UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

SERVICIOS Y VISUALIZACIÓN WEB PARA EL CONTROL DE SIMULACIONES

Grado en Ingeniería Informática

Trabajo fin de Grado

Junio 2015

Autor: Cristopher del Carmelo López Santana

Tutores: José Juan Hernández Cabrera y José Évora Gómez
Dpto. Informática y Sistemas.

Agradecimientos

Que yo haya llegado hasta aquí, no es un simple hecho de superación individual, sino que es la historia de muchas personas que han aportado todo lo que podían para que yo pudiera estar hoy aquí. A todos ellos, aunque alguno se me olvide añadirlo en este apartado, siempre tendrá un hueco dentro de mi.

Me gustaría empezar por mi familia, en especial a mis padres y mi hermana. Que han permitido después de muchos esfuerzos no solo económicos, que en los tiempos que corren han sido muchos, sino también todo ese tiempo, esfuerzo, cariño y esmero que han sabido dedicarme a lo largo de toda mi vida. Sin ellos hoy no sería ni la mitad de lo que soy ahora.

A la familia Jerez Flores, en especial a Macarena, por su apoyo durante todo este tiempo. Donde me han abierto las puertas de su casa para que el gasto económico que me suponía, ir a mi casa fuera mucho menor, aportando no solo una casa si no un hogar donde sentirse acogido y apoyado.

A mis tutores José Évora y José Juan, los cuales creyeron desde el principio en mi y que supieron darme la ayuda, la guia y las fuerzas que necesitaba. Siendo no solo unos meros mentores profesionales sino también personales.

Al laboratorio, donde he realizado mi proyecto, en el cual se encontraban Mario y Rubén, dos grandes apoyos a la hora de resolver mis dudas y preguntas para seguir avanzando.

A mis amigos que siempre han estado ahí para preguntarme, ¿Qué tal vas?, y por entender que muchas veces no podíamos vernos, no porque no quisiera o no los echara de menos, sino porque estaba allanando el camino de mi vida.

A la familia Cortés Cardona, por saber ayudarme en los momentos en que lo necesitaba, que para Johan en especial, fueron muchos. Y a Lisseth por ofrecernos esas cenas tan espectaculares cuando nos veía dejándonos hasta la sangre en el camino. Su hijo, aún bastante joven, ha conseguido hacerme entender muchas cosas de la vida, incluso ha sabido sacarme una sonrisa cuando no tenía ganas de volver a hacerlo.

Por último, pero no por ello menos importante, a la comunidad de desarrollo ágil que hemos formado entre todos en la isla. Porque me ha permitido seguir mejorando y aprendiendo sobre nuestra profesión. Además, he descubierto lo que tanto buscaba, y es la existencia de más gente con la misma pasión.

Son todos ellos los que han conseguido, con sus esfuerzos presentar este proyecto y yo soy solo un mero focalizador de su fuerza y empeño. A todos ellos, no hay palabras para expresar lo que hoy siento así que lo expresaré diciendo, muchísimas gracias.

Índice

Motivación y contextualización

Mis objetivos para realizar el trabajo de fin de grado, estaban orientados al uso de metodologías de desarrollo ágil, el clean code y el uso de tecnologías web. Esto me llevó a desarrollar mi proyecto en la división CES (calidad, eficiencia y sostenibilidad) del instituto universitario SIANI.

Una de sus líneas de investigación de esta división trata sobre el estudio de redes eléctricas inteligentes (Smart Grids). Concretamente, en esta división se usa la simulación como herramienta de análisis para la evaluación de nuevas estrategias de gestión en redes eléctricas, como por ejemplo, la gestión de la demanda. La gestión de la demanda conlleva la introducción de tecnologías de la información para realizar una gestión inteligente de la demanda con el objetivo de conseguir adaptar la demanda de energía a las limitaciones y necesidades de las infraestructuras de la red eléctrica.

Conceptos propios de las redes eléctricas inteligentes como la gestión de la demanda, generación distribuida y el uso eficiente de la energía, conllevan el desarrollo de nuevos estudios que requieren nuevas aproximaciones [1]. En el pasado, los estudios estaban centrados, principalmente, en cómo programar las unidades de generación de acuerdo a la demanda, considerando a ésta como una carga agregada. La realización de los nuevos estudios de gestión de la demanda requiere nuevas aproximaciones dónde se representa a la red eléctrica como un sistema complejo [2].

Los sistemas complejos están formados por un gran número de entidades con un alto grado de interacción entre sí, procesos o agentes, el entendimiento que el desarrollo requiere y el uso de nuevas herramientas científicas, modelos no lineales y simulaciones hechas por ordenador [Fig. 1]. Una característica propia de los sistemas complejos es la heterogeneidad de las entidades y de sus interacciones, lo cual hace que el comportamiento que emerge del sistema sea prácticamente impredecible [4].

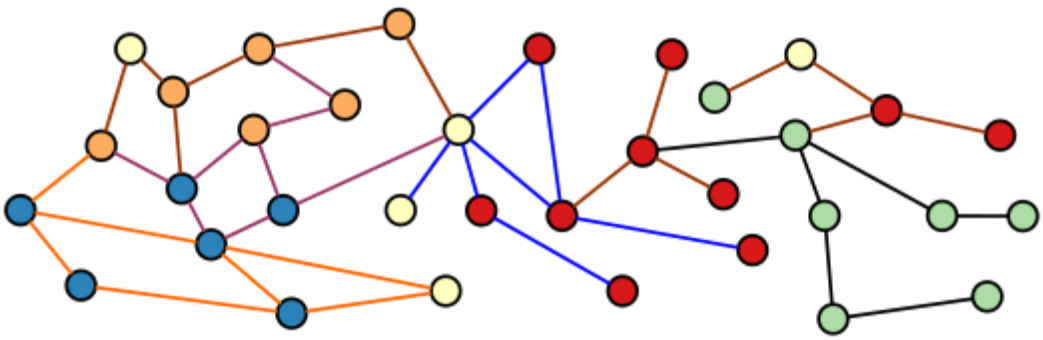


Figura 1

Ilustración de un sistema complejo. Una red de diferentes tipos de entidades e interacciones como ejemplo de un sistema complejo

En el caso de los sistemas complejos, el comportamiento de cada una de las entidades es modelado individualmente de acuerdo al comportamiento que se espera de cada entidad. Por ello, el comportamiento emergente del sistema es la consecuencia de los comportamientos ejecutados por cada una de las entidades así como de las interacciones entre éstas.

En el caso de las redes eléctricas, los estudios de gestión de la demanda requieren la representación de todos y cada uno de los distintos componentes que conforman la demanda (cada nevera, cada bombilla). Viendo que no existían simuladores que se adaptaran a las necesidades de la división CES, el propio grupo de investigación decidió crear su propio simulador para sistemas complejos, Tafat, de cara a poder simular redes eléctricas inteligentes.

Tafat es un framework para el desarrollo de simuladores para sistemas complejos que permite construir modelos complejos en los cuales la escena se descompone en muchas entidades [5]. Cada entidad del modelo se representa estáticamente a través de sus atributos y variables. De este modo, se desarrolla la parte estructural del modelo. Por otro lado, los comportamientos de las entidades forman parte de la representación dinámica, la cual describe cómo evolucionan las diferentes entidades a lo largo del tiempo, así como, su interacción con otras entidades.

En este proyecto se afronta el reto de añadir la capacidad de interacción sobre una simulación de sistemas complejos en tiempo de ejecución, tanto para modificarla como para simplemente observarla. Esto implica una serie de cuestiones, como por ejemplo, cómo abordar la representación de un sistema complejo, de tal manera que sea fácilmente gestionable por una persona, o como ofrecer una manera sencilla para alterar la simulación. Con esta idea nace SGI (Simulation Gateway Interface), un framework que permite hacer accesibles las simulaciones a través de una interfaz gráfica para poder analizarlas o modificarlas.

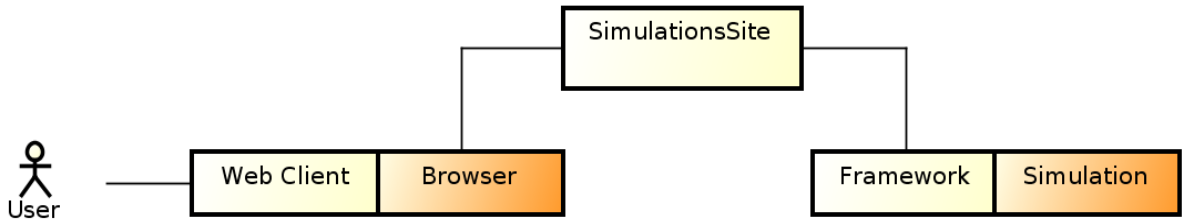
Para ello, este framework, que se debe acoplar programáticamente a la simulación, contiene un servicio web, el cual ofrece la capacidad de interactuar con la simulación. Estas interacciones se transforman en operaciones de acceso y modificación a la simulación. Estas operaciones deben ser proporcionadas por el creador de la simulación, puesto que es éste quien sabe cómo acceder a los datos. Una vez se ha integrado la simulación con el framework, dicho servicio web expondrá una interfaz gráfica que permitirá observar y modificar la simulación de un modo sencillo.

El documento se divide en diversos capítulos. El siguiente capítulo describe el producto que se ha desarrollado. Después, se comenta todo lo relacionado con el proceso de desarrollo del producto. Finalmente, se mencionan las conclusiones aportaciones y trabajo futuro.

El producto software

Simulation Gateway Interface o **SGI**, nace con la intención de poder visualizar y modificar en tiempo de ejecución una simulación. Se pretende representar de una manera sencilla, un modelo complejo, para que el usuario, perciba lo que necesita de la simulación.

Para poder adaptar la simulación y poder ser visualizada, se ha dotado a este sistema de un framework rápido de implementar y entender. Además, se ha buscado la manera de que el desarrollador que ya dispone de una simulación, pueda usarlo sin necesidad de hacer una gran modificación sobre la simulación ya desarrollada.

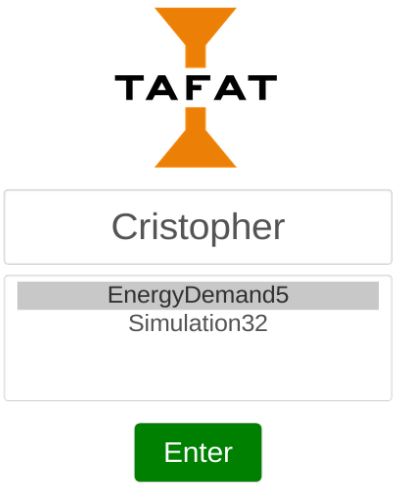


No obstante, del objetivo principal se ha conseguido alcanzar una solución más, que le añade versatilidad al sistema. Y es que SGI no solo ofrece un visualizador, si no que también transforma una simulación a un servicio web. De esta forma, el sistema consigue ofrecer la capacidad de interactuar a la simulación.

La aplicación web

Se optó por realizar una aplicación web para este proyecto, ya que permite ser ejecutada en cualquier ordenador, sin necesidad de una instalación o descarga implícita por parte del usuario.

La aplicación web dispone de dos pantallas diferenciadas, la landing page y la página principal del entorno. Ésta última contiene la mayoría de las acciones disponibles sobre la simulación.



Landing Page

Al arrancar, el usuario deberá introducir un nombre de usuario válido y seleccionar una de las simulaciones que están ejecutándose dentro de la red.

En caso de que todo vaya correctamente, se accederá a la página principal de la aplicación.

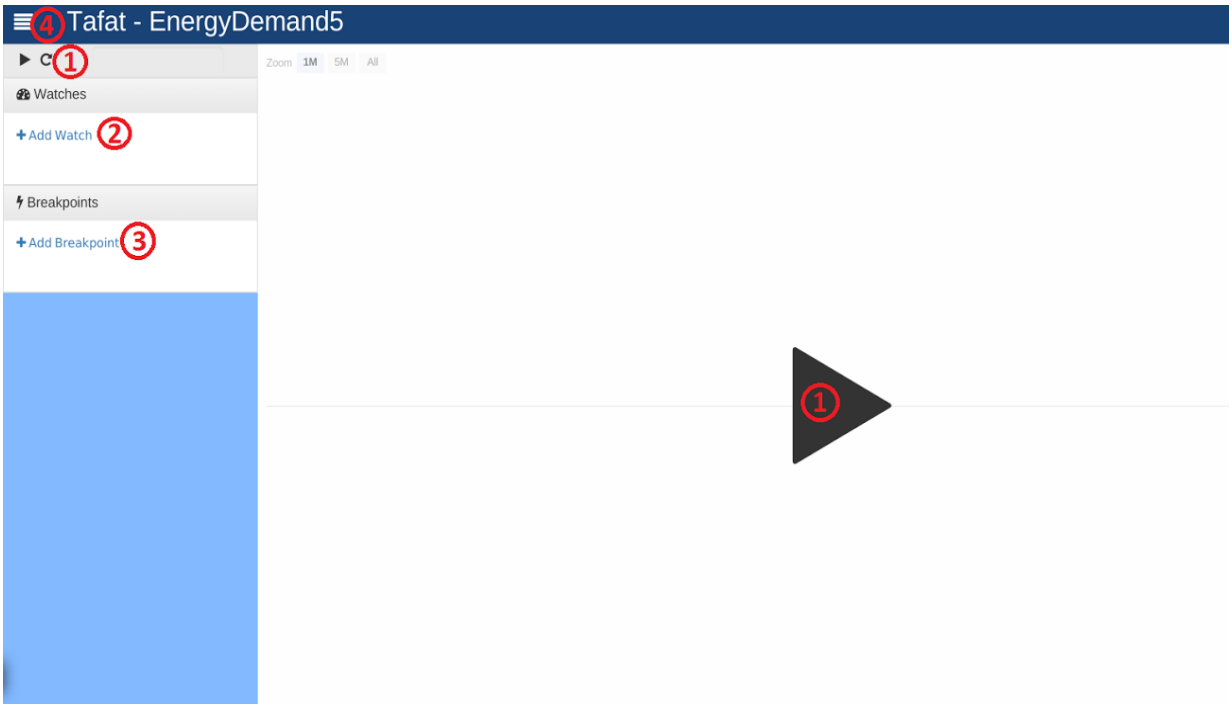
Página principal

Antes de comenzar a explicar las acciones, debemos explicar los dos conceptos básicos de nuestra aplicación.

Breakpoint : Es una parada intencional por parte del usuario, para comprobar el estado de la simulación en un instante de tiempo. En dicho instante, el usuario podrá modificar el estado de los watches que tenga seleccionados.

Watch: Los watches o visores, son los encargados de observar en qué estado se encuentra un atributo de un determinado objeto, en cada instante de tiempo.

Ya con estos conceptos claros, podemos pasar a explicar la interfaz de la que disponemos en la página principal.



Esta es la pantalla es la siguiente a la Landing Page. Esta Página dispone de una serie de acciones principales enumeradas en la imagen.

1) Ejecutar la simulación

Esta acción solo se ejecutará en el momento en el que añadamos un watch a nuestra simulación, por lo que se trata de un requisito indispensable para la ejecución.

2) Añadir un Watch

Al pulsar sobre esta acción, aparecerá un diálogo poco intrusivo, con el que podremos descender por la jerarquía de objetos de la simulación, para poder visualizar uno de sus atributos en el gráfico.

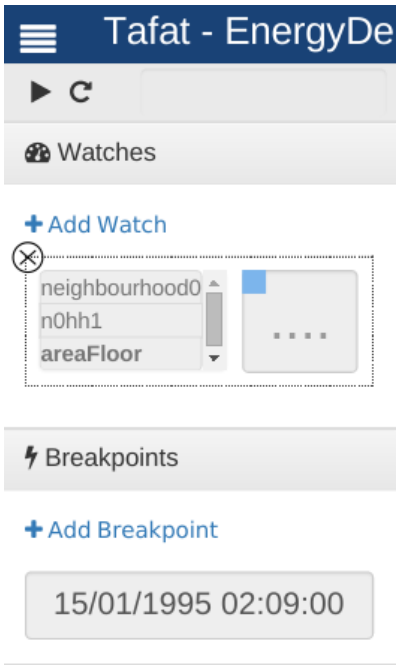
3) Añadir un Breakpoint

Cuando ejecutemos esta característica, se nos desplegará un datepicker, que estará acotado al tiempo de la simulación. Al seleccionar una fecha determinada y pulsar el botón de “Done”, estaremos añadiendo un breakpoint a la simulación.

4) Descargar una imagen del gráfico

Gracias a esta característica, se ofrece una forma de poder obtener una imagen del gráfico en el estado actual, en distintos formatos, PNG, JPG, PDF. Además, permite la posibilidad de imprimirlo directamente.

Una vez hayamos seleccionado un watch y un breakpoint, dispondremos de dos nuevas acciones.



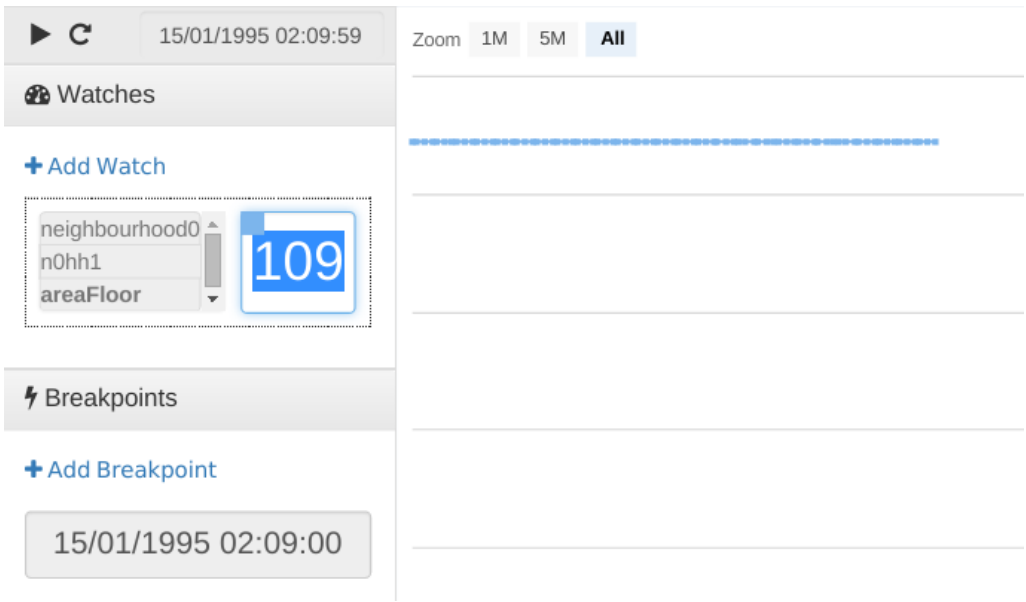
1) Eliminar Watch

Una vez que hayamos seleccionado un watch y se nos muestre en el menú, si pasamos el ratón por encima del objeto, se nos mostrará un botón de cerrar o eliminar. Si pulsamos en él, no continuaremos viendo la evolución de dicho atributo.

2) Eliminar Breakpoint

Esta funcionalidad se realiza exactamente igual que “Eliminar watch”. Al pasar el ratón por encima, aparece un botón de cerrar o eliminar, que al ser pulsado, eliminará el Breakpoint de la lista.

Ya con algunos watches y breakpoints seleccionados, podremos ejecutar la simulación. Es al caer en un Breakpoint, cuando aparecerá otra acción.



1. Modificar el valor de un atributo

En la imagen superior, se muestra una simulación que ha caído en un breakpoint. Como se explicaba al inicio, esta situación nos permite modificar el valor del atributo, para posteriormente continuar la simulación y comprobar cómo se comporta con el nuevo valor. Ésto nos lleva a una serie implicaciones muy interesantes las cuales le dan muchísimo juego a la aplicación.

El framework

El framework que se le ha ofrecido al desarrollador de simulaciones intenta ser poco intrusivo, fácil de entender e implementar. Para conseguirlo hemos ofrecido una interfaz a implementar sencilla y con métodos entendibles, que comentaremos más adelante.

Otro aspecto interesante, es que se evita que el desarrollador de simulaciones gestione los usuarios, los breakpoints, los watches, etc. Sino que esto es transparente. El objetivo es que el desarrollador solo deberá preocuparse exclusivamente de cómo el framework accede a la simulación y cuando tiene que refrescar los datos.

El Framework se distribuye en formato .jar, por lo tanto, la forma de integrarse con el mismo, es añadirlo como un jar externo e implementar el SGIFramework.

Las funciones de la interfaz a implementar o accesibles para el desarrollador son las siguientes:

1. Refresh: esta función, por el contrario que las demás, no es abstracta. Es una función a la que el desarrollador llamará, cada vez que quiera dejar paso al framework para manejar las solicitudes pendientes y refrescar la vista con los nuevos datos.
2. Roots: sirve para acceder a los objetos raíces de la simulación.
3. ChildrenOf: permite obtener los objetos que contiene un objeto.
4. AttributesOf : devuelve los atributos de un objeto determinado.
5. ValueGet y ValueSet: estas operaciones permiten obtener (Get) y establecer (Set) el valor un atributo de un objeto.
6. Play arranca o reanuda la simulación.
7. Pause : pausa temporalmente la simulación hasta que se ejecute el play.
8. Reset : reinicia la simulación hasta su estado inicial.

Desarrollo del software

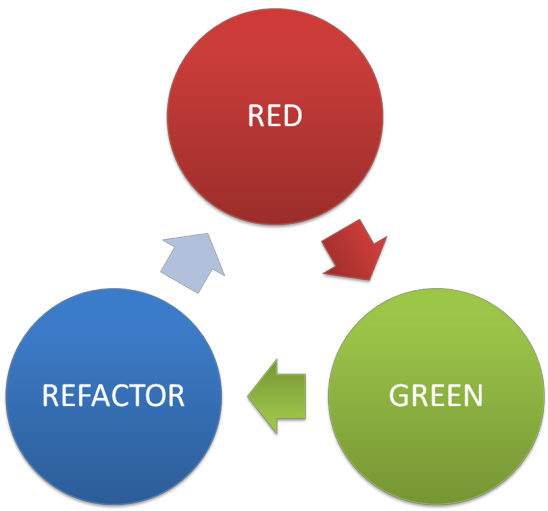
Aproximación metodológica

Mi visión sobre el desarrollo del software es principalmente agilista y evolutiva. Como se suele decir dentro de entornos agilistas, el software es un ente vivo, que crece y se desarrolla y por ende se adapta y evoluciona. La idea de evolucionar es que se adapte a los nuevos cambios que van surgiendo y que las partes que van surgiendo de él sean partes adaptadas al cambio. De lo contrario el software morirá frente a un ambiente demasiado rudo para él por su poca flexibilidad a los cambios.

Para crear el software de este proyecto de manera adaptativa, fue primordial el hecho de tener reuniones prácticamente semanales. Durante estas reuniones la concepción del Software fue evolucionando y madurando, así, las pérdidas en una solución incorrecta eran imperceptibles. Por este motivo, es clave contar con discusiones sobre las soluciones alcanzadas con profesionales del sector.

Igualmente, considero que no debemos prediseñar mucho más allá de un posible esqueleto del sistema, ya que deben crecer de forma natural, para ir adaptándose al medio y no morir precozmente, porque si no consiguen adaptarse demasiado bien a los cambios, nuevos pensamientos o concepciones sobre el propio sistema desarrollado.

Además de todo lo anteriormente citado, el TDD ha sido fundamental para llevar a cabo este desarrollo inmerso en los cambios, las nuevas concepciones y constante mejora de las soluciones adoptadas.



De esta manera, gracias a las pequeñas iteraciones del TDD, donde creamos una pequeña necesidad, la abordamos y finalmente la refactorizamos, nos ha permitido que varias micro-iteraciones, se conviertan al final en grandes iteraciones. Así como si de un reloj se tratase, han surgido no solo arquitecturas de software emergentes, sino que también ha emergido prácticamente solo el orden de las iteraciones.

Código limpio y refactoring

Antes de comenzar, vamos a citar a dos grandes autores para definir que es refactoring y clean code. El Refactoring “Es una técnica disciplinada para reestructurar un código existente alterando su estructura interna, pero sin cambiar su comportamiento externo” Martin Fowler. El Clean Code o código limpio, *“*es simple y directo, se lee como la prosa bien escrita. El código limpio nunca oscurece u oculta la intención del diseñador sino que debe contener abstracciones nítidas y líneas de control directas y legibles..” Grady Booch.

De esta manera, siguiendo estas definiciones, podemos entender que para hacer clean code, primero hay que hacerlo para luego cambiar su estructura e intentar transformarlo. Con esto en mente, las iteraciones del TDD, no es más que la práctica continuada de ese principio.

Mi brújula a la hora de saber donde aplicar las distintas técnicas de refactoring han sido los principios SOLID, ya que estos, consiguen ofrecer una serie de buenas reglas para la mantenibilidad del software durante el tiempo. Además, los Code Smells que he usado para comprobar qué técnicas de refactoring aplicar han sido:

Malos nombres, clases largas, código astronómico, obsesión primitiva, operaciones de switch/case, envidia de características, duplicidad del código, comentarios innecesarios...

Por último ya una vez detectado donde se encuentran los distintos malos olores del código, se pasaría a aplicar las distintas técnicas de refactoring pero antes tenemos

En este proyecto se ha perseguido no solo se ha desarrollado un producto software sino se ha perseguido la excelencia dentro del código. Para ello, las técnicas más aplicadas en el han sido:

Extract Method, Inline Method, Replace Temp with Query, Remove Assignments to Parameters, Move Method, Extract Class, Replace Record with Data Class, Renames, Pull Up Method, etc.

Para aplicar las técnicas de refactoring, hace falta una buena herramienta que nos ayude, sino, todo esto podría ser mucho más laborioso. Para ello, nos hemos apoyado en el IDE de Jetbrains, el cual posee entornos como Intellij para Java, y de WebStorm para el desarrollo con tecnologías Web.

Intellij y Webstorm, aunque también tienen algunos contras, he de decir que son rápidamente eclipsados por la cantidad de ventajas que ofrece a la hora de desarrollar de manera cómoda y rápida.

Arquitectura y Diseño del software

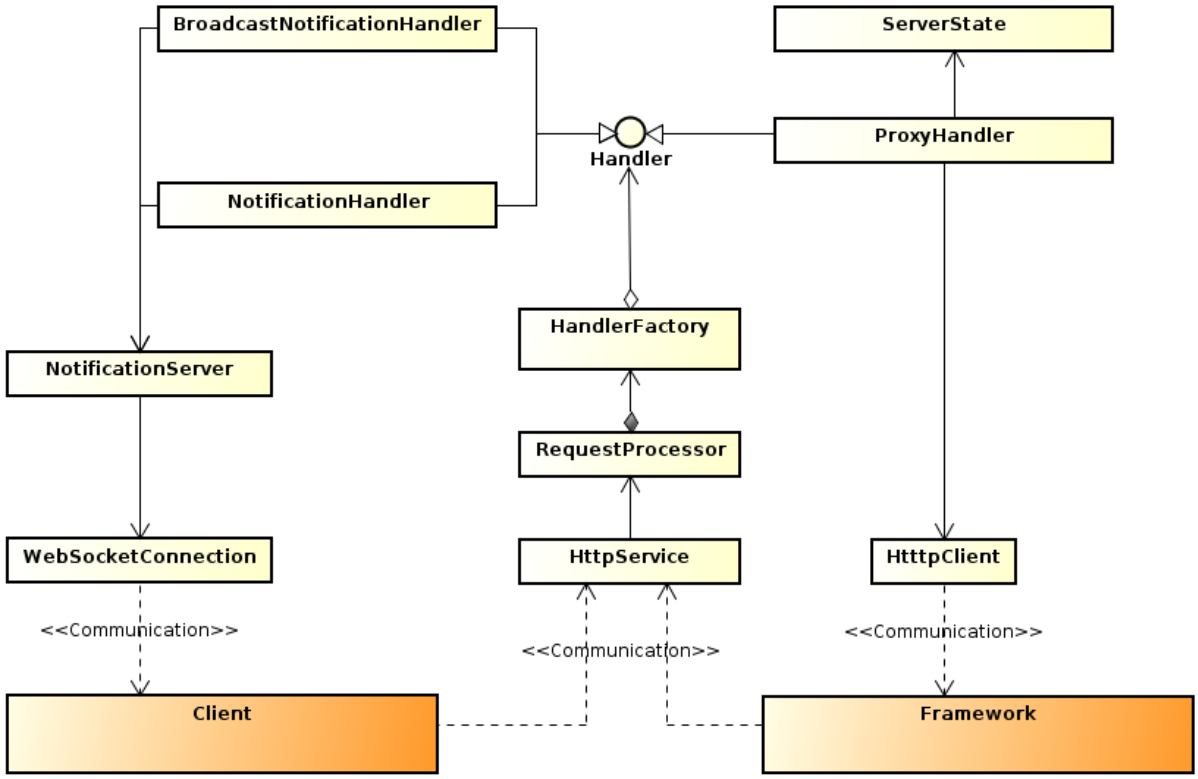
En la siguiente sección, paso a comentar la arquitectura de la que dispone mi proyecto, esta arquitectura no ha sido prediseñada, si no que ha surgido a medida que se iba resolviendo cada micro iteración de TDD.

Así pues, pasaremos a ver como una petición va navegando a través del software que hemos construído, parándonos en las clases con mayor relevancia. Veremos como estas peticiones van navegando a través de nuestro software hasta conseguir la respuesta deseada y cómo ésta afecta al estado de nuestro software.

De este modo, comenzaremos hablando de la arquitectura que existe en el Simulations Sites y posteriormente pasaremos a hablar sobre la del Framework.

Simulations Site

A continuación, pasamos a describir la arquitectura del Simulations Site, el que como hemos comentado, se encarga tanto de recibir las peticiones externas, como de atender a las peticiones del Simulation Framework.



En este diagrama volvemos a tener representaciones de otros dos sistemas, el Framework y el Cliente. Éstos serán ahora los que desde la perspectiva del Simulations Site, hagan las distintas interacciones.

Cuando se recibe una petición para un framework, el primer sitio al que llega es al HttpService. Éste, como en el Framework, se encarga de transformar el objeto HttpExchange y de ahí pasarlo al RequestProcessor, propio del Simulation Sites. En este caso, se personalizó el RequestProcessor, porque las peticiones necesitaban un tratamiento distinto al normal en líneas generales. Para terminar, el RequestProcessor es inyectado en la librería junto al HandlerFactory.

Estas peticiones provocan la ejecución de un ProxyHandler. Este handler, comprueba en qué dirección se encuentra la simulación a la que va dirigida en el ServerState. Para concluir, se procede a reenviar la petición HTTP con el HttpClient de

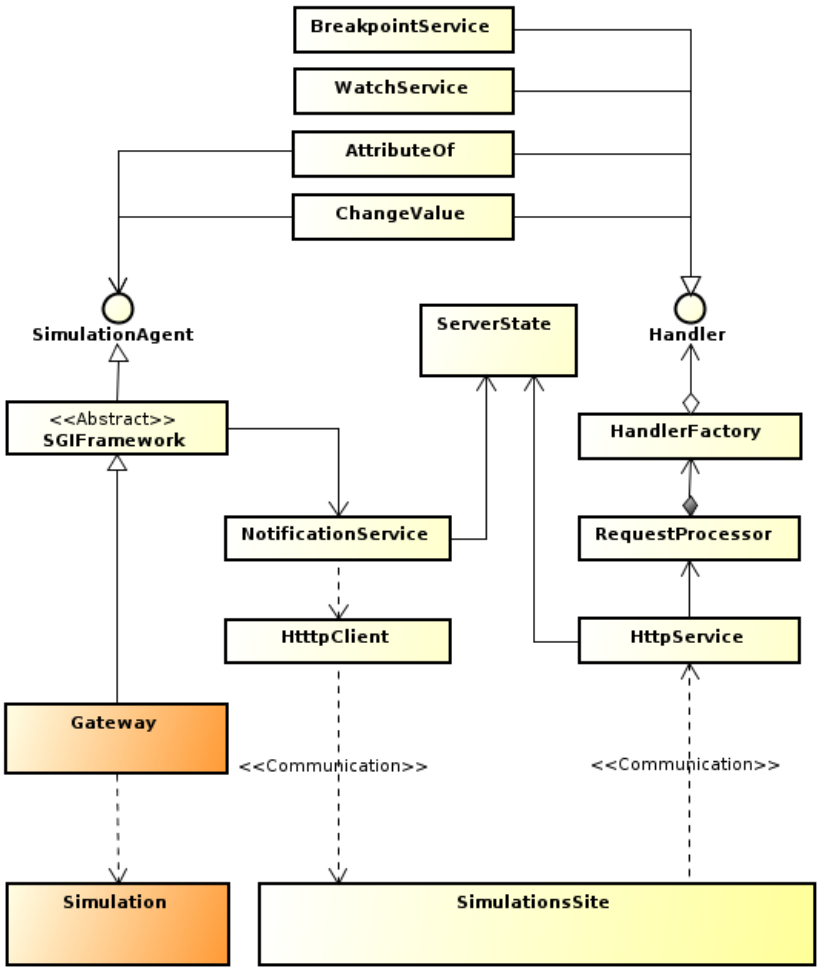
nuestra librería, hacia el Framework. Una vez que el Framework responde, se devuelve la respuesta al Cliente, que había quedado a la espera.

En el caso de las notificaciones, es algo distinto a los casos anteriores. Como habíamos comentado, el Framework envía una petición HTTP para provocar una notificación sobre un usuario o usuarios. Dependiendo de si es una petición de notificación individual o masiva, se ejecutará el NotificationHandler o el BroadcastNotificationHandler respectivamente.

En el caso del NotificationHandler, éste delega la responsabilidad de enviar la notificación push al NotificationServer. Éste comprueba a qué usuario y de que simulación se ha hecho la petición y se envía por el Websocket asignado. En el caso del BroadcastNotificationHandler, es muy parecido al Notification Handler, solo que obtiene todos los usuarios asignados a una simulación en concreto y se les envía a todos el mensaje.

Si todo ha ido bien, el Simulation Sites, le enviará una respuesta al Framework de que todo ha salido correctamente.

Framework



El diagrama muestra dos representaciones en un color naranja, las cuales no son clases en sí, sino que son una representación de los sistemas adyacentes como son el Simulations Site y la Simulación. La relación con el Simulations Site es vía comunicaciones de red. Además también se encuentra representada la librería HttpHandler en un paquete distinto.

Cuando una petición dirigida a una simulación, es enviada al Simulation Sites, éste la trata y la envía al Framework. Como estamos usando la librería nativa de Java para tratar las comunicaciones por HTTP, el objeto que recibimos es un HttpExchange. Este objeto nativo de Java, contiene tanto la información de la request como la forma de enviar la response.

Por ello, nuestra clase HtttpService, perteneciente a la librería HttpHandler, lo primero que hace es partir la responsabilidad entre dos clases, el Request y el Response. Una vez partida la responsabilidad, se envía al Request Processor el objeto Request

obtenido. Este objeto, se encarga de enviar dicha petición al HandlerFactory y éste devolverá un Handler determinado.

El Handler Factory es implementado por cada proyecto que use la librería y posteriormente es inyectado. Por tanto, los handlers que se encargan de tratar las Requests también son propios de cada proyecto.

Cuando el Framework tiene que responder a las peticiones, se diferencian dos casos base.

1. Una petición hacia un caso resuelto únicamente por el Framework, como son los Breakpoints y los Watches. Éstas son resueltas por los servicios del propio Framework.

2. Una petición que necesita de la implementación del SimulationAgent. Cada acción es representada por un objeto que la abstrae. Por ejemplo: AttributeOf, ChildrenOf, etc.

Una vez conseguida la respuesta, ésta será posteriormente transformada en el HttpService en un HttpExchange para poder ser enviada al Simulations Site.

Cuando sea necesario enviar una notificación, el SGIFramework a través del ServiceNotification, hará una petición al Simulations Sites. La dirección de este último se encuentra almacenada en el Server State, por lo que tendrá que obtenerla primero. El ServiceNotification, envía una petición HTTP al Simulations Sites a través del HttpClient y de forma asíncrona, esperará una respuesta por parte de éste.

Por último, las notificaciones podrán ser de dos tipos:

- Individuales, para un solo usuario.
- Masivas, para cada usuario suscrito a la simulación.

Iteraciones del desarrollo

Primera iteración : La conceptualización.

Durante la primera iteración, basé mis esfuerzos en identificar las necesidades y buscar las soluciones tecnológicas que me permitieran abordarlas. Para ello, diseñé un mockup de la interfaz web.

Este mockup se diseñó con Balsamiq, ya que me pareció la herramienta más acorde a mis necesidades. Quería que la realización del mockup fuera muy sencilla para alcanzar un diseño conceptual, en el que no tuviera que centrarme por ahora, en los detalles menores que la interfaz debía asumir (Colores, logos, imágenes, tipografías...).

Estas decisiones eran debidas a que mi objetivo principal no era el diseño de la página, si no que necesitaba la obtención de requisitos de los que aun no me había percatado con el mínimo coste posible. Esto, incluso, me permitiría ser más preciso con las tecnologías que debía aprender. En un par de reuniones, se consiguió alcanzar nuevos requisitos. Para ello, en cada reunión, se adaptaba rápidamente el mockup a las distintas conclusiones alcanzadas obteniendo un Mockup que plasma las distintas necesidades.

Al mismo tiempo, también repasé sobre ciertos protocolos usados en la actualidad y su funcionamiento básico tal como DHCP el cual guarda ciertas relaciones con el inicio del protocolo SGI.

Una vez alcanzados los nuevos requisitos empecé a investigar sobre las tecnologías que necesitaría y la forma de usarlas. Por consiguiente, mientras eran aprendidos, necesitaba practicarlos y ponerlos en acción para comprobar que los conocimientos adquiridos eran correctos, para ello realice una serie de experimentos:

1. Un respondedor de conexiones HTTP (libreria nativa de Java)

El respondedor de conexiones HTTP no hacía mucho más que responder ante una petición dada, en este experimento buscaba como enrutar peticiones, que sucedía frente a un bloqueo del servidor, o como detectar si la necesidades pudieran ser otras como la de obtener ficheros estáticos.

2. Creación de un pequeño protocolo por sockets (Uso de sockets TCP).

El experimento del protocolo hecho por sockets, nos permitió ver, que para el envío de mensajes era necesario la creación de un protocolo previo el cual ya no los ofrecía HTTP en su gran mayoria, asi que se optó por dejar este camino. No obstante, esta solución me dio experiencia para siguientes iteraciones, donde necesitaría de alguna forma la interacción de bajo nivel en la red.

3. En cada experimento use Top-down de TDD.

Por otra parte, también me impuse ciertas técnicas a usar sobre las cuales fui experimentando, aprendiendo y adaptando a mi, como por ejemplo fueron:

- Top-down o outside-in “es una metodología de desarrollo, con Test Driven Development, en la cual, se comienza escribiendo test desde las capas externas del software (presentación, GUI) adentrándonos paso a paso. La idea es dirigir el diseño del código al uso que va a tener, en vez de intentar anticipar los requisitos” Harry J.W. Percival

- Git-flow “es un modelo de ramificación, creado por Vincent Driessen para la gestión de las ramas en git, para ayudar a los desarrolladores a mantener el curso de las características, la corrección de fallos y las publicaciones en los proyectos software grandes.” Jeff Kreeftmeijer

- Pomodoro es un método para la administración del tiempo desarrollado por Francesco Cirillo a fines de los años 1980. La técnica usa intervalos de 25 minutos separados por pausas. Esta técnica se ha adaptado para ser usado en el desarrollo ágil de software.

Segunda Iteración: Top-Down API

En esta iteración con los conocimientos necesarios y con Top-Down de la mano, comencé a generar la arquitectura de mi servidor. De tal modo, comencé definiendo la API que iba a tener mi servidor y la arquitectura fue emergiendo por sí misma. Dando lugar a las primeras entidades que acabaría usando en todo el proyecto.

En esta iteración realice las primeras aproximaciones a los estados de la simulación, la obtención de los valores, las primeras lógicas del envío, recepción y tratamiento de mensajes.

Hasta esta iteración, el servidor, al que iban a atacar los usuarios y el framework de la simulación eran el mismo. En este momento, es donde se observó el potencial de separarlo debido a que el cliente solo debía acceder a un punto al que acceder.

Tercera Iteración : El comienzo de SGI

Uno de los objetivos principales es ofrecer un servicio, que fuera fácil y sin apenas configuración. Es por este motivo que se optó por crear un protocolo automático para detectar donde se encontraba dentro de la red el Simulation Sites y se suscribieran al mismo.

Para dar esta solución opte por usar sockets sobre UDP, ya que estos me permiten enviar una señal de broadcast por toda la red con una comunicación sencilla. Esto me daba una gran versatilidad a la hora de crear la conexión, liberandonos de la necesidad de la configuración manual.

El protocolo que creé tiene tres estados.

Descubrimiento: el cliente, lanza una señal de broadcast con la esperanza de detectar dónde se encuentra el Simulations Site. Si existe un Simulations Site en la red, este le informará de cuál es su IP, para empezar con la negociación del puerto donde el dispositivo, va a ejecutar dicha simulación.

Negociación: el cliente le pedirá suscribirse al Simulations Site. Para ello, el servidor tendrá que darle un puerto que su IP tenga libre. Por defecto, el rango de puertos ofrecidos comienza por el 50000. En caso de que ya existiera una simulación ejecutándose en ese puerto, se incrementa en uno el ultimo puerto conocido.

Por ejemplo:

Cliente	Puerto	Nombre
192.168.1.35	50000	Simulación 43
192.168.1.35	50001	Coche eléctrico
192.168.1.35	50002	Simulación Edificio

Aceptación: En este paso el servidor, envía un puerto al cliente y este le responde aceptandolo en caso de que todo haya ido bien. El servidor, después de haber recibido un "OK", procederá a añadir la simulación en su tabla de suscripción de simulaciones.

Cuarta Iteración : Ascendiendo en los protocolos

Durante la fase anterior, conseguimos la primera comunicación entre el Simulation Sites y el Framework. En esta comunicación, el puerto en el que el servidor HTTP va a escuchar es obtenido. Así que lo que necesitamos ahora, es construir dicho servidor.

Gracias a los experimentos de la primera iteración, me familiaricé con la librería nativa que ofrece Java para el manejo de peticiones y respuestas de HTTP. Esta librería formó las bases de la arquitectura y de las abstracciones que generé para desacoplar esta funcionalidad, pudiendo en el futuro usar otras librerías para este fin.

Así que, con esto en mente, abstraje las necesidades de ambos y creé una librería para el manejo personalizado de peticiones HTTP, evitando la repetición de código y aumentando la eficiencia en el mantenimiento.

La arquitectura de esta librería, a la cual nombré como Http Handler, se basa en el polimorfismo. Por este motivo, aunque muchas clases las he construído para facilitar el uso de la librería, han habido casos donde he necesitado adaptar la librería a ciertas necesidades. Para ello, simplemente he creado una clase que satisfaga las necesidades individuales haciendo que cumpla el contrato.

El diseño de esta librería se basa en el principio de HTTP: toda petición (request) debe tener una respuesta (response). Para que una Petición sea respondida, antes debe ser procesada. Para eso, es necesario pasar por un Procesador de Peticiones (RequestProcessor), el cual se encarga de adaptar la petición si fuera necesario y detectar qué Manejador(Handler) es el encargado de responderle. Por último, una vez procesada la Petición y obtenida la Respuesta, esta es enviada a su destino desde el HttpService.

Quinta Iteración: Refactor

Una vez conseguida la correcta interacción entre ambos servicios, se pasó a una fase de Refactor, ya que habían partes mejorables y otras pendientes de mejora.

Durante esta iteración se pensó en una forma de tratar las excepciones de una manera más limpia y clara que el uso de los try-catch, ya que ensuciaba el código y su tratamiento no iba más lejos que un mensaje de error.

Así pues, se realizó una librería para el tratamiento de excepciones a la cual llamé ExceptionRunner. Esta librería usa inferencia de clases y expresiones lambdas para crear una forma más cómoda y sencilla de tratarlas. Esto se alcanzó gracias al principio de SRP: si el tratamiento de excepciones es una responsabilidad única, ¿porque no delegar esta responsabilidad en un agente externo?

Así se detectaron dos uso comunes de los try-catch: para obtener un objeto o para ejecutar un método. Una comparación del resultado:

Método vacío.

1) Ejecución sin la librería

```
Try{
      método(parametro1, parametro2)
}catch(Exception exception){
      System.out.err(message);
}
```

2) Ejecución con la librería

```
runSafe(()->método(parametro1,parametro2), message);
```

Método que devuelve un objeto

1) Ejecución sin la librería

```
Try{
      Integer i = devuelveInteger(parametro1, parametro2)
}catch(Exception exception){
      System.out.err(message);
}
```

2) Ejecución con la librería

```
Integer i = getSafe(()->devuelveInteger(parametro1, parametro2));
```

Como se observa, el manejo de excepciones acaba siendo mucho más claro y expresivo, además también admite un método para el manejo del error.

También comprendí el uso de módulos en Intellij, ya que cada librería o parte del proyecto estaba dispersa en un proyecto de Intellij distinto. Esto me hizo ganar en productividad y rapidez.

Sexta Iteración: Construcción del Framework para la simulación

Ya con todo lo demás superado, era hora de empezar a perfilar distintos detalles del framework de la simulación. Durante las iteraciones anteriores se habían alcanzado pequeñas aproximaciones, donde comenzaba a emerger un esqueleto de la arquitectura con la que podría resultar al final. Así que durante esta iteración, empezamos a realizar tests para terminar de darle forma.

Esta interfaz, tenía que ser no solo útil para el desarrollador, sino también versátil y fácil de usar. Debido a esto, fue necesario durante su realización, la interacción con los desarrolladores de simulaciones, para pedirles su opinión y resolver mis dudas.

Así fue como se detectaron los distintos servicios que tenía la simulación, tales como el Servicio de estados de la simulación, los watches, la jerarquía de objetos, etc. El desarrollador debía de obtener estas cuatro interfaces e implementarlas. Posteriormente tendría que configurar el Framework para añadir la ruta donde había implementado los servicios.

Los servicios eran accesibles no solo por el Framework si no que se ofrecía una clase de utilidad llamada ServicesManager. Dicha clase volvía a jugar con la inferencia de clases, así su uso era muy cómodo:

```
ServicesManager.get(NotificationService.class)
                .push(“Username”, “Mensaje”);

ServicesManager.get(SimulatorStatesService.class).play();
```

Las clases pasadas por parámetros, no eran la implementación de las interfaces, si no las interfaces en sí mismas. El ServicesManager, se encargaba de implementarlas y devolverlas.

Para la búsqueda de las clases y detección de las implementaciones, usé la librería de Reflection. Así podía ver qué clases eran realmente implementaciones de qué interfaces y podía llamar a sus constructores. El coste en eficiencia, no era demasiado alto, ya que reflection solo era usado al principio de la ejecución del Framework.

Séptima Iteración: Construcción del Simulation Sites

Durante esta iteración me ocupé de terminar ciertos aspectos del encargado de almacenar y redirigir las peticiones de las simulaciones, llamado Simulation Sites.

El Simulation Sites, se encarga de esperar por las peticiones de nuevas simulaciones y una vez establecida la conexión, almacena el nombre, la dirección y el puerto. También se encarga de atender las peticiones de los clientes de la aplicación web. En lo que respecta a las distintas peticiones de un usuario, el Sites, solo hará de proxy, pero para el cliente web la simulación la contiene el Sites, de ahí su nombre.

Además, también se encarga de las notificaciones a los usuarios que las simulaciones quieran hacer. Éstas son realizadas a través de Websockets, el usuario crea la conexión e informa al sites bajo que simulación y con qué nombre está suscrito. Así cuando la simulación quiera notificar a un usuario, ésta mandará una petición al Sites.

Octava Iteración: Integración del Sites y el Framework

Una vez terminadas las dos estructuras principales, me dispuse a realizar los tests de integración para comprobar que ambos sistemas funcionaban correctamente juntos y observar todo aquello que pudiera faltar. Para conseguirlo, me cree un módulo para testear la integración del uno con el otro.

Realicé distintos test de sistemas, para comprobar las distintas interacciones tales como el comienzo del protocolo SGI, las distintas peticiones externas hacia el interior, el envío de notificaciones, etc.

La realización de esta integración con TDD resolvió rápidamente muchos problemas que de otro modo me hubiera supuesto un esfuerzo mucho mayor. Además de de emerger la arquitectura faltante, TDD, me ayudó a completar las partes en las que aun no me había percatado.

Después de realizar la integración con los distintos test, hice un pequeño Mock de una simulación, para integrar en un futuro también el cliente web.

Novena Iteración: Maquetación Cliente Web

Una vez alcanzada la integración entre el Framework y el Sites, necesitábamos crear ya el cliente Web, para alcanzar la integración entre todos los componentes del sistema. Pero antes, de empezar a trabajar con la lógica del cliente, empecé a desarrollar la interfaz que iba a tener. Para ello usé tres herramientas principales que me ayudaron mucho, JQuery, Bootstrap y Highcharts.

Gracias a Bootstrap, conseguí realizar la estructura de la página en distintas secciones, siendo su único problema la altura, ya que el ancho, Bootstrap lo gestiona con su concepto de Grid. Por otra parte, este mismo sistema es tratado de manera responsive, ya que dependiendo del tamaño de la pantalla, es capaz de reorganizar la página, para adaptarse.

La Gráfica, en un principio se planteó hacerlo a través de vectorizados, el problema es que esta solución podría llevar demasiado tiempo de aprendizaje y desarrollo, así que se optó por buscar otra alternativa. La alternativa fue Highcharts, una herramienta para la gestión de gráficos de una forma más sencilla, así que optamos por usarla con un resultado muy profesional y dinámico.

Además, durante la maquetación se empezó a hacer algún experimento de la gestión de código con AMD (Asynchronous Module Definition). Esta API, nos permite la gestión de las dependencias, de una forma más responsable. Además, las dependencias externas también las gestiona, así que también pueden ser configuradas.

Décima Iteración: Integración Cliente Web

En esta iteración el objetivo principal, fue conseguir la integración del Cliente, con el resto del sistema. Probablemente se haya pasado con deuda técnica, pero el objetivo fue asi, por la necesidad de aprender exclusivamente como usar las distintas herramientas.

Mi opinión era que consiguiendo este hito podríamos llegar al Refactor, de otra forma hubiera sido más difícil, porque no nos hubiéramos centrado solo en aprender las distintas herramientas, sino también cómo integrar con las distintas técnicas.

Para alcanzar este hito, comenzamos a usar los Websockets en Javascript, cuya API es prácticamente inmejorable; las llamadas a la API de AJAX de JQuery y continuamos usando RequreJS, el cargador de módulos AMD (Asynchronous Module Definition).

Para la gestión de notificaciones por Websocket, cree una clase que comprende los distintos tipos de notificaciones y los redirige a su clase asociada, de esta forma puedo generar distintas notificaciones y crear más versatilidad a la hora de representar distintos objetos. Así se consigue presentar de manera distinta por ejemplo, los sensores.

En cuanto a la gestión de peticiones HTTP, genere una clase que se encarga de crear las Peticiones (Request) y otra que gestiona las peticiones a través de AJAX.

Por último, AMD me ha permitido mayor versatilidad a la hora de crear mi propia arquitectura y manejar las distintas dependencias. Así pues, he podido diversificar el uso de clases, en vez acabar en la perdición de un script entero para un proyecto o de tener varios ficheros y gestionar manualmente el orden de los imports.Además, RequireJS obliga al uso del patrón module, permitiendo una orientación a objetos más cercana y más limpia.

Undécima: Pair Programming

Ya con el sistema perfectamente integrado y funcionando, era hora para ver cómo reaccionan los usuarios frente al producto. Así que enseñé el resultado del framework al desarrollador de simulaciones, el cual se esperaba una interfaz más sencilla de implementar.

Es por esto, que durante esta iteración, nos sentamos y empezamos a escribir juntos la nueva la interfaz, además de ciertos aspectos internos del framework.

El sistema resultante cambió radicalmente, fue un acierto el sentarnos juntos, ya que aseguramos el acuerdo entre el desarrollador del framework y el de simulaciones rápidamente. Esto nos permitió alcanzar un estado mucho mejor que el de ambos por separado y aprendiendo muchísimo por el camino.

Finalmente, la clase abstracta que resultó, la llamamos SGI Framework, clase de la que extienden los desarrolladores de simulaciones. Dicha clase abstracta, implementa una interfaz llamada SimulationAgent, que es la que ofrece realmente los métodos a implementar. Esto podría llegar a ofrecer en un futuro la posibilidad de usar el framework también como una librería.

Además, se ocultó la gestión de breakpoints y la suscripción a los watchers, siendo esto, transparentes para el desarrollador. Ésto le permite centrarse en lo que realmente le interesa, la simulación y cómo se accede a ella.

Duodécima: Iterando en la interfaz

Esta iteración será más extensamente descrita en la sección de “Diseño de la interfaz y obtención de requisitos”.

En esta iteración se basa en la continua mejora de la interfaz de usuario, intentando siempre, conseguir una interfaz lo más limpia e intuitiva posible. Ésto se debe a que aunque la interfaz era prácticamente la copia directa de Mockup, el cual no era más que una primera aproximación.

Así que se incrementó el uso de diálogos frente al de modales y la mejor situación de los distintos logos del framework al que se dirige este proyecto. También, se observa que existen distintas utilidades innecesarias que se acaban quitando, como por ejemplo, los ajustes de la simulación.

Durante estas iteraciones, se va alcanzando una interfaz mucho más usable, cuidada y profesional.

Diseño de la Interfaz y Obtención de requisitos

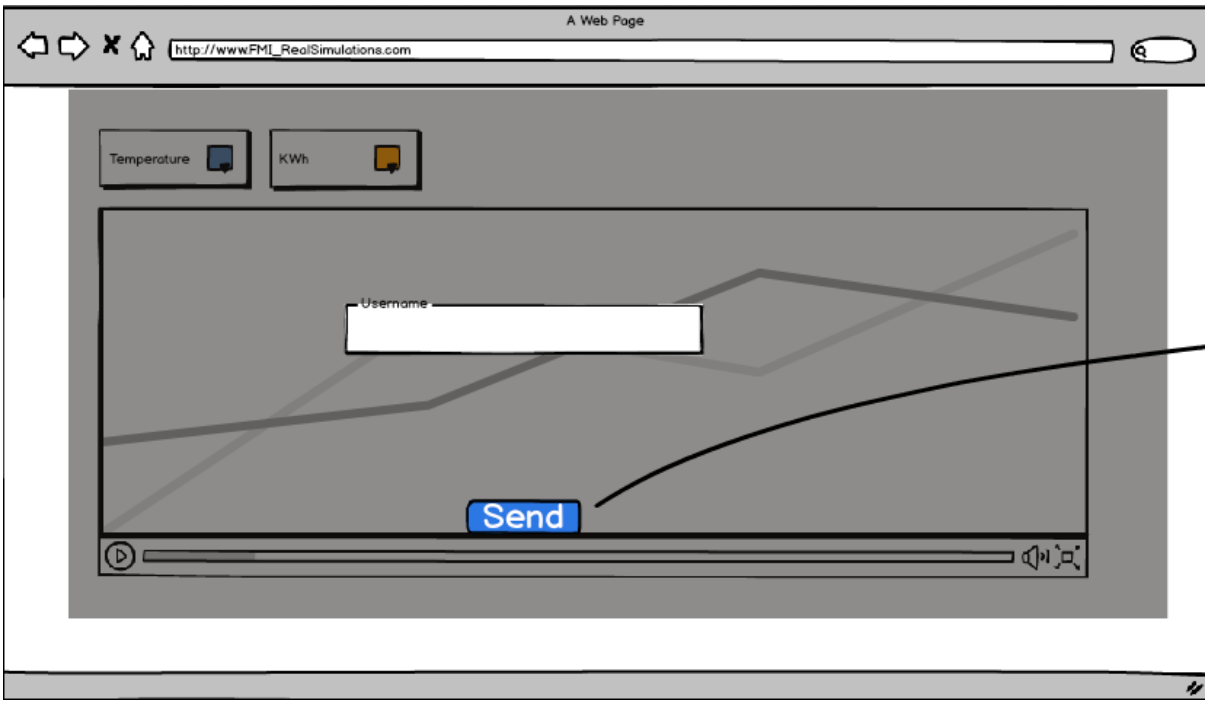
MockUp

1º Iteración

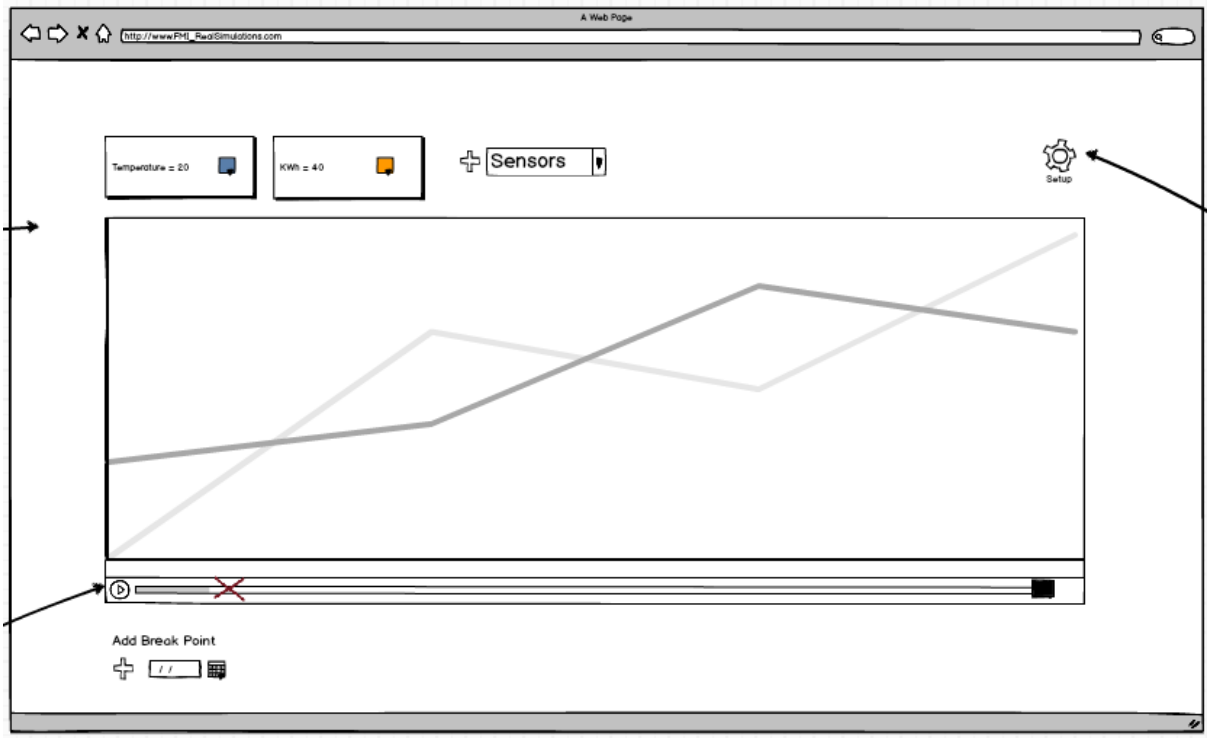
El mockup y su obtención final fue un resultado de una serie de iteraciones, de las cuales, se podían extraer gran cantidad de información acerca del proyecto. Los mockups fueron afinados en dos iteraciones.

En la primera iteración, desarrollé el mockup, con los requisitos que sabía hasta la fecha. Para ello, nos basamos en los videos de Youtube para mostrar los datos y el estado de la simulación.

Al no estar dentro de los objetivos la gestión de usuarios se podría elegir el nombre que se deseara. Para ello se ideó un modal, que ofreciera introducir su nombre de usuario nada más aterrizar en la web.



En la siguiente imagen se puede apreciar cuál fue mi primera aproximación al cliente web. Disponíamos de un diálogo integrado en el cual se seleccionaron los sensores e iríamos agregándolos en la lista superior. Además, dentro de los recuadros de los sensores seleccionados, nos irán saliendo los valores de los sensores. En el momento de un breakpoint, podríamos modificar el valor dentro del mismo.



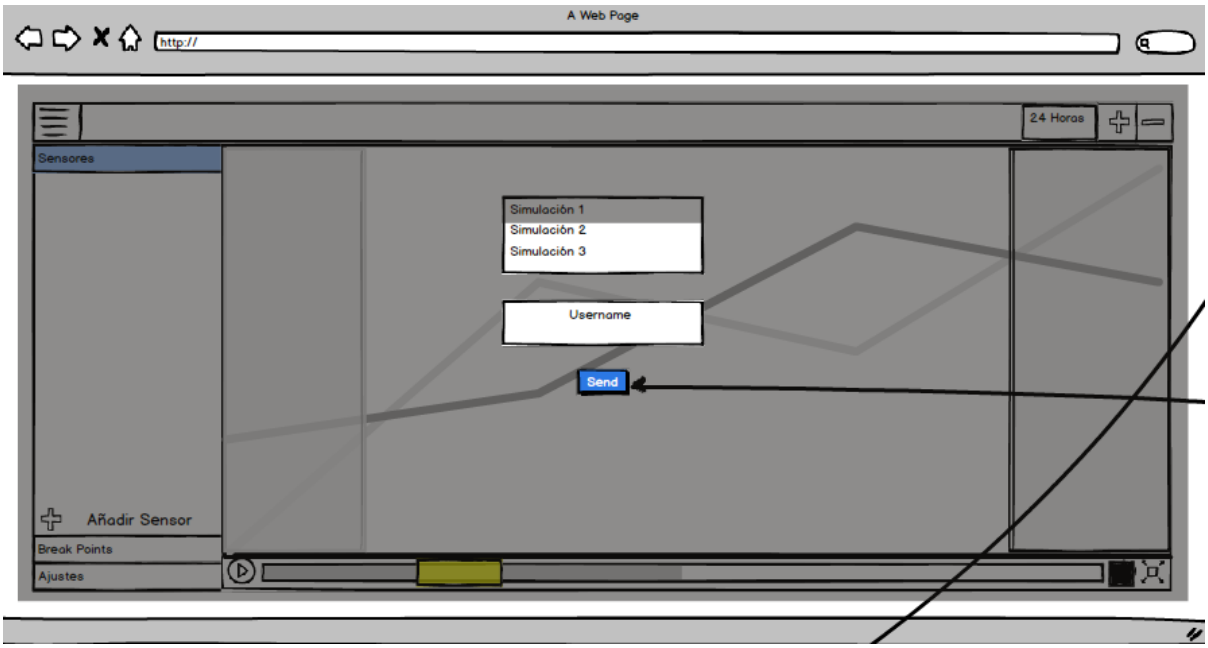
En la parte inferior se encuentra la acción de añadir breakpoints. Con un datepicker se puede elegir fácilmente cuando deseamos establecer nuestra parada.

2º iteración

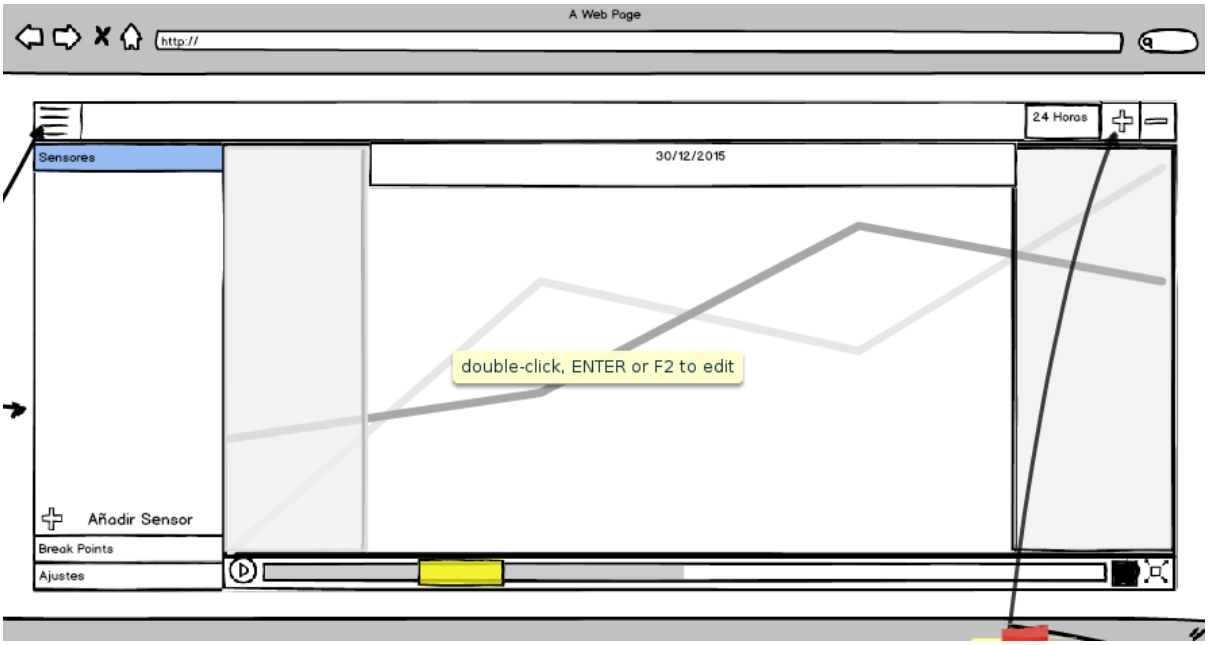
Después de la primera reunión, se llegaron a varios acuerdos entre los que destacan:

- Pueden coexistir varias simulaciones a la vez.
- Para establecer un sensor, hay que descender por una jerarquía de objetos
- El usuario podrá navegar por la gráfica con una ventana de tiempo.

Por esto, se pasó a modificar el mockup y reestructurarlo para adaptarlo mejor a las nuevas necesidades.



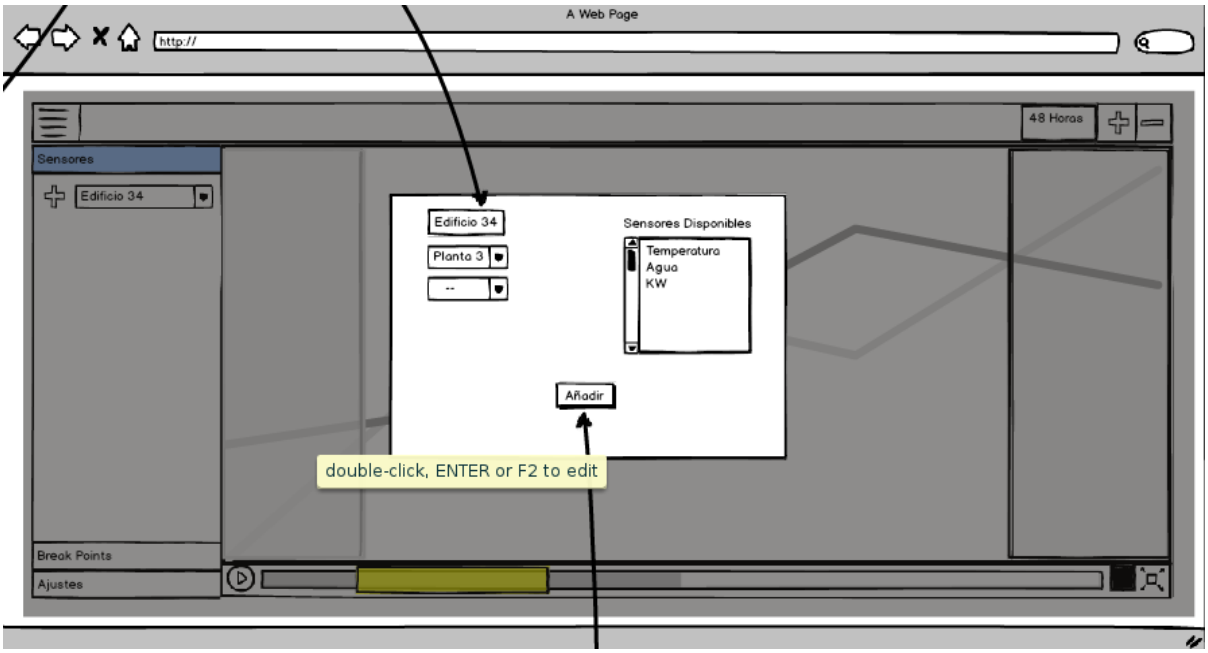
Como se observa en esta iteración se añadió al modal anterior una lista, para que se pudiera seleccionar entre más de una simulación



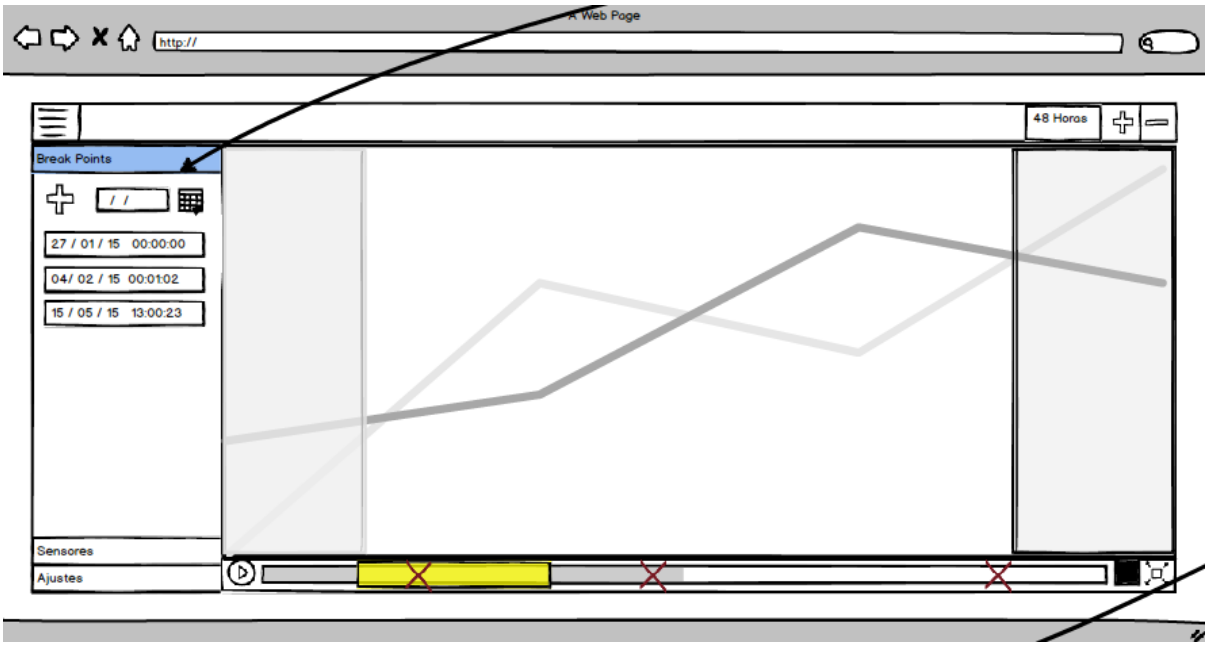
Además, se usó más los laterales, que los altos y bajos, ya que la tendencia de las pantallas es hacia ratios panorámico.

Por otro lado también, podemos ver que hay una marca amarilla en la barra de progreso, que indica la ventana de tiempo que estamos visualizando en la gráfica Esta ventana temporal puede ser modificada con el + y - que se observa en la parte superior. También es posible arrastrar a lo largo de la barra de progreso.

Otra aspecto nuevo es la parte superior de la gráfica, en la cual tendremos el nombre de la simulación, identificando así la simulación en la que estamos.



En esta ventana vemos el modal, que da soporte al requisito de navegar el modelo de la simulación a través de su jerarquía de contención. En cada selección del objeto, aparecerán los sensores disponibles. Posteriormente, al darle a añadir nos instalaría el sensor



En esta ventana,se muestra como se representan los breakpoints. En vez de estar en la parte inferior está en un lateral aprovechando el diseño panorámico de la mayoría de monitores

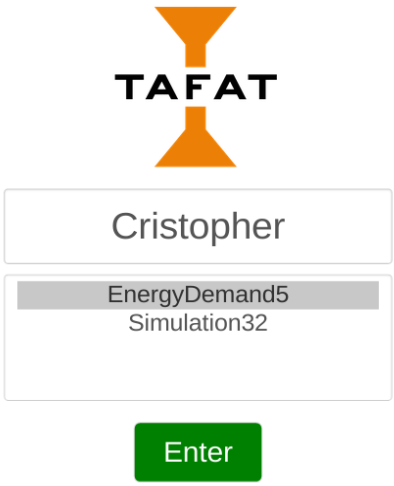
Iteraciones sobre la interfaz Gráfica

Durante esta sección me encargare de ir comentando la historia detrás de cada pantalla que sea mostrada además de explicar su sentido dentro de la aplicación.

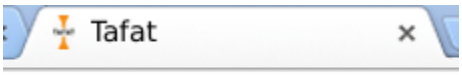
Después de muchas y pequeñas iteraciones que hemos ido superando, conseguimos algo bastante limpio y digno del esfuerzo del trabajo que hay, no solo en el cliente, sino también en el back-end del proyecto.

Antes de comenzar a explicarlo todo hay que comentar de que al principio la interfaz era muy similar a la del mockup, esa interfaz ha ido evolucionando hasta alcanzar lo que se presenta. Otro detalle es que el idioma elegido ha sido el inglés para darle la adaptabilidad a un entorno más internacional.

Pantalla de recibimiento

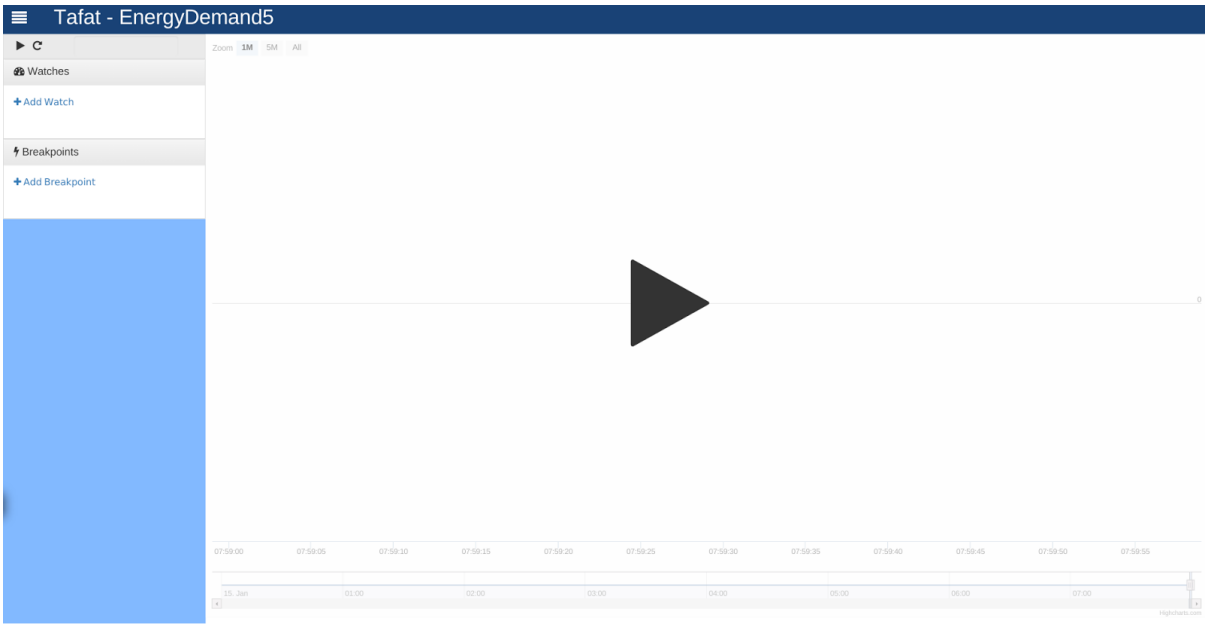


En la pantalla de inicio de usuario, vemos que la idea es la misma, pero ya no es una pantalla modal, si no que ahora es una pantalla de recibimiento o “landing page”. A esta página le hemos añadido el logo de TAFAT, agrandado el tamaño de letra y hecho más elegante el botón de entrar.



Además en la pestaña vemos que se ha incluido el logo y el nombre del framework de Tafat, esto cambiará al entrar dentro de la simulación.

Pantalla inicial

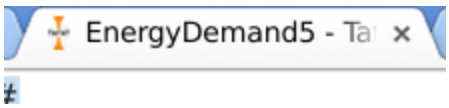


Como podemos ver esta pantalla ha evolucionado bastante desde el mockup. Para empezar el boton de play grande con un fondo degradado dejando entrever el gráfico, nos permite lo siguiente, y es que será imposible pulsar el botón de Play hasta que sea añadido al menos un watcher. Una vez añadido podremos ejecutarla.

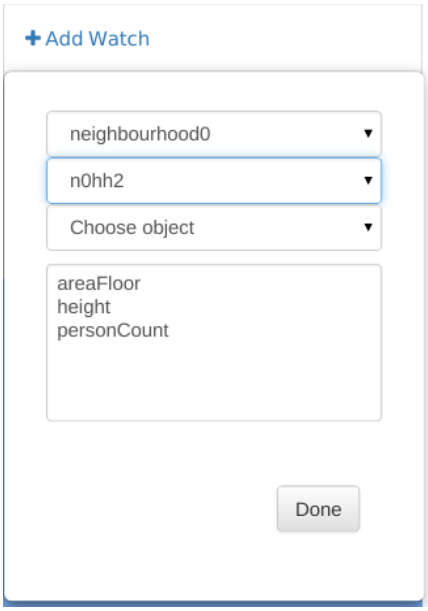
También se ha cambiado la situación del nombre de la simulación, antes en el gráfico, se ha colocado en la barra superior, junto con Tafat. Asimismo, se quitaron los botones de zoom y se integraron dentro del navigator (la barra de navegación del gráfico).

Los botones de play/pause y stop, se situaron junto con los menús. Así conseguimos seguir la diagonal de lectura natural (desde la esquina superior izquierda a la esquina inferior derecha), situando los botones más importantes en la esquina superior izquierda. El botón de play grande se encuentra en centro de dicha diagonal

Para continuar si nos volvemos a fijar en la pestaña del navegador, observamos lo siguiente.

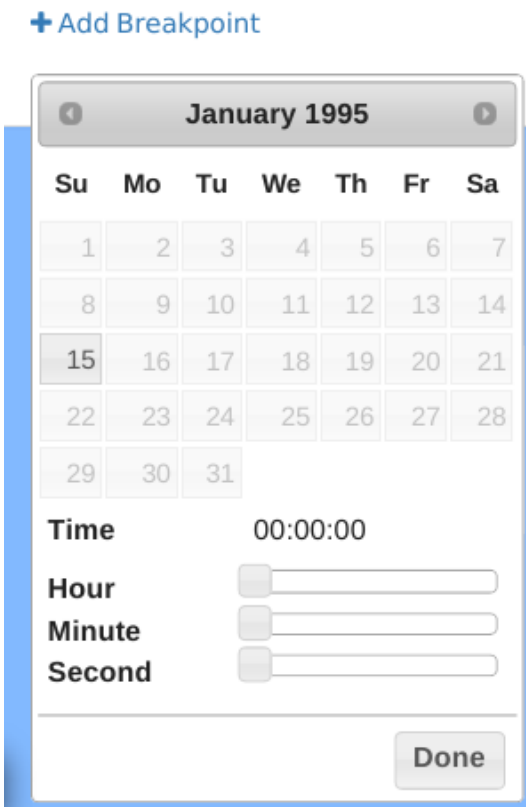


Se ve como el título de la pestaña cambia a "nombre de la simulación - Tafat" esto nos permitiría tener situadas varias pestañas con varias simulaciones dentro del mismo navegador, de esta manera podríamos situar su localización más cómodamente.



Diálogo de watches

Esta que se muestra es la apariencia final del diálogo de los watches, lo que antes era un modal. De esta forma, se consigue que el usuario disponga de una forma menos intrusiva y directa de bajar en la jerarquía de objetos. Una vez alcanzado el objeto deseado se puede seleccionar el watcher para así poder ver su evolución.

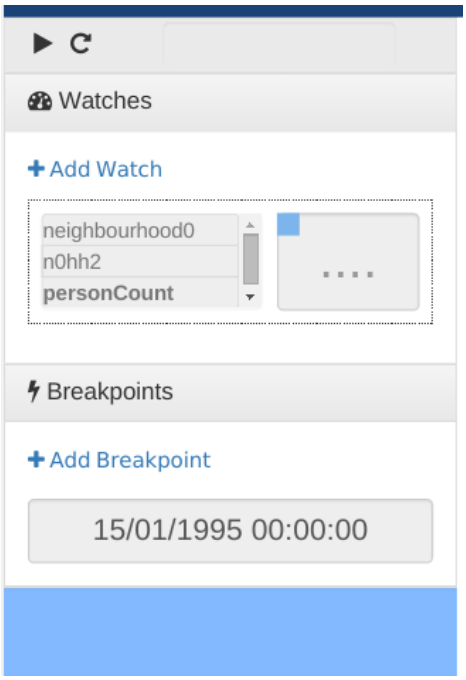


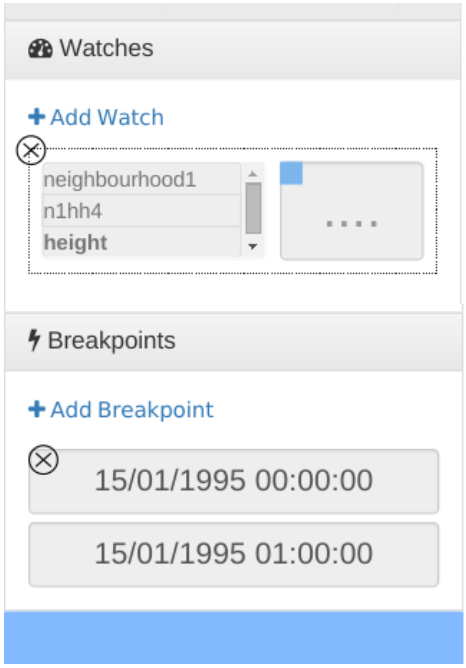
Diálogo de los breakpoints

El diálogo del breakpoint, es un datepicker, el cual nos permite seleccionar la fecha dentro del rango de tiempo de la simulación. Por ello los días que no están dentro de este tiempo, están deshabilitados.

Una vez seleccionado el momento concreto, se aprieta el botón “Done” y se añadirá dicho breakpoint a la simulación.

Representación de los breakpoints y los watches

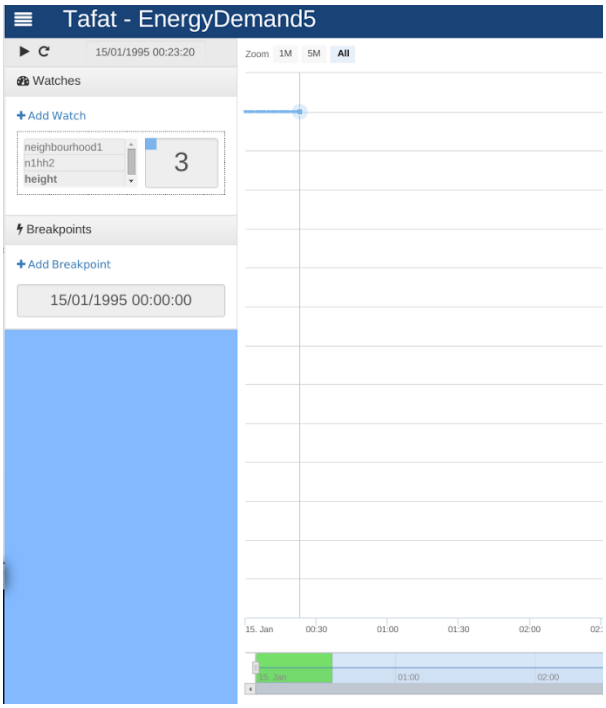


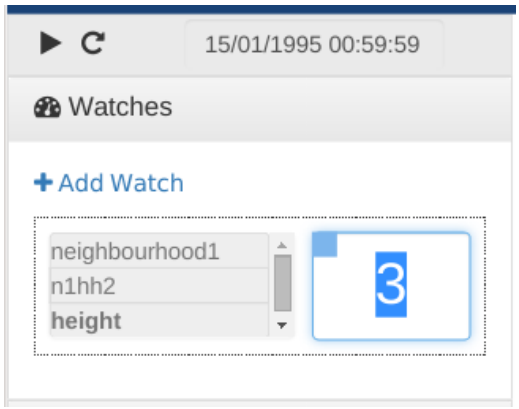


En la imagen superior vemos cómo ha evolucionado el concepto para representar los watches y los breakpoints. Esta representación es mucho más compacta que las anteriores, ya que se aprovecha mucho mejor el espacio del que se dispone. Otro aspecto interesante, es que el valor del watch antes de ejecutar la simulación o de haber recibido un valor, se mantendrá con puntos, ya que intenta representar que está a la espera de ser actualizado.

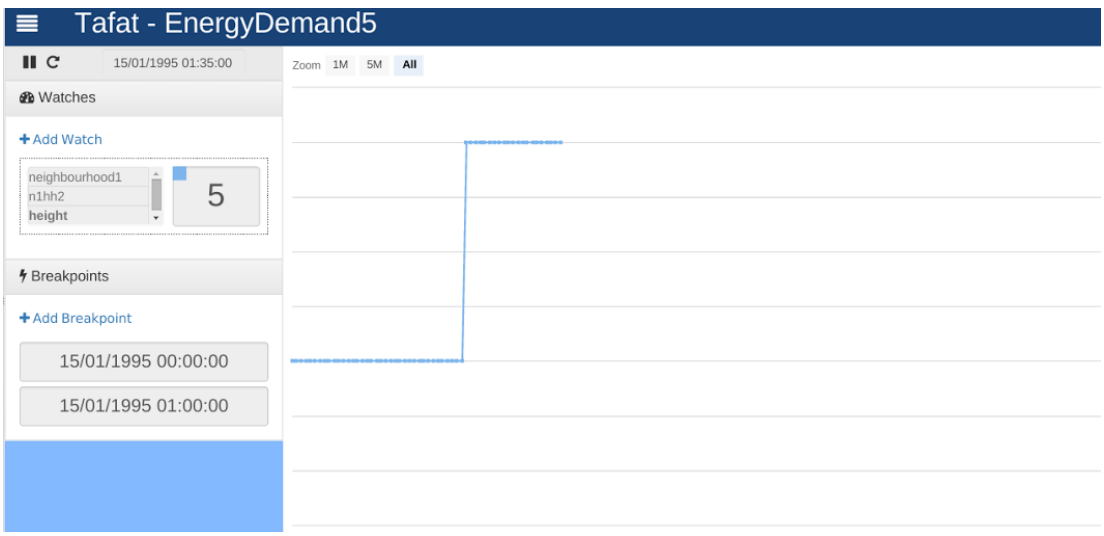
Además, los botones de eliminar watch o breakpoint, que se aprecian en la imagen de la derecha, solo aparecen al pasar el ratón encima del elemento. Así, no hay nada que contamine la vista hasta que sea necesaria su aparición.

Ejecución visual de los breakpoints



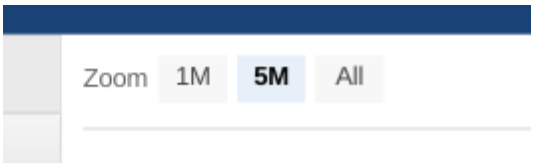


En la ejecución normal, se puede apreciar en la imagen de la izquierda, donde la línea de puntos va expandiéndose, en este caso con un valor constante de tres. Más tarde, al ejecutarse un breakpoint, la vista del watch cambiará ligeramente. Este cambio, es la activación de los inputs de los watches. Así se podrá, cambiar el valor para ver como este evoluciona al reanudar la simulación.



El la imagen superior se comprueba el resultado del cambio del valor en el watch.

Botones de zoom



Además de poder cambiar la ventana temporal en el navegador de la gráfica, en la parte superior izquierda de la gráfica se puede cambiar el nivel de zoom a tres por defecto. Estos son :

- “1M” 1 minuto
- “5M” 5 minutos
- “All” Todo el tiempo que dura la simulación.

Esto ofrece la comodidad, de que si el usuario no quiere usar la barra de navegación para adaptarla a su necesidades, tiene la otra posibilidad de usar los botones.

Botón para imprimir el gráfico

La otra funcionalidad de la que se dispone, es la de poder obtener una imagen del gráfico. Esta funcionalidad, se encuentra en el botón, situado en la barra superior al lado del nombre. En un principio este botón, estaba junto al gráfico, pero se comprobó que era mucho más natural situarlo en la esquina superior.

Este botón nos permite descargar tres tipos de documentos, PNG, JPG y PDF, mientras que en la última opción imprime directamente el gráfico.

Tecnología usada

En un principio se me dieron a elegir, distintas tecnologías de las cuales tuve que decidir cuál era la más apropiada o cual era la mejor para mis objetivos tanto personales como profesionales.

Para el cliente, se me habló sobre la tecnología GWT, una tecnología de google muy potente, que permite escribir código Java para posteriormente traducirlo a Javascript. Esto permite, trabajar con el tipado de Java, y el seguimiento en el debug de manera mucho más controlada que con Javascript. ¿Entonces por qué elegirlo?

Mi elección por Javascript, es porque considero que adaptar la forma de programar a distintas filosofías de programación es muy enriquecedor, porque te permite aumentar tu perspectiva. Este, es un lenguaje con una filosofía distinta y casi contrapuesta a la de Java, así que me pareció muy atractivo elegirlo. Además, laboralmente es muy demandado y era un lenguaje que no dominaba bien y por el cual, tenía especial interés en aprender en más profundidad.

De lado del servidor, se me dio la posibilidad de usar un framework para la gestión de las peticiones HTTP, pero me negué. Esto es debido a que mi experiencia con los Frameworks, no solo de Java, ha sido bastante mala, y los frameworks que han llegado a gustarme eran bastante minimalistas. Por este motivo, comencé a buscar la solución en Java nativo, y aunque se que en realidad no estaba inventado nada nuevo, estaba creando una forma de liberarme de la necesidad de usar un framework. Mi aversión a los frameworks es debido a que obligan a aprender una forma de pensar, que más que ayudarme a alcanzar una buena solución, en muchos casos obliga a ofuscarla para alcanzarla.

A la hora de trabajar con el cliente, llegó un momento donde me fue imprescindible el uso de un lenguaje de plantillado. Buscaba un lenguaje no dependiente de la plataforma del servidor, ni de frameworks, que no tuviera que ser renderizado en el servidor, si no en el navegador y Dust, fue la solución.

En mi búsqueda un compañero me facilito un artículo de Linkedin, en el que probaban distintos lenguajes de plantillado en una hackaton realizada por la empresa. Posteriormente se consultaba la sensación de los desarrolladores hacia los lenguajes usados. Al final se establecían ciertas métricas para alcanzar un lenguaje que llegará a cumplir sus objetivos y el ganador fue Dust. En aquella época era una proyecto de software libre descontinuado por su creador, así que Linkedin realizó Fork y continuó con el proyecto.

Dust es un lenguaje que me pareció sencillo, versátil y se adapta perfectamente a mi objetivo de un lenguaje de plantillado en el browser. El lenguaje permite el compilado en el navegador, pero recomienda que esta utilidad solo se use en fase de desarrollo ya que implica una carga extra para el navegador.

Java

Java es un lenguaje de programación tipado y orientado a objetos, creado con la intención de compilarlo una vez y ejecutarlo en cualquier dispositivo (WORA, “Write Once, Run Anywhere”). Esto lo consigue gracias a su máquina virtual, la cual da una plataforma común para ser ejecutada en cualquier sistema operativo, sin preocuparse, de tener que recompilar el programa.

La sintaxis de Java, es muy parecida a la de C y C++, pero no es un lenguaje orientado a bajo nivel. Además, ofrece un recolector de basura, por lo que no necesita de la implementación de destructores y nos evita problemas con las fugas de memoria.

Java es uno de los lenguajes más usados de nuestra época, disputándose el primer lugar junto con C, según estadísticas oficiales de TIOBE, particularmente es muy usado para aplicaciones de cliente-servidor.

Este es mi caso, ya que Java, es el responsable de responder antes las distintas peticiones de los clientes web y gestionar el transcurso de las conexiones de los frameworks.

Javascript

Javascript, al contrario que Java, es un lenguaje de tipado dinámico, basado en prototipos, interpretado y orientado a eventos. Este lenguaje fue creado por Brendan Eich de Netscape se comenzó llamando Mocha, luego LiveScript y finalmente Javascript, justo cuando Sun y Netscape se unieron. Se pretendía que fuera un lenguaje de scripting complementario a Java.

Javascript, cuenta con gran apoyo, no solo para el de desarrollo de clientes web, si no que también está comenzando a incrementarse para el desarrollo de aplicaciones del lado del servidor. Actualmente la plataforma más conocida para este fin es Node.js, pero la realidad es que desde el principio llegó a contar con dicha posibilidad con Netscape Enterprise Server, lanzado en diciembre de 1994.

Javascript es uno de los 10 lenguajes de programación más usados del mundo según TIOBE por debajo de Python y su uso se incrementa cada vez más, debido su versatilidad y al amplio espectro de uso del que dispone.

Dentro de mi proyecto, es usado principalmente para el cliente web y también para compilar las plantillas Dust desde el servidor gracias a Node.js.

Asynchronous Module Definition (AMD)

AMD o Asynchronous Module Definition (Definición Asíncrona de Módulos) especifica un mecanismo para definir módulos, que serán cargados de manera asíncrona junto con sus dependencias. Esto es perfecto para los navegadores, ya que la carga síncrona incurre en problemas de ejecución, usabilidad y búsqueda de errores.

¿Pero qué es un módulo en Javascript y cual es su propósito? El patrón Module tiene como objetivo encapsular una pieza de código dentro de una unidad, dejando solo una interfaz pública, permitiendo así el poseer métodos y variables privadas.

De esta manera no solo se convierte en una herramienta para la carga de dependencias y módulos de Javascript, sino también en una buena práctica para estructurar el código. Además, conseguimos erradicar la práctica de hacer los imports en un orden determinado para conseguir el correcto funcionamiento del cliente.

Para usar esta tecnología utilicé la librería RequireJS, la cual era recomendada en la documentación oficial de JQuery, como una buena práctica a la hora de trabajar con Javascript.

HyperText Markup Language (HTML)

HTML o Lenguaje de Marcas de hipertexto, permite estructurar los componentes de una página web. Este lenguaje, se basa en que no dispone de los elementos necesarios para la entera construcción de la página, si no que usa la referenciación, de ahí, hipertexto.

Se considera un subconjunto de SGML, un conocido lenguaje de etiquetas del año 1989, creado por Tim Berners-Lee. Además junto con HTML se crea HTTP, URL y con todo esto las bases fundamentales de la World Wide Web.

HTML a lo largo de la historia ha ido incorporando nuevas características y suprimido otras, con el fin de hacerlo más eficiente y compatible con los distintos navegadores para las distintas plataformas. Pretende ser escrito una vez y que bajo cualquier navegador actualizado entienda la versión y sea capaz de interpretarlo.

Algo a tener muy en cuenta es que HTML, solo representa la estructura de un documento. Aunque, HTML ofrezca opciones para representar ciertos aspectos visuales es una mala práctica, la cual debemos desplazar a las hojas de estilo o CSS.

Cascading Style Sheets (CSS)

CSS u Hojas de estilo en cascada, es un lenguaje usado para definir el estilo de una página HTML o XML. Es la forma para separar el estilo de la estructura de de un documento, escrito en HTML, así, se consigue alcanzar un documento mucho más limpio y fácil de leer.

CSS trabaja con selectores, los cuales son los que enlazan el estilo con el que se va a mostrar cada elementos del documento seleccionado, de esta forma, podemos describir varios reglas de estilo para un mismo selector.

JQUERY

JQuery es una librería de Javascript, que facilita el manejo del DOM entre otras muchas cosas, tal como el manejo de eventos, animaciones, peticiones Ajax. Por todo esto y por la facilidad de uso que ofrece se ha convertido en una de las más usadas y populares en la web.

JQuery, es un proyecto de software libre, pero dispone de doble licencia, la Licencia del MIT y la Licencia Pública General de GNU, esto permite ser usado tanto en proyectos libres como privados.

Esta librería es usada por empresas importantes como Microsoft la cual ha añadido en su IDE Visual Studio y tambien es usada en sus frameworks ASP.NET, ASP.NET MVC y AJAX.

DUST

Dust es un motor de plantillas para uso en Javascript. Hereda de la familia de los lenguajes de CTemplate, anteriormente llamado Google Templates, un proyecto originalmente usado por Google en su motor de búsqueda. El cambio de nombre fue para darle un nombre más genérico y acorde a la comunidad que ahora lo mantiene.

Por su familia CTemplate, es un motor diseñado para ejecutarse de manera asíncrona tanto en servidores como en navegadores. Dust procede de un autor individual distinto, pero debido a la falta de mantenimiento del proyecto, Linkedin interesado por él hizo un fork y lo continúa manteniendo cuatro años después.

Dust promete las siguientes características y objetivos:

- Menos lógica, pero no exento de esta.
- Carga, renderizado y streaming de manera asíncrona.
- Plantillas integrables (Permite unir distintas plantillas entre sí).

- Agnóstico del formato (No solo permite generar HTML).
- Alto rendimiento.
- Ampliamente soportado.

APACHE

Apache es un servidor web, que se encarga de servir contenido estático y dinámico. Es un proyecto de código abierto, basado inicialmente en el NCSA Httpd, servidor web conocido en el año 1995, aunque posteriormente Apache acabo sobreescribiendo el código al completo.

El nombre de Apache le viene de su creador Behelendorf, el cual quería representar algo firme y que mantuviera internet tal y como los primeros ingenieros informáticos lo habían creado antes de que las grandes compañías llegarán para cambiarlo todo a placer.

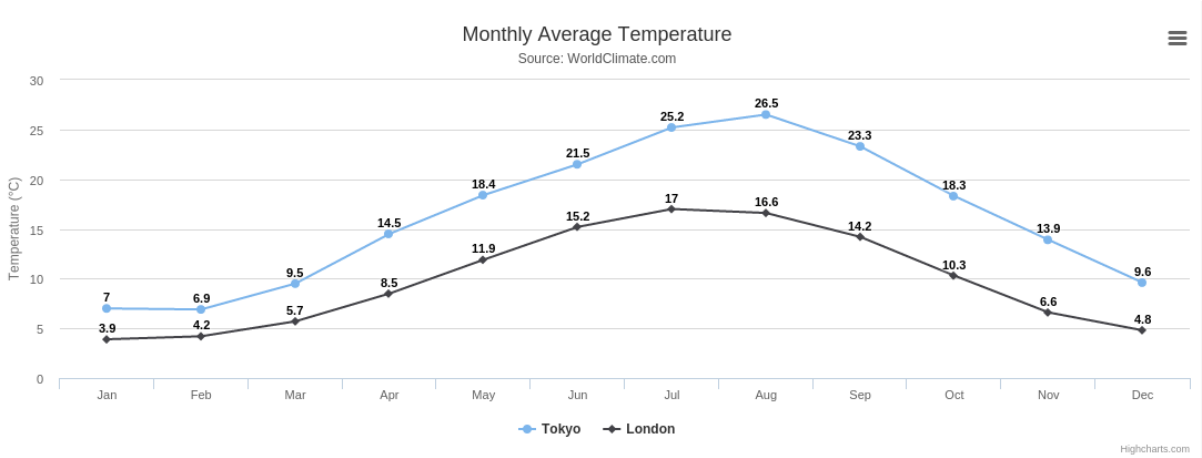
Gracias a la noción de sitio virtual, nos permite, redirigir las peticiones a servidores de contenido dinámico, pudiendo centralizar todas las peticiones sobre el mismo servidor y así entre mezclar los servidores para dar la sensación de que es el mismo servicio el que se encarga de ofrecer el contenido.En mi proyecto fue usado con este fin, el contenido estático es el de la página web y el dinámico era servido por el Simulation Sites.

Highcharts

Higcharts es una potentísima librería de Javascript, la cual te permite generar una gran cantidad de tipos de gráficos, desde mapas a velocímetros, hasta gráficos lineales como el que usamos en nuestro trabajo. Highcharts dispone también para generar gráficos dinámicos, es decir, que el gráfico sea un objeto vivo, que puede ir siendo modificado en tiempo real.

Esta librería, nos permite hacer uso de ella de una manera limpia gracias a su API, limpia y clara. Además cuenta con una documentación tan extensiva, como el desarrollador quiera y con un incontable número de ejemplos los cuales enseñan de forma práctica su forma de uso.

La única, problemática que podría tener Higcharts, es que el renderizado continuo implica un coste de eficiencia, teniendo que controlar su uso de alguna forma. En cuanto a los precios la compañía, nos permite usarlo de manera gratuita, ya que es para propósito educativo.

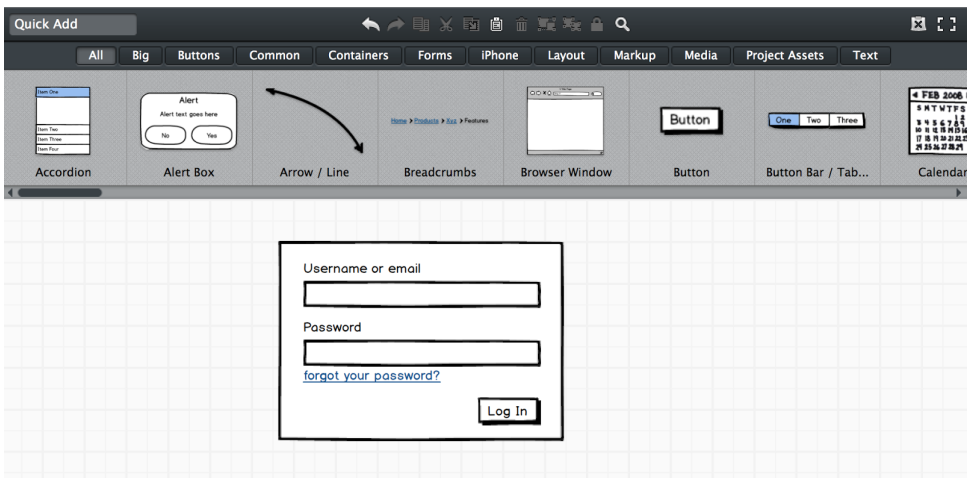


Balsamiq

Es una herramienta para realizar mockups, la cual dispone de muchísimos componentes predefinidos para construir rápidamente las ideas sobre tu aplicación. Esto permite no perder la mayor parte del tiempo en construir componentes que siempre son muy parecidos, o preocuparte sobre detalles que en un principio, no tienen importancia, como el color, etc.

Balsamiq es una herramienta, que no solo permite construir prototipos, sino que también ayuda a captar de forma rápida y ágil los requisitos del cliente, pudiendo llegar a un acuerdo sobre la marcha mientras retocas el mockup.

La herramienta existe desde el 2008 y su principal objetivo es ayudar a los desarrolladores a crear aplicaciones o páginas web más fáciles de usar.



Protocolos

User Datagram Protocol (UDP)

UDP o User Datagram Protocol, es un protocolo no orientado a conexión, que nos permite el envío de datos, de una manera rápida. Algo a comentar sobre UDP es que no está orientado a conexión, por lo tanto no asegura el correcto envío de los datos.

Ya que UDP no necesita de una contesta para su funcionamiento, ni de una señal de que el datagrama ha llegado bien, UDP puede ser usado para el envío de señales broadcast por una red, para de esta manera detectar distintas localizaciones de servicios en la misma.

Así mismo, se pueden enviar peticiones de broadcast buscando un servicio determinado (Por ejemplo, un servidor DHCP). Por ese mismo motivo ha sido necesario en mi proyecto, ya que uno de los requisitos, es que la propia simulación detectará automáticamente donde se encontraba el Simulation Sites. De esta manera no hay que preocuparse de la distribución de los servicios dentro de la red.

HyperText Transport Protocol (HTTP)

HTTP o Hypertext Transport Protocol, es un protocolo creado para trabajar por encima de TCP/IP, sobre la capa de aplicación. HTTP es un protocolo orientado a transacciones y sigue el esquema de petición/respuesta. Además, HTTP no conserva estado, si no que esto recae en el cliente o el servidor que realizan las distintas peticiones, en caso de ser necesario.

HTTP es muy utilizado entre las peticiones entre navegadores web y servidores, ya que HTTP es una de las grandes bases del Hipertexto. Por tanto, todo elemento enlazado e identificado por una URL, es requerido a través de una petición HTTP al servidor del que disponga de él.

Pero este protocolo, puede ser usado también para el envío de información, ya que permite enviar texto entre sus datos. De esta manera, puede haber una comunicación directa entre el cliente y el servidor.

Dentro de mi proyecto, una vez detectado dónde está el Simulation Sites, este ofrece un puerto en el que ejecutarse un socket TCP en el Framework. A partir de aquí, comienza una comunicación por HTTP. Esta comunicación entre ambos permite el redireccionamiento de las peticiones de los clientes hacia el Framework, así como las peticiones del Framework hacia el Simulation Sites.

Websocket

Websockets es un protocolo que proporciona una forma de comunicarse por el mismo puerto usado por el protocolo HTTP (80) de forma bidireccional. Esto otorga una manera de comunicación directa, en la que se pueden enviar notificaciones rápidamente sin preocuparse de los puertos que pudieran haber cerrados.

El protocolo Websocket fue normalizado por el IETF en el RFC 6455 mientras que la API está actualmente siendo normalizada por el W3C. Además, esta tecnología es actualmente soportada por la mayoría de los navegadores, Mozilla Firefox 8, Google Chrome 4, Safari 5, Internet Explorer 10 y la versión móvil de Safari en el iOS 4.2.

El funcionamiento básico del protocolo se inicia en un principio desde el protocolo HTTP, este le permite empezar a generar las distintas peticiones para la negociación de la coneccion por Websockets. Así pues, el cliente mandará un petición de negociación de WebSocket y el servidor le responderá. Después de terminar el handshake se podrá empezar a transmitir en modo Full-duplex, donde se enviarán tramas de únicamente 2 bytes.

Las tramas pueden ser en formato binario o en formato texto, no obstante, las tramas binarias no están aún soportadas por la API.

Conclusiones y trabajos futuros.

Conclusiones

En este proyecto se ha afrontado el reto de añadir la interacción humana sobre la simulación de sistemas complejos en tiempo de ejecución tanto para modificarla como para simplemente observarla. Esto ha implicado una serie de cuestiones tales cómo abordar la representación de un sistema complejo o cómo ofrecer una manera sencilla para alterar la simulación.

Así es que, con este reto entre manos, creamos **Simulation Gateway Interface** o **SGI**. Una forma de visualizar y modificar una simulación en tiempo de ejecución. Asimismo, se consigue representar de una manera sencilla, un modelo complejo, para que el usuario pueda interactuar fácilmente con el modelo simulado. Por otro lado, se ha creado un framework que permite exponer una simulación a través de un servicio web, el cual se implementa fácilmente y es poco intrusivo.

Pero este software, no es algo meramente conceptual, el proyecto ha sido integrado dentro de una simulación real, realizada en Tafat, framework desarrollado por el SIANI, que se ha usado extensivamente en proyectos de I+D con empresas como EIFER (European Institute For Energy Research) o EDF (Électricité de France).

Posteriormente, SGI ha sido mostrados a los usuarios de Tafat para comprobar si las necesidades de interacción con las simulaciones están cubiertas. Esta ha sido positivamente acogida por los usuarios de Tafat. Este proceso de validación es necesario para identificar la magnitud que SGI puede tener en el mundo de la visualización de simulaciones de sistemas complejos.

Profesionalmente, el trabajo de fin de grado, me ha supuesto una experiencia muy enriquecedora, ya que me ha permitido trabajar en profundidad con diversas tecnologías como, por ejemplo, las comunicaciones de red, la creación de servicios web, el desarrollo de un cliente web, etc.

Asimismo, el poder trabajar con un proyecto más grande de lo que estoy acostumbrado me ha hecho ver la importancia de analizar correctamente las necesidades y requisitos y abstraerse de como implementarlo, algo bastante complejo cuando se parte de una profesión técnica como la informática.

En cuanto a mi evolución personal, considero que lo más importante que he aprendido, es a no sentir vergüenza por enseñar una solución no finalizada o no del todo correcta. Ésto me ha permitido ser más ágil a la hora de enfrentarme a las dificultades que se me han ido planteando por el camino.

Como trabajo futuro, considero cuatro líneas de trabajo que serían muy interesantes a continuar:

1. Una de las modificaciones, podría ser crear un standalone, para que un desarrollador de simulaciones, no deba preocuparse de ningún tipo de configuración de Apache ni del Simulation Sites.
2. Dar la facilidad para seleccionar, varios watches, a través del uso patrones o expresiones regulares. Por ejemplo “fridges*”.
3. Una posibilidad interesante sería poder definir indicadores de alto nivel, para ofrecer al usuario una forma más avanzada de detectar características intrínsecas de la simulación.
4. Otro trabajo muy interesante sería dotar de más tipos de representaciones gráficas como, por ejemplo, mapas que permitan geolocalizar elementos de la simulación, mapas de calor, gráficas de correlación, etc.

Bibliografía

[1] Evora, J. (2014). A Methodological Approach on Software Engineering applied to Smart Grids using Complex Systems. ULPGC.

[2] Palensky, P., Kupzog, F., Zaidi, A. A., & Zhou, K. (2008). Modeling domestic housing loads for demand response. In Industrial Electronics, 2008. IECON 2008. 34th Annual Conference of IEEE (pp. 2742–2747).

[3] Advances in Complex Systems. (2010). World Scientific Publishing Co. Retrieved from http://www.first-pages.com/ukss/systems/journals/JournalDetails?journal_id=24

[4] Nicolis, G., & Nicolis, C. (2012). Foundations of Complex Systems: Emergence, Information and Predicition. World Scientific.

[5] Evora, J., Hernandez, J. J., & Hernandez, M. (2014). Advantages of Model Driven Engineering for studying complex systems. Natural Computing, 1–16.

[6] Robert C. Martin (2008). Clean Code, A Handbook of Agile Software Craftsmanship

[7] Martin Fowler 2000. Refactoring, Improving the Design of Existing Code.

[8] Jeff Kreeft Meije (2010). Using git-flow to automate your git branching workflow http://jeffkreeftmeijer.com/2010/why-arent-you-using-git-flow/

[9] Vincent Driessen (2010). A successful Git branching model http://nvie.com/posts/a-successful-git-branching-model/