, y devuelve una variable *boolean* con valor true si no hay ningún error, y false en el caso contrario. Las validaciones realizadas son las siguientes:

- Debe de haber un nodo de tipo AwsStartNode en la Scene para poder lanzar el experimento.
- Debe de haber un nodo conectado al nodo AwsStartNode, siendo el nodo origen de la conexión AwsStartNode.
- Debe de haber un nodo de tipo AwsStopNode en la Scene para poder lanzar el experimento.
- Debe de haber un nodo conectado al nodo AwsStopNode, siendo el nodo destino de la conexión AwsStopNode.
- Se comprueba el número de nodos y de conexiones en la Scene. Para que el experimento pueda lanzarse, debe de haber un nodo más que conexiones en la Scene. Es decir, si hay 10 nodos en la Scene debe de haber 9 conexiones. En caso contrario, habrá algún nodo sin conectar.
- No puede aparecer ninguna conexión antes del AwsStartNode o después del AwsStopNode.
- En caso de que el usuario haya introducido un bucle en la Scene, el número de repeticiones debe de ser mayor a 0.

Si una de las condiciones anteriores no se cumpliera, el método devolvería un false y se mostrará un mensaje de error explicando el origen del mismo.

Los métodos *getNodes(), getConnections(), getLoop(), getNodeIDCounter()* y *getEdgeIDCounter*(), son invocados en el momento que el usuario pulsa el botón Save Experiment. Estos métodos recopilan y envían a la clase FluidicJob toda la información necesaria para reconstruir la Scene en un futuro.

El método *clearScene()* busca todos los nodos que hay en la Scene en el momento de ejecutar la acción y elimina tanto los nodos como las conexiones asociadas a ellos. Además se reinician todos los contadores que establecen la ID de nodos y conexiones para poder diseñar un nuevo experimento en esa misma Scene.

Por último el método *clearLoop()* elimina el bucle que haya en el Scene, en caso de que hubiese alguno. También se reinicia el contador que indica que hay un bucle en la Scene.

La clase CreateScene es invocada desde el método *getVisualRepresentation()* de la clase FluidicJob y realiza la reconstrucción de la Scene, quedando ésta de la misma manera a cuando fue guardada. Contiene los métodos *addAwsNodesToScene(), addAwsConnectionsToScene()* y *addLoopToScene()*.

El método *addAwsNodesToScene()* recibe la información de los nodos que se encuentran dentro del archivo .xml y crea un nuevo objeto del tipo correspondiente con los parámetros que tenía en su momento (ID, posición en la Scene, nombre…)

El método *addAwsConnectionsToScene()*, reconstruye las conexiones que había en la Scene a partir del ID de los nodos source y target.

Por último, el método *addLoopToScene()*, crea un loop en a partir del punto inicial y final que ocupaba previamente en la Scene.

Las clases WidgetMenu, EdgeMenu y SceneMenu son las encargadas de la creación de los menús que aparecen en la Scene al hacer click derecho sobre conexiones, Scene o nodos respectivamente (Ver Figs. 32a, 32b y 32c).

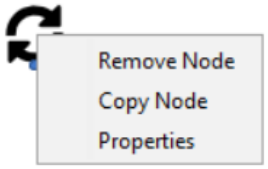


Fig.32a Menú creado por la clase WidgetMenu

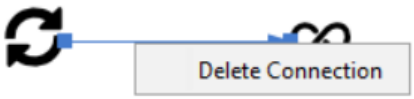


Fig.32b Menú creado por la clase EdgeMenu

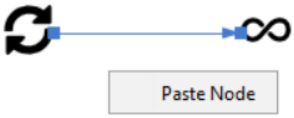


Fig.32c Menú creado por la clase SceneMenu

Estas clases implementan las interfaces PopupMenuProvider y ActionListener. La interfaz PopupMenuProvider devuelve un JPopupMenu y contiene JMenuItems, cada uno con la funcionalidad que el programador quiera. Por otro lado, ActionListener se usa para detectar y manejar eventos de acción, es decir, los que tienen lugar cuando se produce una acción sobre un elemento del programa. Cuando se pulsa sobre un elemento del menú, se invoca al método correspondiente para realizar la acción.

SceneConnectProvider y SceneReconnectProvider son las clases encargadas de la creación de conexiones entre nodos. La clase SceneConnectProvider antes de la creación de una conexión entre dos nodos realiza dos acciones:

1) Comprobar que ambos nodos (source y target) son objetos y de tipo AwsNode.

2) Realizar las siguientes validaciones (en caso de que no se cumplan, la conexión no es creada y saltará un mensaje de aviso):

- Si el nodo destino de la conexión es de tipo AwsStartNode no se crea la conexión, ya que el nodo AwsStartNode indica el comienzo del experimento.
- Si el nodo en el que se origina la conexión es de tipo AwsStopNode, no se crea la conexión, ya que el nodo AwsStopNode indica la finalización del experimento.
- Si uno de los nodos en los cuales el usuario desea realizar una conexión ya tiene una conexión asociada a él, no se permitirá ya que solo debe de haber un flujo temporal en el experimento.

Por otro lado, la clase SceneReconnectProvider se invoca cuando sobre una conexión ya creada, se modifica la fuente o el destino de la conexión (Ver Fig.33)

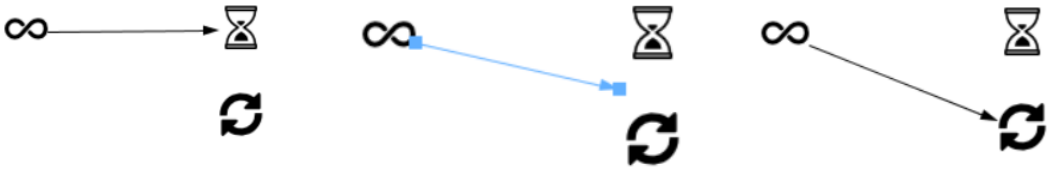


Fig.33 Momento en el que se hace uso de la clase SceneReconnectProvider.

3.1.3.4 Paquete com.aws.fluidicjob.actions

El paquete com.aws.fluidicjob.actions contiene la clase NewFluidicJobAction y define las acciones que se realizan cuando en la barra de herramientas superior (Ver Fig.17), se pulsa el botón utilizado para la creación de un nuevo experimento programado (FluidicJob).

En la Fig. 34a se muestra el panel de la aplicación AWS Suite desarrollada por AWSensors, con una flecha se ha indicado el botón que debe presionar el usuario cuando quiere iniciar la programación de un experimento. Al pulsar dicho botón se arranca la aplicación desarrollada en este TFM y aparece un DialogDescriptor (Fig.34b) que permite al usuario introducir el nombre que desea asignar al experimento programado que está creando. El FluidicJob creado aparece en el árbol de proyectos (Ver Fig.35) de la aplicación AWS suite, dentro de la carpeta y proyecto correspondiente, y se abre la Scene (TopComponent), para que el usuario comience a diseñar su experimento. La clase encargada de crear el FluidicJob (data object) es la clase FluidicFactoryService (paquete com.aws.fluidicjob.services), la cual es invocada por la clase NewFluidicJobAction, que envía el nombre introducido y el proyecto al que pertenece, para crear el data object.

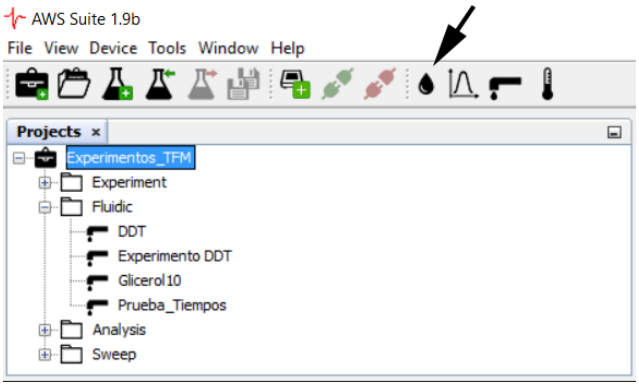


Fig.34a Botón "New Fluidic Job" de la barra superior de herramientas.

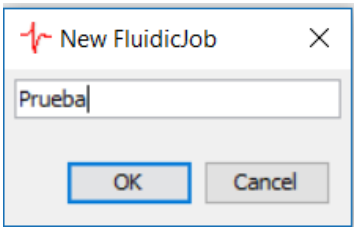


Fig.34b Dialog Descriptor creado para obtener el nombre del FluidicJob

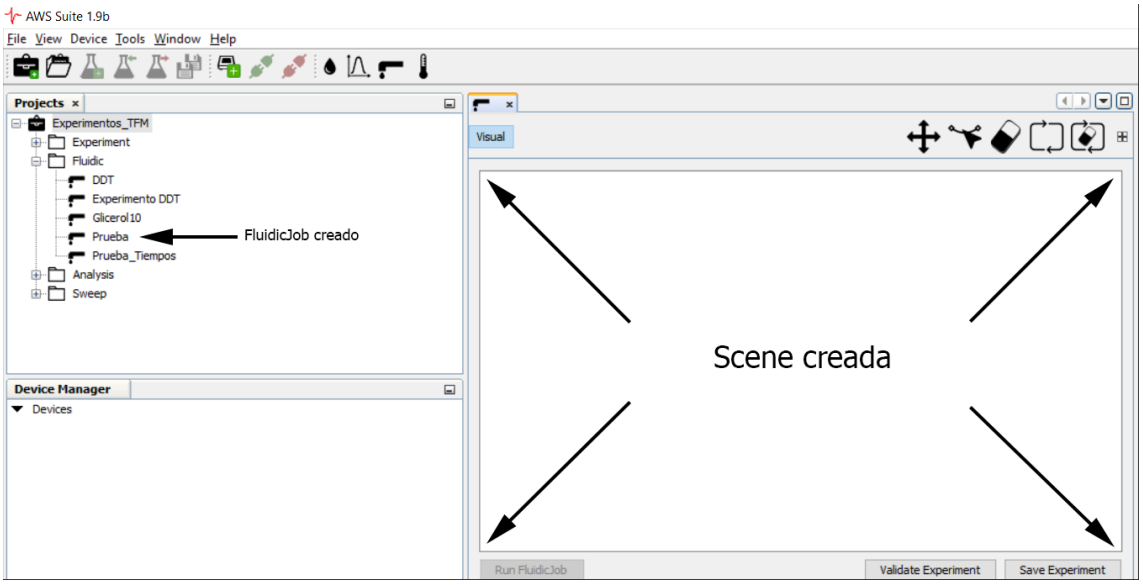


Fig.35 FluidicJob y Scene creados.

3.1.3.5 Paquete com.aws.fluidicjob.nodes

Dentro del paquete com.aws.fluidicjob.nodes, se encuentran las clases FluidicNodeFactory, FluidicFolderNode y FluidicDataNode. Las clases FluidicNodeFactory y FluidicFolderNode se invocan cuando se abre un proyecto dentro del árbol de proyectos, y su función es añadir dentro de dicho proyecto la carpeta *Fluidic* (Ver Fig.36). Por otro lado, la clase FluidicDataNode busca en su lookup si hay algún FluidicJob creado o guardado previamente y si es así lo añade a la carpeta *Fluidic*.

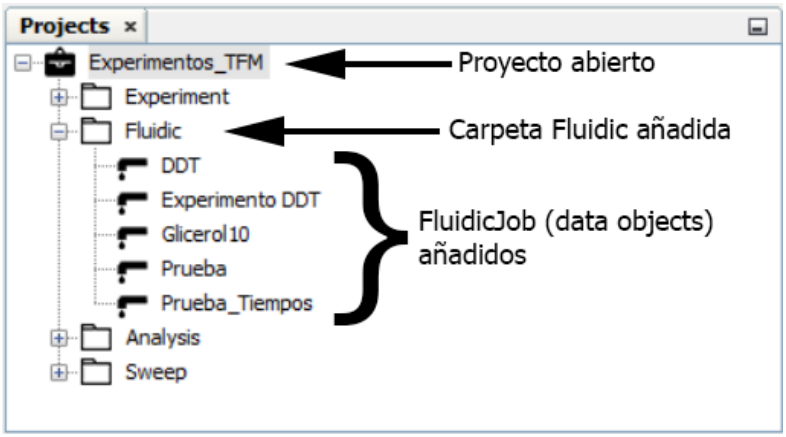


Fig.36 Apertura de un proyecto y FluidicJob asociados.

3.1.3.6 Paquete com.aws.fluidicjob.palette

El paquete com.aws.fluidicjob.palette contiene todas las clases encargadas de la creación de paleta de comandos, de la creación de los nodos contenidos en ella, los atributos de los mismos y la información a transferir cuando un nodo se arrastra a la Scene. Cada elemnto de la paleta está asociado a la ejecución de una acción en el sistema de fluídica de la plataforma F20. Las clases que conforman este paquete son:

- Category: esta clase define las propiedades que posee cada categoría de la paleta. Únicamente tiene la propiedad name(*String*).

- CategoryChildren: esta clase desciende de la clase Children.Keys. Esta clase utiliza un conjunto de "keys", cada una de ellas representaa un nodo (Category). Utiliza el método *addNotify()*. Este método se llama la primera vez que se necesita la lista de categorías, es decir, cuando se llama a *addNotify()* se instancia una nueva categoría. Previamente se ha definido en un Array el nombre de las categorías que compondrán la paleta. Actualmente solo hay una categoría (Fluidic) (Ver Fig.37).

```
//Creación de las categorías
@Override
protected void addNotify() {
    super.addNotify();
    Category[] objs = new Category[Categories.length];
    for (int i = 0; i < objs.length; i++) {
        Category cat = new Category();
        cat.setName(Categories[i]);
        objs[i] = cat;
    }
    setKeys(objs);
}
```

Fig.37 Método addNotify() de la clase CategoryChildren.

- CategoryNode: esta clase recibe las categorías creadas y las muestra en la paleta (Ver Fig.38).

```
public CategoryNode(Category category) {
    super(new AwsShapeChildren(category), Lookups.singleton(category));
    setDisplayName(category.getName());
}
```

Fig.38 Clase CategoryNode.

- AwsShape: esta clase define las propiedades que posee cada elemento de la paleta. Estos atributos son:
    - Category: de tipo *String*, esta propiedad determina la categoría a la que pertenece el nodo.
    - Image: de tipo *String*, esta propiedad establece la ruta de la imagen asociada al nodo.
    - Command: de tipo *String*, esta propiedad determina la acción a realizar por el nodo cuando se genere el experimento.
    - ItemName: de tipo *String*, determina el nombre que aparece a la derecha de cada nodo en la paleta.

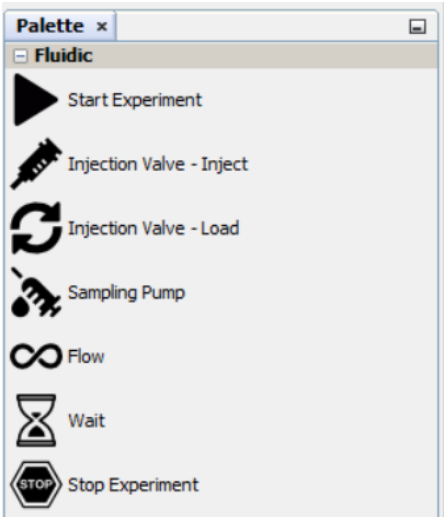


Fig.39 Paleta de comandos generada por las clases del paquete com.aws.fluidicjob.palette.

- AwsShapeChildren: esta clase crea los nodos que pertenecen a la categoría correspondiente. Esta clase desciende de la clase Index.ArrayChildren, se utiliza para definir los nodos y sus propiedades en un matriz (Ver Fig.40). Cada fila de dicha matriz corresponde a un nodo y cada columna a un atributo del nodo. Para ir añadiendo los nodos a la categoría correspondiente se utiliza el método *java.util.List initCollection()* (Ver Fig.41). Este método comprueba el atributo Category de cada nodo de la matriz, crea los objetos de tipo AwsShape, establece los atributos definidos en la matriz y los añade a la categoría correspondiente, haciendo uso del método *childrenNodes.add()*. Finalmente se crean objetos del tipo AwsShapeNode.

```
private final String[][] items = new String[][]{{"Start Experiment", "Fluidic", "com/aws/fluidicjob/resources/start32.png","START"},
                                                 {"Injection Valve - Inject", "Fluidic", "com/aws/fluidicjob/resources/inject32.png","INJECT"},
                                                 {"Injection Valve - Load", "Fluidic", "com/aws/fluidicjob/resources/load32.png","ILOAD"},
                                                 {"Sampling Pump", "Fluidic", "com/aws/fluidicjob/resources/samplingpump32.png","SOLENOIDAL"},
                                                 {"Flow", "Fluidic", "com/aws/fluidicjob/resources/infinite32.png","INFINITEFLOW"},
                                                 {"Wait", "Fluidic", "com/aws/fluidicjob/resources/wait32.png", "WAIT"},
                                                 {"Stop Experiment", "Fluidic", "com/aws/fluidicjob/resources/stop32.png","STOP"}};
```

Fig.40 Matriz contenedora de los atributos de los nodos de la paleta.

```
@Override
protected java.util.List initCollection() {
    ArrayList childrenNodes = new ArrayList(items.length);
    for (int i = 0; i < items.length; i++) {
        if (category.getName().equals(items[i][1])) {
            AwsShape item = new AwsShape();
            item.setItemName(items[i][0]);
            item.setCategory(items[i][1]);
            item.setImage(items[i][2]);
            item.setCommand(items[i][3]);
            childrenNodes.add(new AwsShapeNode(item));
        }
    }
    return childrenNodes;
}
```

Fig.41 Método java.util.List initCollection.

- AwsShapeNode: esta clase desciende de la clase AbstractNode, obtiene de los nodos (la información de los nodos proviene del método *java.util.List.initCollection()* de la clase AwsShapeChildren) su propiedad Image e ItemName y los muestra en la paleta.

```
public AwsShapeNode(AwsShape key) {
    super(Children.LEAF, Lookups.fixed(new Object[]{key}));
    this.shape = key;
    setIconBaseWithExtension(key.getImage());
    setName(key.getItemName());
}
```

Fig.42 Clase AwsShapeNode.

- PaletteSupport: esta clase hace uso de las clases PaletteController y DragAndDropHandler. PaletteController proporciona acceso a datos en la paleta. Si una instancia de esta clase está en el Lookup de cualquier TopComponent, entonces la paleta se abre y muestra su contenido cuando se abre el TopComponent. Por otro lado la clase DragAndDropHandler se utiliza para poder arrastrar y soltar los nodos creados en la paleta en la Scene. Para transferir los datos desde la paleta al Scene se hace uso de los DataFlavor. Un DataFlavor proporciona meta información sobre los datos. Los DataFlavor se suelen utilizar para acceder a los datos en el portapapeles o durante una operación de arrastrar y soltar (Drag and Drop). Los DataFlavor que se han utilizado son un *imageFlavor*, con el que transferimos al Scene la imagen asociada al nodo seleccionado y un *stringFlavor* que transfiere el comando asociado al nodo seleccionado.

```
private static class MyDnDHandler extends DragAndDropHandler {

    public void customize(ExTransferable exTransferable, Lookup lookup) {
        AwsShape shape = lookup.lookup(AwsShape.class);
        final Image image = ImageUtilities.loadImage(shape.getImage());
        final String command = shape.getCommand();
        exTransferable.put(new ExTransferable.Single(DataFlavor.imageFlavor) {
            protected Object getData() throws IOException, UnsupportedFlavorException {
                return image;
            }
        });

        exTransferable.put(new ExTransferable.Single(DataFlavor.stringFlavor) {
            protected Object getData() throws IOException, UnsupportedFlavorException {
                return command;
            }
        });
    }
}
```

Fig.43 Clase DragAndDropHandler.

3.1.3.7 Paquete com.aws.fluidicjob.panels

El paquete com.aws.fluidicjob.panels contiene los paneles o ventanas de propiedades que se abren al hacer doble click sobre ciertos nodos, o al abrir el menú de un nodo y hacer click *Properties* (Ver Fig.44). Los nodos que tienen una ventana de propiedades asociada son AwsInfiniteFlowNode (Fig.45a), AwsInjectNode (Fig.45b), AwsLoadNode (Fig.45b), AwsSamplingPumpNode (Fig.45c) y AwsWaitingTimeNode (Fig.45d).

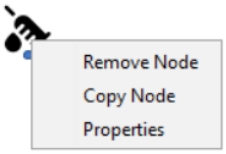


Fig.44 Menú de un nodo de tipo AwsSamplingPumpNode.

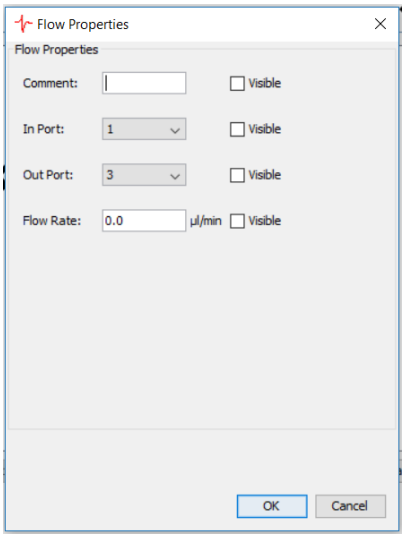


Fig.45a Panel asociado a un nodo de tipo AwsInfiniteFlowNode

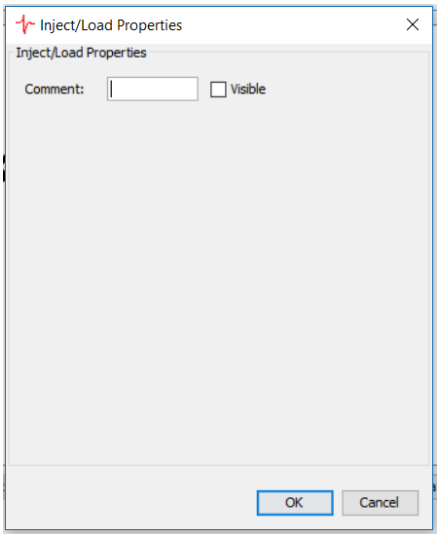


Fig.45b Panel asociado a un nodo de tipo AwsInjectNode/AwsLoadNode

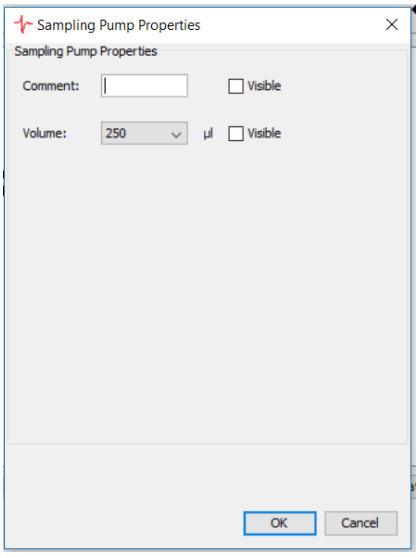


Fig.45c Panel asociado a un nodo de tipo AwsSamplingPumpNode

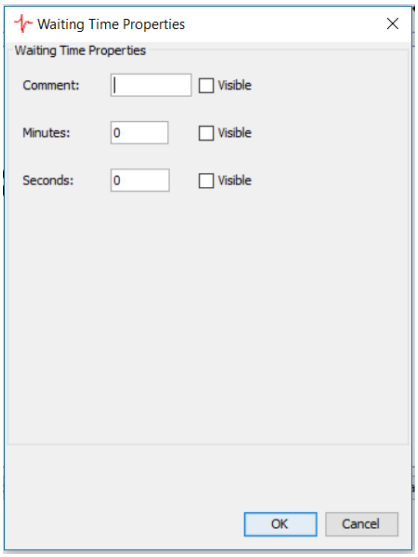


Fig.45d Panel asociado a un nodo de tipo AwsWaitingTimeNode

.

Estos paneles se invocan desde los métodos *getConfig()* de las clases AwsInfiniteFlowNode, AwsInjectNode, AwsLoadNode, AwsSamplingPumpNode, AwsWaitingTimeNode. Dentro de estos paneles, que en realidad son clases, se encuentran los métodos necesarios para guardar la información que se introduce en ellos cuando el usuario pulsa el botón *OK*, y los métodos que recuperan la información almacenada en los nodos (mostrando dicha información en el panel cuando se abre). En algunos paneles, como es el caso de AwsInfiniteFlowProperties, AwsSamplingPumpProperties y AwsWaitingTimeProperties, se realizan unas validaciones con el objetivo de evitar que el usuario introduzca valores o caracteres incorrectos, inhabilitando el botón de *OK* y mostrando el error por pantalla (Ver Fig.46).

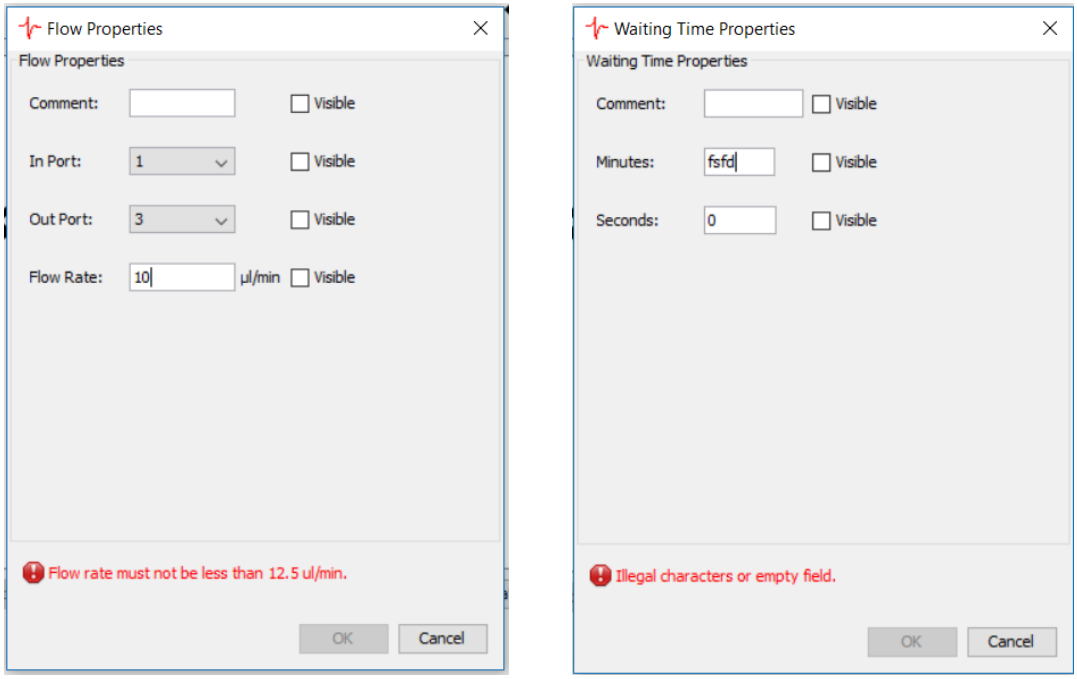


Fig.46 Ejemplo de mensaje de validación en la ventana de propiedades.

3.1.3.8 Paquete com.aws.fluidicjob.topcomponents

El paquete com.aws.fluidicjob.topcomponents contiene la clase FluidicVisualElement que desciende del elemento JPanel. JPanel es considerado un objeto contenedor, ya que sirve para contener a otros objetos como botones, cuadros de texto, otros JPanel… En este caso el JPanel contiene la Scene (que es añadida a un JScrollPane) obtenida al invocar al método *getVisualRepresentation()* de la clase FluidicJob, una barra de herramientas para interactuar con la Scene, y tres botones: 1) RunFluidicJob, 2) ValidateExperiment y 3) SaveExperiment (Ver Fig.47).

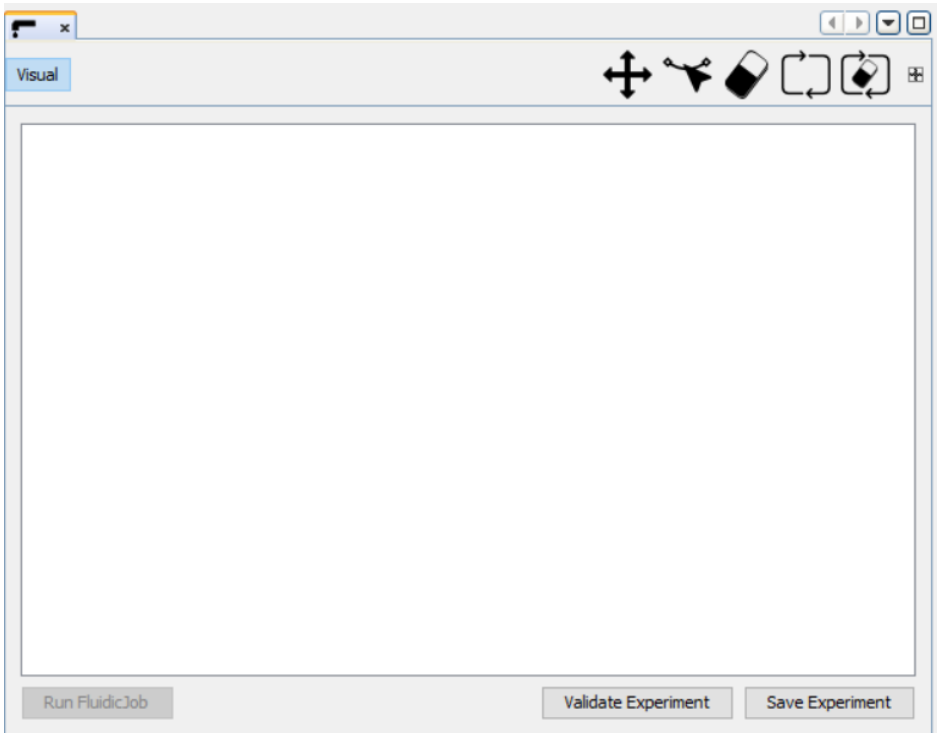


Fig.47 TopComponent creado por la clase FluidicVisualElement.

Los botones de la barra de herramientas (Fig.47) de la esquina superior derecha y que son creados en esta clase, invocan a métodos de las clases GraphSceneImpl y SceneUtilities, realizan las siguientes acciones:

Acción de mover nodos

Crear conexión entre dos nodos

Borrar todos los nodos y conexiones de la Scene

Crear Loop

Borrar Loop

Fig.48 Botones de la barra de herramientas.

Por otro lado las acciones que realizan los botones situados en la parte inferior del TopComponent son:

- RunFluidicJob: invoca al método *RunFluidicJob()* de la clase FluidicJob, encargado de enviar los comandos a la plataforma. Este botón se habilita cuando la validación del experimento ha sido satisfactoria.
- Validate Experiment: invoca al método *validateExperiment()* de la clase SceneUtilities. Comprueba que el experimento generado por el usuario se puede lanzar, ya que en el caso de que se encuentre algún error, devolverá un mensaje en una ventana emergente con dicho error.

- Save Experiment: al pulsar sobre este botón se invoca a métodos de la clase FluidicJob y SceneUtilities. Se recoge la información que hay en ese momento en la Scene y se guarda en un archivo .xml para que la Scene pueda ser reconstruida posteriormente.

3.1.3.9 Paquete com.aws.fluidicjob.util

Dentro del paquete com.aws.fluidicjob.util se encuentran las clases FluidicJobCommand y FluidicJobTimerTask que se utilizan para llevar a cabo el envío de datos a la plataforma.

FluidicJobCommand es una estructura de datos que contiene toda la información necesaria para la ejecución de un comando en la plataforma F20. Los atributos de esta clase son:

Tipo	**Nombre Variable**
FluidicJobCommandType	command
int	channel
int	solenoidalVolume
int	inPort
int	outPort
double	flowRate

El tipo FluidicJobCommandType es a su vez del tipo enumerado. Un enumerado (o *enum*) es una clase "especial" que limita la creación de objetos a los especificados explícitamente en la implementación de la clase. La única limitación que tienen los enumerados respecto a una clase normal es que si tiene constructor, éste debe ser privado para que no se puedan crear nuevos objetos. En el *enum* declarado en esta clase se muestra en la Fig.49.

```
public enum FluidicJobCommandType {

    START,
    ILOAD,
    INJECT,
    FINITEFLOW,
    INFINITEFLOW,
    SOLENOIDAL,
    UPDATEINFINITEFLOW,
    STOP;
}
```

Fig.49 Tipo FluidicJobCommandType definido como enum.

Este *enum* contiene los comandos que identifican a cada a nodo, a excepción del nodo de tipo AwsWaitingTimeNode, ya que este tipo de nodo representa un tiempo de espera y no un comando. La creación de los objetos FluidicJobCommand se realiza en la creación de los nodos. Por ejemplo, el constructor que aparece en la clase AwsInfiniteFlowNode es el mostrado en la Fig.50.

```
public AwsInfiniteFlowNode(String id, Image icon, String command) {
    super(id, icon, command);
    super.fluidicCommand = new FluidicJobCommand(FluidicJobCommand.FluidicJobCommandType.INFINITEFLOW);
}
```

Fig.50 Creación de un objeto FluidicJobCommand.

Se puede observar, como se crea en el constructor de la clase AwsInfiniteFlowNode un objeto de tipo FluidicJobCommand (este objeto es un atributo de la clase AwsInfiniteFlowNode) y se establece que el FluidicJobCommandType es INFINITEFLOW. Posteriormente, cuando el usuario establezca los parámetros de cada comando la correspondiente ventana de propiedades, se guardará dicha información en esta clase. Podría decirse, que la información guardada en las

clases que pertenecen al paquete com.aws.fluidicjob.model, es utilizada para manipular, interactuar y reconstruir la Scene, mientras que la información de la clase FluidicJobCommand es la que se envía a la plataforma, aunque ambas informaciones son idénticas.

Por último, la clase FluidicJobTimerTask es la encargada de enviar los comandos a la plataforma F20. Es invocada desde la clase FluidicJob, y recibe los objetos de tipo FluidicJobCommand y la plataforma a la que hay que enviar los comandos. En función del FluidicJobCommandType de cada nodo recibido, se decide en un estamento *switch* que comando enviar a la plataforma.

3.1.3.10 Otros paquetes

- com.fluidicjob.resources: este paquete únicamente contiene todas las imágenes (archivos .png) que se utilizan dentro del módulo.
- com.aws.fluidicjob.services: la clase de este paquete crea el FluidicDataObject desde 0.

3.1.4 Validación del módulo desarrollado

Una vez desarrollada la aplicación para la programación de experimentos, es necesario validarla para comprobar que funciona adecuadamente. Para ello se definieron los 2 experimentos que a continuación se describen.

Experimento 1: Detección de cambios en la viscosidad y densidad en medios fluidos.

Una de las aplicaciones más extendidas del cristal de cuarzo es su uso para detección de cambios en la viscosidad y densidad de medios fluidos. Cuando sobre un sensor QCM se produce un cambio en el medio que está en contacto con él, se modifican dos parámetros de su impedancia eléctrica: su resistencia y su frecuencia de resonancia. Existen modelos físicos que relacionan las variaciones en la resistencia y frecuencia de resonancia de un sensor QCM con cambios en la densidad y viscosidad del medio. Para validar la aplicación desarrollada, en este primer experimento se provocará un cambio en el medio que hay depositado sobre un sensor QCM de 10MHz. Para ello, durante un intervalo de tiempo controlado se hará pasar un flujo constante de agua por la superficie de un sensor QCM, y en un instante determinado se cambiará el flujo de agua por un flujo glicerol con una concentración del 5%. Para ello, el protocolo que debe seguirse es el siguiente:

1. Se hace pasar agua bidestilada por el sensor con una velocidad (flow rate) de 50 µl/min. Para ello se succionará el agua bidestilada almacenada en un reservorio por el puerto 1 de la válvula de distribución (distribution valve en Fig.8) y se dispensará por el puerto 3 de la misma válvula. Para encaminar el agua hacia la superficie del sensor se deberá colocar la válvula de inyección en posición *Load* (injection load en Fig.8). Este flujo se mantiene hasta la finalización del experimento.
2. Una vez transcurridos los 5 minutos desde el inicio del experimento se cargan 500 µl de glicerol almacenado en un reservorio haciendo uso de la bomba solenoidal de la plataforma AWS F20 (sample pump en Fig.8). Los 500 µl de glicerol se dejan almacenados en un tramo de tubo del sistema de fluídica (número 6 de la Fig.8).
3. Una vez transcurridos 10 minutos desde el inicio del experimento se modifica la posición de la válvula de inyección pasándola a modo *Inject*. De este modo, el flujo de agua bidestilada para por el tramo de tubería donde estaba almacenado el glicerol y lo arrastra hasta la superficie del sensor.
4. Una vez transcurridos 15 minutos desde el inicio del experimento se vuelve a cambiar la válvula de inyección a la posición Load, de modo que el flujo de agua bidestilada deja de pasar por el tramo de tubería en el que había almacenado el glicerol. En estas condiciones vuelve a pasar agua bidestilada por la superficie del sensor.
5. Se cambia la posición de la válvula de inyección a *Load* (vuelve a pasar agua bidestilada por el sensor).

6. Una vez transcurridos 20 minutos desde el inicio del experimento vuelven a repetirse los pasos 2 a 5 otras dos veces y se acaba el experimento.

El diseño de la programación del protocolo descrito se muestra en la Fig.51.

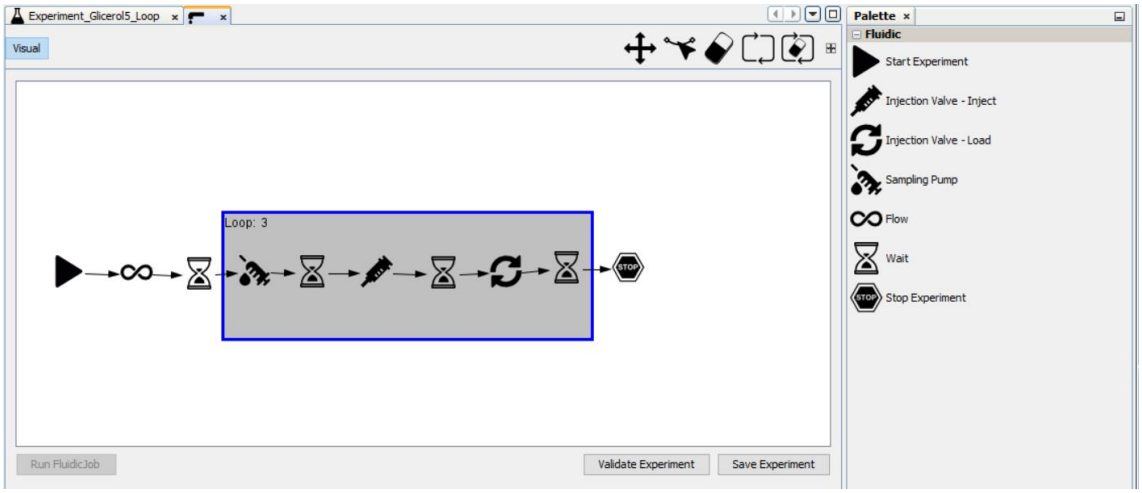


Fig.51 Programación del experimento consistente en un cambio del medio en contacto con el sensor.

El primer elemento ▶indica el inicio del experimento, tras él se introduce un elemento de flujo infinito ∞. En los atributos de este elemento se debe indicar la velocidad del flujo (50µl/min según el protocolo) y los puertos de entrada y dispensación de la válvula de distribución (puertos 1 y 3 según el protocolo). Este flujo se denomina infinito porque se mantiene así hasta que aparece un botón que indica finalización del experimento STOP, o aparece otro elemento de flujo infinito cambiando las propiedades del anterior (velocidad o puertos de entrada y dispensación). El siguiente elemento marca el instante en el que se produce el siguiente evento en el experimento, se trata de un elemento wait ⧖ en el que se debe programar el tiempo en el que se desea que se produzca el siguiente evento, ese tiempo se cuenta desde el inicio del experimento si no hay ningún elemento wait antes de él. En caso de que exista otro elemento wait antes, entonces se computa desde el último. Puesto que se desea realizar la carga de la muestra 5 minutos después del inicio del experimento se programará este tiempo en las propiedades del elemento wait. El siguiente evento consiste en cargar la muestra en un tramo de tubería reservado para ello, con este fin se incluye el elemento , en él debe programarse el volumen de muestra que se desea cargar (500µl según el protocolo). El siguiente evento debe producirse 10 minutos después del inicio del experimento, se trata de la inyección de la muestra. Para ello, debe incluirse otro elemento wait con un tiempo programado de 5 minutos, y a continuación un elemento que ordene la inyección de la muestra para hacerla pasar sobre el sensor 10 minutos después del inicio del experimento. Tras una espera de otros 5 minutos (elemento wait), se hace pasar la válvula de inyección a la posición load, para ello se introduce el elemento , y otra espera de 5 minutos hasta volver a repetir 2 veces de nuevo el protocolo desde la carga de la muestra. Para ello se implementa un loop (cuadrado sombreado que aparece en la Fig.51) englobando los elementos que se deben repetir. Una vez finalizadas las dos repeticiones se detiene el experimento introduciendo el elemento STOP.

El experimento se realizará utilizando el modo Tracking de la plataforma AWS A20. Este modo de funcionamiento realiza un barrido inicial de la respuesta en frecuencia del sensor para detectar la frecuencia de resonancia del mismo (Ver Fig.52) y lo que hace es ir siguiéndola durante los cambios se van produciendo en la impedancia del sensor. Al tratarse de un sensor QCM de 10MHz se espera la frecuencia de resonancia alrededor de dicho valor.

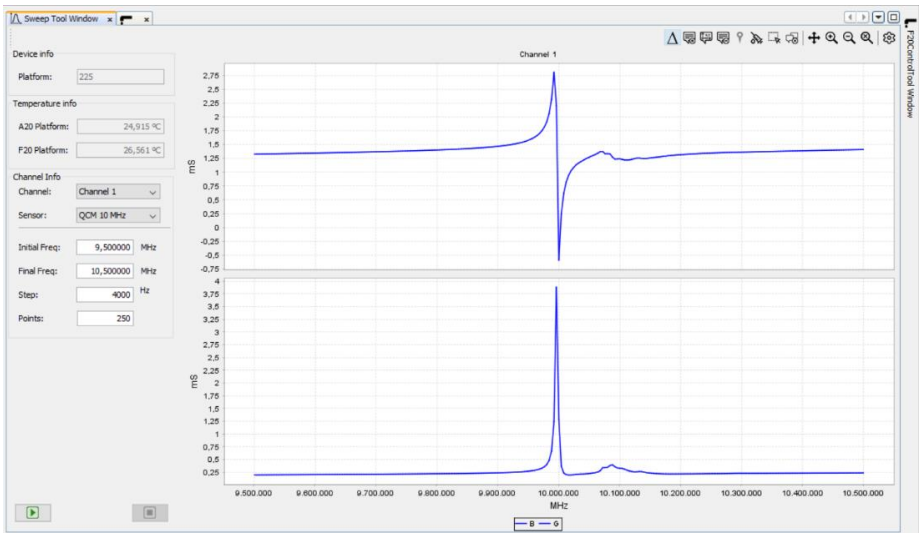


Fig.52 Respuesta en frecuencia de un sensor QCM a 10MHz.

Una vez detectada la frecuencia de trabajo y antes de lazar el experimento programado, es conveniente dejar el equipo funcionando unos 30 minutos haciendo que circule por el sensor agua bidestilada, con ello que se pretende estabilizar la temperatura del equipo y obtener la línea de medida de referencia (línea base). La obtención de una línea base estable es necesaria antes de iniciar cualquier experimento, esta línea no debe presentar cambios bruscos debidos a efectos ajenos al cambio físico que pretende detectarse para no falsear las medidas proporcionadas por el sensor. En la Fig.53 se muestra una línea base obtenida, los picos que aparecen en la respuesta en frecuencia son debidos a la carga y descarga de la jeringa del sistema de fluídica.

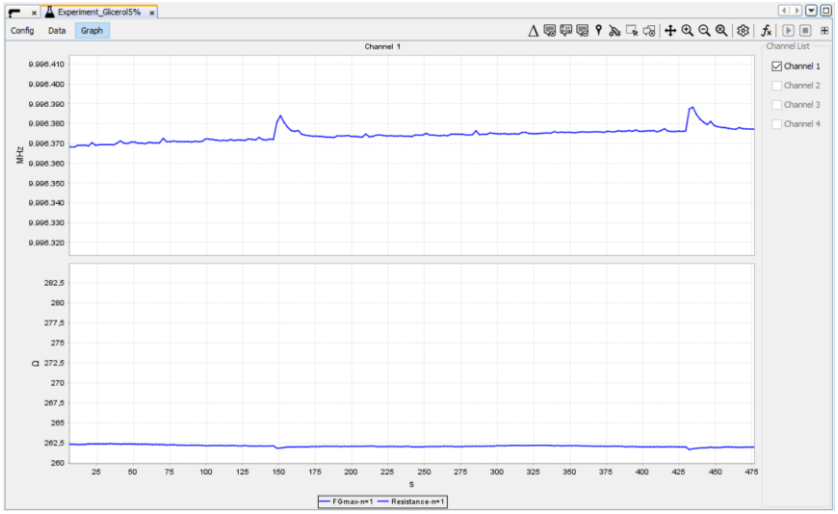


Fig.53 Obtención de la línea base.

Una vez conseguida la línea base, se lanza el experimento programado que se ha representado en la Fig.51. Los resultados de los cambios medidos en la resistencia y frecuencia de resonancia del sensor se observan en la Fig.54.

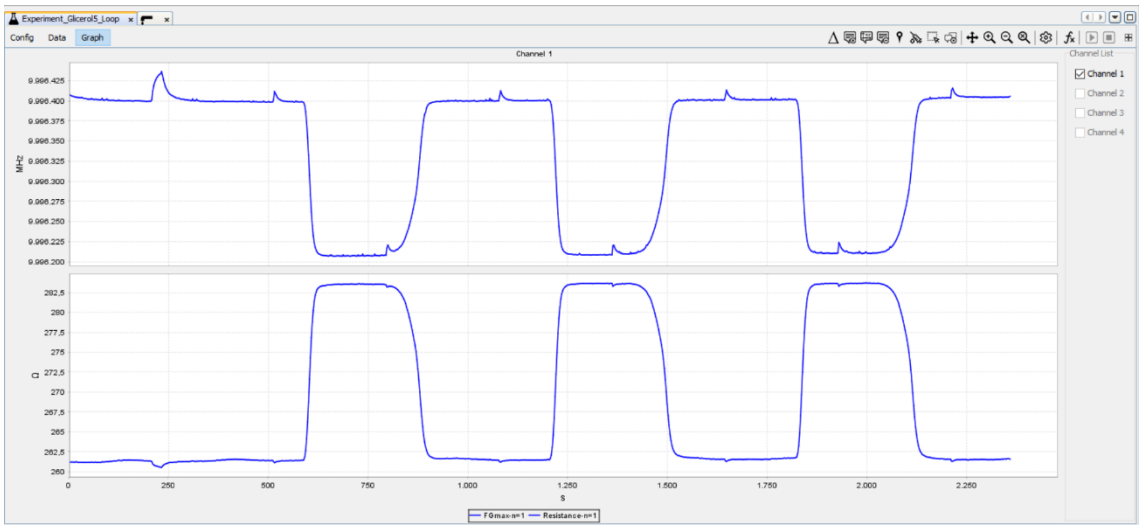


Fig.54 Resultado del experimento diseñado para glicerol al 5%.

En el experimento se han programado tres pasos de glicerol por la superficie del sensor, cada vez que se produce el cambio de medio agua bidestilada-glicerol 5% se produce un decremento de la frecuencia de resonancia y un incremento en la resistencia. Los cambios se deben al aumento en la densidad y viscosidad del glicerol en relación con la del agua bidestilada. Los cambios se producen en los instantes esperados según el tiempo programado en el experimento, se observa también como las tres repeticiones del mismo experimento son prácticamente iguales.

El mismo experimento programado se aplicó para una cambio de medio agua bidestilada-glicerol al 10%. Los resultados se muestran en la Fig.55.

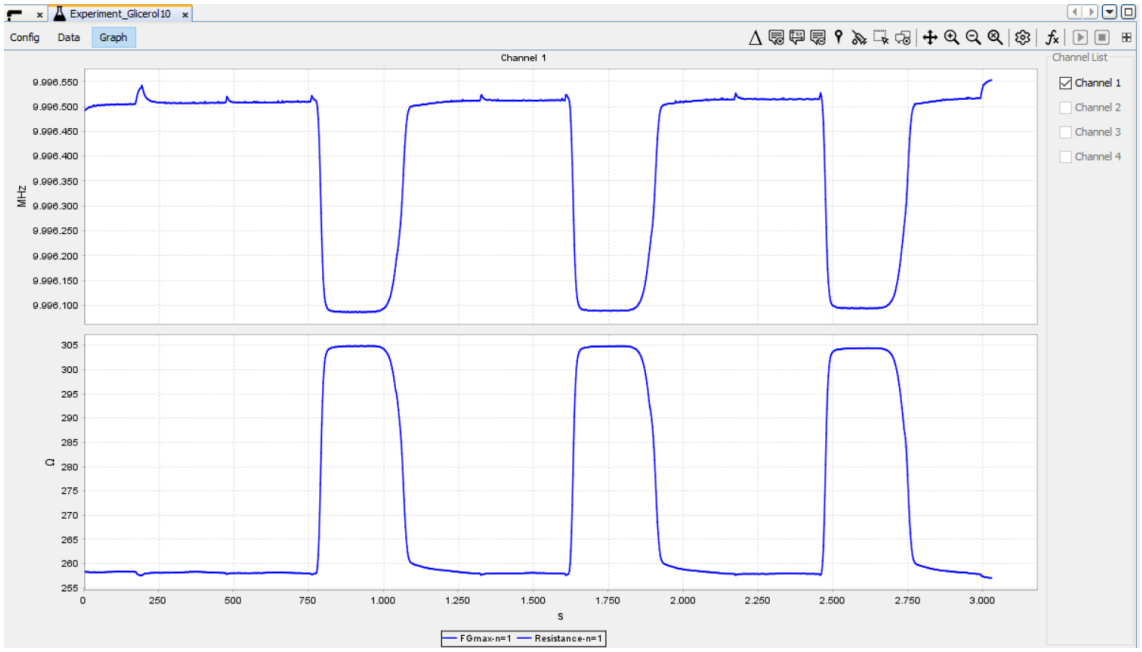


Fig.55 Resultado del experimento diseñado para glicerol al 10%.

Al elevar la concentración del glicerol se produce un aumento en la densidad y viscosidad del fluido resultante, esto resulta en una variación mayor tanto en la frecuencia de resonancia como en la resistencia.

Experimento 2: Detección del pesticida DDT para una aplicación como biosensor.

Este experimento se realizó en una plataforma que tiene instalada la empresa AWSensors en el Centro de Investigación e Innovación en Bioingiería (CI2B) de la Universidad Politécnica de Valencia. Investigadores de este centro están ejecutando un proyecto de investigación de la convocatoria Retos de la sociedad (AGL2013-48646-R) junto con investigadores del Instituto Universitario de Ingeniería de Alimentos para el Desarrollo (IUIAD) -Unidad de Control y Gestión de Calidad- para la detección de pesticidas y antibióticos en mieles. En particular se está trabajando sobre la detección del pesticida DDT. Los pesticidas son moléculas de muy bajo peso molecular, por ello utilizan cristales HFF-QCM de 100 MHz. Las medidas las realizan utilizando el modo High-Resolution que tiene disponible la plataforma AWS A20. Como se comentó en el epígrafe 1.1.2, en este modo de operación se selecciona y se fija la frecuencia de excitación óptima del sensor, que para el caso de un sensor HFF-QCM es la que proporciona la máxima conductancia del mismo. Una vez fijada esta frecuencia de excitación el sistema monitoriza los cambios en la fase y en la amplitud de la respuesta eléctrica del sensor en función del tiempo. El hecho de utilizar sensores de alta frecuencia amplifica el nivel de señal entregada por el sensor, esto unido a que la metodología de medida limita el ruido permite aumentar el límite de detección necesario en este tipo de aplicaciones. Para lograr la detección selectiva del pesticida, la superficie del sensor se inmoviliza con una sustancia que reconoce biológicamente al pesticida a detectar. En esta aplicación se ha utilizado un reconocimiento antígeno-anticuerpo. El antígeno corresponde al DDT y el anticuerpo al Anti-DDT. Cuando en la muestra se encuentra la sustancia a detectar (pesticida DDT) se produce una reacción antígeno anticuerpo que modifica la masa de la superficie del sensor produciendo cambios en su frecuencia de resonancia. Al tratarse de moléculas de pequeño tamaño, los cambios en la resistencia son prácticamente despreciables. A continuación se describe el protocolo a seguir para llevar a cabo la realización de este experimento.

1. Obtención de línea base, para ello se hace pasar por la superficie del sensor un flujo constante del medio biológico adecuado para que se produzca la reacción antígeno-anticuerpo (PBS). Este medio se succiona de un reservorio por el puerto 1 de la válvula de distribución y se dispensa por el puerto 3 de la válvula de distribución con una velocidad de 20 µl/min. Este flujo debe pasarse hasta el final del experimento.
2. Después de 30 minutos de espera para conseguir una buena línea base, se cargan 500 µl de la muestra en la que se quiere detectar si está presente pesticida. La carga se hace en un tramo de tubería en el que queda almacena a la espera de ser inyectada en la superficie del sensor. Para hacer la carga de la muestra se utiliza la sampling pump que aparece en la Fig.8.
3. Tras un minuto de espera, se modifica la posición de la válvula de inyección pasándola a posición *Inject*. En esta situación el flujo de PBS "arrastra" la muestra que estaba almacenada en un tramo de tubería y la hace pasar por el sensor.
4. Se deja un tiempo de espera de 20 minutos para que se produzca la reacción antígeno-anticuerpo.
5. Una vez transcurridos los 20 minutos se vuelve a cambiar la posición de la válvula de inyección a *Load* para que el flujo siga circulando por el canal principal.
6. Una vez que se ha producido la interacción se regenera la superficie del sensor y se prepara para otra repetición del mismo experimento, para ello se hace pasar por el sensor ácido clorhídrico a una velocidad de 250 µl/min , el cuál se toma por el puerto 5 de la válvula de distribución (a este puerto está conectado el reservorio en el que se almacena este ácido) y se dispensa por el puerto 3 para hacerlo pasar por la superficie del sensor. Este medio se hace pasar durante un intervalo de 4 minutos. Este medio deshace la unión biológica que se produce durante el reconocimiento de la muestra.
7. A continuación se hace pasar por el sensor a la misma velocidad (250 µl/min) PBS para eliminar los restos de ácido clorhídrico que pueda haber. Este fluido se hace pasar durante 5 minutos.
8. Se hacen otras dos repeticiones del experimento, repitiendo los pasos 3 a 7.

La programación del protocolo correspondiente a este experimento se muestra en la Fig.56.

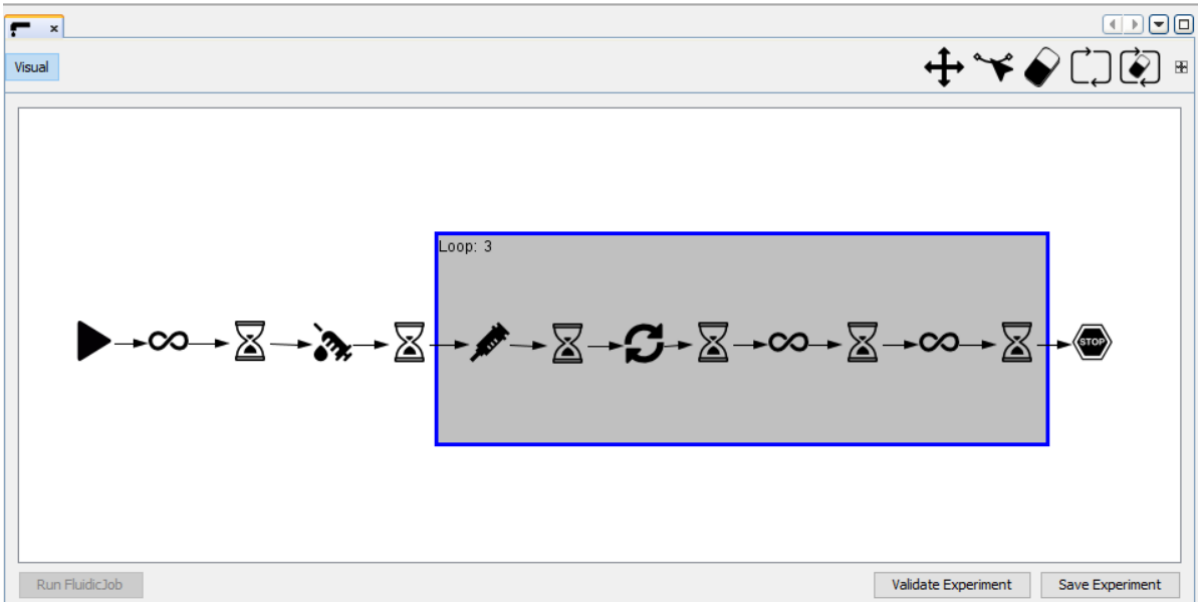


Fig.56 Experimento diseñado para la detección de DDT.

Los resultados del experimento se muestran en la Fig.57. En el panel superior se muestra en color azul la variación en la frecuencia (obviar la curva roja), y en el panel inferior se muestra en color azul los cambios en la resistencia (obviar la curva roja). Se observan tres repeticiones del mismo experimento (se ha recortado el tiempo en el que se estuvo obteniendo la línea base inicial para tener un detalle mayor de los instantes en los que se produce la reacción). Desde el instante en el que se produce la inyección de la muestra, transcurren unos minutos hasta que se produce la reacción biológica de reconocimiento, la cual corresponde al decremento en la frecuencia de resonancia (cercano a los 2k Hz). Aunque también se producen cambios en la resistencia del sensor, su magnitud es de unos pocos ohmios. Este resultado es muy frecuente en aplicaciones en las que las moléculas que se unen a la superficie del sensor son de tamaño muy pequeño. Entre un ensayo y la siguiente repetición se produce la regeneración de la superficie del sensor, esta regeneración consiste en romper la unión biológica de reconocimiento antígeno-anticuerpo, para ello se hace pasar ácido clorhídrico por la superficie del sensor a una velocidad más elevada. El cambio de velocidad produce un cambio en la respuesta del sensor, que se recupera cuando vuelve a disminuirse la velocidad para arrancar el siguiente ensayo. Los "picos" observados en la respuesta de la señal se deben al movimiento de la jeringa, la carga de la jeringa se produce a una velocidad elevada siendo el resultado la aparición de los picos. Al igual que en el experimento anterior los resultados son muy repetitivos.

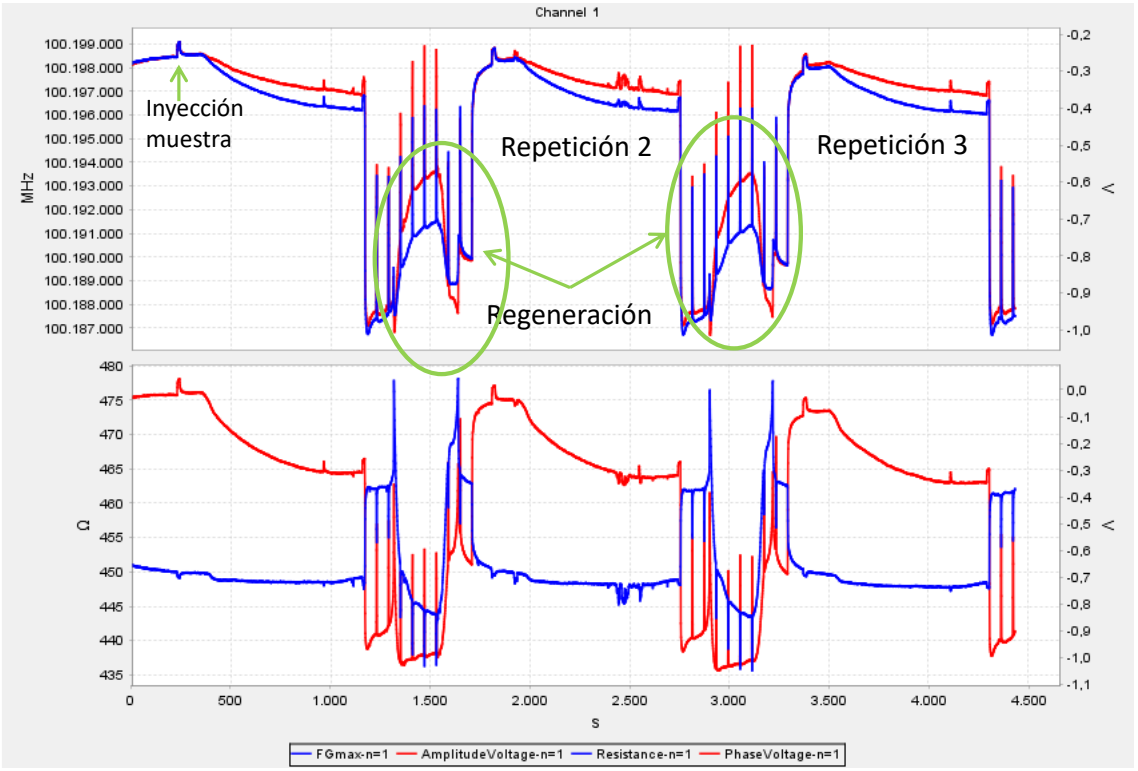


Fig.57 Resultado del experimento diseñado para la detección de DDT.

3.2 Estudio de viabilidad de la incorporación de un nuevo sistema de bombeo basado en una micro annular gear pump

Uno de los retos pendientes de la tecnología de sensores acústicos es la optimización del set up experimental con el fin de incrementar la fiabilidad de la respuesta de estos sensores. Para ello, es necesario minimizar efectos externos tales como las variaciones de la temperatura o las perturbaciones que introduce el sistema de fluídica. Los sistemas de bombeo empleados actualmente en la plataforma F20 son costosos, generan burbujas y alteraciones en la línea base cuando se produce la recarga de la jeringas. Por este motivo, desde el departamento de I+D de la compañía AWSensors se buscaron otras soluciones para sustituir a los sistemas de bombeo actuales por otros que permitan abaratar su coste y reducir los efectos observados. Una de las soluciones a explorar son las bombas *micro annular gear pump*. Se trata de un sistema de bombeo más económico que no utiliza jeringas. La bomba fue seleccionada por el grupo de I+D de la compañía, y se definió como objetivo de este TFM su puesta a punto y programación, así como su evaluación. Para llevar a cabo dicha evaluación se propuso la realización de medidas comparativas entre los dos sistemas de bombeo (el actual y el nuevo a evaluar).

3.2.1 Principio de operación de la micro annular gear pump.

Actualmente, la dosificación precisa en el rango de microlitros y mililitros con flow rates del orden de µl/min, son requerimientos cada vez más demandados en los ámbitos de instrumentación analítica, medicina, biotecnología o producción industrial. En este sentido, las **micro annular gear pumps** son bombas empleadas en la dosificación con precisión en numerosas aplicaciones.

En la fabricación de las micro annular gear pumps se utilizan materiales de alta tecnología, que posibilitan la compatibilidad química con los medios a dosificar, y mecánica de precisión, que proporcionan volúmenes de dosificación y flow rates muy bajos.

La gama de productos de las micro annular gear pumps comprende cuatro líneas diferentes (Ver Fig.58). En función de la bomba escogida, se puede alcanzar flow rates de hasta 1 µl/h y volúmenes de dosificación a partir de 0.25 µl (Ver Fig.59).

High performance pump series | Hermetic inert pump series | Low pressure pump series | Modular pump series

Fig.58 Tipos de bombas comercializados por HNP Mikrosysteme.

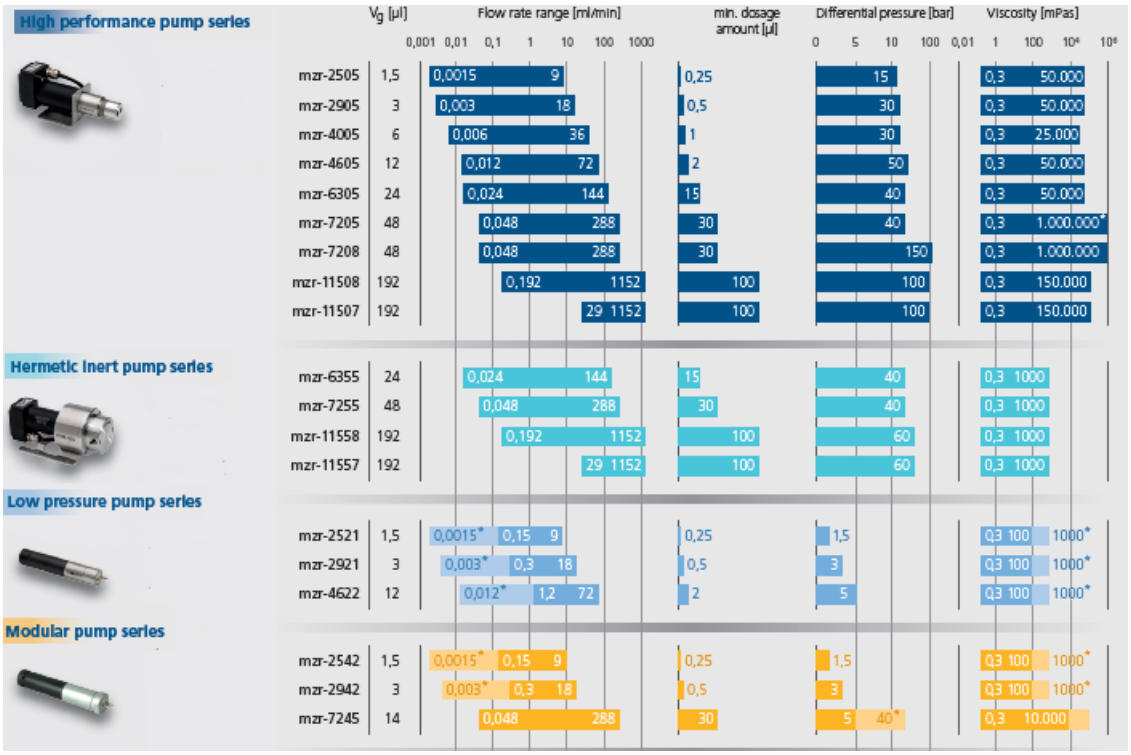


Fig.59 Flow Rate y volumen de dosificación en función del tipo de bomba.

Las micro annular gear pumps son bombas reciprocas y rotatorias que están provistas de un rotor interno dentado externamente y un rotor externo dentado internamente girando alrededor de ejes ligeramente excéntricos (Ver Fig.60). Las características de los dos rotores forman, durante la rotación, un sistema de varias cámaras de bombeo selladas.

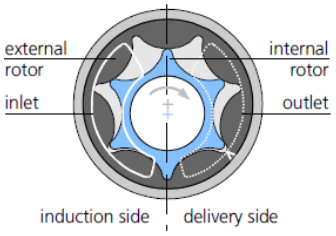


Fig.60 Estructura de la micro annular gear pump.

A medida que los rotores giran alrededor de sus ejes, se va generando un flow rate homogéneo entre la entrada y la salida (Ver Fig.61).

Fig.61 Principio de funcionamiento de las micro annular gear pumps.

La bomba está conectada directamente al eje del motor por medio de un acoplamiento flexible resistente a la torsión. En cuanto al sellado entre la parte que contiene el líquido y el resto de la bomba, está asegurado por medio de una junta rotatoria.

El caudal Q, o flow rate, en este tipo de bombas viene dado por la siguiente ecuación:

$$Q = \eta_{vol} \cdot V_g \cdot n \quad (2)$$

Donde V_g es el volumen de desplazamiento suministrado por la bomba (este valor depende del modelo de bomba escogido), *n* el número de revoluciones y η_{vol} la eficiencia volumétrica, que representa la desviación entre el flujo real y el teórico. Las diferencias entre ambos flujos se pueden ser debidas a las fugas, o a la viscosidad del medio de bombeo.

En este tipo de bombas pueden surgir efectos de cavitación. La cavitación puede derivar en una limitación de la velocidad. Cuando la presión estática alcanza la presión de vapor del líquido a la entrada de la bomba, se forma un gas que dificulta la alimentación de la bomba. En estas condiciones, un incremento de la velocidad no se traduce en un aumento del flow rate.

3.2.2 Puesta en marcha de la bomba y montaje.

La bomba que se ha utilizado en este TFM es la mzr-7240 (Ver Fig.62), la cual se encuentra dentro del grupo *Modular pump series*. Las micro annular gear pump modulares son adecuadas para el uso con líquidos poco corrosivos. Este tipo de bombas cubre desde aplicaciones de instrumentación analítica a aplicaciones químicas.

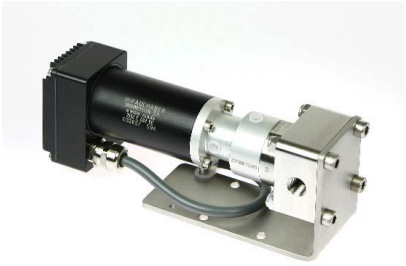


Fig.62 Bomba mzr-7240.

Esta bomba tiene un volumen de desplazamiento V_g de 48 µl a 3000 rpm, y un rendimiento volumétrico de un 100%. A partir de la ecuación (2) se obtiene un caudal de 144 ml/min. La Tabla 1 muestra el caudal suministrado en función del número de revoluciones (se asume η_{vol} = 100%).

Speed **[rpm]**	Q **[ml/min]**	Q **[ml/h]**
1	0.048	2.88
100	4.8	288
500	24	1440
1000	48	2880
2000	96	5760
4000	192	11520
6000	288	17280

Tabla 1. Caudal característico de la micro annular gear pump.

Como se desprende de la tabla, el límite inferior del caudal son 48 µl/min y el superior 288 µl/min, se escogió este modelo su rango es el que mejor se ajusta a los flow rates empleados en las aplicaciones en las que se utiliza este tecnología (entre 20 µl/min y 200 µl/min). Aunque el límite inferior está ligeramente por encima, las bombas que bajan por debajo de 48 µl/min tienen flow rates máximos alrededor de 70 µl/min (Ver Fig.59).

Para controlar la bomba se ha utilizado el Terminal Box S-G05 (Ver Fig.63). Dicho terminal permite el control de la bomba utilizando uno de los siguientes modos: 1) con un potenciómetro, 2) remotamente con una señal analógica externa de 0-10 V, 3) con una señal de corriente externa (0-20 mA o 4-20 mA), o 4) con un interfaz RS232. Con los tres primeros modos, únicamente se puede controlar la velocidad del motor, variando la posición del potenciómetro, incrementando o disminuyendo la tensión, e incrementando o disminuyendo la corriente. Si el usuario desea controlar parámetros tales como calibración, paso del motor, velocidad, tiempos de dispensación/pausa y dirección del flujo, el fabricante proporciona dos software (Ver Fig.64a y Fig.64b) que se comunican con la mzr-7240 a través de la interfaz RS232. Con el primero de los software se puede seleccionar casi toda la gama de bombas de HNP Mikrosysteme, y controlar los parámetros antes mencionados. El segundo software se utiliza para enviarle los comandos directamente a la mzr-7240 y tener un control más exhaustivo de la bomba. Por ejemplo el comando *V1000* indica que la bomba trabajará a una velocidad de 1000 rpm.

Fig.63 Terminal Box S-G05.

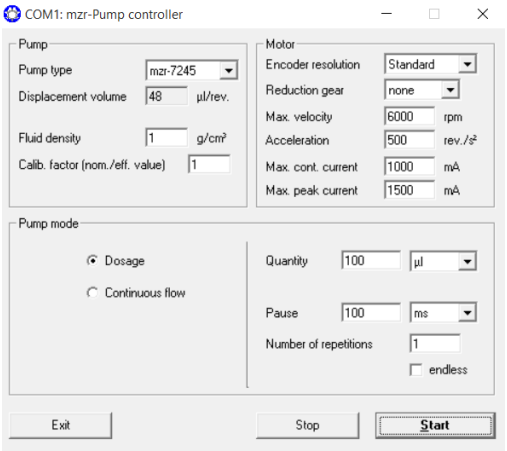


Fig.64a Software mzr-pump controller

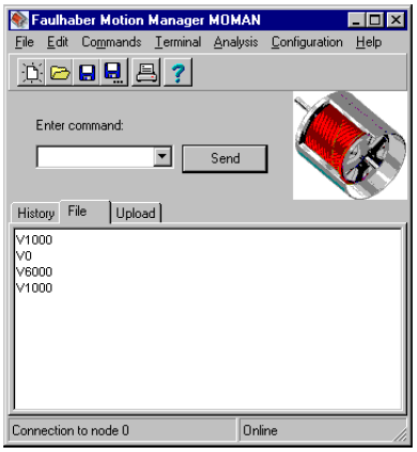


Fig.64b Software Motion Manager

Una vez realizadas las conexiones entre la mzr-7240 y el Terminal Box S-G05 tal y como se indica en la Fig.65 y se resumen en la Tabla 2, se siguió el siguiente protocolo para controlar la bomba a través la interfaz RS232.

1. Para prevenir una puesta en marcha descontrolada de la bomba, se debe llevar la rueda del potenciómetro a la posición null, girando la rueda en el sentido de las agujas del reloj hasta el límite.

2. Colocar el interruptor DIP en la posición “Poti”.
3. Conectar la interfaz RS232 del Terminal Box al ordenador. Usar para ello un cable cruzado de 9 pines.
4. Conectar la fuente de alimentación de 24 VDC (en este caso utilizamos una fuente de alimentación externa).
5. Proveer de un suministro de líquido constante a la bomba para evitar la operación en seco del dispositivo.
6. Poner en funcionamiento la bomba con uno de los dos software proporcionados.

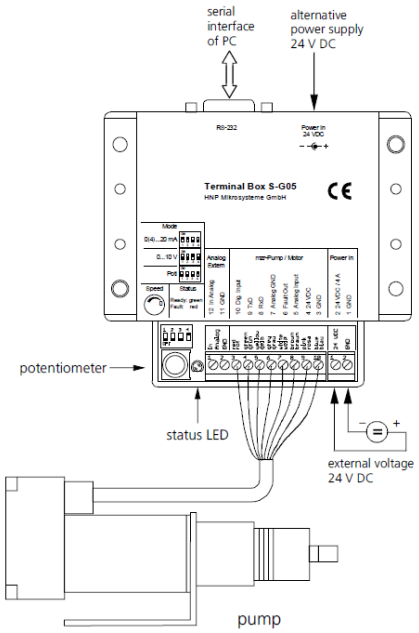


Fig.65 Esquema de conexiones mzr-7240 – Terminal Box S-G05.

Wire	Function	Terminal Box
Blue	Ground	GND
Pink	Voltaje supply	24VDC
Brown	Analog input	Analog input
White	Error output	Error out
Gray	Ground analog input	Analog GND
Yellow	RS-232 interface signal reception	RxD
Green	RS-232 interface signal transmission	TxD
Red	Digital input	Dig.Input

Tabla 2. Conexiones entre la mzr-7240 y el Terminal Box S-G05.

En la Fig. 66 se muestra una fotografía del set up empleado para la puesta en funcionamiento de la bomba y su posterior validación.

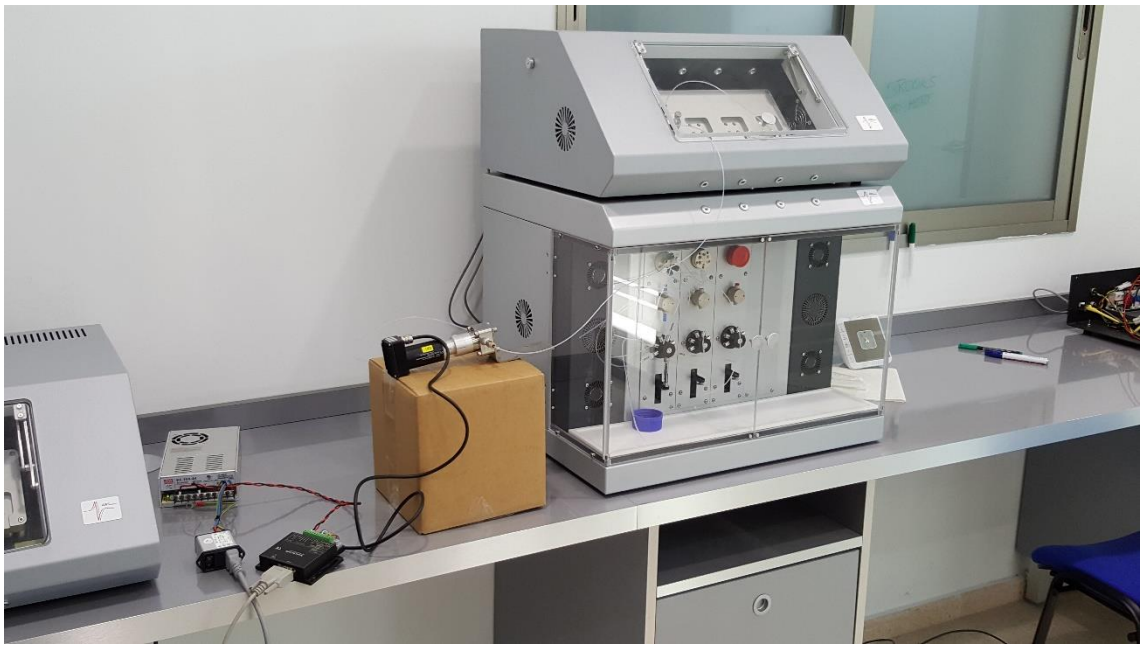


Fig.66 Montaje de la bomba mzr-7240 y conexión con la plataforma A20.

3.2.3 Estudio comparativo entre el sistema de bombeo actual y el sistema a evaluar

En las Fig.67a y 67b se muestra un detalle de la bomba a validar y de la jeringa empleada por el sistema de bombeo actual de la plataforma AWS F20, dicha jeringa se conecta a un módulo PSD que la controla junto con una válvula de distribución. En las tablas 3 y 4 se han resumido las prestaciones de la bomba mzr-7240 (Tabla 3) y las del sistema que hay implementado actualmente en la plataforma AWS F20, la Hamilton PSD4 5495-30 (Tabla 4). En términos de precisión y caudal el módulo actual supera el módulo a evaluar, sin embargo su conste también es superior. Lo interesante es ver las respuestas de ambos sistemas en funcionamiento para comprobar si se produce una mejora en el registro de la línea base con el nuevo sistema a evaluar.

Fig.67a Bomba mzr-7240

Fig.67b Hamilton PSD4 5495-30

			mzr-7240
Displacement volumen [µl]			48
Measurements [mm]	L x W x H		150 x 100 x 42
Flow rate	Q[ml/min]	min.	0.048
	[ml/min]	max.	288
Smallest dosage volume [µl]			5
Precision CV [%]			1

Tabla 3. Prestaciones de la bomba mzr-7240.

			Hamilton PSD4 5495-30
Accuracy			± 1% @ 100% stroke
Precision			≤ 0.05% @ 100% stroke
Syringe Volume [µl]			250
Flow rate	Q[ml/min]	min.	0.0125
	[µl /min]	max.	14.500
Resolution			Selectable 3000 steps (standard) / 24000 steps (high)
Syringe Drive Mechanism			Stepper motor driven lead screw and optical encoder
Communication Protocol			RS-232. RS-485 or CAN

Tabla 4. Prestaciones de la Hamilton PSD4 5495.30.

Para comprobar si la bomba puede ser integrada en la evolución del nuevo producto ASW F30 se realizaron una serie de experimentos simples consistentes en realizar registros de la línea base haciendo pasar un flujo continuo de agua bidestilada sobre el sensor. Estos experimentos se hicieron con el sistema de bombeo actual disponible en la plataforma AWS F20, y con la bomba mzr-7240 a validar. Sobre dicha línea base se debe chequear aspectos tales como la aparición de burbujas, desviaciones de la línea base o flow rates permitidos. Se utilizarán dos tipos de sensores: QCM de 9MHZ y HFFQCM de 100MHZ y se utilizarán flow rates permitidos por ambos sistemas de bombeo y similares entre sí para poder realizar la comparación (250 µl/min y 100 µl/min para el sistema PSD, y 240 µl/min y 96 µl/min para el sistema mzr-7240). El experimento se realizará empleando el método de medida High Resolution.

En la Fig.68, se muestra la respuesta generada para un sensor de 100MHz y para dos flow rates diferentes 250 µl/min hasta el instante 18:07 y 100 µl/min a partir de dicho instante y hasta el final del experimento. Los sensores HFFQCM son sensores extremadamente sensibles a cambios en las propiedades de los medios depositados sobre él, pero también a otros agentes externos como la temperatura o la velocidad del medio que se hace pasar sobre él. Se trata de una lámina muy fina del cuarzo (alrededor de 16 micras) por lo que el cambio de 250 µl/min a 100 µl/min lo detecta. Los “picos” observados picos se deben a la carga de la jeringa. La carga de la jeringa se hace una velocidad elevada (10000 µl/min) mientras que la dispensación se hace a 250 µl/min hasta el instante 18:07 y a 100 µl/min después. Durante los reducidos instantes temporales en los que la jeringa se tiene que recargar del medio líquido, el flujo por el sensor cesa temporalmente produciéndose el “pico” de la respuesta.

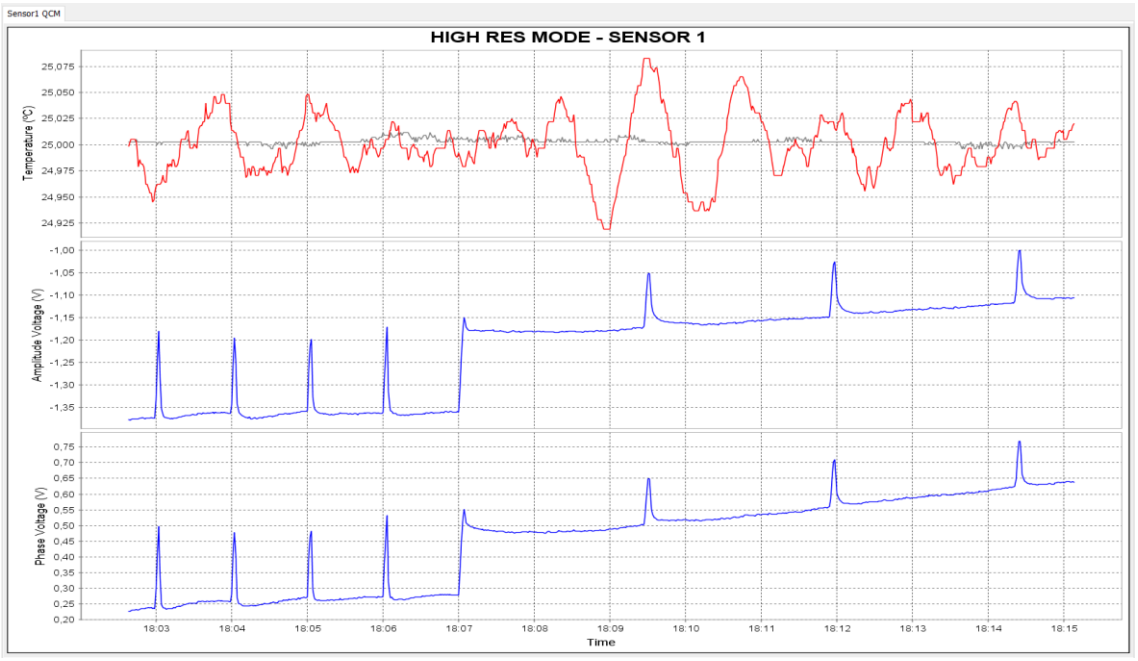


Fig.68 Modo HighRes – Sensor HFF-QCM 100 MHz – Sistema actual con la jeringa (Flow rate 250ul/min - 100ul/min).

Los resultados obtenidos con el nuevo sistema de bombeo a evaluar (bomba mzr-7240) se muestran en la Fig. 69. Se utilizaron dos flow rates diferentes lo más similares posible a los empleados con el sistema PSD: 240 µl/min hasta el instante 18:35:30 y 96 µl/min (correspondientes a una velocidad de 5 y 2 rpm respectivamente). Al igual que sucedía con el sistema PSD, el cambio de flujo sobre la superficie del sensor produce una modificación en la respuesta del sensor por los mismos motivos expuestos para el sistema PSD.

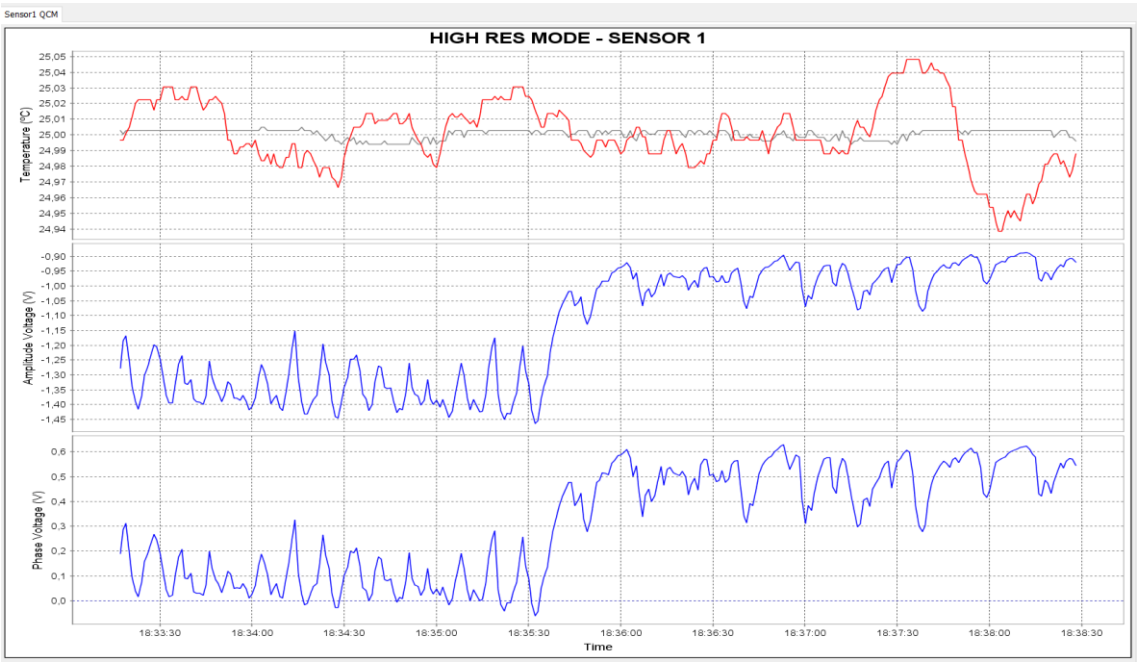


Fig.69 Modo HighRes – Sensor HFF-QCM 100 MHz – Bomba mzr-7240 (Flow rate 240ul/min - 96ul/min).

Las mismas medidas con ambos sistemas se repitieron para un sensor QCM de 9 MHz, con ello lo que se pretendía comprobar es si el efecto del sistema de bombeo sobre sensores de más baja frecuencia era diferente.

En la Fig.70 y Fig.71 se muestran los resultados obtenidos con el sistema de bombeo PSD y el sistema mzr-7240, respectivamente. Como era de esperar, la amplitud de las variaciones observadas en el sensor QCM de 9 MHZ es menor, su espesor es mucho menor, por lo que las variaciones debidas a efectos externos como la vibración que introduce el sistema de fluídica se reducen.

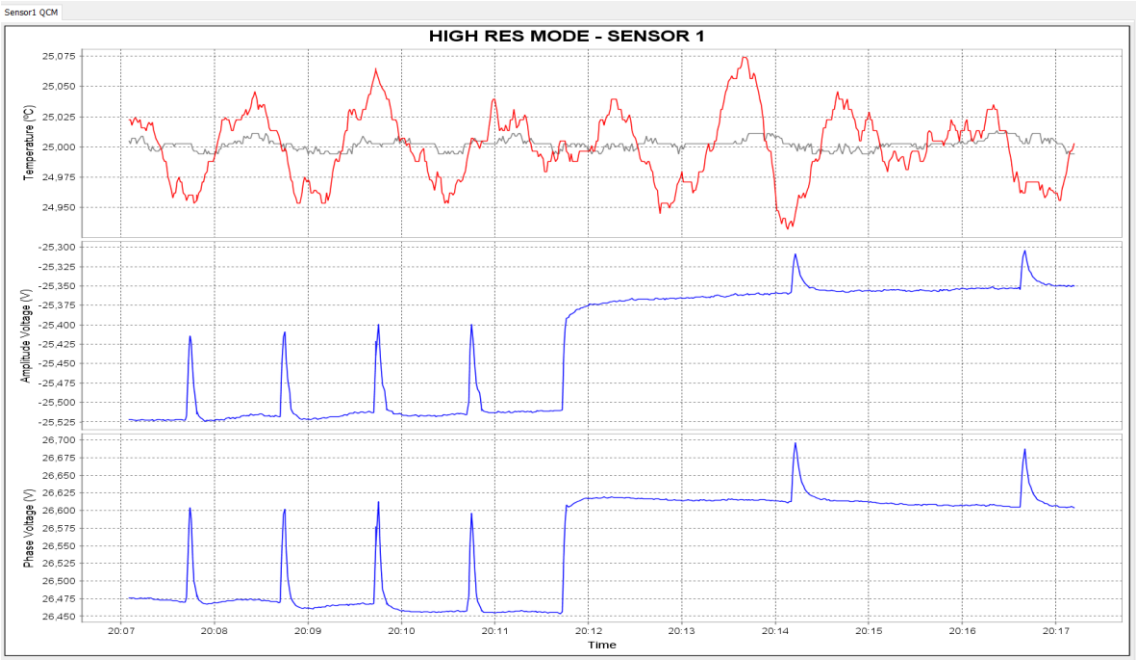


Fig.70 Modo HighRes – Sensor QCM 10MHz - Sistema actual con la jeringa (Flow rate 250ul/min - 100ul/min).

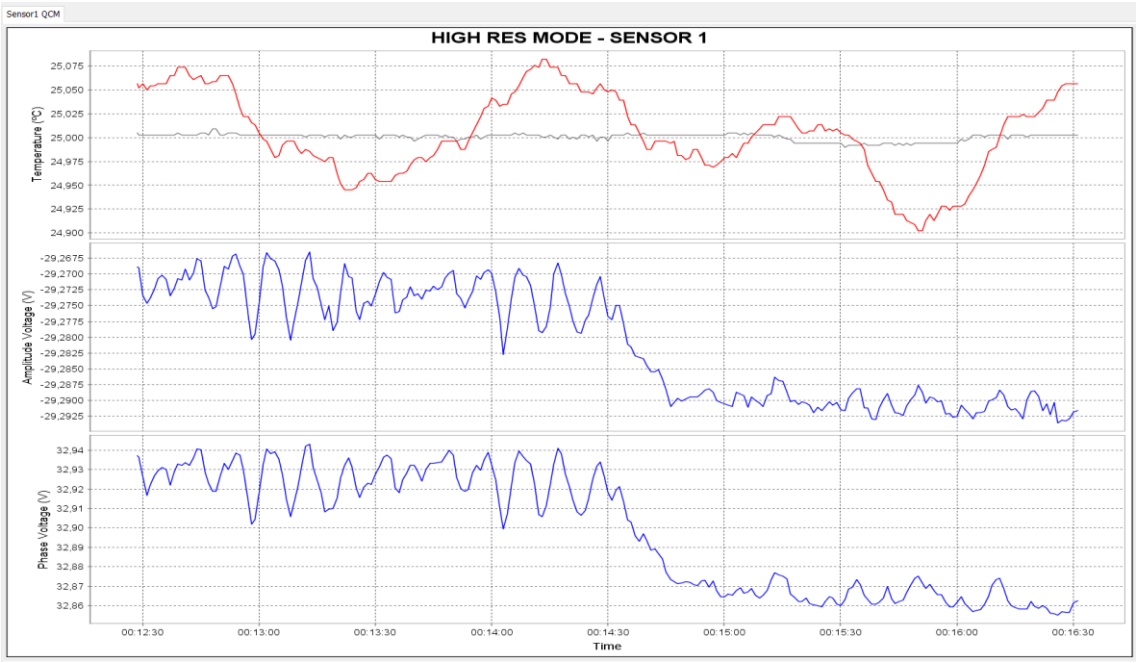


Fig.71 Modo HighRes – Sensor QCM 10MHz - Bomba mzr-7240 (Flow rate 240ul/min - 96ul/min).

Las conclusiones que se extraen de los experimentos realizados son las siguientes:

1) La bomba mzr-7240 no presenta los picos típicos del sistema PSD de jeringa, sin embargo las oscilaciones que se observan en un experimento en el que se tendría que ver una respuesta plana son excesivamente elevadas. Su magnitud es del mismo orden que las variaciones de señal que se observan en experimentos de alta

resolución con biosensores. Esto es un gran impedimento para su uso en ese tipo de aplicaciones.

2) Tal y como se ha comentado anteriormente, en un experimento típico con un biosensor es necesario trabajar en un rango de flow rates comprendido entre 20 µl/min y 200 µl/min. Estos rangos son los que ofrece la bomba Hamilton PSD4 5495-30 que está instalada actualmente en la plataforma AWSF20, ya que permite flow rates comprendidos entre de 12.5 µl y 14500 µl. En la bomba mzr-7240 el límite inferior son 48 µl/min (correspondiente a 1 rpm), que aunque podría ser un flow rate aceptable, cuando se ha probado en un experimento real se ha observado que debido a la baja presión la bomba no funciona correctamente. En las figuras anteriores se ha mostrado la respuesta para un flow rate de 96 µl/min (correspondiente a 2 rpm), y se observa un ruido base mucho mayor que el sistema PSD4 de Hamilton.

La discusión anterior permite concluir que no se mejora en prestaciones con el cambio de sistema de bombeo, por lo que se aconseja su cambio.

4. Conclusiones y futuras líneas

Las conclusiones de este Trabajo Fin de Máster son las siguientes:

1. Se ha desarrollado una aplicación software que permite a un usuario de la plataforma AWS F20 realizar la programación de experimentos. La aplicación ha sido integrada satisfactoriamente en el software AWS Suite comercializado por la empresa AWSensors.
2. La aplicación desarrollada ofrece al usuario un entorno amigable para programar diferentes protocolos, programación que puede cambiar tantas veces como quiera mientras se define el protocolo final de un experimento, lo que le confiere a la instrumentación mucha flexibilidad.
3. La aplicación introduce la gran ventaja de permitir a un usuario invertir el tiempo que antes dedicaba al manejo de la fluídica (cambios de válvulas, accionamiento de bombas, ajuste de velocidades…) en cada paso del protocolo (en muchas ocasiones entre unas acciones y otras pasaban muy pocos minutos) en otras tareas.
4. La aplicación desarrollada ha sido evaluada en 2 experimentos: Detección de cambio de propiedades de un fluido y detección del pesticida DDT. Los resultados han sido satisfactorios.
5. Se han evaluado las prestaciones de un nuevo sistema de bombeo (mzr-7240) que permita solventar algunas desventajas del sistema actual (Hamilton PSD4 5495-30). Se ha puesto en marcha, se ha adaptado a la plataforma de medida AWS-A20 y se ha programado para hacer pasar por dos tecnologías de sensores de cuarzo (QCM y HFFQCM) un fluido a diferentes flow rates.
6. Se han evaluado las prestaciones de ambas bombas con experimentos, pero no se han observado mejoras en la línea base de la respuesta del sensor. En relación a los flow rates alcanzados por el sistema de bombeo a evaluar, éstos no abarcan el rango necesario en experimentos de alta resolución de forma fiable. Aunque su coste es menor, las prestaciones empeoran por lo que se aconseja no sustituir el sistema actual por el evaluado.

Las líneas futuras que deja este Trabajo Fin de Máster abiertas son:

1. Mejora de la interfaz gráfica de la aplicación desarrollada, fundamentalmente en lo referente al diseño de los iconos.
2. Incorporar a la aplicación una representación gráfica del experimento en varios ejes temporales, con el fin de proporcionar al usuario información más clara de cuándo se producen los diferentes eventos en el experimento.
3. Superponer la información de las acciones realizadas en el sistema de fluídica sobre las repuestas gráficas proporcionadas por el software AWS Suite.

4. Añadir nuevos iconos relacionadas con nuevas acciones sobre el sistema de fluídica: Dispensar todo el volumen de la jeringa (independientemente del estado de ésta), dispensar sólo una cantidad definida por el usuario…
5. Explorar sistemas de fluídica basados en pocillos sobre los que se deposite o se extraiga un fluido mediante micropipetas sujetas por brazos robotizados. Este tipo de sistema de fluídica es el más adecuado para aplicaciones en las que se utilicen arrays de sensores,

5. Bibliografía

[1] *The Theory of Sound. s.l.* : **Rayleigh, J.W.S** Dover Publications New York, 1945.

[2] *Verwendung von Schwingquarzen sur Wägung Dünner Schichten und zur Mikrowägung. s.l*. **Sauerbrey, G.;.** : Z. Physik, 1959, Vol. 155, págs. 206-222.

[3] *Piezoelectri Crystals as Detectors for Liquid Chromatography. s.l.* : **Konash, P.L.; Bastiaans, G.J.** Analytical Chemistry, 1980, Vol. 52, págs. 1929-1931.

[4] *Characterization of a Thickness Mode Quartz Resonator with Multiple Nonpiezoelectriv Layers. s.l.* : **Granstaff, V.E.; Martin, S.J.** J.Appl.Phys, 1994, Vol. 75, págs. 1319-1329.

[5] *High-Frecuency Phase Shift Measurement Greatly Enhances the Sensitivity of QCM Immunosensors.* **March, Carmen; García, José V.; Sánchez, Angel; Arnau, Antonio; Jiménez, Yolanda; García, Pablo; Manclús, Juan J.; Montoya, Ángel;.** 65, s.l. : ELSEVIER, 2015, Biosensors and Bioelectronics.

[6] *Application of a Quartz-Crystal Microbalance to Measure Ionic FLuxes in Microcorporous Carbons for Energy Storage.* **Levi, Mikhael D.; Salitra, Grigory; Levy, Naomi; Aurbach, Doron; Maier, Joachim;.** 2009, Nature Materials.

[7] *A Different Point of View on the Sensitivity of Quartz Crystal Microbalance Sensors.* **Arnau, Antonio; Montagut, Yeison; García, Jose Vicente; Jiménez, Yolanda;.** 26 de October de 2009, Measurement Science and Technology.

[8] *A piezoelectric immunosensor for the detemination of pesticide residues and metabolites in fruit juices*. **March, A; Manclús, J.J; Jiménez, Y; Arnau, A; Montoya, A.** Talanta Volume 78, Issue 3, 15 May 2009, Pages 827–833

[9] *Fundamentals of Piezoelectric Immunosensors*. **Montoya, A; Ocampo, A; March, A**. Piezoelectric Transducers and Applications 2008, pp 289-306

[10] *The Definitive Guide to NetBeans Platform 7 (Expert's Voice in Java). s.l.* : **Heiko Bock.** Apress; 1st ed. edition, 2011.

[11] *Visual Library 2.0 - Documentation.* **Wielenga, Geertjan.** **http://bits.netbeans.org/dev/javadoc/org-netbeans-api-visual/org/netbeans/api/visual/widget/doc-files/documentation.html.**

[12] ***NetBeans API List.*** **10 de Marzo de 2014. http://bits.netbeans.org/8.0/javadoc/.**

[13] *Operating manual for micro annular gear pump mzr-7240* **HNP Mikrosysteme GmbH**

[14] *Faulhaber 3564K024B CS Instruction Manual* **FAULHABER**