¿Es Angular un framework adecuado para implementar un SCADA near real-time?

Trabajo fin de máster en desarrollo de sitios y aplicaciones WEB

Fecha: 06/06/2021	*Amaya Gamarra Lousa*

Tabla de contenido

Tabla de figuras

Indice de tablas

1. INTRODUCCIÓN

Un *SCADA* es una aplicación que permite monitorizar y controlar un proceso, a menudo en un entorno industrial.

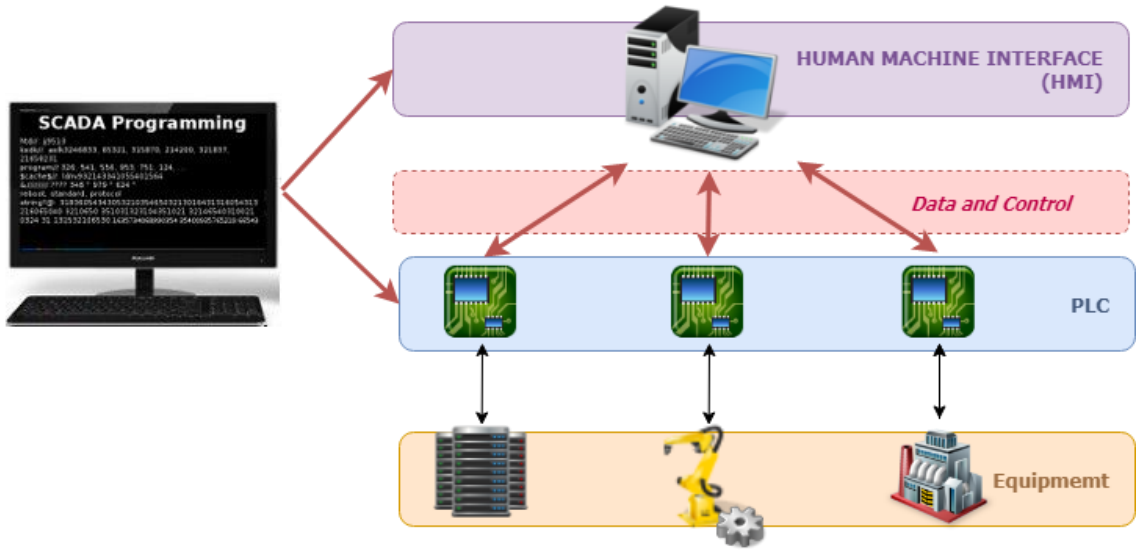


Figura 1 :Arquitectura de supervisión/control de equipos industriales

La Figura 1 muestra los diferentes niveles aplicativos que intervienen en la supervisión y en el control de equipos o maquinaria industrial. La primera capa la componen los equipos electrónicos de adquisición y control que generalmente se implementan con *PLC*s. Estos dispositivos son los encargados de transformar las señales eléctricas en señales lógicas y viceversa. Los *PLC*s se comunican con el *SCADA* vía protocolos industriales que permiten remontar las señales adquiridas y recibir las órdenes que este último les envía y que a menudo han sido iniciadas por un operador.

El *SCADA,* a su vez, asegura las funciones de tratamiento de datos como la conversión de las señales adquiridas a valores de ingeniería o el cálculo de nuevas variables como resultado de aplicar un cálculo a un conjunto de medidas. Por otro lado es la interfaz hombre máquina y por lo tanto implementa una interfaz gráfica con el que el usuario interactúa.

En muchos casos el desarrollo de las aplicaciones de *SCADA* es realizado por los fabricantes de *PLC*s y, por ello, la elección del primero suele venir condicionada por la marca que se seleccione del segundo. Éste hecho conduce frecuentemente a situaciones en las que en una misma instalación industrial conviven diferentes *SCADA*s, cada uno destinado a controlar una parte de la misma.

Otro inconveniente es que los *SCADA*s suelen implementarse sobre sistemas operativos de la familia *Microsoft Windows*, con los problemas que ello conlleva en términos de mantenibilidad.

En los últimos años se percibe una evolución en las arquitecturas de control industrial. La cada vez más habitual práctica de integrar los datos de mantenimiento y de producción en la toma de decisiones empresariales ha provocado la aparición aplicaciones de “hipervisión” que actúan como sintetizadores de los datos de supervisión generados por los *SCADA*s y que los remontan de manera gráfica a los ejecutivos de las empresas para que estos puedan integrarlos en sus informes. Es a nivel de estos hipervisores que se observa una tendencia hacia el uso de aplicaciones *WEB*.

Por otro lado, numerosos protocolos industriales que comunican los *PLC*s con los *SCADA*s han tendido a la estandarización, actualmente la mayoría de aplicaciones integran drivers para protocolos como *Modbus-TCP*, *Modbus-RTU* e incluso *EtherCAT*.

Se podría dar un paso más hacia la mantenibilidad de un *SCADA* si se implementara como una aplicación *WEB* lo que además conllevaría la posibilidad de utilizar una misma compilación desde diferentes plataformas y tipos de dispositivos. La implementación de la interfaz gráfica debe, no obstante, ser ágil para el usuario y, aún más importante, esta interfaz debe ser lo suficientemente rápida como para ofrecer datos de supervisión fiables y precisos respecto a la evolución física de las señales adquiridas.

La arquitectura orientada a componentes de *Angular* me ha parecido una buena manera de cumplir con el primer objetivo: si los símbolos del *SCADA* se implementan mediante componentes con un comportamiento intrínseco, sólo nos quedaría encontrar una solución ergonómica que permita al usuario la distribución estos componentes fácilmente sobre una vista, de preferencia mediante un mecanismo *drag & drop*.

Por otro lado, el uso de RxJS y de una arquitectura basada en un patrón de flujo de datos Redux debería facilitar el cumplimiento de los requisitos de rendimiento.

2. DEFINICIONES Y ABREVIACIONES

Abreviación	*Sign*ificado
IHM	Interfaz Hombre Máquina
LAN	Local Area Network
PLC	Programmable Logic Controller
PWA	Progressive Web Application
SCADA	Supervisory Control And Data Acquisition
WLAN	Wireless Local Area Network

3. LOS PRODUCTOS

El resultado de este trabajo es un demostrador, el perímetro del proyecto es muy ambicioso respecto al tiempo del que he dispuesto, y por lo tanto he priorizado el desarrollo de las funciones y su optimización en detrimento de la estética del mismo. Así mismo, he dado prioridad al desarrollo de la interfaz gráfica del *SCADA* con el fin de llegar a medir el rendimiento de *Angular* para este tipo de aplicaciones. Finalmente, la arquitectura de la aplicación está muy meditada y orientada a simplificar la edición de nuevas vistas y/o de nuevos símbolos por parte de personal sin conocimientos de *Angular*. En las instalaciones industriales, el mantenimiento de los sistemas de control suele realizarlo el personal de la fábrica, el cual raramente tiene conocimientos de programación *Web.*

3.1. DESCRIPCIÓN SUMARIA

Los productos que he desarrollado son:

- *Simlib*: una librería reducida de símbolos para usar en las vistas de la interfaz gráfica,
- *Angular-RT*-IHM: Una interfaz gráfica de alta reactividad que cumpla las funcionalidades más básicas de la IHM de un *SCADA*
- Un conjunto de simuladores, que generen las señales para animar las vistas y que también reciban los comandos generados desde éstas para provocar reacciones en los diferentes símbolos,

Los próximos capítulos ofrecen más detalles sobre las funciones y la implementación de los mismos. Aunque inicialmente la había especificado, me ha faltado tiempo para desarrollar una aplicación que permita crear/editar las vistas del *front end*. En la memoria ya había previsto su especificación e incluso le había dado un nombre : *Drawview*. He optado por dejar los requisitos por si pudiera servir de idea para algún futuro trabajo final de máster.

3.1.1. *Simlib*

Todo *SCADA* ofrece al usuario una librería de símbolos para representar sus procesos gráficamente. Un *SCADA* de gran distribución necesita ofrecer una gran variedad de estos para aumentar las posibilidades de que cada usuario encuentre exactamente los símbolos que mejor representan el sistema que se quiere controlar y supervisar. En mi caso, sólo pretendo tener un conjunto reducido de símbolos que permita:

1. Probar la funcionalidad y el rendimiento de la interfaz gráfica.
2. Determinar cómo se implementarán los símbolos, ¿deberán ser componentes *Angular* desarrollados en *Typescript*? o ¿es relativamente simple crear un editor de símbolos de

alto nivel que genere el código a partir de un diseño gráfico realizado por un usuario con pocos o nulos conocimientos en programación?

Por todo esto, los símbolos que he desarrollado han sido seleccionados para permitir la representación de variables binarias, enteras o reales así como el envío de comandos de esos tipos. La Tabla 1 ofrece un poco más de detalle de los símbolos que implementaré.

Símbolo	Representación gráfica	Comportamiento
Piloto de estado binario (verde/rojo)		Rojo cuando la variable binaria es false. Verde cuando ésta es true. Se dibujará un marco en rojo alrededor del símbolo cuando el valor se considere no fiable.
Indicador numérico (ya sea de valores enteros o reales)		Fondo blanco cuando el valor es fiable y dentro de unos márgenes predefinidos. Fondo rojo cuando el valor está fuera de márgenes. Se dibujará un marco en rojo alrededor del símbolo cuando el valor se considere no fiable.
Interruptor		Un interruptor es un símbolo que debe enviar comandos y a la vez representar el cambio de estado de un equipo al aplicarse el comando. Por tanto, se representará en rojo cuando el equipo esté apagado (variable de powered on status es false) y en verde cuando el equipo esté encendido (variable de powered on status es true) Se dibujará un marco en rojo alrededor del símbolo cuando el valor de la variable de estado sea poco fiable.
Control analógico		Un control analógico es un símbolo que debe enviar comandos analógicos y a la vez mostrar el valor con el que se ha configurado el equipo que lo recibe. Por tanto, debe

Símbolo	Representación gráfica	Comportamiento
		estar asociado a una variable de comando y a otra de adquisición. Se dibujará un marco en rojo alrededor del símbolo cuando el valor mostrado sea poco fiable.

Tabla 1 Símbolos contenidos en Simlib

En todos los símbolos se hace referencia a la fiabilidad del valor mostrado. Más en particular, se tratan los casos en los que se ha perdido la comunicación con el proceso (ej, se ha perdido la comunicación entre el *SCADA* y el *PLC*) y, aunque se conoce el último valor recibido, no es seguro que este no haya evolucionado desde la desconexión.

3.1.2. *Angular-RT-IHM*

Angular-RT-IHM es una interfaz gráfica que debe cumplir los requisitos funcionales y de rendimiento de la IHM de un *SCADA* industrial. Si analizamos el uso que se hace de estos productos deberíamos ser capaces de:

- Animar los símbolos gráficos de una o varias vistas a partir de los valores recibidos del *back end*.
- Enviar comandos desde la interfaz que provoquen cambios en el proceso. Mostrar el resultado de la transmisión de estos.
- Mostrar y gestionar una lista de alarmas que pueden ser “reconocidas” por el usuario.
- Mostrar los *logs* con los eventos datados que se han realizado desde cada instancia de la interfaz (ej. qué comandos se han enviado y qué alarmas se han reconocido).
- Disponer de una IHM que se pueda ejecutar simultáneamente desde varios dispositivos y que la información que estos presentan esté perfectamente sincronizada.
- Mantener una tasa de refresco coherente con los tiempos de ciclo de los *PLC*s y/o de los protocolos industriales. Una frecuencia de refresco no inferior a 20Hz (tiempo de ciclo de 50ms) me parece un buen objetivo desde el cual podemos partir.
- Ser capaz de gestionar varias fuentes de datos simultáneamente. La gran diversidad de marcas y de modelos de autómatas provocan que en muchas instalaciones haya diferentes buses de *PLC*s para controlar diferentes procesos. Así por ejemplo, el sistema eléctrico puede estar controlado con una gama de *PLC*s A mientras que la parte mecánica lo está con la gama B. Para mi demostrador propongo gestionar señales en proveniencia de 3 fuentes de datos.

Por otro lado, el IHM se limita a mostrar datos sin realizar ninguna combinación o conversión de estos. Los cálculos previos que deseen realizarse son responsabilidad de un proceso que en nuestro caso se encontraría en el *back end*.

Angular RT IHM será desarrollado en *Angular*/*Angular material* y para optimizar su rendimiento usaré RxJS + una arquitectura basada en un patrón de flujo de datos Redux.

Para las pruebas puedo usar hasta 7 dispositivos conectados a una red local inalámbrica, 4 navegadores en *desktop* y 3 en *tablet*.

3.1.3. *Los simuladores*

Para realizar las medidas de rendimiento de la IHM voy a necesitar 3 tipos de simuladores "*back end*":

- Un simulador tipo "*Signal manager*" que generará cambios aleatorios y a alta frecuencia sobre una lista de señales y se los transmitirá mediante un *message broker* a las IHMs. En el sentido contrario, el simulador recibirá comandos de la IHM que podrán provocar cambios en las señales que éste emite.
- Un simulador tipo "*alarm manager*" que simulará el envío de alarmas y su cambio de estado ligado al reconocimiento de estas.
- Un simulador tipo "*event manager*" que registrará las acciones realizadas en las diferentes IHM y las difundirá para su posterior presentación en el *eventlog*.

Los simuladores se instalarán en tres *raspberris pi* que cuentan con un procesador Armv71 (32 bits y 4 núcleos).

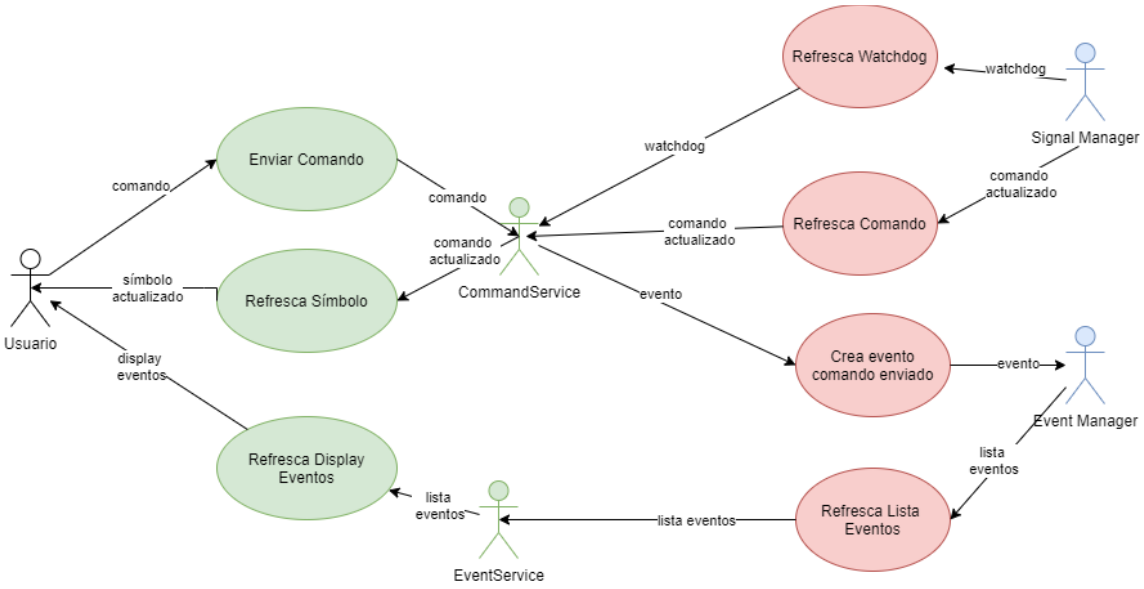
3.2. REQUISITOS TÉCNICOS

3.2.1. *Casos de uso*

Los requisitos funcionales que se presentan en esta sección responden a los siguientes casos de uso:

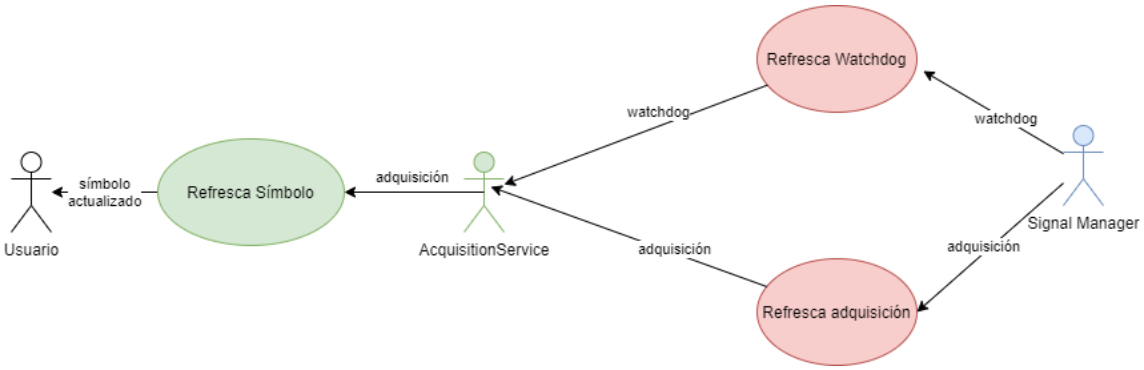
3.2.1.1. UC-0001 Ejecución de comandos

Este caso de uso detalla las acciones que se realizan cuando un usuario envía un comando desde la interfaz gráfica. Igualmente identifica los actores implicados.



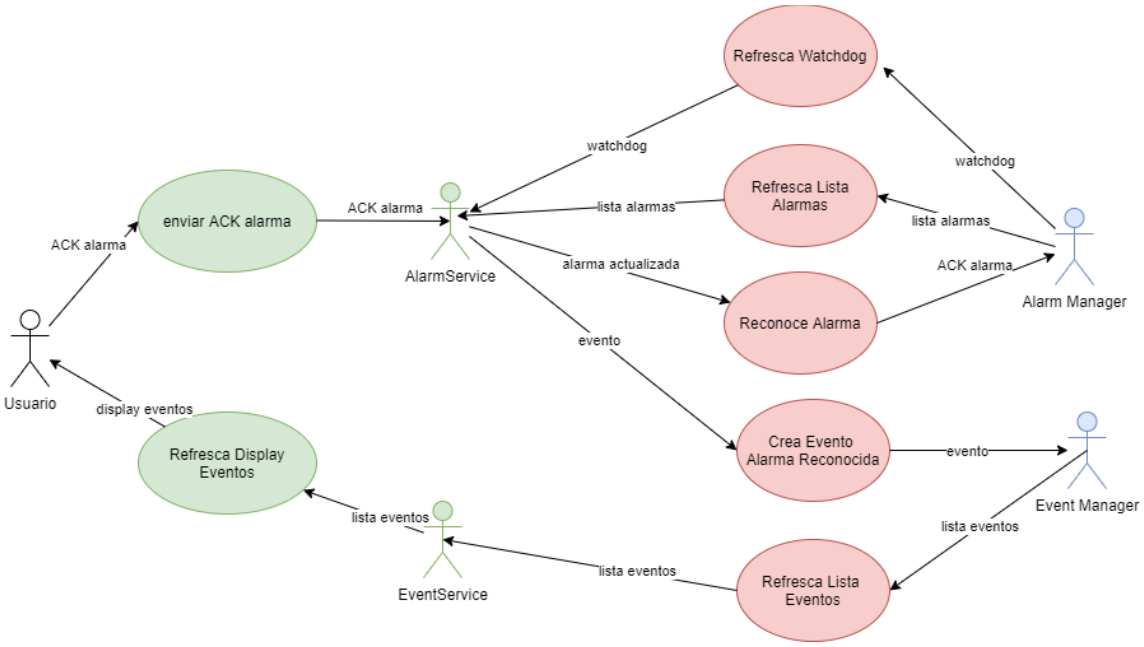
3.2.1.2. <u>UC-0002 Presentación de adquisiciones</u>

Este caso de uso detalla las acciones que se realizan cuando una señal cambia en una fuente de datos y se transmite ese cambio al usuario. Igualmente identifica los actores implicados.



3.2.1.3. <u>UC-0003 Tratamiento de alarmas</u>

Este caso de uso detalla las acciones que se realizan cuando una alarma se genera y es trasmitida al usuario así como cuando este último procede a reconocerla. Igualmente identifica los actores implicados.



3.2.2. *Requisitos funcionales*

3.2.2.1. Simlib

REQ-FUNC-000010	
Description	*Simlib* debe contener los siguientes símbolos: • Pilotos de estado binarios (verde/rojo). • Indicadores numéricos (ya sea de valores enteros o reales). • Interruptores. • Controles analógicos (ya sea de valores enteros o reales)..
Comentarios	
Justificación	
Método de verificación	Inspección

REQ-FUNC-000020	
Description	La definición de un símbolo incluirá la lista de tipos de comandos y de adquisiciones que se necesitarán para animarlo así como su apariencia para cada combinación de estas.

Comentarios	Ej el interruptor estará asociado a un comando ON, a un comando OFF y a un estado isOn. Su apariencia dependerá de si su variable ON es true o false.
Justificación	
Método de verificación	Inspección

REQ-FUNC-000030	
Description	Toda adquisición tendrá asociada una variable de validez. Si esta variable indica que el valor de la adquisición es inválido (poco fiable) el símbolo se animará rodeado de un marco rojo.
Comentarios	Ver UC-0002 Presentación de adquisiciones
Justificación	
Método de verificación	Inspección

REQ-FUNC-000040	
Description	El símbolo piloto: • Tendrá asociada una variable de estado binaria isOn. • Se mostrará rojo cuando isOn es false. • Se mostrará verde cuando cuando isOn es true. • Se dibujará un marco en rojo alrededor del símbolo cuando el valor se considere no fiable.
Comentarios	Ver UC-0002 Presentación de adquisiciones
Justificación	
Método de verificación	Test

REQ-FUNC-000050	
Description	El símbolo de indicador numérico:

	• Tendrá asociada una variable de estado entera o real de tipo value. • Tendrá asociada una máscara de formato de presentación del valor (tipo POSIX printf). • Mostrará el valor de acuerdo al formato. • Tendrá asociado un rango de valores dentro del cual el valor se mostrará como nominal (texto negro sobre fondo blanco). • Se dibujará un marco en rojo alrededor del símbolo cuando el valor se considere no fiable.
Comentarios	Ver UC-0002 Presentación de adquisiciones
Justificación	
Método de verificación	Test

REQ-FUNC-000060	
Description	El símbolo de interruptor: • Tendrá asociada una variable de estado binaria de tipo isOn. • Se mostrará rojo cuando isOn es false. • Se mostrará verde cuando cuando isOn es true. • Tendrá asociada un comando de tipo comandoOn y otro de tipo comandoOff. • Sólo permitirá el envío de comandoOn cuando la variable isOn sea false y fiable. • Sólo permitirá el envío de comandoOff cuando la variable isOn sea true y fiable. • Se dibujará un marco en rojo alrededor del símbolo cuando de la variable de estado se considere no fiable.
Comentarios	Ver UC-0001 Ejecución de comandos
Justificación	
Método de verificación	Test

REQ-FUNC-000090	
Description	El símbolo de control analógico: • Tendrá asociada una variable entera o real de tipo value. • Tendrá asociada una máscara de formato de presentación del valor (tipo POSIX printf). • Mostrará el valor de acuerdo al formato. • Tendrá asociado un rango de valores dentro del cual el valor se mostrará como nominal (texto negro sobre fondo blanco). • Tendrá asociado un comando entero o real de tipo setValue. • Incluirá un botón (Set) que desplegará un cuadro de texto en el que se podrá introducir el valor del comando setValue. • Se dibujará un marco en rojo alrededor del símbolo cuando el valor se considere no fiable.
Comentarios	Ver UC-0001 Ejecución de comandos y UC-0002 Presentación de adquisiciones
Justificación	
Método de verificación	Test

3.2.2.2. Drawview

REQ-FUNC-0000100	
Description	Drawview es el software de generación de vistas para su representación en *Angular-RT*-IHM:
Comentarios	
Justificación	
Método de verificación	Test

REQ-FUNC-000110

Description	Drawview debe permitir al usuario la inserción en las vistas de los símbolos de la librería *Simlib* así como la configuración de cada una de las instancias (variables asociadas, duración de los comandos en el pulsador temporal, rango de validez de los valores).
Comentarios	
Justificación	
Método de verificación	Test

REQ-FUNC-000120	
Description	Drawview debe permitir al usuario la inserción de las siguientes formas básicas no animadas: • conectores rectos y curvos. • rectángulos. • círculos y elipses. • etiquetas de texto.
Comentarios	
Justificación	
Método de verificación	Test

REQ-FUNC-000130	
Description	La inserción de símbolos y de formas no animadas en Drawview se realizará mediante el mecanismo *drag&drop.*
Comentarios	
Justificación	
Método de verificación	Test

REQ-FUNC-000140

Description	Drawview incluirá un mecanismo para la alineación vertical de formas y símbolos respecto a: • Su eje horizontal superior. • Su eje horizontal central. • Su eje horizontal inferior.
Comentarios	
Justificación	Test
Método de verificación	

REQ-FUNC-000150	
Description	Drawview incluirá un mecanismo para la alineación horizontal de formas y símbolos respecto a: • Su eje vertical izquierdo. • Su eje vertical central. • Su eje vertical derecho.
Comentarios	
Justificación	
Método de verificación	Test

REQ-FUNC-000160	
Description	Drawview incluirá un mecanismo para la superposición de formas y símbolos (Z) que permita: • Enviar una forma al fondo. • Enviar una forma al frente.
Comentarios	
Justificación	
Método de verificación	Test

REQ-FUNC-000170	
Description	Drawview tomará como parámetro de entrada la lista de todas las variables y comandos disponibles para ser usados en *Angular-RT-IHM.*
Comentarios	
Justificación	
Método de verificación	Test

REQ-FUNC-000180	
Description	Drawview realizará un control de coherencia de las variables usadas en la configuración de los símbolos respecto a las variables que se le pasan como parámetro.
Comentarios	
Justificación	
Método de verificación	Test

REQ-FUNC-000190	
Description	Drawview incluirá un mecanismo de compilación que genere vistas *Angular* en formato *HTML*). La compilación no será posible si el control de coherencia ha fallado previamente.
Comentarios	
Justificación	
Método de verificación	Test

3.2.2.3. Angular-RT-IHM

REQ-FUNC-000200	
Description	*Angular-RT-IHM* debe representar variables binarias, enteras o reales.
Comentarios	
Justificación	
Método de verificación	Test

REQ-FUNC-000220	
Description	*Angular-RT-IHM* recibirá cambios de variables desde tres fuentes de datos diferentes
Comentarios	Las variables de cada fuente de datos serán distintas entre ellas, no se pretende que la aplicación compare valores de una misma variable que le hayan llegado desde orígenes diferentes.
Justificación	Ver UC-0001 Ejecución de comandos, UC-0002 Presentación de adquisiciones y UC-0003 Tratamiento de alarmas
Método de verificación	Test

REQ-FUNC-000230	
Description	*Angular-RT*-IHM debe permitir el envío de comandos mediante la selección de éstos en un menú contextual.
Comentarios	Ver UC-0001 Ejecución de comandos
Justificación	
Método de verificación	Test

REQ-FUNC-000240	
Description	*Angular-RT-IHM* debe presentar permanentemente el estado de la comunicación con los procesos remotos ya sean los que gestionan los comandos enviados o bien con alguna de las fuentes de datos.
Comentarios	Ver UC-0001 Ejecución de comandos, UC-0002 Presentación de adquisiciones y UC-0003 Tratamiento de alarmas
Justificación	
Método de verificación	Test

REQ-FUNC-000250	
Description	*Angular-RT-IHM* debe presentar permanentemente el estado de su sincronización con un servidor *NTP* remoto.
Comentarios	
Justificación	
Método de verificación	Test

REQ-FUNC-000260	
Description	Tras el envío de un comando, *Angular-RT-IHM* debe mostrar al usuario el resultado de la transmisión: Éste será: • el comando que ha enviado no está definido en la aplicación. • el comando se ha enviado correctamente. • el comando no se ha podido enviar por un problema de comunicación con el *back end*.
Comentarios	Ver UC-0001 Ejecución de comandos, UC-0002 Presentación de adquisiciones y UC-0003 Tratamiento de alarmas
Justificación	
Método de verificación	Test

REQ-FUNC-000270	
Description	Tras la pérdida de comunicación con cualquier fuente de datos, *Angular-RT-IHM* marcará todos los símbolos que se animen con variables provenientes de las fuentes de datos perdidas como valores poco fiables.
Comentarios	Ver UC-0001 Ejecución de comandos y UC-0002 Presentación de adquisiciones
Justificación	
Método de verificación	Test

REQ-FUNC-000280	
Description	*Angular-RT-IHM* debe presentar una lista de alarmas que contenga todos los eventos relacionados con las alarmas que se hayan producido en las últimas 24 horas o, desde el último reinicio del *SCADA*. En particular: • Todas la activaciones de alarma. • Todas la desactivaciones de alarma. • Los reconocimientos de estas junto con el nombre del usuario que los ha realizado.
Comentarios	Ver UC-0003 Tratamiento de alarmas
Justificación	
Método de verificación	Test

REQ-FUNC-000290	
Description	*Angular-RT*-IHM debe permitir el reconocimiento de alarmas mediante la selección de éstos en un menú contextual.
Comentarios	Ver UC-0003 Tratamiento de alarmas
Justificación	
Método de verificación	Test

REQ-FUNC-000300	
Description	*Angular-RT*-IHM debe mostrar los eventos que se produzcan en cualquiera de sus instancias en todas ellas. En particular: • Ejecución de comandos. • Activación de una alarma. • Desactivación de una alarma. • Reconocimiento de una alarma.
Comentarios	Ver UC-0001 Ejecución de comandos, UC-0002 Presentación de adquisiciones y UC-0003 Tratamiento de alarmas
Justificación	
Método de verificación	

3.2.3. *Requisitos de rendimiento*

REQ-PERF-000010	
Description	*Angular-RT*-IHM debe permitir la representación de hasta 50 símbolos por vista
Comentarios	
Justificación	
Método de verificación	Test

REQ-PERF-000020	
Description	*Start render time* en cualquier vista de *Angular-RT-IHM* debe ser igual o inferior a 1 segundo.
Comentarios	
Justificación	
Método de verificación	Test

REQ-PERF-000030	
Description	*Time to interact* en cualquier vista de *Angular-RT*-IHM debe ser igual o inferior a 2 segundos.
Comentarios	
Justificación	
Método de verificación	Test

REQ-PERF-000040	
Description	*Angular-RT*-IHM debe soportar una tasa de refresco de10Hz con 7 instancias ejecutándose simultáneamente
Comentarios	Se deberá probar con 1500 variables cambiando a 10Hz : 100 son las que se mostrarán en las vistas abiertas y el resto servirá para comprobar que el filtro de los símbolos a modificar se realiza a buena velocidad.
Justificación	
Método de verificación	Test

REQ-PERF-000050	
Description	Un comando enviado desde *Angular-RT-IHM* debe llegar al *back end* en un tiempo no superior a 100 ms.
Comentarios	
Justificación	
Método de verificación	Test

REQ-PERF-000060

Description	El reconocimiento de una alarma realizado desde una instancia de *Angular-RT-IHM* debe representarse en las otras instancias en un tiempo no superior a 1 segundo.
Comentarios	
Justificación	
Método de verificación	Test

4. DISEÑO E IMPLEMENTACIÓN

4.1. *SIMLIB*

4.1.1. *El modelo de datos*

Los símbolos de la librería *Simlib* se basan en el modelo de datos presentado en la Figura 2, siendo el modelo de símbolo el que se identifica como *Sign* en la figura (*Symbol* es palabra reservada). Cada símbolo cuenta con un identificador de órgano que se representa, por ejemplo una instancia de un sensor, de un interruptor, de una batería. Los datos para animar cada instancia de símbolo son estáticos, se generan durante la fase de preparación de las vistas y se gestionan dentro de ficheros JSON. Cada símbolo integra:

- un atributo de validez: que permitirá modificar la apariencia del componente para que el usuario identifique la fiabilidad del estado que se muestra,
- un camino hacia el directorio donde se encuentran las imágenes que componen la animación del símbolo. Éste directorio será común a todas las instancias,
- una lista de las medidas (adquisiciones) que servirán para animar el símbolo,
- una lista de las órdenes (comandos) que servirán para animar el símbolo y que podrán generarse desde el símbolo,
- una lista de las condiciones que se evaluarán para la animación del componente.

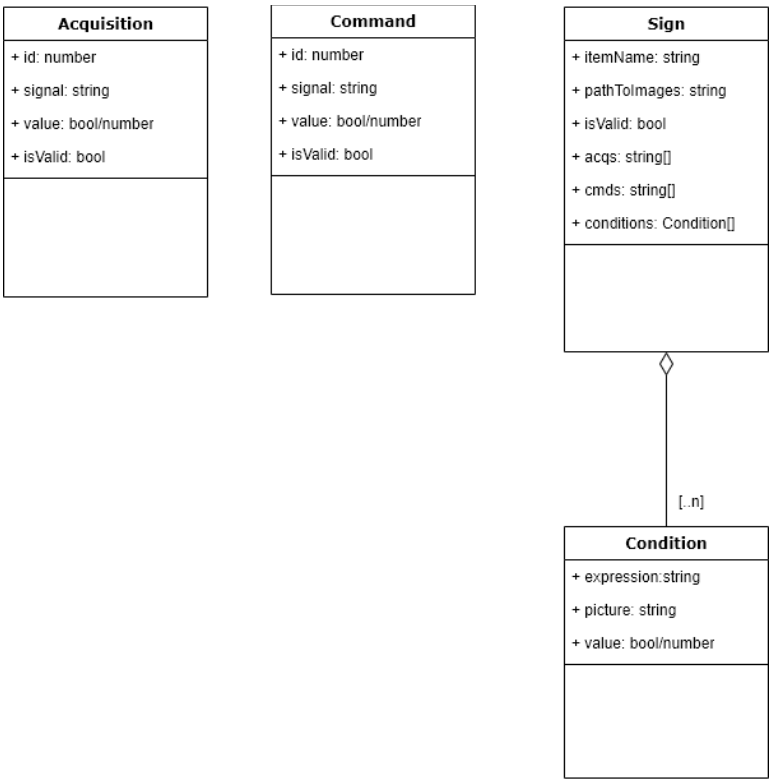


Figura 2 Símbolos : Modelo de datos

Cada condición se compone de una expresión *javascript* compuesta por variables de adquisición y de comandos, que de cumplirse, forzará que se muestre la imagen asociada, cuyo nombre de fichero se guarda en el campo *picture*. Además, el resultado de evaluar la expresión se guarda en el campo *value*.

Todas las instancias de símbolos de la misma clase deberían guardar cierta coherencia en términos de la expresión que se evalúa y de las imágenes que se muestran. Es una coherencia que debe asegurarse durante la fase de preparación de las vistas. Un ejemplo de símbolo se muestra a continuación:

```
{
    "itemName": "lamp1Status",
    "pathToImages": "assets/light-indicator/",
    "isValid": true,
    "acqs": [
        "lamp1Status_on"
    ],
    "cmds": [],
    "conditions": [
        {
            "expression": "lamp1Status_on",
            "picture": "light-indicator_on.png",
            "value": false
        },
        {
            "expression": "!lamp1Status_on",
            "picture": "light-indicator_off.png",
            "value": true
```

```
            }
        ]
    },
```

El símbolo muestra el estado del *item Lamp1*. Tiene asociada la medida (*lamp1status_on*). Cuando esta es true, la primera condición de la lista se cumple y en la vista se muestra la imagen *assets/light-indicator/light-indicator_on.png.* En caso contrario, se mostrará la imagen *assets/light-indicator/light-indicator_off.png*

El código siguiente corresponde a un símbolo que se anima exclusivamente con comandos:

```
{
    "itemName": "switch3",
    "pathToImages": "assets/switch/",
    "isValid": true,
    "acqs": [],
    "cmds": [
        "switch3_on",
        "switch3_off"
    ],
    "conditions": [
        {
            "expression": "switch3_on",
            "picture": "switch_on.png",
            "value": false
        },
        {
            "expression": "switch3_off",
            "picture": "switch_off.png",
            "value": true
        }
    ]
},
```

El símbolo muestra el estado del *item switch3*. Tiene asociado el comando (*switch3_on*). Cuando este es *true* (se ha cerrado el interruptor), la primera condición de la lista se cumple y en la vista se muestra la imagen *assets/switch/switch_on.png*. En caso contrario, se mostrará la imagen *assets/switch/switch_off.png*.

4.1.1.1. Los componentes

Para la animación de los símbolos y buscando que el usuario a cargo de la preparación de las vistas no necesite tener grandes conocimientos de programación, he creado un componente *AbstractSymbolComponent* del que deberán derivar todos los otros componentes símbolos. Este componente contiene un menú contextual (*contextMenu*) que listará los comandos que se pueden enviar desde una instancia de símbolo.

Por otro lado, *AbstractSymbolComponent* gestiona una lista de observables *Acquisition* y otra de observables *Command* a las cuales se subscribe para gestionar la evolución de la animación según los valores de las diferentes señales. Un símbolo sólo se subscribe a las señales que tiene asociadas, durante las pruebas de rendimiento, he detectado que el rendimiento disminuye

conforme aumenta el número de símbolos que se muestran en pantalla. Si los símbolos se subscribieran a la totalidad de señales del sistema, sus *callbacks* estarían continuamente solicitados, lo que afectaría al rendimiento de su animación.

También se han implementado cuatro *stores*, para las adquisiciones, para los comandos, para las alarmas y para los eventos En el caso de los símbolos, tienen *observables* sobre los de adquisiciones y comandos que reaccionan a cambios en cualquiera de ellos y provocan la ejecución del método *resolveConditions*. Este método permite evaluar las condiciones del símbolo ante cualquier cambio en las variables implicadas. Estos dos *stores* están conectados a dos servicios (*AcquisitionsService* y *CommandService*) que se están subscritos al *Sign*alManager del *back end*,

El menú contextual de cada símbolo contiene los comandos que están disponibles para enviar en cada momento. En esta aplicación se trata de los comandos que no se han enviado, por ejemplo, si un interruptor está *on*, sólo se podrá enviar el comando *off* y viceversa. Este mecanismo permite integrar gestión de privilegios por perfiles, ciertos comandos críticos podrían estar reservados a un perfil determinado de usuario. El menú se crea dinámicamente durante su inicialización y se actualiza cada vez que se produce un cambio en uno de los comandos implicados.

Para acabar, es importante remarcar que los símbolos se crean dinámicamente a la carga de los sinópticos mediante llamadas a sus *factories*. Es por esta razón que los cambios en estos componentes no se propagan desde el componente padre y que debemos forzar su propagación. Para ello al componente Abstract*Symbol*Component se le inyecta un *ChangeDetectorRef* cuyo método *detectChanges*() permite forzar la detección de cambios en su animación.

4.1.1.2. Los templates

La implementación de Abstract*Symbol*Component permite, por herencia, crear símbolos que simplemente difieren de éste en su plantilla HTML. Esto facilitaría la creación de una herramienta "gráfica" que generara los símbolos sin necesidad de programar. La herramienta podría generar la plantilla *HTML* y el fichero *Typescript* de manera mecánica y repetitiva.

De las plantillas de los símbolos cabe destacar que la invalidez se gestiona mediante directivas *ngIf*. Cuando una instancia de un símbolo procesa valores inválidos de las variables asociadas, esta se dibuja con un marco rojo que se genera mediante un gráfico *SVG*. El código siguiente muestra un ejemplo.

```
<div *ngIf="isValid; else withFrame">
    <div [matMenuTriggerFor]="contextMenu" (contextmenu)="onContextMenu($event)">
        <img src="{{ currentPicture}}" width="56" height="34" />
    </div>
</div>
<ng-template #withFrame>
    <div [matMenuTriggerFor]="contextMenu" (contextmenu)="onContextMenu($event)">
        <svg width="72" height="58">
```

```
                <rect x="0" y="0" width="72" height="58"
                    style="fill:blue;stroke:red;stroke-width:10;fill-opacity:0;stroke-
opacity:1" />
                <image x="5" y="5" width="62" height="48" [attr.xlink:href]="currentPicture"
/>
            </svg>
        </div>
    </ng-template>
```

4.2. *ANGULAR-RT-IHM*

Angular-RT-IHM permite:

- la supervisión en tiempo real del estado de los simuladores,
- la visualización de sinópticos,
- la visualización de eventos,
- la visualización y el reconocimiento de alarmas.

4.2.1. *La infraestructura*

Angular-RT-IHM cuenta con un servicio *DataService* que carga los ficheros estáticos (JSON) al inicializarse la aplicación, procesa la información que contienen, genera las estructuras de datos para facilitar su uso (generalmente mapas) y lo pone a disposición del resto de componentes. Este servicio está implementado como un *singleton*, lo que permite asegurar que sólo se carga una vez y que mantiene los datos en memoria.

La inicialización de la aplicación se realiza ejecutando el siguiente código integrado en el fichero app.module.ts:

```
export function appInit(dataConfigSvc: DataService) {
  return () => dataConfigSvc.load();
}
  providers: [AcquisitionsService, DataService,
    {
      provide: APP_INITIALIZER,
      useFactory: appInit,
      multi: true,
      deps: [DataService]
    }
  ]
```

Por otro lado, los componentes de la IHM están contenidos en el componente *ihm-framework*. Éste contiene:

- una cabecera con:
    - un reloj que se refresca a partir del tiempo sistema de la máquina en la que se ejecuta el *front end*
    - un componente *health-display* con tres indicadores gráficos que muestran el estado de los generadores de señales, eventos y alarmas. Este componente está subscrito a unas variables de *watchdog* gestionadas en los servicios

acquisition, *alarm* y *event*. Los servicios gestionan temporizadores que anulan las variables de *watchdog* cada 90ms si éstas no han sido previamente refrescadas por los servicios. Éstos reciben la variable de *watchdog* de los managers vía *SocketIO*. Dado que las señales de *watchdog* sólo se emiten desde el *back end* y que este no necesita ser informado de un eventual cambio de estado en el *front end*, estas no son gestionadas en ningún store.

- un grupo de tabuladores que permite acceder a los componentes de visualización de sinópticos, de eventos o de alarmas.

4.2.2. *La visualización de sinópticos*

De cara a simplificar la preparación de las vistas, mi intención inicial era crear un componente sinóptico genérico al cual se le pudiera pasar su plantilla extraída de un fichero. De esta manera, la herramienta *Drawview* no necesitaría generar código de componente y, tal y como ya se hace en el caso de los símbolos, sólo sería necesario generar el código *html* de la vista.

Esta estrategia no es posible con *Angular* porque la plantilla se carga en tiempo de compilación. Quedaba la alternativa de crear una plantilla única que contuviera la totalidad de plantillas de sinópticos asociadas a una directiva con el identificador de cada vista, tal y como se muestra en el código siguiente:

```
<ng-template #syno_1 [appLoadSynoTemplate]="'syno_1'">
Syno 1 is loaded
        <div class="flex-box">
        <SymbolContainer #myLight itemName="lamp1Status"></SymbolContainer>
        <SymbolContainer #lightIndicatorContainer itemName="lamp2Status"></SymbolContainer>
        <SymbolContainer #lightIndicatorContainer itemName="lamp3Status"></SymbolContainer>
        <SymbolContainer #lightIndicatorContainer itemName="lamp4Status"></SymbolContainer>
        </div>
</ng-template>
```

La aplicación implementa un componente *SynoManagerComponent* que contiene un árbol de vistas y que, al seleccionar cualquiera de las hojas del árbol, provoca la carga del componente *SynopticComponent* con la plantilla asociada (ej. Syno_1 para el primer sinóptico, Syno_2 para el segundo, etc.

El problema radica en que *SynopticComponent* ya es un componente que descubre los símbolos definidos en los *Symbol*Container y crea las instancias de esos símbolos dinámicamente.

Angular no detecta contenedores dentro de *ng-template/ng-content* por lo que los símbolos contenidos en las plantillas de los sinópticos no se llegan nunca a dibujar. La directiva *ng-template* de *Angular* es una declaración de intenciones que se transforma en *ng-content* en el momento en que se dibuja. Durante este proyecto no he logrado conseguir que ninguno de los dos componentes (*SynoManagerComponent* y su hijo *SynopticComponent*) detecten los símbolos que se tienen que instanciar, mediante llamadas a *ViewChildrenContent* y/o *ContentChildren*

Al final he optado por mantener la misma estrategia que había funcionado con los símbolos y he creado un componente *AbstractSynopticComponent* cuya única función es crear los componentes de la plantilla (i.e. los símbolos) de manera dinámica y a partir de *factories*.
Los diferentes sinópticos heredan de este componente y se diferencian entre ellos en la implementación de la plantilla HTML.

4.2.3. *La visualización de eventos*

Los eventos se presentan en un *"virtual scroll"* de *Angular* como una lista.
El componente *event-display* se subscribe al store de eventos el cual, a su vez, tiene un interfaz directo con el servicio de eventos.

4.2.4. *La visualización y el reconocimiento de alarmas*

Las alarmas se presentan en una tabla de *Angular* con paginador. Aparte, se aplican estilos diferentes según si han sido reconocidas o no.
El componente *alarm-display* se subscribe al store de alarmas el cual, a su vez, tiene un interfaz directo con el servicio de alarmas.

4.3. LOS SIMULADORES

Tras una primera prospección y mucha lectura sobre los pros y los contras sobre los diferentes sistemas de mensajería disponibles, he optado por usar *ZeroMQ*. Esta librería parece ser la que mejor ratio de rendimiento versus complejidad de uso presenta. Por otro lado, aunque las *APIs* están disponibles en todos los lenguajes imaginables, he escogido *nodejs* por ser multiplataforma.

Por otro lado, *ZeroMQ* aún no soporta *Websockets* que, por lo que he podido deducir después de una prospección intensiva, es la única tecnología que permite una comunicación bidireccional en tiempo real con una aplicación web. Esto complica su integración en *Angular* y, por lo tanto, he implementado la comunicación final entre el *front end* y el *back end* con socket.io (ver Figura 3).

Figura 3 Comunicación entre el back end y el front end

El lector se puede preguntar por qué he seguido manteniendo *ZeroMQ* en mi proyecto y la respuesta está relacionada con la representatividad de mi prototipo. En un *SCADA* el *front end* es sólo un componente, antes de que los datos le sean enviados, éstos han pasado por varios

procesos que aseguran, entre otros, la adquisición desde el *PLC*, su conversión digital a analógica, la comparación respecto a márgenes preestablecidos para decidir si los valores remontados son motivo de alarma, etc. Estos procesos necesitan comunicarse entre ellos y un sistema de mensajería como *ZeroMQ* suele ser una muy buena opción, por lo tanto éste es el protocolo desde el que se va a iniciar la generación de los datos que llegarán a nuestro *front end*.

La implementación de los simuladores responde al diagrama de clases que se muestra en la Figura 4.

En este diagrama de clases se pueden observar tres bloques diferenciados:

- Los procesos "*generator*" son aquellos que generarán los datos que queremos ver refrescados en el *front end*. Éstos contienen cada uno una instancia de *Publisher* que es la clase que implementará el socket *ZMQ.Publisher*. Por este socket se emitirán los datos aplicativos (adquisiciones, alarmas o eventos) y también se emitirá una señal de *watchdog* que permitirá al *front end* conocer el estado de la comunicación con el proceso final, para así poder invalidar los símbolos que muestra al usuario en caso de que ésta se haya interrumpido. Cada proceso *generator*
- Los procesos "manager" son aquellos que reciben las publicaciones de los generadores (*ZeroMQ*) y las transmiten al *front end* mediante *Websockets SocketIO*. Cada proceso *manager* incluye un *Subscriber* especializado. Los *Subscriber*s reciben los datos publicados por el "*Publisher*" asociado y los retransmiten por todos aquellos *Websockets* que tengan conectados. Los *Subscriber* se especializan para simplificar el tratamiento de subscripciones que llegan del *front end*. Cada uno de los subscriptores tendrá un interfaz directo a uno o dos servicios del *front end* :
    - *SignalSubscriber* comunica con el *AcquisitionsService* y con el *CommandsService* del *front end*.
    - *AlarmSubscriber* comunica con el *AlarmsService* del *front end*.
    - *EventSubscriber* comunica con el *EventsService* del *front end*.
- Finalmente, los procesos *Stub* tienen por objetivo la recepción de los datos que cambian en el *front end* (comandos + estado de las alarmas) y su registro en el fichero de log para, más tarde, medir el tiempo que los cambios tardan en viajar por la aplicación.

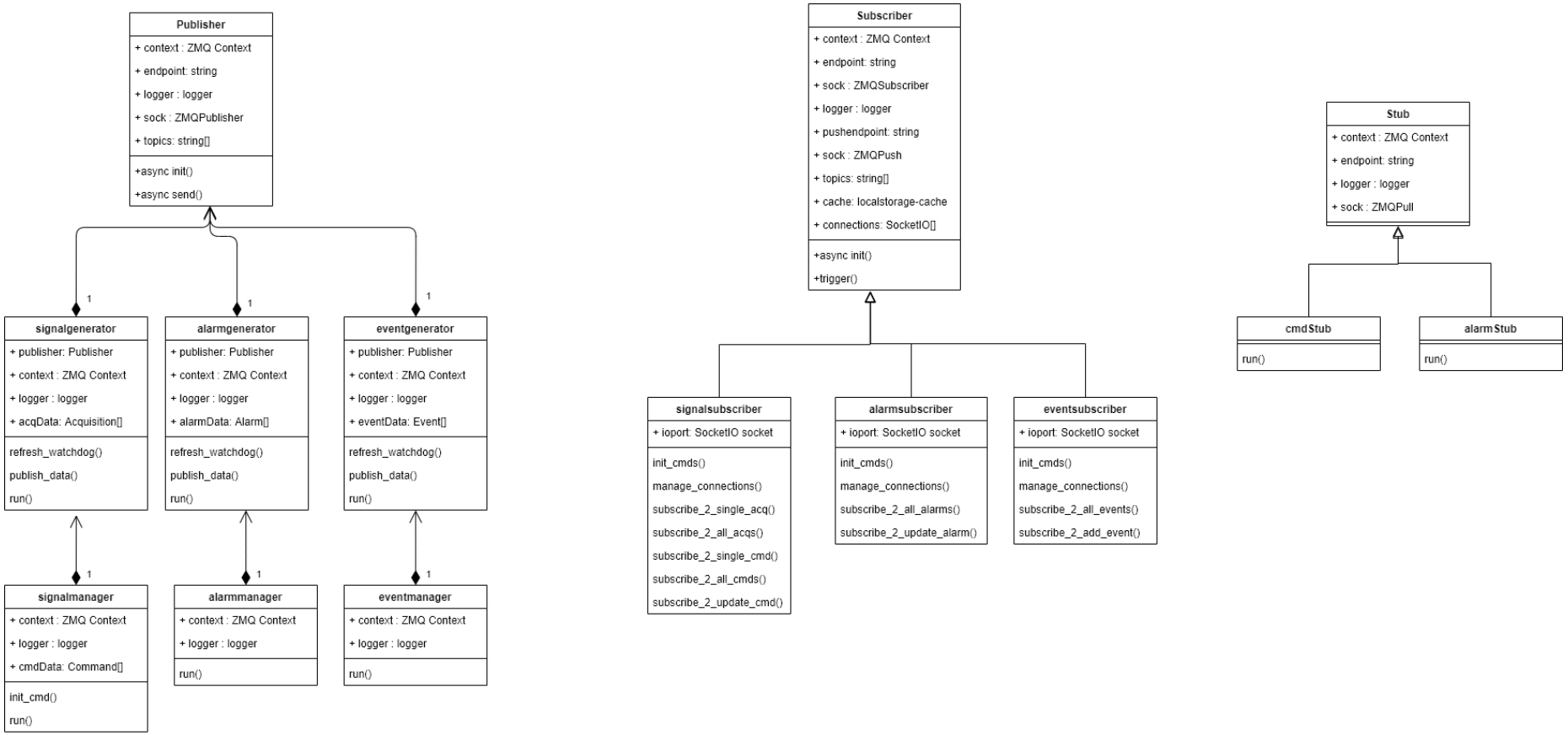


Figura 4 : Diagrama de clases de los simuladores

La distribución de los simuladores en los diferentes hosts se muestra en la Figura 5 y responde al rendimiento de cada máquina.

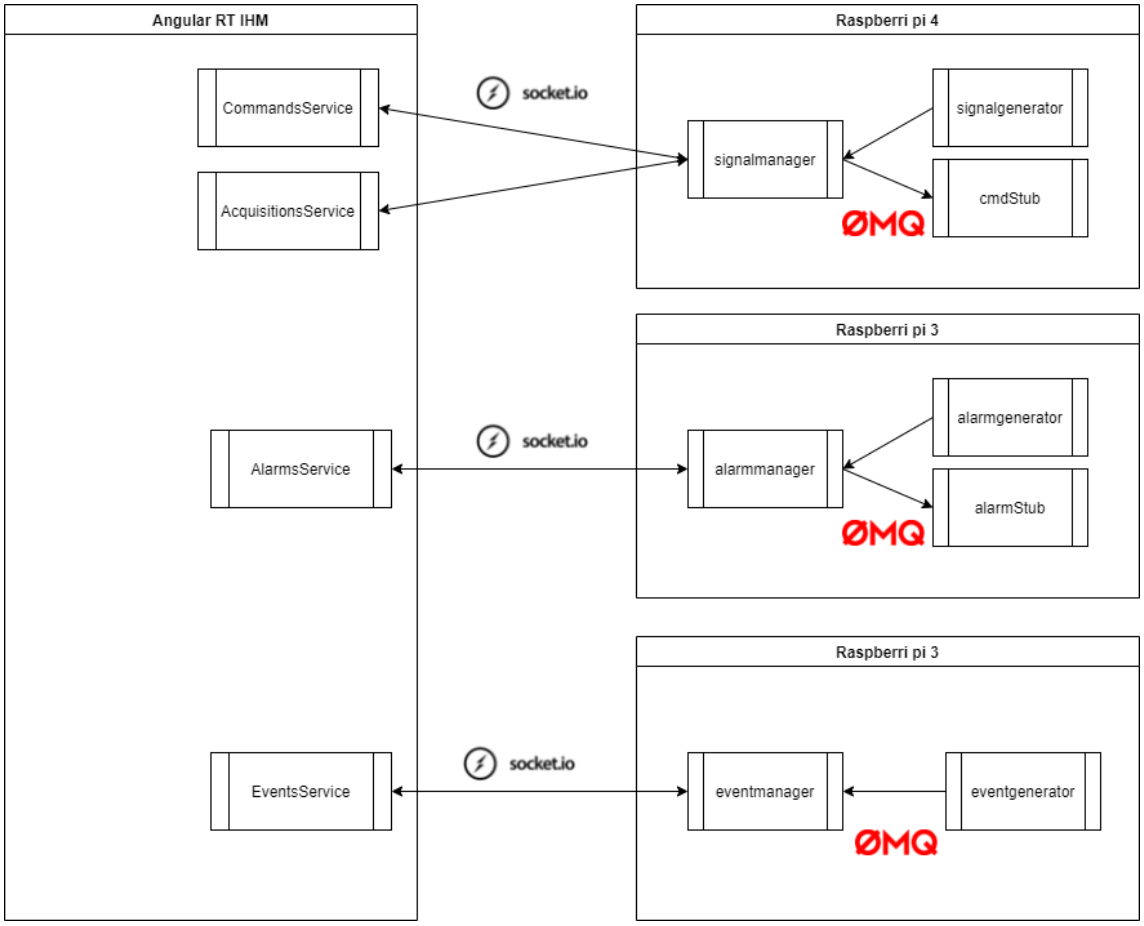


Figura 5 Diagrama de despliegue de los simuladores

Es importante resaltar los sentidos de la comunicación que se muestran en la Figura 5:

- Los generadores sólo publican mensajes *ZeroMQ*.
- Los managers se subscriben a los tópicos de los generadores (*ZeroMQ*) y cuando necesitan emitir los comandos o los reconocimientos de alarmas que reciben del *front end*, utilizan un *socket Push* de *ZeroMQ*.
- Los servicios del *front end* se subscriben a los managers (*SocketIO*) mediante enlaces bidireccionales. El *front end* puede así utilizar la misma conexión para, por ejemplo, recibir alarmas o para enviar los reconocimientos de éstas.

4.4. LAS TECNOLOGÍAS

La Tabla 2 presenta las tecnologías que se han usado en los productos principales del proyecto

Producto/componente	Tecnología/herramienta
Simlib	Los símbolos son componentes *Angular*. Ver diseño y detalles de implementación en §**¡Error! Marcador no definido.**
***Angular-RT*-IHM**	Se usará *Angular* con RxJS + Redux. Ver diseño y detalles de implementación en §4.2
Simuladores	*ZeroMQ* y *SocketIO* tal y como he comentado en §4.3

Tabla 2 Tecnologías propuestas para el desarrollo de los diferentes productos

4.5. LAS PRUEBAS

4.5.1. *La plataforma de pruebas*

La **¡Error! No se encuentra el origen de la referencia.** muestra la plataforma de pruebas que he implementado e utilizado, tanto las pruebas funcionales como para las pruebas de rendimiento.

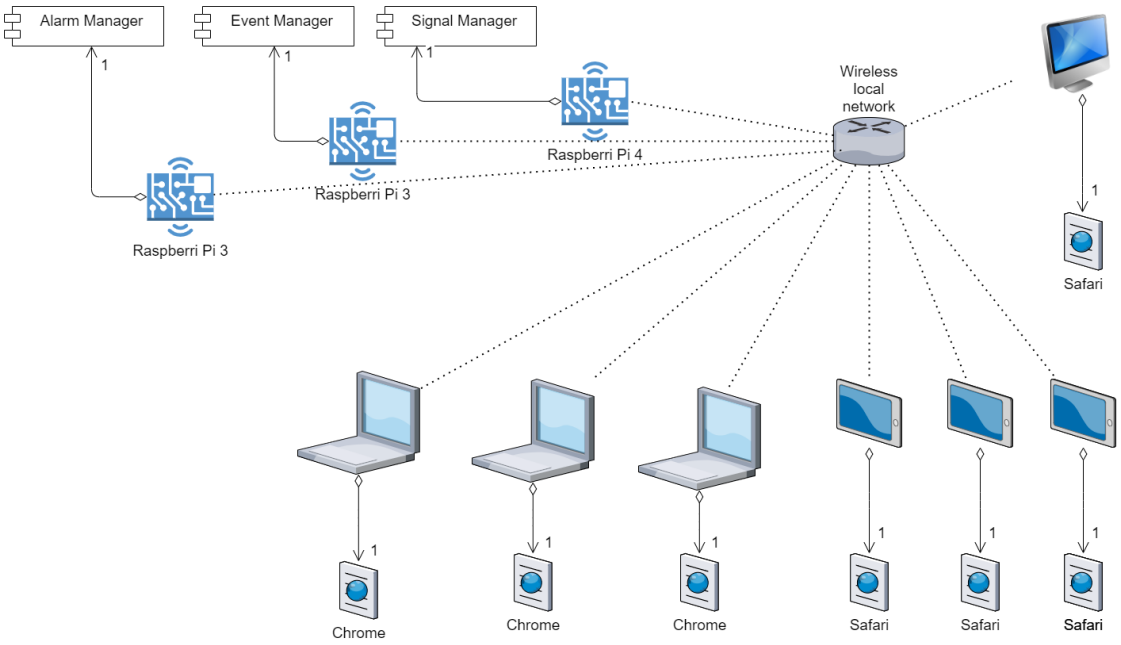


Figura 6 Plataforma propuesta para las pruebas funcionales y de rendimiento

Esta plataforma contiene las siguientes máquinas:

- 3 *raspberri pi* que juegan el rol de *back end*. Estas máquinas disponen de un procesador con 4 cores y, aunque mi idea inicial era usar una sola para alojar los tres simuladores, he tenido que acabar distribuyéndolos en 3 *hosts* diferentes porque una sola no aguantaba la carga.

- 1 ordenador de sobremesa Imac con navegador *Safari* el cual es, además, el servidor de *front end.*
- 3 portátiles, 2 de ellos con sistema operativo *Windows* y navegador *Chrome*. El tercero con sistema operativo *Linux* y navegador *Mozilla Firefox*.
- 2 tabletas tipo *Ipad* con navegador *Safari* y 1 tableta *Android* con navegador *Chrome*.

Todas estas máquinas se conectan a una *WLAN* compuesta por 3 *routers* conectados en *LAN* (con cables) y distribuidos en tres niveles de un edificio. La edad de las máquinas cliente es variable, algunas tienen hasta 10 años, pero las versiones de los navegadores es actual.

4.5.2. *Los casos de prueba funcionales*

Las pruebas se han iniciado siempre con las siguientes condiciones iniciales:

- los tres simuladores en marcha y el nivel de debug a "info" (para que la escritura en disco no penalizara el rendimiento). Las señales de watchdog de los tres *back ends* se transmiten en ciclos de 50 ms hacia los *front ends.* Inicialmente probé con ciclos de 25 ms pero *Socket.io* no era capaz de gestionar las conexiones.
- El *signal manager* genera cambios en 1500 adquisiciones cada 100 ms.
- El *alarm manager* inicializa una lista de 500 alarmas que sólo cambian por interacción del usuario.
- El *event manager* inicializa una lista de 200 eventos que se amplía con los eventos que se generan automáticamente durante el uso de la aplicación.

Los casos nominales de prueba han sido los siguientes:

1. Visualización de un sinóptico con 50 símbolos. He verificado que todos los símbolos con adquisición asociada se actualizaban simultáneamente en todas las máquinas cliente. He tenido que aumentar el periodo de refresco a 500 ms porque, en caso contrario, las pérdidas de comunicación con el signal manager eran demasiado frecuentes (varias por minuto).
2. Envío de comandos desde un sinóptico. Verificación en todos los clientes del cambio de estado del símbolo, así como de la aparición de nuevos eventos indicando el envío de cada uno de los comandos.
3. Visualización de eventos en cada uno de los clientes.
4. Visualización de la lista de alarmas en cada uno de los clientes. Reconocimiento de una alarma en uno de ellos y verificación en todos ellos del cambio de estado de la alarma así como de la aparición de un evento indicando este cambio de estado.

Los casos degradados de prueba han sido los siguientes:

1. Visualización de un sinóptico con 50 símbolos. Interrupción del *Signal manager*, verificación de que todos los símbolos se invalidan en todos los *front ends* y de que el piloto asociado en el *health display* indica una pérdida de comunicación con este

servicio. Comprobación de la aparición del evento de pérdida de comunicación en la lista de eventos.

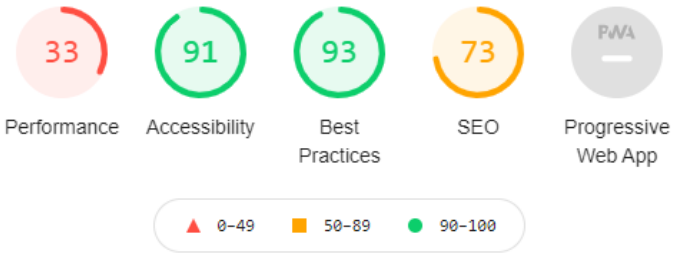
2. Reconexión del *Signal manager.* Verificación de que todos los símbolos se validan en todos los *front ends* y de que el piloto asociado en el *health display* indica el establecimiento de comunicación con este servicio. Comprobación de la aparición del evento de restablecimiento de comunicación en la lista de eventos.
3. Desconexión del *Alarm manager*. Verificación de que la lista de alarmas se vacía en todos los *front ends* y de que el piloto asociado en el *health display* indica una pérdida de comunicación con este servicio.
4. Reconexión del *Alarm manager*. Verificación de que la lista de alarmas se rellena en todos los *front ends* y de que el piloto asociado en el *health display* indica un restablecimiento de comunicación con este servicio.
5. Reconexión del *Event manager*. Verificación de que la lista de eventos se rellena en todos los *front ends* y de que el piloto asociado en el *health display* indica un restablecimiento de comunicación con este servicio.

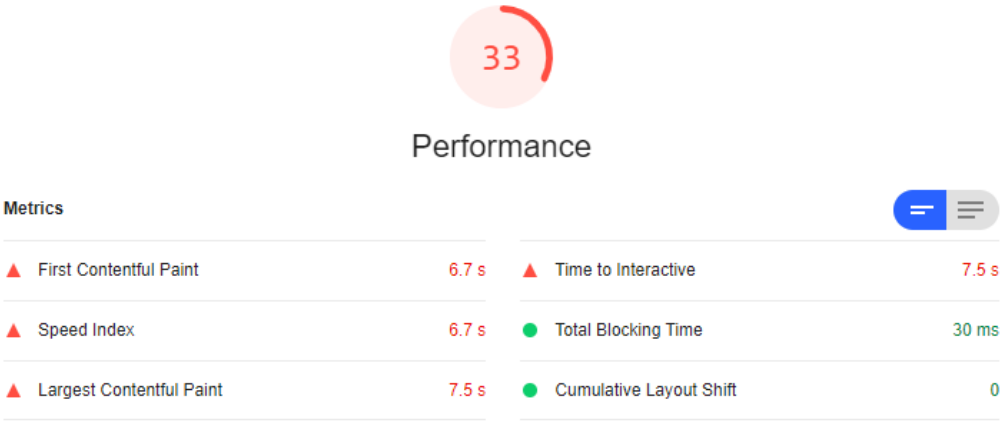
5. ANÁLISIS DE RESULTADOS

Los resultados de las pruebas funcionales muestran que la aplicación implementa correctamente las funciones que se le asignan. Los datos que se muestran en todos los *front ends* son idénticos y esto es debido al uso del patrón de flujo de datos Redux que centraliza los cambios.
No obstante, el rendimiento no es el esperado. Los principales problemas son:

- con 1500 actualizaciones de símbolos cada 100 ms, las pérdidas de comunicación con el *signal manager* son demasiado frecuentes (1 cada 10 segundos). He aumentado el tiempo de refresco a 500 ms y la aplicación se muestra bastante más estable, no obstante, se observa una degradación del comportamiento con el aumento de *front ends*. A más *front ends,* más frecuentes son las pérdidas de comunicación con los simuladores (en particular con el *signal manager).*
- Si reducimos el número de señales que cambian en cada ciclo, el comportamiento es mucho más estable y las pérdidas de comunicación disminuyen.
- Si reducimos el número de *front ends* conectados, el comportamiento es mucho más estable y las pérdidas de comunicación disminuyen. Quedaría ver si el problema está ligado a la capacidad de los simuladores para refrescar todas las subscripciones activas o si es una limitación del *framework.*
- El *scroll view* de eventos deja de dibujarse cuando el número de estos aumenta demasiado rápido (que es cuando se producen pérdidas de comunicación y se generan los eventos asociados).
- El paginador de alarmas deja de funcionar cuando la lista de alarmas supera las 200 unidades.

- El análisis *Lighthouse* sobre la carga de la aplicación en los *front ends* es el siguiente:





Los "problemas" de rendimiento son principalmente debidos a la enorme cantidad de datos que viajan por la red. Si tenemos en cuenta el tipo de aplicación con el que estamos tratando, sería relativamente simple instalar la aplicación como una PWA y reducir así el coste en tiempo que supone su carga inicial.

6. EL PROYECTO

6.1. PLAN DE PROYECTO

La Tabla 3 lista las tareas en las que he descompuesto el proyecto. Están ordenadas en el tiempo, si bien, hay dos tareas que pueden avanzarse o que pueden realizarse en paralelo de las otras. Se trata de las tareas 3 y 6.

Posición	Tarea	Estado de cumplimiento al entregar la PAC2
Tarea 1	Desarrollo de los símbolos.	100%
Tarea 2	Diseño de *Angular-RT*-IHM. Prototipo del *front end.*	100%
Tarea 3	Diseño e implementación "*Signa*l manager", "alarm manager" y "event manager".	100%

Posición	Tarea	Estado de cumplimiento al entregar la PAC2
Tarea 4	*Angular-RT*-IHM: pruebas funcionales.	100%
Tarea 5	Pruebas de rendimiento de *Angular-RT*-IHM.	100%

Tabla 3 Tareas del proyecto

6.2. DESCRIPCIÓN DETALLADA DE LAS TAREAS

Tarea 1		**Desarrollo de los símbolos**	
Fecha prevista de inicio	02/03/2021	**Fecha prevista de fin**	14/03/2021
Entradas			
• Requisitos funcionales de entrada de *Simlib*.			
Actividades			
• Diseño de la plantilla de un símbolo. • Desarrollo de los símbolos especificados. • Desarrollo de las pruebas unitarias para los símbolos. • Diseño e implementación de una vista. • Tests de integración símbolos-vista.			
Salidas			
• Componente de pilotos de estado binario validado unitariamente. • Componente indicador numérico (ya sea de valores enteros o reales) validado unitariamente. • Componente pulsador validado unitariamente. • Componente interruptor validado unitariamente. • Componente batería (con nivel de carga porcentual) validado unitariamente. • Componente control analógico validado unitariamente. • Prototipo de vista.			

Tarea 2		**Diseño de *Angular-RT*-IHM. Prototipo del *front end***	
Fecha prevista de inicio	15/03/2021	**Fecha prevista de fin**	11/04/2021
Entradas			
• Requisitos funcionales de entrada de *Angular-RT*-IHM. • Componente de pilotos de estado binario validado unitariamente.			

- Componente indicador numérico (ya sea de valores enteros o reales) validado unitariamente.
- Componente pulsador validado unitariamente.
- Componente interruptor validado unitariamente.
- Componente batería (con nivel de carga porcentual) validado unitariamente.
- Componente control analógico validado unitariamente.
- Prototipo de vista.

Actividades

- Diseño e implementación del módulo de autenticación (contra un servicio fake).
- Diseño e implementación de las ventanas de la IHM.
- Diseño e implementación del store.
- Pruebas de integración IHM + store.

Salidas

- *Front end* implementado y listo para integrar con *back end*.

Tarea 4		**Diseño e implementación "*Sign*al manager", "alarm manager" y "event manager"**	
Fecha prevista de inicio	12/04/2021	**Fecha prevista de fin**	02/05/2021

Entradas

- *Front end* implementado.
- Requisitos funcionales de entrada de *Angular-RT*-IHM.

Actividades

- Desarrollo de *Sign*al manager.
- Desarrollo de event manager.
- Desarrollo de alarm manager.
- Integración message broker en *back end*.

Salidas

- *Back end* desarrollado y listo para integrar con *front end*.

Tarea 5		***Angular-RT*-IHM: pruebas funcionales**	
Fecha prevista de inicio	03/05/2021	**Fecha prevista de fin**	16/05/2021

Entradas

- *Front end* implementado.
- *Back end* desarrollado.

Actividades

- Implementación de la plataforma de pruebas.

• Pruebas de integración *front end – back end*. • Pruebas funcionales
Salidas
• *Angular-RT*-IHM probado funcionalmente.

Tarea 6		**Pruebas de rendimiento de *Angular-RT*-IHM**	
Fecha prevista de inicio	17/05/2021	**Fecha prevista de fin**	30/05/2021
Entradas			
• *Angular-RT*-IHM probado funcionalmente.			
Actividades			
• Pruebas de rendimiento.			
Salidas			
• *Angular-RT*-IHM 100% probado.			

7. CONCLUSIÓN

7.1. DIFICULTADES ENCONTRADAS Y CÓMO SE HAN SUPERADO O ESQUIVADO

Durante el diseño y desarrollo de las diferentes aplicaciones que componen este trabajo he encontrado las siguientes dificultades:

1. Uno de mis objetivos principales era conseguir que las vistas y los símbolos que se muestran en el *front end* no fueran componentes *Angular*, es decir, que un usuario sin conocimiento del *framework* fuera capaz de diseñar sus propios símbolos o sinópticos. Idealmente un símbolo o un sinóptico debería poder cargarse a partir de un archivo HTML. Si bien, no he conseguido conseguir plenamente mi objetivo, he conseguido reducir el impacto de desarrollo mediante el uso de componentes abstractos que contienen un código *Typescript* genérico. La solución ideal pasaría por el desarrollo de herramientas gráficas de diseño de símbolos y vistas que generaran los ficheros HTML, *Typescript y CSS* que la aplicación *front end* dibujará.
2. La implementación de la comunicación entre los simuladores y el *front end* me ha exigido mucho esfuerzo porque la documentación existente, tanto de *ZeroMQ* para *nodejs* como de *socket.io* es bastante reducida y se limita a proporcionar ejemplos demasiado básicos. Al final, he tenido que desarrollar mis propios procesos “simples” de prueba para identificar los errores de mi código. A modo de ejemplo, tardé una

semana en descubrir que la versión de *socket.io* de las raspberris era incompatible con la que tenía en el *front end.*

7.2. ¿ES ANGULAR UN FRAMEWORK ADECUADO PARA IMPLEMENTAR UN SCADA NEAR REAL-TIME?

En este punto de la memoria debemos afrontar la respuesta a la pregunta que motivó inicialmente el trabajo. Empezaremos por la afirmación más simple:

- *Angular Material* no tiene la madurez suficiente para implementar un *SCADA near real-time.*La limitación del *scrollview* así como de los paginadores cuando el número de *items* a dibujar supera un par de centenas no lo hace demasiado fiable. Es cierto que simplifica la configuración de la apariencia mediante el uso de temas pero yo usaría más funcionalidades de esta librería.
- Como ya he mencionado anteriormente, *Angular* ofrece las funcionalidades y la arquitectura adecuadas para desarrollar la aplicación y su arquitectura de componentes simplifica su mantenimiento.
- Desde el punto de vista del rendimiento, sería posible usar *Angular* desarrollando componentes gráficos simples y optimizados (como es el caso de mis sinópticos), sin usar componentes *Material.* Pienso que esto sólo sería adecuado para sistemas en los que el número de variables que evolucionan por segundo sean pocas (una centena). Un buen ejemplo sería un *SCADA* dedicado a la supervisión de instalaciones en los que las variables que se adquieren no tienen una evolución demasiado rápida (ej. valores de temperatura o de presión). Con un volumen de cambios reducido, el funcionamiento de la aplicación es más que adecuado.

7.3. APRENDIZAJE

Este trabajo me ha permitido poner en práctica y de manera autónoma, gran parte de los conocimientos adquiridos en las asignaturas del máster. El diseño de la interfaz gráfica tiene en cuenta los aprendizajes de “Diseño de interfaces interactivas” así como de la familia de asignaturas *HTML* y *CSS.* El uso de *nodejs* para la implementación del *back end* me ha permitido poner en práctica parte de lo aprendido en “Programación en JavaScript para programadores” y el uso de *Angular* como *framework* de desarrollo no habría sido posible sin mi paso por las dos asignaturas de ”Desarrollo *front end*”.

Por otro lado, este trabajo me ha permitido aventurarme en el uso de otras tecnologías que llevaba tiempo queriendo aprender (*ZeroMQ*) lo que ha añadido un reto y mucha diversión a la tarea.

Tal y como he mencionado con anterioridad, el desarrollo de *Drawview* sería un buen ejercicio para algún alumno falto de ideas para su trabajo fin de máster.

Para finalizar, espero haber despertado el interés del lector con esta memoria y quedo a su disposición para contestar las eventuales dudas que pudieran surgirle.