DEPARTAMENTO DE INGENIERÍA TELEMÁTICA

PROYECTO FIN DE CARRERA

INGENIERÍA EN INFORMÁTICA

APLICACIÓN WEB PARA AUTOMATIZACIÓN Y VISUALIZACIÓN DE SIMULACIONES DE REDES PRIME

AUTOR: ANDRÉS J. FERRÉ COLLADO

TUTOR: DR. GREGORIO LÓPEZ LÓPEZ

CO-DIRECTOR: DR. JAVIER MATANZA DOMINGO

LEGANÉS, OCTUBRE DE 2015

TÍTULO: APLICACIÓN WEB PARA AUTOMATIZACIÓN Y VISUALIZACIÓN DE SIMULACIONES DE REDES PRIME.
AUTOR: ANDRÉS J. FERRÉ COLLADO
TUTOR: DR. GREGORIO LÓPEZ LÓPEZ
CO-DIRECTOR: DR. JAVIER MATANZA DOMINGO

EL TRIBUNAL

PRESIDENTE: _____

VOCAL: _____

SECRETARIO: _____

REALIZADO EL ACTO DE DEFENSA Y LECTURA DEL PROYECTO FIN DE CARRERA EL DÍA 29 DE OCTUBRE DE 2015 EN LEGANÉS, EN LA ESCUELA POLITÉCNICA SUPERIOR DE LA UNIVERSIDAD CARLOS III DE MADRID, ACUERDA OTORGARLE LA CALIFICACIÓN DE

VOCAL

SECRETARIO PRESIDENTE

A mi madre

Agradecimientos

A Gregorio, mi tutor, por ser un guía excepcional, por su apoyo y dedicación, por sus ánimos, por hacerlo posible, por estar ahí cuando ha hecho falta y por darme la oportunidad de trabajar con él, incluso estando yo a 9000km de distancia y con 7 horas de diferencia.

A todas las personas que han contribuido a mi formación y educación, que con su dedicación, me han ayudado a cumplir mis metas.

A mis amigos de la universidad Carlos III, a los de BEST y a los de mi otra universidad, la de Twente, con los que he compartido aventuras y desventuras, horas de biblioteca, días y noches de prácticas, con los que he celebrado grandes ocasiones y que me han ayudado a remontar de todos mis tropiezos, con los que he cruzado fronteras en bici y con los que me he reencontrado en otro continente. Aunque ya no compartamos nuestro día a día, me habéis ayudado en esta lucha y me habéis hecho como soy. Aunque muchos de nuestros caminos se han separado, espero que algún día se vuelvan a cruzar. Sé que de esta experiencia me llevo amigos para siempre, que han visto mi mejor y mi peor cara. En especial, Castre, Lau, Irene, Alba, Miguel, Nuria, Alberto, Dani... gracias. Dankje Nikos, Ignacio en Diego (look! A dom!).

Al resto de gente que me importa, a mis amigos de Pinto, Lega y el pueblo, a Paloma, Elena, Estibaliz, Ángel y Pablo, por los buenos momentos y por aguantarme y entenderme. A David, que a veces lo hace mejor que yo. Espero que llegue el día en que pueda dedicaros el tiempo que os merecéis.

A mi familia, por su apoyo.

A mi padre, porque sé que estaría orgulloso.

A mi madre, por su amor y su apoyo incondicionales, por sostenerme cuando todo parecía derrumbarse. Te quiero.

“One day ladies will take their computers for walks in the park and tell each other, ‘My little computer said such a funny thing this morning’”

Alan Turing

Resumen

Las comunicaciones PLC de Banda Estrecha están ganando importancia como tecnología de última milla en despliegues de infraestructuras avanzadas de medición. PRIME es un estándar de comunicaciones PLC de Banda Estrecha ampliamente utilizado en España (por ejemplo, por Unión Fenosa o Iberdrola), y con proyección internacional. Las herramientas de simulación son especialmente importantes en este ámbito, pues permiten planificar, evaluar y tomar decisiones relativas a redes PRIME, minimizando riesgos, tiempo y coste. Sin embargo, la complejidad y curva de aprendizaje de los simuladores de redes de comunicaciones dificulta su manejo en entornos operativos. El objetivo de este Proyecto Fin de Carrera es diseñar, desarrollar y validar una aplicación Web que permita ejecutar simulaciones y visualizar resultados de forma amigable.

El presente Proyecto Fin de Carrera ha sido desarrollado dentro del ámbito del proyecto de investigación nacional OSIRIS (Optimización de la Supervisión Inteligente de la Red de Distribución), financiado por el Ministerio de Economía y Competitividad y liderado por Unión Fenosa Distribución (tercera distribuidora eléctrica a nivel nacional).

Palabras clave

Simulador de Redes de Comunicaciones; Comunicaciones PLC de Banda Estrecha; PRIME; Red Eléctrica Inteligente; Aplicación Web.

Abstract

Narrow Band PLC communications are gaining importance as last mile technologies in Advanced Metering Infrastructures. PRIME is a Narrow Band PLC communications standard widely used in Spain (e.g., by Unión Fenosa or Iberdrola), and with international projection. Simulation tools are particularly important in this area, as they allow planning, evaluating and making decisions about PRIME networks, minimizing risks, time and costs. However, the complexity and learning curve of the communication network simulators makes difficult to handle them in operational environments. The aim of this Thesis is to design, develop and validate a Web application which allows running simulations and visualizing their results in a human-friendly manner.

This Thesis has been developed within the scope of the national research project OSIRIS (Optimization of the Distribution Network Intelligent Monitoring), funded by the Spanish Ministry of Economy and Competiveness and led by Unión Fenosa Distribución (third company in the Spanish electricity distribution market).

Keywords

Communications Network Simulator; Narrowband Power Line Communications; PoweRline Intelligent Metering Evolution; Smart Grid; Web Application.

Índice general

Índice de figuras

Capítulo 1

Introducción

Este capítulo proporciona una visión global del Proyecto Fin de Carrera ayudando a contextualizarlo. En él se explican los motivos que han llevado a acometer el proyecto así como los objetivos que se pretenden alcanzar. Por último se incluye una guía en la que se describen la estructura y contenido de la presente memoria.

1.1 Contexto

1.1.1 Deficiencias de las Redes Eléctricas Tradicionales

La topología básica de la red eléctrica tradicional es estrictamente jerárquica, con una clara diferenciación entre sus distintos subsistemas de generación, transporte, distribución y consumo, como muestra la Figura 1. Otra de las principales características de estas redes es que son unidireccionales por naturaleza: la energía fluye desde las plantas de generación hasta el consumidor final.

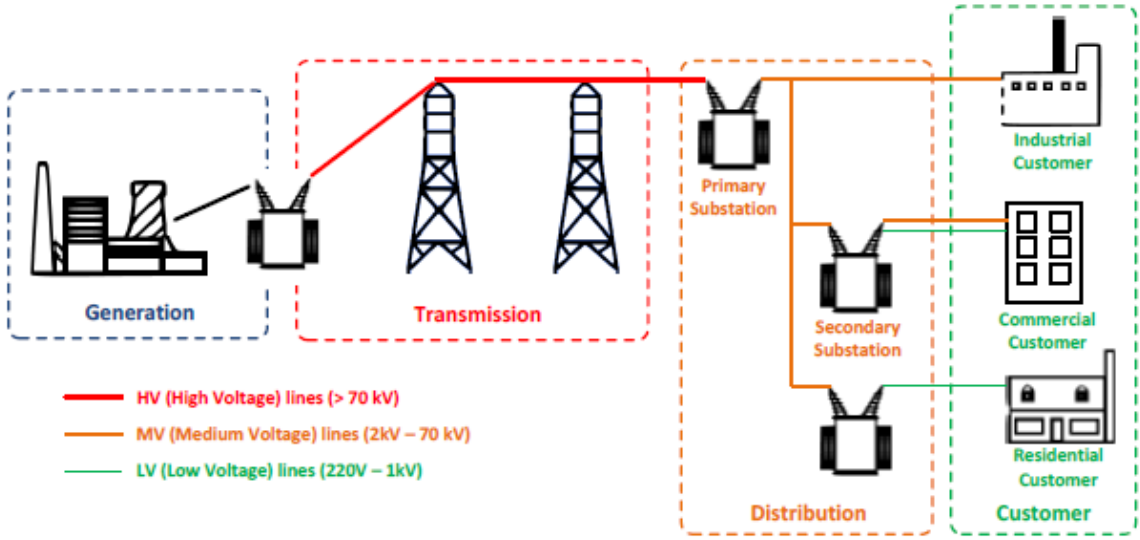


Figura 1: Jerarquía de las Redes Eléctricas Tradicionales.

Esta topología tradicional provoca una serie de ineficiencias, como que la capacidad de generación deba ser mayor de lo necesario para cubrir momentos pico de demanda o que la falta de vigilancia en tiempo real del sistema pueda producir apagones u otros fallos, lo que es inaceptable para una infraestructura tan crítica e implica unos costes muy altos para los operadores de estas redes. Además, este tipo de red no está preparada para cumplir con los nuevos requisitos y características que la sociedad actual espera de ellas.

Las redes eléctricas actuales se enfrentan a retos que las topologías tradicionales no pueden asumir. Las energías renovables generan una cantidad variable de energía que depende de factores que quedan fuera del control del operador eléctrico y han contribuido a aumentar la dispersión geográfica de las plantas generadoras de electricidad, reduciendo los costes e incrementando la capacidad de generación y la escalabilidad de la red, pero también su complejidad.

Además, los vehículos eléctricos van a suponer un gran incremento en la demanda de energía eléctrica. Sin embargo, también podrían ayudar en la gestión de la red mediante el control de sus ciclos de carga y el uso de sus baterías como un almacén local de energía, proveyendo la flexibilidad necesaria para asegurar la coincidencia entre la generación y el consumo de energía [1].

1.1.2 Redes Eléctricas Inteligentes

Las Redes Eléctricas Inteligentes (REI o, del inglés, *Smart Grids*) están concebidas para solucionar las principales deficiencias encontradas en las redes tradicionales y proporcionar nuevas o mejores soluciones a las necesidades actuales de los operadores y de la sociedad.

Este nuevo paradigma hace que las redes eléctricas dejen de ser unidireccionales, ya que la energía puede fluir en ambos sentidos entre los generadores y los consumidores, y deje de poder asumirse que la cantidad de energía demandada en un determinado momento será la que determine la cantidad de energía que deba producirse en ese momento, gracias a elementos que pueden posponer el consumo de esa energía (como sistemas de calefacción eléctrica por acumulación) o incluso revertirlo (como las baterías de los vehículos eléctricos) [2].

Además, introduce la posibilidad de que los consumidores pasen a ser pequeños productores que reduzcan su dependencia de la red eléctrica o incluso que vuelquen sus excedentes de producción a la misma, dando lugar a lo que se conoce como *prosumidores*.

Este tipo de redes requieren cada vez más que el consumidor final sea una parte más activa en la gestión de la red eléctrica e incentivan un uso más racional de la misma, por ejemplo, mediante políticas de precios dinámicos (como se da en España con la tarifa con discriminación horaria de dos y tres periodos o la tarifa por horas).

1.2 Motivación

Las Tecnologías de Información y Comunicaciones (TIC) son un punto fundamental de las *Smart Grids*, compuestas por una gran cantidad de dispositivos, como sensores y actuadores, que deben ser vigilados y controlados, además de por sistemas de información donde se ejecutan los procesos de optimización. Por esta parte, los avances en tecnologías como la minería de datos, *Big Data* y la Computación en la Nube (o *Cloud Computing*) permitirán gestionar, procesar y tomar decisiones basadas en grandes conjuntos de información a su debido tiempo.

Por la parte de las redes de comunicaciones, destacan las comunicaciones Máquina-Máquina (M2M o *Machine-to-Machine*), que engloban todas las comunicaciones entre dispositivos con conectividad de red que no requieran intervención humana. Entre estos sistemas que hacen uso de redes de comunicaciones M2M cabe destacar las Infraestructuras de Medición Avanzadas (AMI o *Advanced Metering Infrastructures*), que incluyen los contadores inteligentes (del inglés, *smart meters*) de los clientes finales que precisan comunicarse con los sistemas de gestión y facturación de la compañía eléctrica [3].

Hay diversos métodos para transmitir todos los datos necesarios en una *Smart Grid*, y generalmente se utilizarán varios de ellos según las necesidades específicas de cada caso. Entre las distintas tecnologías de transmisión de datos disponibles en el mercado, cabe destacar las comunicaciones mediante cable eléctrico (PLC o *Power Line Communications*), que permiten aprovechar la red de transmisión de energía existente para la transmisión bidireccional de datos.

Los cables de bajo voltaje son un medio un tanto hostil desde el punto de vista de la transmisión de datos (p.ej., debido al ruido, a la variación de la impedancia). Sin embargo, las comunicaciones por ese medio proporcionan muchas ventajas a los operadores de redes eléctricas (p.ej., no es necesario desplegar infraestructura). Por este motivo, se han especificado una serie de estándares PLC de Banda Estrecha (NB-PLC o *Narrowband Power Line Communications*) que mitigan en parte los inconvenientes del medio, lo que les ha llevado a ser la solución preferida por la mayoría de distribuidores eléctricos para la última milla de los despliegues de AMI.

Entre las distintas tecnologías NB-PLC disponibles en el mercado destaca PRIME, por tratarse, junto a *Meters and More* [4], de la tecnología NB-PLC más desplegada en España (Iberdrola, Fenosa) así como por su clara vocación internacional (también se está utilizando en otros países como Portugal, Reino Unido, Polonia, Brasil o Australia). El protocolo PRIME lo define la *PRIME Alliance* [5], cuyo objetivo es establecer un conjunto de estándares públicos y no propietarios que permiten la interoperabilidad completa entre equipamiento y sistemas de distintos proveedores, aunque los niveles físicos y de enlace también fueron aceptados como estándar por la ITU-T en 2012 [6].

Los componentes de la arquitectura PRIME no están sujetos a derechos de propiedad intelectual. Las especificaciones son lo suficientemente exhaustivas y detalladas para que cualquier nuevo actor sea capaz de proveer al mercado con soluciones interoperables. De esta forma se consiguen soluciones económicamente eficientes que benefician a las compañías eléctricas y a los consumidores.

Sin embargo, las comunicaciones NB-PLC en general aún presentan importantes retos tecnológicos, por lo que deben desarrollarse paralelamente herramientas de simulación que permitan planificar, evaluar y tomar decisiones relativas a este tipo de redes minimizando riesgos, tiempo y costes [7].

SimPRIME es un simulador de redes PRIME, basado en Matlab y OMNeT++, que persigue dichos objetivos. El núcleo del simulador fue desarrollado como parte de la tesis doctoral de Javier Matanza Domingo [8]. Si bien es cierto que el simulador ha despertado interés por parte de distribuidoras eléctricas como Unión Fenosa o Iberdrola, también es cierto que la complejidad de su uso y la curva de aprendizaje asociada han impedido que el interés inicial vaya a más y el simulador comience a ser utilizado en entornos de operación. Esto ilustra la importancia de disponer de una interfaz de uso amigable para maximizar el impacto del simulador en el mercado, lo que supone el principal motivo que ha llevado a acometer este Proyecto Fin de Carrera.

Así, cabe destacar que el presente Proyecto Fin de Carrera ha sido desarrollado dentro del ámbito del proyecto de investigación nacional OSIRIS (Optimización de la Supervisión Inteligente de la Red de Distribución) [9], financiado por el Ministerio de Economía y Competitividad y liderado por Unión Fenosa Distribución (tercera distribuidora eléctrica a nivel nacional).

1.3 Objetivos

El objetivo principal de este Proyecto Fin de Carrera es desarrollar una interfaz amigable para el simulador de redes PRIME SimPRIME, que permita reducir drásticamente su complejidad de uso, acercándolo así al mercado.

Para conseguir dicho objetivo global, se han establecido una serie de subobjetivos o hitos descritos a continuación:

- Analizar el estado del arte de las distintas tecnologías implicadas en la simulación de redes PRIME, comprendiendo así las necesidades de los usuarios de estos sistemas.
- Analizar el estado del arte de las distintas tecnologías que permitan el desarrollo e implementación de una aplicación que permita actuar de capa intermedia entre el usuario y el simulador proporcionando una interfaz amigable, para conocer las distintas alternativas y sus principales características.
- Diseñar la arquitectura del software, seleccionando las tecnologías y herramientas más adecuadas de entre las analizadas en el punto anterior.
- Desarrollar la aplicación siguiendo el diseño resultante del punto anterior, que permita a los usuarios del sistema interactuar con el simulador de una forma más amigable.
- Validar el funcionamiento de la aplicación resultante y el cumplimiento de los requisitos establecidos por parte de la misma.
- Documentar adecuadamente el desarrollo realizado.

1.4 Estructura de la memoria

El presente documento se organiza siguiendo la siguiente estructura:

- **Capítulo 1: Introducción**

Este capítulo proporciona una visión global del Proyecto Fin de Carrera ayudando a contextualizarlo. En él se explican los motivos que han llevado a acometer el proyecto así como los objetivos que se pretenden alcanzar. Por último se incluye una guía en la que se describen la estructura y contenido de la presente memoria.

- **Capítulo 2: Estado del arte**

Este capítulo revisa el estado del arte de las tecnologías relacionadas con el trabajo desarrollado, incluyendo las tecnologías necesarias tanto para el despliegue y simulación de redes NB-PLC (con especial énfasis en PRIME) como para el desarrollo de la aplicación que hará de interfaz entre el usuario y el simulador de redes PRIME SimPRIME. La selección de estas últimas se explicará razonadamente en el capítulo 3, en base al análisis de las mismas realizado en este capítulo. El funcionamiento del simulador SimPRIME también se presenta brevemente en este capítulo, por considerarse parte del estado del arte.

- **Capítulo 3**: **Diseño**

El objetivo de este capítulo es describir el proceso de diseño de la arquitectura *software* de la aplicación a desarrollar así como la selección de plataformas y tecnologías a utilizar. Para ello, se presentan en primer lugar las distintas herramientas y los distintos requisitos funcionales y no funcionales proporcionados y, a continuación, las principales decisiones técnicas y de diseño que se han tomado de cara al desarrollo de la aplicación, descrito en el capítulo siguiente.

- **Capítulo 4: Desarrollo y Puesta en Marcha**

El objetivo de este capítulo es detallar la implementación de cada uno de los módulos que conforman el sistema, siguiendo los principios de diseño definidos en el capítulo anterior. También incluye detalles sobre el procedimiento para la puesta en marcha del sistema.

- **Capítulo 5: Validación**

Este capítulo muestra las validaciones realizadas para comprobar el correcto funcionamiento del *software*, y las validaciones incorporadas en el formulario de solicitud de simulaciones para verificar que los datos introducidos por el usuario son correctos.

- **Capítulo 6: Conclusiones y Trabajos Futuros**

Este capítulo resume las principales conclusiones obtenidas tras la ejecución del presente Proyecto Fin de Carrera, además de una serie de posibles mejoras que se podrían realizar en el sistema desarrollado en un futuro.

Capítulo 2

Estado del arte

Este capítulo revisa el estado del arte de las tecnologías relacionadas con el trabajo desarrollado, incluyendo las tecnologías necesarias tanto para el despliegue y simulación de redes NB-PLC (con especial énfasis en PRIME) como para el desarrollo de la aplicación que hará de interfaz entre el usuario y el simulador de redes PRIME SimPRIME. La selección de estas últimas se explicará razonadamente en el capítulo 3, en base al análisis de las mismas realizado en este capítulo. El funcionamiento del simulador SimPRIME también se presenta brevemente en este capítulo, por considerarse parte del estado del arte.

2.1 Tecnologías PLC de Banda Estrecha

2.1.1 Visión global

Múltiples estudios [10] [11] [12] [13] y proyectos piloto desarrollados por grandes distribuidoras eléctricas [14] apuntan a que NB-PLC es la tecnología más apropiada especialmente para la última milla de despliegues AMI, como interfaz entre los concentradores de datos y los contadores inteligentes.

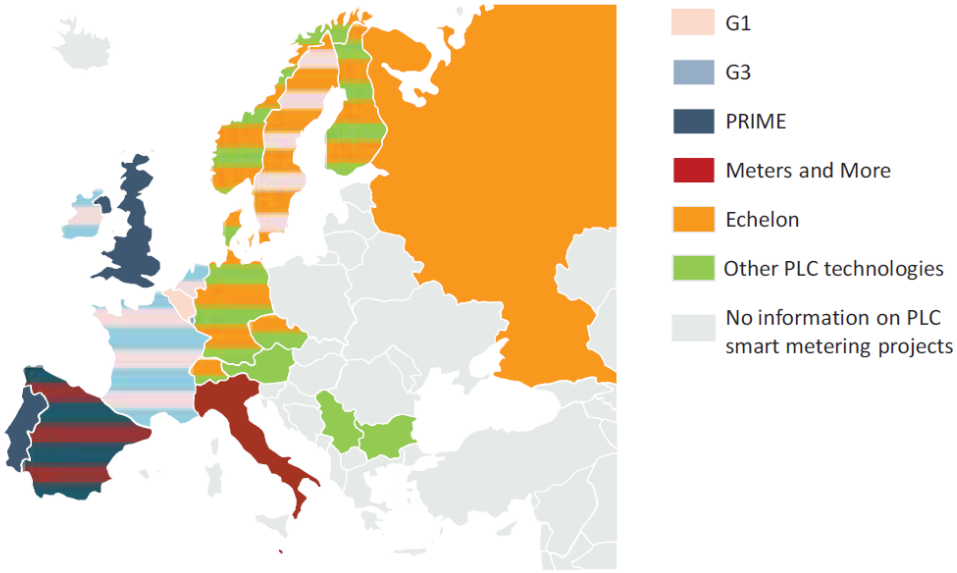


Figura 2: Mapa del despliegue de tecnologías NB-PLC en Europa [15]

Existen de hecho diferentes estándares NB-PLC en el mercado, con distintas características técnicas y niveles de penetración, como ilustra la Figura 2:

- Meters and More es una tecnología cuya especificación lidera el grupo Enel, aunque sus capas inferiores también están siendo estandarizadas por IEC/CENELEC (CLC TS 50568-4). Presenta sus mayores niveles de penetración en mercados donde opera el grupo ENEL, tales como Italia (el 100% de los contadores italianos usan esta tecnología actualmente) o España (aproximadamente la mitad del parque de contadores inteligentes españoles utilizarán esta tecnología en 2018). Funciona en la banda CENELEC-A. Utiliza una única portadora, destacando por su robustez, aunque alcanza tasas de transmisión de datos bajas (9,6 kbps).
- Open Smart Grid Protocol (OSGP) es una tecnología promovida por el fabricante norteamericano Echelon, aunque sus capas inferiores están también estandarizadas por el IEC (IEC 14908.1). Presenta sus mayores tasas de despliegue en los países nórdicos y Rusia. Funciona en la banda CENELEC-A. Se trata de una tecnología monoportadora que alcanza tasas de transmisión de datos de hasta 3,6 kbps.
- CX1 es una tecnología promovida por Siemens y cuyas capas inferiores están siendo estandarizadas también por IEC/CENELEC (CLC TS 50590). Actualmente se

está desplegando en Austria. Funciona en la banda CENELEC-A. Utiliza una modulación multiportadora adaptativa que le permite alcanzar tasas de hasta 64 kbps.

- G3 es una tecnología cuya especificación lidera la distribuidora EDF y el fabricante de chipsets Maxim, aunque sus capas PHY y de enlace han sido publicadas como estándar por la ITU-T recientemente (ITU-T G9903). Presenta sus mayores niveles de penetración en mercados donde opera EDF, como Francia. Inicialmente definida para que funcionase en la banda CENELEC-A, ha sido extendida recientemente para poder ser utilizada en el mercado americano (FCC) y asiático (ARIB). Se trata de una tecnología multiportadora que alcanza tasas de hasta 34 kbps y que ha destacado desde sus inicios por ser especialmente robusta.
- Como ya se ha comentado, PoweRline Intelligent Metering Evolution (PRIME) es una tecnología promovida por la PRIME Alliance, liderada por distribuidoras eléctricas españolas como Iberdrola y Unión Fenosa Distribución y por fabricantes de chipsets como Texas Instruments. Sus capas PHY y de enlace también han sido publicadas como estándar por la ITU-T (ITU-T G.9904) [6]). Al igual que G3, fue inicialmente definida para que funcionase en la banda CENELEC-A y ha sido extendida recientemente para poder ser utilizada en el mercado americano (FCC) y asiático (ARIB). Es una tecnología multiportadora que puede llegar a alcanzar tasas de hasta 128,6 kbps (en su versión para la banda CENELEC-A).
- G.hnem [16] se trata de un esfuerzo de la ITU-T por homogeneizar G3 y PRIME, aunque su implementación es computacionalmente más compleja que las anteriores. Su utilización actualmente es baja.
- IEEE 1901.2 es la especificación NB-PLC del IEEE. Su utilización actualmente es muy baja.

PRIME destaca como una tecnología NB-PLC prometedora dentro de las disponibles. Está siendo implantada en países como España, Portugal, Reino Unido, Polonia, Brasil o Australia, existiendo de manera oficial más de 5 M de contadores con tecnología PRIME instalados en campo. Además, en 2018 habrá alrededor de 15 M de contadores con tecnología PRIME desplegados únicamente en España (debido a la legislación española IET/290/2012) y la nueva versión del estándar (v1.4) amplía su dominio al resto del mundo, incluyendo bandas de frecuencia para los mercados americano y asiático.

Como ya se ha comentado, PRIME es el estándar que implementa la herramienta de simulación SimPRIME, en la que se basa este PFC. Por ello, el siguiente apartado profundiza en ella, resumiendo brevemente sus principales detalles técnicos.

2.1.2 PRIME

PRIME es una tecnología NB-PLC de segunda generación cuyo desarrollo fue inicialmente liderado por la PRIME Alliance, estando considerada la versión 1.3.6 de la especificación de las capas físicas, MAC y de Convergencia, como un estándar por la ITU-T desde 2012. La versión 1.4 expande el espectro de frecuencia utilizado para poder operar en los mercados americano y asiático e incluye algunas funciones para incrementar la robustez de la comunicación a nivel de la capa física y MAC.

Desde una perspectiva de la capa física, PRIME opera en la banda de 41-89 kHz (v1.3.6) o de 3-500 kHz (v1.4), usando la modulación OFDM para hacer un uso más eficiente del espectro. Las compañías pueden usar las modulaciones DPSK, DQPSK o D8PSK. Además, permite la utilización de FEC para recuperarse ante posibles errores introducidos por el canal. A nivel de capa física, la velocidad de transmisión en redes PRIME puede ir desde los 5.4 kbps a 1028.8 kbps, dependiendo de la combinación de modulación digital y FEC utilizadas (también conocido como modo de comunicación).

A nivel MAC, se definen dos tipos de nodos: Base y de Servicio. Sólo se permite un Nodo Base por cada red PRIME, ya que actúa de coordinador. En terminología AMI, el Nodo Base es conocido como concentrador de datos, o simplemente concentrador. Aunque los Nodos de Servicio suelen ser contadores inteligentes, dependiendo de las necesidades de la red, también pueden funcionar como *switches*. El objetivo principal de los *switches* es incrementar el rango de la señal en el cable mediante un mecanismo de repetición, mitigando así los efectos del ruido y la atenuación. Así, la topología física de este tipo de redes es en *bus*, mientras que la topología lógica tiene típicamente forma de árbol. Un aspecto a considerar es que la posición de estos *switches* puede influir en el comportamiento de la red, como se discute en [17]. A pesar de ello, el estándar no da indicaciones sobre qué criterios seguir para introducir *switches*.

En relación a los mecanismos MAC, aunque el estándar define un periodo de con contienda (SCP) y un periodo libre de contienda (CFP), actualmente sólo el SCP está implementado por los fabricantes. A pesar de ello, se está investigando sobre los beneficios potenciales de usar CFP para nuevos servicios de Redes Eléctricas Inteligentes [18]. En el SCP se utiliza CSMA/CA como técnica de acceso al medio.

La capa de Control de Enlace Lógico (LLC), perteneciente a la capa de Convergencia, es responsable de manejar las conexiones lógicas. Identifica cada transacción con un número de identificación y realiza los procesos de control de flujo. El control de flujo está implementado en PRIME mediante el establecimiento de una Unidad Máxima de Transmisión (MTU) y usando un procedimiento de ventana deslizante. La MTU define la longitud en *bytes* del mayor paquete de datos que puede encapsular el nivel MAC. En caso de que la aplicación intente enviar un mensaje mayor, la capa LLC fragmenta el mensaje en varios paquetes, ninguno de los cuales será mayor que la MTU. Cada uno de estos paquetes (también conocidos como fragmentos) está etiquetado con un identificador para que la parte receptora pueda reensamblarlos.

Con respecto al procedimiento de ventana deslizante, PRIME establece diferentes valores permitidos para el Tamaño de Ventana (WS). El valor del WS puede jugar un rol importante en el rendimiento de la red en términos de latencia. Además, los dispositivos PRIME pueden implementar o no capacidades de Repetición de Solicitudes (ARQ) para asegurar la correcta recepción de todos los mensajes. Dado que los parámetros ARQ son negociados en la fase de conexión, este proceso funciona de punto a punto entre el transmisor final y el receptor, siendo por tanto los *switches* transparentes al mismo.

En la capa de aplicación, DLMS/COSEM es el estándar utilizado sobre todas las tecnologías NB-PLC disponibles en el mercado. En concreto, COSEM (IEC 62056-61/62) es un perfil del protocolo de aplicación DLMS (IEC 62056-53) especialmente diseñado para la medición de energía [19] [20]. Como tal, DLMS/COSEM incluye modelos de datos para representar

parámetros comunes relacionados con la energía junto con un protocolo de comunicación diseñado para transmitir este tipo de información.

2.2 Simuladores de redes de comunicaciones

Esta sección analiza algunas de las soluciones disponibles en el mercado para desarrollar herramientas de simulación de redes de comunicaciones. Existen multitud de alternativas. Dentro del ámbito de los simuladores de redes de comunicaciones propiamente dichos, esta sección se centra en los siguientes, por tratarse de los tres con más aceptación en el mercado:

- Riverbed Modeler (anteriormente, OPNET) [21].
- NS3 [22].
- OMNeT++ [23].

Sin embargo, también pueden desarrollarse herramientas de simulación en base a software de cálculo (p.ej., Microsoft Excel o Matlab) o en lenguajes de scripting (p. ej., Python), dependiendo de las características de las tecnologías de simulaciones a simular.

2.2.1 Riverbed Modeler

Riverbed Modeler (antes conocido como OPNET Modeler), producto actualmente perteneciente a Riverbed, es un potente simulador de redes de comunicaciones basado en eventos discretos que incluye modelos muy precisos de tecnologías de comunicaciones como, por ejemplo, las celulares (2G, 3G). Sin embargo, no existe modelo aun para redes PLC, aunque podría desarrollarse partiendo de los módulos ya implementados (p. ej., Ethernet). El lenguaje de programación que se utiliza en este simulador es C++.

La versión comercial de Riverbed Modeler requiere del pago de una licencia que incluye la utilización del simulador así como soporte técnico y formación. En cualquier caso, las universidades y centros de investigación actualmente tienen la posibilidad de obtener una licencia de pruebas de 6 meses, que puede renovarse siempre y cuando se hagan públicos los resultados de investigación obtenidos.

2.2.2 NS3

NS3 es la última versión del popular simulador de redes de comunicaciones NS2. NS2 y NS3 son bastante comunes en entornos académicos, llegando a ser casi de uso obligatorio para que los resultados obtenidos al simular determinados protocolos tengan validez dentro de la comunidad investigadora (p.ej., para protocolos del IETF – *Internet Engineering Task Force*).

Existe una implementación de PLC para NS3 [24] que podría utilizarse como referencia y adaptarse a PRIME.

NS es software libre que se ofrece bajo la versión 2 de la GNU (*General Public License*).

2.2.3 OMNeT++

OMNeT++ es un simulador de redes de comunicaciones modular basado en eventos discretos. En OMNeT++ existen dos tipos de módulos:

- Simples: que representan elementos atómicos;
- Compuestos: compuestos – valga la redundancia – por una combinación de módulos simples.

En OMNeT++ se manejan tres tipos de ficheros:

- .cpp: definen funcionalidad, utilizando C++ como lenguaje de programación;
- .ned: definen la topología de red;
- .ini: configuran las simulaciones a realizar (parámetros de entrada, parámetros de salida, duración, etc.).

No se ha encontrado implementación oficial de PRIME para OMNeT++. Sin embargo, como ya se ha comentado, en [8] se presenta la versión inicial de SimPRIME, una implementación de PRIME para OMNeT++ desarrollada a partir de la implementación de Ethernet disponible en el INET Framework.

OMNeT++ se distribuye bajo *Academic Public License*, que no permite su uso comercial. La versión comercial de OMNeT++ es OMNEST.

2.3 Simulador SimPRIME

Como ya se ha comentado, el núcleo del simulador de redes PRIME SimPRIME fue desarrollado como parte de la tesis doctoral de Javier Matanza [8]. Este simulador combina MATLAB y OMNeT++ para modelar los efectos de la capa física (PHY) y de las capas superiores (MAC, LLC y aplicación) respectivamente, como ilustra la parte derecha de la Figura 3Figura 3: Esquema de la arquitectura de SimPRIME.. Como también muestra la Figura 3, y en consonancia con lo comentado en la sección 2.1.2, la implementación de las capas PHY, MAC y LLC se ajusta a la especificación PRIME, mientras que la capa de aplicación se modela en base a DLMS/COSEM.

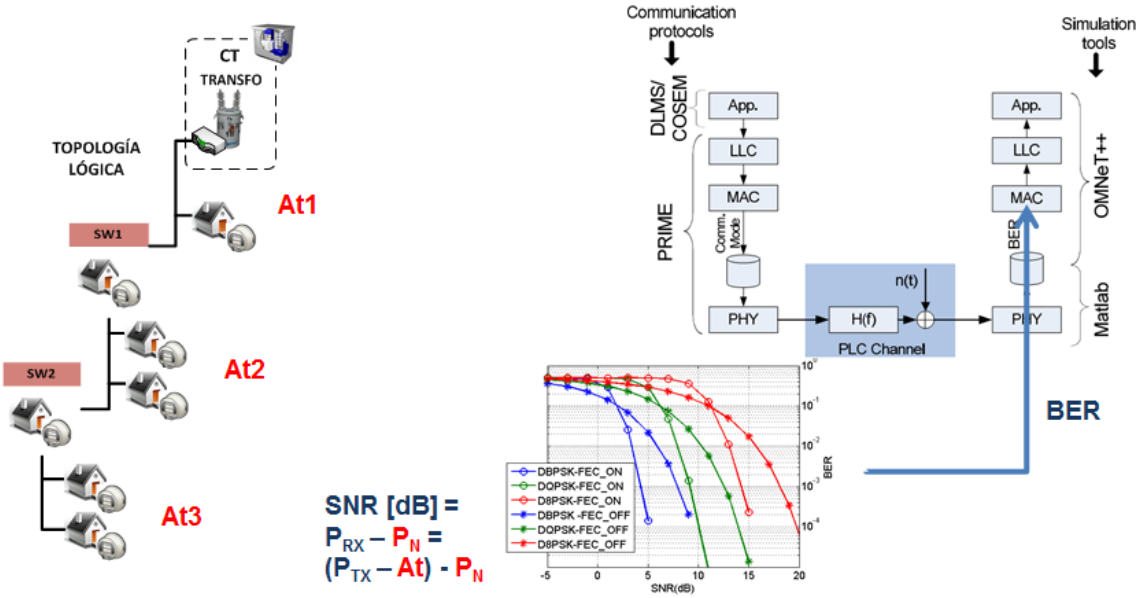


Figura 3: Esquema de la arquitectura de SimPRIME.

La Figura 3 también ilustra la interacción entre MATLAB y OMNeT++. Como puede observarse, conocidos la potencia de transmisión y el nivel de ruido base, cuando un nodo envía un mensaje a otro, se calcula la potencia recibida en base a una matriz de atenuaciones que contiene la atenuación entre cada par de nodos. Con la potencia recibida y la potencia de ruido base se obtiene la relación señal a ruido (SNR), tal y como muestra la ecuación incluida en la Figura 3. Conocidos la SNR y la constelación utilizada, se obtiene la tasa de error de bit (BER) mediante las curvas SNR frente a BER que también muestra la Figura 3. Dichas curvas se calculan a priori en base al estándar usando para ello MATLAB. OMNeT++ utiliza el valor de BER resultante para decidir si el mensaje recibido contiene errores (y por tanto debe ser descartado) o por el contrario fue recibido correctamente (y puede ser procesado por las capas superiores).

La versión original del simulador ha sido extendida permitiendo, por ejemplo, simular no sólo topologías aleatorias sino también topologías fijadas a priori, como la que aparece en la parte izquierda de la Figura 3. Esta nueva característica permite simular topologías de red reales obtenidas a partir del procesamiento del fichero de topología estándar que proporcionan los concentradores (S11.xml). En esta misma línea, el simulador original también ha sido extendido para que las atenuaciones entre cada par de nodos no sólo puedan obtenerse en base a teoría de líneas de transmisión [25], sino que también puedan obtenerse a partir de determinadas BER fijadas para cada par de nodos de la red dependiendo del nivel

lógico en el que se encuentren. En la Figura 3, por ejemplo, se propone asumir una BER para todos los nodos que se encuentran en un mismo nivel de la topología lógica (dando lugar a At1), otra entre los nodos que están a un nivel de distancia (dando lugar a At2) y otra entre los nodos que están a más de un nivel de distancia (dando lugar a At3).

2.4 Lenguajes y *frameworks* de desarrollo

Los lenguajes de programación son aquellos que nos permiten comunicarnos a bajo nivel con las computadoras, expresando con ellos el comportamiento que esperamos de las mismas. Es decir, nos permiten programar a las máquinas para que ejecuten procesos que implementan cierta funcionalidad.

Los *frameworks* nos permiten escribir en esos lenguajes de una forma más eficiente; abstrayendo el lenguaje para, por ejemplo, implementar de una forma más sencilla operaciones muy habituales o patrones de diseño.

No debe confundirse un *framework* con una librería o biblioteca de *software*. Estas últimas son implementaciones de *software* ya realizadas de forma reutilizable, mientras que los *frameworks* suelen controlar el flujo de ejecución del programa; y las implementaciones que pone al servicio del programador, además de proveer un comportamiento por defecto, están generalmente diseñadas para ser extendidas y reescritas según las necesidades del proyecto.

En esta sección se analizan algunas de las opciones disponibles para implementar sistemas de *software*.

2.4.1 Java (plataforma)

Además de un lenguaje de programación, Java es una plataforma que provee de una Interfaz de Programación de Aplicaciones (API o *Application Programming Interface*) y de una Máquina Virtual de Java (JVM o *Java Virtual Machine*), además de ciertas herramientas de desarrollo, como un compilador para el lenguaje de programación Java [26]. La plataforma Java se utiliza ampliamente para el desarrollo de aplicaciones, tanto de escritorio como web.

Actualmente hay tres implementaciones de la plataforma de Java:

- **Java SE:** Es la plataforma más conocida de Java, y de hecho, cuando la mayoría de la gente piensa en el lenguaje de programación Java, sólo piensa en esta plataforma.
  Provee el *core API*, la JVM estándar y los componentes de la antigua plataforma JavaFX, que fue integrada a partir de la versión Java SE 7 Update 6 [27].
- **Java EE:** Esta plataforma está construida sobre Java SE y proporciona un API extendido y un entorno de ejecución para construir aplicaciones empresariales a gran escala.
  Este tipo de aplicaciones se ejecutan en servidores de aplicaciones, y la comunicación con otros usuarios o con el sistema se realiza mediante servicios web.
- **Java ME:** Provee un API reducido con respecto al de Java SE y la JVM está optimizada para ser ejecutadas en pequeños dispositivos, como teléfonos móviles.

2.4.1.1 Lenguajes de programación sobre la Plataforma Java

La JVM no es más que la abstracción de una computadora. Este enfoque permite que el *software* compatible con una plataforma Java, pueda ejecutarse en diferentes arquitecturas y sistemas operativos, siempre que exista una JVM implementada para ese sistema [28].

Aunque el lenguaje de programación Java fue el lenguaje con el que nació la plataforma Java, y ambos resultan indistinguibles para mucha gente, existen multitud de lenguajes compatibles con la plataforma Java, al poder compilar sus códigos a *bytecode* (que es el código máquina que entiende la JVM).

JAVA (LENGUAJE DE PROGRAMACIÓN)

Java es un lenguaje de programación compilado, de propósito general, concurrente, basado en clases y orientado a objetos. Está diseñado para ser simple y fácil de aprender. Es un lenguaje estático y fuertemente tipado [29].

SCALA

Según la especificación del lenguaje en [30], Scala es un lenguaje de programación similar a Java, lenguaje con el que puede interoperar. Trata de unificar la orientación a objetos (todo valor es un objeto, definido por una clase) y la programación funcional (toda función es un valor).

GROOVY

Según la especificación del lenguaje en [31], Groovy es un lenguaje ágil y dinámico construido sobre las fortalezas de Java, pero con poderosas características adicionales inspiradas en lenguajes como Python, Ruby y Smalltalk. Está diseñado de forma que su aprendizaje sea sencillo para programadores de Java, integrándose de forma transparente con las clases y librerías de Java ya existentes.

2.4.1.2 Frameworks Java

SPRING FRAMEWORK

Spring es un *framework* modular para desarrollar aplicaciones principalmente en el lenguaje de programación Java, aunque también soporta Groovy, JRuby y BeanShell [32]. El uso de módulos permite usar sólo las partes que sean necesarias para el *software* desarrollado, y además permite abarcar una gran cantidad de soluciones a necesidades muy diversas. Es el *framework* más usado para el desarrollo de aplicaciones Java [33].

Entre los módulos más importantes se encuentran spring-mvc, que facilita el desarrollo de un programa con el patrón modelo-vista-controlador [34], y sprint-batch, que facilita la implementación de tareas en segundo plano [35].

PLAY 2

Play es un *framework* orientado a la implementación de aplicaciones web con soporte nativo para el patrón MVC y los lenguajes de programación Java y Scala. Provee de una alternativa ágil al modo tradicional de desarrollo de aplicaciones empresariales en Java, haciendo hincapié en la productividad del programador [36].

2.4.2 Python

Python es un lenguaje de programación interpretado de alto nivel con semántica dinámica y dinámicamente tipado. Tiene una sintaxis sencilla y fácil de aprender, que enfatiza la legibilidad del código y por tanto reduce el coste de mantenimiento del *software* [37]. Python

representa una de las opciones más utilizadas actualmente para desarrollar soluciones de todo tipo, tanto de *frontend* como de *backend*.

Los programas Python, pueden parecer lentos comparados con aquellos que han sido programados en Java [38]. Sin embargo, un código en Python suele ser entre 3 y 5 veces más corto que su equivalente en Java. Esto se debe a los tipos dinámicos de alto nivel y al tipado dinámico de Python [39].

Actualmente se mantienen dos ramas de Python con ciertas incompatibilidades entre sí: Python 2.7 y Python 3. Esto es debido a que la rama 3.X, lanzada en 2008, no incluye retrocompatibilidad[1] para algunos elementos de versiones anteriores. Se recomienda el uso de la rama 3.X de Python excepto si eso supone un problema de compatibilidades [40].

Debido a la incompatibilidad entre versiones, se siguió manteniendo la versión 2, hasta que en 2010 se lanzó la versión 2.7. En ese momento los desarrolladores indicaron que no habría una versión 2.8 de Python y a partir de entonces sólo saldrían actualizaciones de mantenimiento. Inicialmente, esa fase de soporte extendido en la que se mantendrá el sistema iba a durar hasta 2015. Sin embargo, debido a la complejidad de la migración para varios proyectos importantes, se decidió extender ese soporte hasta 2020 [41].

El intérprete oficial de Python, CPython, está escrito en C, pero existen otras implementaciones alternativas [42]:

- Jython es la implementación de Python en la plataforma Java SE. Permite la integración total con aplicaciones Java existentes y el acceso a las clases y a la API de la plataforma Java, ejecutando el código Python compilado en la JVM.
- IronPython es la implementación de Python en la plataforma .NET, permitiendo el acceso a las bibliotecas de esta plataforma y exponiendo el código de Python a otros lenguajes de .NET.
- PyPy es un intérprete de Python escrito en Python. Su objetivo principal es mejorar el rendimiento de la ejecución del código manteniendo la máxima compatibilidad con CPython.

Las implementaciones alternativas sólo ofrecen soporte estable para Python 2.7. Sin embargo, muchas de ellas ya han empezado con el desarrollo de sus implementaciones para Python 3.

2.4.2.1 Django

Django es un *framework* para Python mantenido por la *Django Software Foundation* [43], orientado al desarrollo de aplicaciones web utilizando el patrón MVC. Está diseñado para facilitar una implementación y un despliegue rápidos de aplicaciones, facilitando la interactuación del sistema implementado con la base de datos. Es fácilmente escalable y ofrece por defecto características de seguridad como protección contra inyecciones SQL o un sistema de gestión de cuentas de usuario [44].

[1] Puede verse una explicación de los motivos que llevaron a crear una nueva versión rompiendo la compatibilidad con versiones anteriores del lenguaje, y una colección de dichas incompatibilidades en [81].

Este *framework* induce a un diseño modular de aplicaciones para fomentar la reutilización del código.

Además, Django incluye un sistema de gestión (django-admin o manage.py). Una herramienta de línea de comandos para realizar tareas administrativas, que permite, por ejemplo:

- Crear aplicaciones y módulos.
- Crear superusuarios.
- Gestionar la internacionalización del sistema (traducciones por cadenas de texto).
- Lanzar un servidor web de desarrollo (incluido en Django).
- Gestionar la base de datos, permitiendo, por ejemplo, importar el modelo de datos provisto por la aplicación y los módulos a la base de datos que utilice el sistema.

Incluye también otros componentes útiles como:

- Serialización de formularios (genera formularios a partir de modelos de datos).
- Validación de formularios según los tipos de datos admitidos y la obligatoriedad de los campos.
- Sistema de plantillas con herencia entre las mismas.

Aunque Django sigue un patrón de diseño MVC, según los desarrolladores del framework, esa terminología es debatible y prefieren referirse a un patrón de diseño Modelo-Plantilla-Vista (MTV o *Model-Template-View*), correspondiéndose los componentes vista y controlador del patrón MVC a los componentes plantilla y vista respectivamente (el componente modelo no varía de nombre) [45]. Para facilitar la comprensión de los lectores no familiarizados con Django, en esta memoria se hace referencia a la terminología estándar de diseño MVC.

2.4.2.2 Celery

Celery es un sistema de colas de tareas distribuidas. Está implementado en Python y, aunque su protocolo puede implementarse con otros lenguajes de programación, ofrece un alto grado de compatibilidad por defecto con aplicaciones escritas en este lenguaje además de con aplicaciones implementadas con Django [46].

Las colas de tareas son un mecanismo para distribuir el trabajo entre distintos hilos o máquinas. Celery se comunica mediante mensajes, normalmente mediante el uso de un servidor de colas, para mediar entre clientes (generadores de tareas) y ejecutores [47].

2.4.3 Otros lenguajes y *frameworks*

Además de los vistos anteriormente, hay innumerables lenguajes de programación disponibles para todo tipo de propósitos. A continuación se puede ver una pequeña muestra de algunos de los más utilizados (excluyendo los ya analizados), sobre todo para el desarrollo de sistemas basados en Web:

2.4.3.1 Ruby

Ruby es un lenguaje de programación dinámico y enfocado en la simplicidad y en la productividad. Su sintaxis “se siente natural al leerla y fácil al escribirla” [48].

RUBY ON RAILS

Rails es el *framework* más usado con Ruby. Está diseñado para facilitar la programación web, haciendo suposiciones sobre lo que cada desarrollador web necesita para empezar, permitiendo escribir menos código [49].

Rails se basa en dos principios básicos:

- El paradigma de programación de Convención sobre Configuración, en el que mientras no se diga lo contrario, el sistema realiza una serie de presuposiciones.
- El principio No te repitas, según el cual toda pieza de conocimiento debe tener, dentro de un sistema, una representación única, no ambigua y acreditada. De esta forma se construye un código más mantenible y menos propenso a errores.

2.4.3.2 Go

Go es un lenguaje de programación desarrollado principalmente por un equipo de Google [50]. Es expresivo, conciso, limpio y eficiente; estáticamente tipado, compilado, orientado a la concurrencia y a la modularidad [51].

2.4.3.3 PHP

PHP es un clásico entre los lenguajes de programación. Es un lenguaje de *scripting* ampliamente utilizado para desarrollo web. Su principal característica es que puede embeberse entre código HTML. Es un lenguaje muy sencillo para principiantes [52].

2.5 Otras tecnologías y herramientas

2.5.1 Bases de datos

Una base de datos es una colección organizada de datos. Su principal uso es asegurar la persistencia de la información almacenada y facilitar su actualización y recuperación, aunque hay usos alternativos como el envío de mensajes o el cacheado de información.

2.5.1.1 MySQL

MySQL es el sistema gestor de bases de datos (SGBD) relacional de código abierto más popular y es desarrollado, distribuido y soportado por Oracle [53].

2.5.1.2 PostgreSQL

PostgreSQL es un SGBD relacional de código abierto. Implementa características sofisticadas como control de transaccionalidad y cumple completamente con el estándar ANSI-SQL:2008 [54].

2.5.1.3 Oracle Database

Como MySQL, es un SGBD relacional desarrollado, distribuido y soportado por Oracle. Sin embargo no es un sistema de código abierto. Soporta una gran variedad de situaciones para acomodarse a las necesidades de los clientes de Oracle. Además de SQL, también tiene un lenguaje propio llamado PL/SQL que, además de permitir la ejecución de consultas muy complejas, permite insertar procedimientos y funciones en el sistema [55].

2.5.1.4 Redis

Redis es un sistema de almacenamiento de datos estructurados en memoria. Se trata, por tanto, de un sistema de almacenamiento de datos volátil. La información almacenada se puede persistir haciendo volcados cada cierto tiempo de los datos en la base de datos o manteniendo un registro con las inserciones que se han realizado en el sistema. Sus principales usos, además del almacenamiento de datos, son el cacheado de información y el intercambio de mensajes [56].

2.5.2 RabbitMQ

Las colas de mensajes (del inglés *Message Queues*) son componentes que proporcionan un protocolo de comunicación asíncrona, es decir, el emisor y el receptor del mensaje no tienen que conectarse simultáneamente a la cola para que la transmisión de los mensajes sea posible.

Estos mensajes son tradicionalmente utilizados en ingeniería de *software* para intercambiar información y lanzar acciones entre distintos procesos. Este es un procedimiento muy utilizado en aplicaciones web que requieren de la ejecución de tareas pesadas en segundo plano. De esta forma, la interfaz web puede comunicarse con un motor o *backend* que ejecute dichos procesos pesados mientras continua interactuando con el usuario.

RabbitMQ es uno de los servidores de colas de mensajes más usados. Implementa el Protocolo Avanzado de Encolado de Mensajes (AMQP o *Advanced Message Queuing Protocol*),

permitiendo el encaminamiento avanzado de mensajes y asegurando la entrega de los mismos (la cola de mensajes es persistente, y se mantiene aunque se reinicie el servicio). [57].

También es posible introducir una instancia de RabbitMQ en un *cluster* de nodos RabbitMQ. Esto puede hacerse bien para asegurar la escalabilidad del sistema, o bien para sistemas de alta disponibilidad con tolerancia a caídas y errores.

2.5.3 Highcharts

Highcharts es una biblioteca escrita en JavaScript pensada para generar gráficas y diagramas de una forma sencilla [58].

Soporta gran cantidad de tipos de gráficas, incluyendo:

- Diagramas de puntos y líneas.
- Líneas de tiempo.
- Diagramas de dispersión.
- Diagrama de áreas.
- Gráficos circulares o de tarta.
- Gráficos circulares de donut.
- Gráficos de radar.
- Gráficos de embudo.
- Diagramas de barras (incluyendo diagramas de cajas y bigotes).

Para que esta biblioteca dibuje la gráfica correspondiente, debe llamarse a la función de JavaScript provista con los datos de configuración del gráfico y una matriz con los datos que deben representarse.

Capítulo 3

Diseño

El objetivo de este capítulo es describir el proceso de diseño de la arquitectura *software* de la aplicación a desarrollar así como la selección de plataformas y tecnologías a utilizar. Para ello, se presentan en primer lugar las distintas herramientas y los distintos requisitos funcionales y no funcionales proporcionados y, a continuación, las principales decisiones técnicas y de diseño que se han tomado de cara al desarrollo de la aplicación, descrito en el capítulo siguiente.

3.1 Modelado de SimPRIME a alto nivel

El simulador de redes PRIME SimPRIME es un *software* que se proporcionó para construir la interfaz por encima. A efectos del presente proyecto, se trata de una caja negra, es decir, lo que importa es qué es lo que hace, pero no cómo lo hace. Esto se tuvo que tener muy en cuenta a la hora de tomar los requisitos y diseñar el sistema.

Para su integración con el resto del sistema, se proporcionó también la interfaz del simulador, es decir, una descripción de sus entradas y salidas. La Figura 4 ilustra dicha interfaz, mostrando en verde algunos de los parámetros de entrada y salida que el simulador considera actualmente, y en naranja los que podría considerar en el futuro.

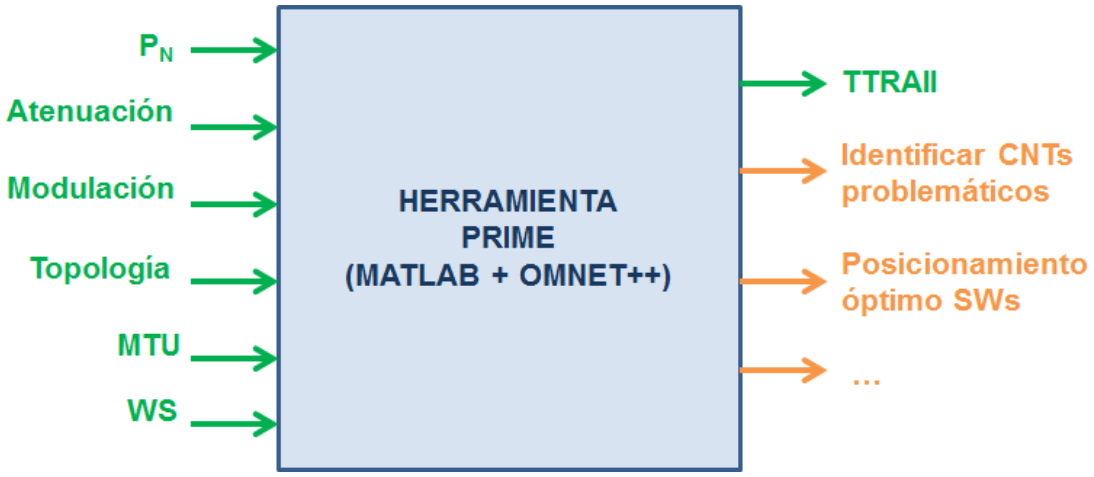


Figura 4: Resumen gráfico de entradas y salidas del simulador.

A continuación, se analizan más en detalle los parámetros y ficheros de entrada del simulador, así como los parámetros de salida.

3.1.1 Entradas del simulador

Al ser una herramienta basada en OMNeT++, se requiere de un fichero de configuración (.ini), del cual se proporcionó un ejemplo al inicio del proyecto (ver ANEXO I). En este fichero se establece cómo debe comportarse el simulador y se incluyen los siguientes parámetros de entrada:

- Identificador (para relacionar los resultados obtenidos con la simulación solicitada en caso de concurrencia de simuladores).
- Tiempo de simulación (en segundos).
- Modo de comunicación inicial.
- Ruta del fichero de atenuaciones.
- Ruta del fichero de topología.
- Número de contadores de la red PRIME.

- Tamaño de los *request* a nivel de aplicación (*AppReqMsgSize*)
- Tamaño de los *response* a nivel de aplicación (*AppResMsgSize*)
- *Maximum Transmission Unit* (MTU) del nivel MAC de PRIME[2]
- *Windows Size* (WS) del nivel LLC de PRIME
- Estrategia de sondeo de contadores
- Tiempo de vida (en segundos)

Como se puede comprobar en los parámetros de entrada mostrados en la Figura 4, además del fichero de configuración se requieren otros dos ficheros:

- Uno describiendo la topología de la red.
- El otro indicando las atenuaciones entre los nodos de la red.

El fichero de topología muestra en texto plano la topología de la red. Cada nodo está representado en una línea distinta y los datos de cada nodo están separados por plecas (|) tal y como se puede observar en la Figura 5.

```
1  0|1|15363|255|0|
2  0|2|15364|255|0|
3  0|3|15366|255|0|
4  0|4|15367|255|0|
```

Figura 5: Ejemplo de fichero de topología.

El fichero de atenuaciones representa la matriz de atenuaciones entre cada par de nodos. Se puede ver un ejemplo de este fichero en la Figura 6.

```
1  45.75|45.75|45.75|...
2  45.75|45.75|45.75|...
3  45.75|45.75|45.75|...
4  45.75|45.75|45.75|...
```

Figura 6: Ejemplo de fichero de matriz de atenuaciones.

3.1.2 Salidas del simulador

El simulador genera, por cada simulación, y por cada repetición de la simulación, un fichero .vec y otro fichero .vci.

Tras el proceso del fichero .vec con la herramienta proporcionada por OMNeT++ "scavetool" se obtiene un archivo de valores separados por comas (CSV) con todos los tiempos de lectura (TTR o *Time to Read*, TTRAll cuando se refiere al tiempo que tarda el concentrador en leer los consumos de todos los contadores de una red PRIME después de haber realizado una solicitud a tal efecto).

[2] Este parámetro en la práctica no se refiere a la MTU del nivel MAC de PRIME, sino al tamaño en bytes del *payload* del nivel MAC de PRIME (la MTU sería el máximo permitido por el estándar).

Como se puede observar en la Figura 7, en la que se puede ver un ejemplo de fichero CSV de salida para TTRAll, una vez descartada la primera línea, que etiqueta las dos columnas presentes en el fichero separado por comas, podemos observar en la primera columna una marca de tiempo, mientras que en la segunda podemos observar un valor booleano, indicando el valor 0 “tiempo de inicio” y el valor 1 “tiempo de finalización”.

```
1  X,PLC_netDLMSCOSEM.PLC_BN.BNApp/DLMSCOSEMAppTTRAllMeters
2  1000,0
3  1913.859388614176,1
4  1913.859388614176,0
5  2828.892175216558,1
```

Figura 7: Ejemplo de fichero CSV de salida para TTRAll.

Cada tiempo de respuesta será el resultado de restar la marca de tiempo indicadora del tiempo de inicio a la posterior marca de tiempo indicadora de tiempo de finalización según la siguiente fórmula:

$$TTR = T_{finalización} - T_{inicio}$$

3.2 Requisitos funcionales

Los requisitos funcionales son aquellos que definen las entradas, las salidas y el comportamiento del sistema.

La Figura 8 ilustra el mapeo entre las entradas y salidas de la aplicación (azul), que se analizan en esta sección, y las entradas (verde) y salidas (rojo) del simulador, explicadas en la sección 3.1.

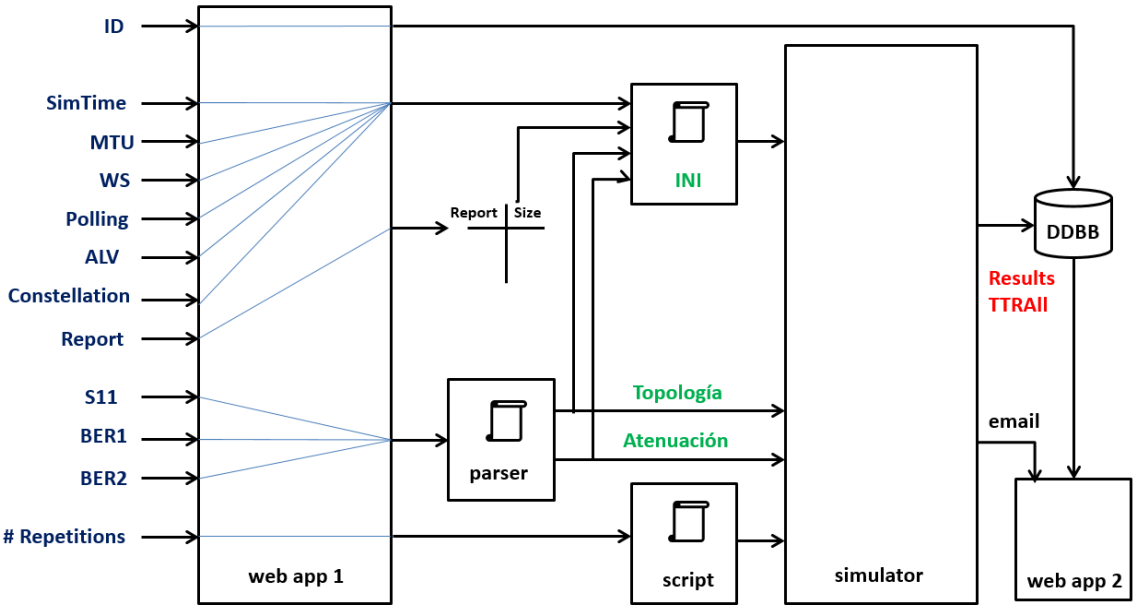


Figura 8: Entradas y salidas del sistema.

3.2.1 FUN01: El sistema deberá proporcionar una interfaz gráfica para el usuario en la que pueda introducir los detalles de la simulación solicitada.

Una vez que el usuario haya accedido al sistema a través de la pantalla de inicio de sesión, debe tener disponible un formulario en el que pueda introducir los detalles necesarios para poder ejecutar la simulación.

Cuando se solicite la simulación con este formulario, el sistema deberá advertir al usuario de los posibles problemas de validación del formulario (por ejemplo, si no se han rellenado todos los campos). En caso de pasar la validación, la simulación debe ejecutarse en segundo plano, mostrándole un mensaje al usuario indicando que se ha programado correctamente y permitiendo al usuario continuar usando el sistema.

Como se puede observar en la Figura 9, los campos mínimos a considerar son los siguientes:

- Identificador de la simulación
- Tiempo de simulación

- Número de ejecuciones
- Informe solicitado (valor por defecto: *S02*)
  Es un desplegable con las siguientes opciones:
  - Informe S02: curva horaria incremental
  - Informe S05: facturación diaria
- Estrategia de sondeo (valor por defecto: *Secuencial*)
  Es un desplegable con las siguientes opciones:
  - Secuencial
  - Simultáneo
- Unidad Máxima de Transferencia (MTU o *Maximum Transmission Unit*)
- Tamaño de la ventana (WS o *Windows Size)*
- Tiempo de vida (valor por defecto: *120 s*)
- Constelación (valor por defecto: *DBPSK FEC ON*)
  Es un desplegable con las siguientes opciones:
  - DBPSK FEC OFF
  - DQPSK FEC OFF
  - D8PSK FEC OFF
  - DBPSK FEC ON
  - DQPSK FEC ON
  - D8PSK FEC ON
  - Dynamic
- Informe S11: informe estándar en formato XML que muestra la topología de la red PRIME en un instante de tiempo determinado
- BER 1: probabilidad de error de bit entre todos los dispositivos que se encuentran en el mismo nivel lógico de la topología
- BER 2: probabilidad de error de bit entre dispositivos que se encuentran en niveles lógicos contiguos de la topología

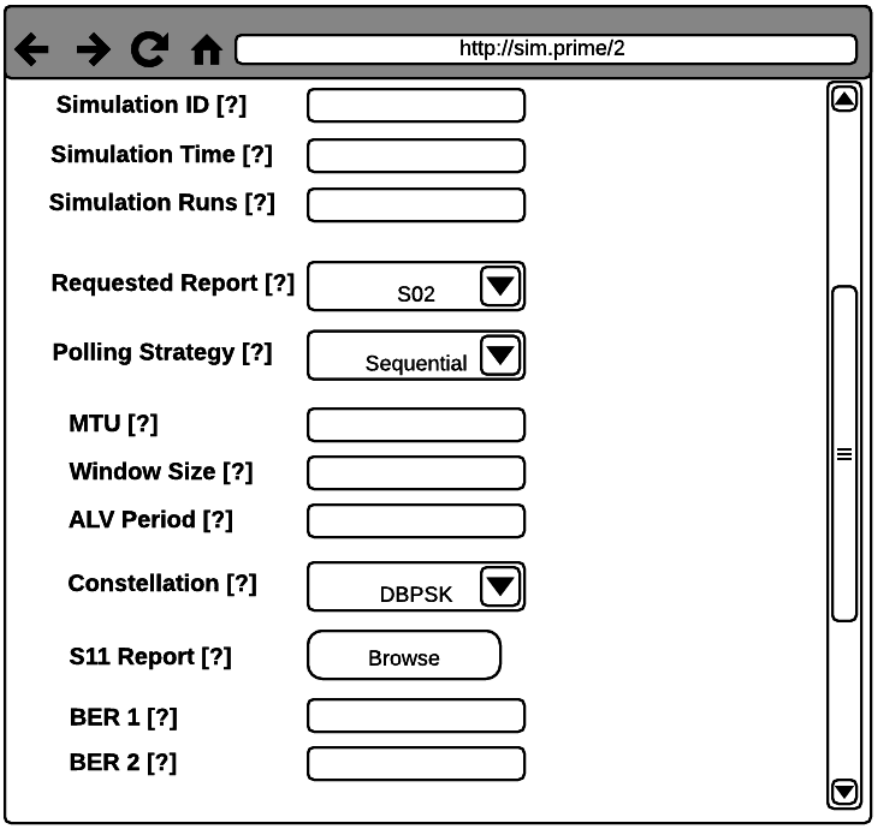


Figura 9: Maqueta proporcionada ejemplificando el formulario de entrada de datos.

3.2.2 FUN02: El sistema deberá generar el fichero de configuración (.ini) del simulador.

Como ya se ha mencionado anteriormente, el simulador requiere de un fichero de configuración con la extensión .ini (en adelante, fichero .ini, o fichero INI) para funcionar.

Este fichero contiene parte de los datos introducidos por el usuario en la interfaz que implemente el requerimiento funcional *FUN01* tal y como se describe en el apartado 3.1.1 del presente documento.

El mapeo de los datos, ilustrado en la Figura 8, es el siguiente:

- Tiempo de simulación → sim-time-limit, **.endSimTime
- Constelación:
  Según la opción marcada en el desplegable, el mapeo de los campos del archivo .ini ***.PLC_SN[*].mac.DEFAULT_COMMUNICATION_MODE* y ***.usePRM* será distinto:
  - DBPSK FEC OFF → *1 y true* respectivamente.
  - DQPSK FEC OFF → *4 y true* respectivamente.
  - D8PSK FEC OFF → *5 y true* respectivamente.
  - DBPSK FEC ON → *0 y false* respectivamente.
  - DQPSK FEC ON → *2 y true* respectivamente.
  - D8PSK FEC ON → *3 y true* respectivamente.

    - Dynamic → *0 y true* respectivamente.
- Informe solicitado:

  Según la opción marcada en el desplegable, el mapeo de los campos será el siguiente:
    - S02 → **.AppReqMsgSize_Bytes = 70B, **.AppResMsgSize_Bytes = 1568 B
    - S05 → **.AppReqMsgSize_Bytes = 70B, **.AppResMsgSize_Bytes = 646 B
- MTU → **.PLC_SN[i].llc.CIMTUSizeBits
- WS → **.PLC_SN[i].llc.defaultWindowSize
- Estrategia de sondeo:

  Según la opción marcada en el desplegable, el mapeo de los campos será el siguiente:
    - Sequential → **.appBehavior = 2001
    - Simultaneous → **.appBehavior = 2002
- Tiempo de vida → **.PLC_BN.mac.DEFAULT_ALV_TIME

Además, los siguientes campos del fichero .ini proporcionarán la información necesaria sobre el resto de ficheros requeridos por el simulador:

- ***.attFileAddress* → Nombre del fichero de la matriz de atenuaciones.
- ***.archFileAddress* → Nombre del fichero de topología de la red.
- ***.nHostsPerBranch* → Número de contadores de la red PRIME (que coincide con el número de líneas del fichero de topología de red que el simulador toma como entrada).

Se entrega junto a los requisitos iniciales, un ejemplo de fichero .ini de configuración con indicaciones de todos los campos que deben modificarse en función de los datos introducidos por el usuario (ver Anexo II).

3.2.3 FUN03: El sistema deberá generar un fichero de topología de red con el formato requerido por el simulador.

Tras configurar el simulador, este debe conocer los detalles de todos los nodos pertenecientes a la red.

Como ilustra la Figura 8, a partir del fichero S11 proporcionado por el usuario en la interfaz que implemente el requisito funcional *FUN01*, deberá generarse este fichero definido técnicamente en el apartado 3.1.1 del presente documento.

Se entrega junto a los requisitos iniciales el código fuente de una implementación en Java del analizador (*parser*) del fichero S11 (ver Anexo III), que incluye, entre otras funcionalidades, la generación de este fichero.

3.2.4 FUN04: El sistema deberá generar la matriz de atenuaciones de los pares de nodos con el formato requerido por el simulador.

El simulador necesita conocer las atenuaciones existentes entre todos los nodos de la red PRIME para poder ejecutar la simulación.

A partir de los campos Constelación, BER 1 y BER 2 proporcionados por el usuario en la interfaz que implemente el requisito funcional *FUN01*, deberá generarse este fichero definido técnicamente en el apartado 3.1.1 del presente documento.

Durante el procesamiento necesario de los datos para generar el fichero de atenuaciones, deben calcularse las atenuaciones baja (para los nodos situados en el mismo nivel lógico de la topología), media (para los nodos situados a un nivel lógico de distancia) y alta (para los nodos situados en otros niveles) a partir de las BER y de la Constelación seleccionada.

Para ello se proporcionan seis diccionarios de conversión, uno por cada constelación posible (excepto para la constelación *Dynamic* ya que es una funcionalidad prevista para futuras versiones del sistema, y en esta versión a efectos de cálculo de atenuaciones será equivalente a *D8PSK FEC ON*):

- D8PSK-FECOFF_HighRes.txt
- D8PSK-FECON_HighRes.txt
- DBPSK-FECOFF_HighRes.txt
- DBPSK-FECON_HighRes.txt
- DQPSK-FECOFF_HighRes.txt
- DQPSK-FECON_HighRes.txt

Según la constelación seleccionada por el usuario, el sistema utilizará el diccionario correspondiente (puede verse un extracto de uno de ellos en la Figura 10). En ese diccionario aparecen tres líneas indicadoras de la constelación para la cual los datos son aplicables y a partir de la cuarta se observan dos columnas separadas por una pleca. La segunda columna establece las relaciones señal / ruido (SNR o *Signal to Noise Ratio*) correspondientes a los valores BER referenciados en la primera columna. Debe seleccionarse la SNR asociada al valor de BER inmediatamente superior al proporcionado por el usuario.

```
1  D8PSK - FEC OFF
2  false
3  D8PSK
4  0,467416|-5,000000
5  0,466812|-4,750000
6  0,464758|-4,500000
7  0,462071|-4,250000
```

Figura 10: Ejemplo de fichero de conversión BER->Atenuación.

Una vez obtenido el SNR, la atenuación correspondiente se calcula con la siguiente fórmula:

$$Atenuación = P_{tx} - P_n - SNR$$

Siendo P_{tx} la potencia de transmisión de la señal y P_n la potencia de ruido de fondo. Para esta versión del sistema se considerarán los siguientes valores fijos:

$$P_{tx} = -3 \text{ dB}$$
$$P_n = -54 \text{ dB}$$

Tras seguir estos pasos, tendremos dos atenuaciones correspondientes a los dos BER proporcionados por el usuario: la atenuación baja será la asociada al BER más bajo y la media la asociada al BER más alto. La atenuación alta se fijará a un valor lo suficientemente grande (p.ej., 9999), ya que se considera que dos nodos situados a más de un nivel lógico de distancia no pueden "verse" entre sí (es decir, no tienen conectividad directa).

Se entrega junto a los requisitos iniciales el código fuente de una implementación en Java del analizador (*parser*) del fichero S11 (ver Anexo III) que incluye, entre otras funcionalidades, la generación de este fichero.

3.2.5 FUN05: El sistema deberá generar un fichero descriptivo de la topología de la red.

El simulador ya dispone de todos los detalles necesarios para comenzar con la simulación. Sin embargo, se requiere de un tercer fichero generado a partir del fichero S11 proporcionado por el usuario en la interfaz que implemente el requisito funcional *FUN01*.

Este fichero en texto plano y en lenguaje natural le proporcionará al usuario una visión global de la topología de la red, tal y como se puede ver en el ejemplo representado en la Figura 11. La finalidad de este fichero no está relacionada por tanto con la ejecución de las simulaciones, sino que se utiliza para comprobar que la topología que se va a simular no contiene incongruencias (p.ej., un nodo que comunica a través de un *switch* que no existe).

```
1  The network logical topology is :
2  I am node with SID=0
3      |
4      |
5      -> I am node with SID = 0 and LNID = 15363
6      |
7      |
8      -> I am node with SID = 0 and LNID = 15364
```

Figura 11: Ejemplo de fichero de árbol de topología.

Se entrega junto a los requisitos iniciales el código fuente de una implementación en Java del analizador (*parser*) del fichero S11 (ver Anexo III) que incluye, entre otras funcionalidades, la generación de este fichero.

3.2.6 FUN06: El sistema deberá ejecutar el simulador proporcionado usando como entrada de datos los ficheros proporcionados.

La máquina que vaya a ejecutar el simulador deberá tener instalado dicho simulador y tener disponibles a través del sistema de ficheros los siguientes ficheros:

- Fichero de configuración .ini (*FUN02*)
- Fichero de topología (*FUN03*)
- Fichero de matriz de atenuaciones (*FUN04*)

Cumplidos estos prerrequisitos, el simulador se ejecutará tantas veces como el usuario haya especificado en el campo "Número de repeticiones" en la interfaz que implemente el requisito funcional *FUN01* (el número de repeticiones es controlado por el propio simulador), como también ilustra la Figura 8.

El comando para ejecutar la simulación será el siguiente:

*/usr/bin/time -o "results/**IDSIMULACIÓN**-time.txt" -f "%E elapsed, \n%U user, \n%S system, \n%M KB memory" opp_runall -j4 ../src/simPRIME -r 0..**NÚMERO_REPETICIONES**' -f **RUTA_A_FICHERO_INI** -c ID**IDSIMULACIÓN** -G -u Cmdenv -n .:../src*

Dependiendo de los valores introducidos, esta simulación podría tardar bastante tiempo en finalizar.

Una vez finalizada la simulación, obtendremos dos ficheros de salida por cada repetición especificada:

- simulation/results/ID**IDSIMULACIÓN**-**NÚMERO_REPETICIÓN**.vec
  Este fichero contiene los resultados de la simulación en bruto.

- simulation/results/ID**IDSIMULACIÓN**-**NÚMERO_REPETICIÓN**.vci
  Este fichero debe ignorarse en la presente versión de la implementación.

Además obtendremos el siguiente fichero independientemente del número de repeticiones solicitadas:

- *results/**IDSIMULACION**-time.txt*
  Este fichero contiene detalles técnicos sobre la ejecución del simulador, como el tiempo que ha transcurrido desde el inicio hasta el final, la capacidad de proceso utilizada o la memoria del sistema utilizada.

3.2.7 FUN07: El sistema deberá preprocesar los datos de salida del simulador.

Una vez ejecutado el simulador (*FUN06*), se obtienen unos datos de salida en bruto que deben procesarse para obtener los datos que debe devolver el sistema.

El procesado de los datos obtenidos por el simulador se realiza mediante la herramienta *scavetool* incluida en la instalación del simulador.

Para ello debe ejecutarse, tantas veces como el usuario haya especificado en el campo "Número de repeticiones" en la interfaz que implemente el requisito funcional *FUN01,* el siguiente comando:

*scavetool vector -p "name(DLMSCOSEMAppTTRAllMeters)" -O "results/**IDSIMULACIÓN**-**NÚMERO_REPETICIÓN**-TTRAll" -F csv "results/ID**IDSIMULACIÓN**-**NÚMERO_REPETICIÓN**.vec*

Una vez finalizada la ejecución de la herramienta *scavetool*, obtendremos un fichero de salida por cada repetición especificada:

- simulation/results/**IDSIMULACIÓN**-**NÚMERO_REPETICIÓN**-TTRAll.csv
  Este fichero contiene los datos TTRAll requeridos por el usuario, tal y como se puede ver en el apartado 3.1.2.

3.2.8 FUN08: El sistema deberá almacenar los resultados de la simulación para poder proporcionárselos al usuario.

Teniendo disponibles los datos TTRAll devueltos por el simulador (*FUN07*), el sistema deberá almacenarlos todos en la base de datos.

Deben procesarse todos los ficheros TTRAll de salida correspondientes a la simulación ya que existirá uno por cada repetición especificada por el usuario en el campo "Número de repeticiones" en la interfaz que implemente el requisito funcional *FUN01*.

El formato seguido por este fichero está especificado en el apartado 3.1.2 y debe seguirse la estructura prevista en dicho apartado para la lectura y almacenamiento de todos los TTR presentes en los ficheros CSV TTRAll de salida.

Todos los TTR resultantes pueden almacenarse juntos y relacionados tan sólo con el identificador de la simulación solicitada, independientemente del fichero de repetición del que procedan.

3.2.9 FUN09: El sistema deberá proporcionar una interfaz gráfica de salida que provea al usuario con los detalles de la simulación en cada momento.

El sistema deberá proveer de una pantalla de detalles para cada simulación independientemente del estado en el que se encuentre.

3.2.10 FUN10: La interfaz de salida con los detalles de la simulación deberá incluir todos los parámetros que el usuario introdujo en la interfaz de entrada.

La interfaz especificada en el requisito *FUN09* deberá mostrar un apartado en el que se muestren los detalles que el usuario introdujo al solicitar la simulación en la interfaz que

implemente el requisito funcional *FUN01* para facilitar la consulta del escenario simulador y por ende la posterior interpretación de resultados

3.2.11 FUN11: Los ficheros generados deberán ser accesibles desde la interfaz de salida.

La interfaz especificada en el requisito *FUN09* deberá facilitar al usuario el acceso a los ficheros de entrada al simulador (*FUN02*, *FUN03* y *FUN04*) y del fichero de descripción de la topología (*FUN05*) según el sistema los vaya generando.

3.2.12 FUN12: La interfaz de salida con los detalles de la simulación deberá proporcionar una tabla de auditoría que incluya datos sobre el estado de procesamiento de la simulación.

La interfaz especificada en el requisito *FUN09* contendrá una tabla informando del estado de la solicitud de simulación para las siguientes tareas:

- Análisis del fichero S11
- Generación del fichero .ini
- Ejecución del simulador

Los siguientes estados deben ser incluidos en la tabla para cada tarea, incluyendo la fecha y hora en la que la tarea entró en cada estado:

- Tarea solicitada
- Tarea iniciada
- Tarea finalizada correctamente
- La tarea finalizó con errores

Además se incluirá la posibilidad de que las tareas informen de otros detalles de interés en un campo de texto libre, bien al informar del cambio de estado de una tarea, o bien añadiendo datos adicionales dentro del mismo estado.

3.2.13 FUN13: Los resultados TTRAll del simulador deben visualizarse gráficamente en la interfaz de salida con los detalles de la simulación.

Una vez estén disponibles los datos de la simulación, tal y como se especifica en el requisito *FUN08*, deberán mostrarse todos los TTR calculados en una nueva sección en la interfaz especificada en el requisito *FUN09*.

Además se incluirá una sección adicional mostrando gráficamente esos resultados en un diagrama de caja y bigotes en el que se puedan visualizar los valores TTR mínimo y máximo, los cuartiles y la mediana.

3.3 Requisitos no funcionales

Los requisitos no funcionales son aquellos que definen el sistema, pero no describen el comportamiento del mismo. En contraposición a los requisitos funcionales, los no funcionales no definen la información que recibe el *software* o la que debe guardarse ni las funciones a realizar.

3.3.1 NFUN01: El sistema debe ser escalable.

La complejidad del procesado de simulación hace que éste lleve mucho tiempo y consuma muchos recursos. Por ello debe permitirse la ejecución de múltiples simulaciones de forma simultánea tanto en una misma máquina (aprovechando así al máximo todos los recursos del sistema) como en diferentes máquinas.

3.3.2 NFUN02: Debe evitarse el llenado del disco duro de las máquinas.

La información en bruto que devuelve el proceso de simulación ocupa un gran espacio en el sistema de almacenamiento de datos de la máquina. Debe evitarse que las máquinas se queden sin espacio libre, eliminando los datos no necesarios después de las simulaciones.

3.3.3 NFUN03: La simulación debe poder ejecutarse en segundo plano.

Al poder tomar una cantidad considerable de tiempo, al solicitar una simulación, ésta debe ejecutarse en segundo plano, permitiendo al usuario continuar con otras tareas o solicitar más simulaciones.

3.3.4 NFUN04: El usuario debe ser avisado por correo electrónico al finalizar la simulación.

Debido a la gran cantidad de tiempo que toman las simulaciones, una vez finalizado el proceso de simulación, el sistema deberá avisar, mediante un correo electrónico al usuario, de la finalización de la ejecución. Este aviso deberá proporcionar un acceso rápido a los resultados de la simulación (p.ej., a través de una URL).

3.3.5 NFUN05: Sólo los usuarios autorizados podrán acceder al sistema.

Se requiere un sistema de gestión de credenciales que gestione el acceso a la aplicación, identifique las simulaciones con los usuarios que las solicitaron y proporcione el correo electrónico de dichos usuarios.

Se ilustra una maqueta de la interfaz de este sistema de gestión de credenciales en la Figura 12.

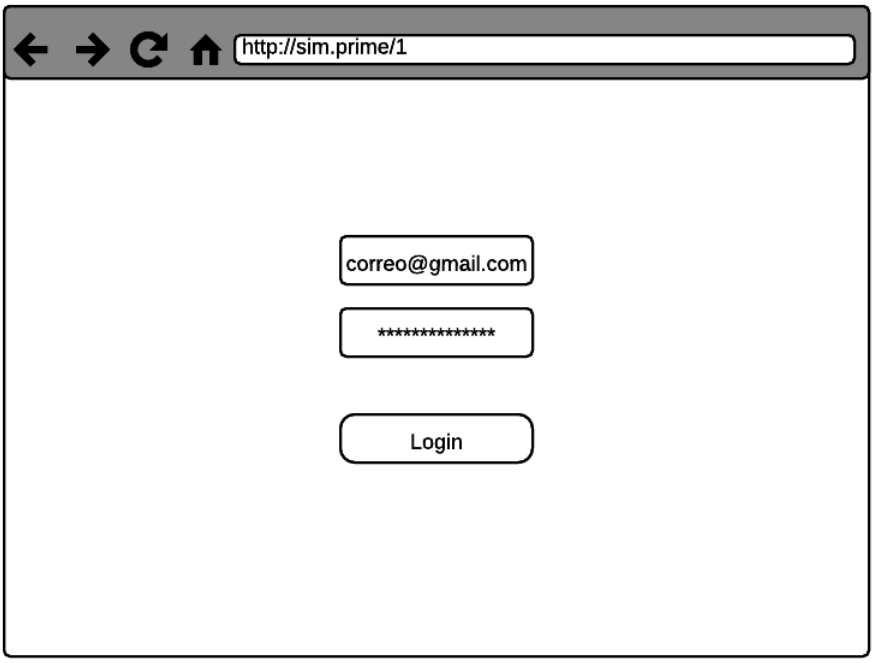


Figura 12: Maqueta proporcionada ejemplificando la interfaz de acceso al sistema.

3.4 Tecnologías seleccionadas

3.4.1 Solución web vs. cliente-servidor.

Aunque desde un inicio se planteó el proyecto desde el punto de vista de una aplicación web, esto no era un requisito, por lo que había que comprobar si era la mejor alternativa para el sistema planteado.

La alternativa más viable al sistema web era una aplicación de escritorio que se comunicara con un subsistema distribuido de simulaciones PRIME. Esta alternativa presentaba las siguientes ventajas con respecto al sistema web:

- Parte del procesamiento podría realizarse en los ordenadores de los usuarios.
- En entornos con una gran volumetría de cálculo, serían necesarios menos servidores de aplicaciones.

Sin embargo, también presenta una serie de desventajas:

- El despliegue es más complicado. La aplicación debería instalarse en los ordenadores de todos los usuarios del sistema, mientras que una aplicación web sólo requiere instalación en el servidor de aplicaciones. El entorno de un servidor es mucho más controlable que el conjunto de entornos heterogéneos y variables de un grupo de usuarios.
- Los usuarios tardarían más en habituarse al sistema ya que además de necesitar formación sobre el sistema, necesitarían formación sobre la aplicación, lo que iría en detrimento de la usabilidad de la aplicación.
- Es una solución más compleja. Desarrollar una aplicación de escritorio con interfaz gráfica es más costoso que desarrollar una aplicación web.
- La comunicación entre los sistemas sería también más compleja y propensa a errores y fallos de seguridad, ya que los sistemas de almacenamiento de datos y de simulación estarían directamente expuestos a la red de los usuarios, y el coste de un sistema que intermedie entre los distintos componentes es similar al de hacer una aplicación web.

Dichas desventajas respaldaron la decisión de desarrollar una aplicación web.

3.4.2 Lenguaje de programación y *Frameworks*

Como se vio en el capítulo 2 “Estado del Arte”, la elección de lenguajes de programación y la de un *framework* están íntimamente relacionadas, por ello se tomaron ambas decisiones de forma simultánea.

Sobre el conjunto de lenguajes analizados, se hizo una pequeña preselección para que la decisión fuera abarcable. Los criterios utilizados fueron:

- El lenguaje utilizado debía tener disponible un *framework* compatible con las necesidades del proyecto, es decir, con las siguientes características:

    - Orientado al desarrollo web.
    - Modelo-vista-controlador (MVC).
    - Abstracción sencilla del modelo de datos.
    - Integración sencilla de motores (*backends*).
- Tanto el lenguaje como los *frameworks* seleccionados debían presentar buen soporte por parte de la comunidad.
- Existencia de herramientas gratuitas que faciliten el desarrollo y el despliegue:
    - Entornos de desarrollo.
    - *Debuggers.*
    - Servidores de aplicaciones.
- Existencia de soluciones de escalabilidad y gestión de tareas *batch*
- Facilidad de aprendizaje. No sólo para el desarrollo inicial sino para su posterior mantenimiento.

Java y Python cumplieron en mayor o menor medida todos los requisitos, teniendo en cuenta los siguientes *frameworks*:

- Java
    - Spring
    - Play
- Python
    - Django

3.4.2.1 La opción de Java

Java destacaba en las herramientas de desarrollo. Dispone de multitud de entornos de desarrollo con funcionalidades de autocompletado y *debugging*, aunque el despliegue es algo más complejo.

Spring es uno de los *frameworks* más maduros del mercado. Sin embargo, también es de los más complejos. Tiene disponible un módulo llamado spring-batch que facilita la gestión de tareas para la realización de tareas más pesadas. El encolamiento de tareas también es posible con el uso de otros módulos de la familia Spring.

Play es bastante más sencillo e igualmente completo. Sin embargo, tiene una base de usuarios más reducida, lo que hace que su soporte de la comunidad no sea tan extenso. Incluye por defecto un gestor de tareas, aunque requiere de módulos para comunicarse con un gestor de colas.

Ambos *frameworks* tienen la desventaja de que, aunque el procesamiento de tareas es trivial, no lo es el procesamiento distribuido de las mismas.

3.4.2.2 La opción de Python

Python en cambio destaca por su flexibilidad y facilidad de aprendizaje. Ofrece más posibilidades que Java a la hora de desplegar el *software* en un entorno productivo y Django es un *framework* potente con una gran comunidad detrás.

Una clara ventaja con respecto a los *frameworks* de Java contemplados es la posibilidad de uso de Celery, un gestor de tareas con un alto grado de integración en Python en general y Django en particular. Además de gestionar las tareas, Celery se encarga de comunicarse con un gestor de colas para poder ejecutarlas de forma distribuida.

Django parecía ser el *framework* que más se adaptaba a las necesidades del proyecto. Además, el grupo de investigación en el seno del cual se ha desarrollado este Proyecto Fin de Carrera está desarrollando otras aplicaciones web relacionadas en Django, por lo que usar Django facilitaría el futuro mantenimiento y desarrollo de la aplicación. Por todos estos motivos, Django fue finalmente la opción escogida.

Al no tener ninguna dependencia con código no compatible con Python 3, ésta fue la versión del lenguaje escogida.

3.4.3 Gestor de tareas y gestor de colas

Para poder ejecutar el motor (*backend*) en segundo plano, es necesario disponer de un sistema que gestione las tareas y la cola de tareas.

Como se ha mencionado en el punto anterior, al usar el *framework* Django, Celery es la opción natural para gestionar las tareas.

Celery necesita además de un *broker* o gestor de colas que permita la comunicación entre los distintos elementos que se encargan de la ejecución de las tareas. Las opciones consideradas para elegir el gestor de colas fueron la base de datos de Django, Redis y RabbitMQ.

3.4.3.1 La opción de la base de datos de Django

Como ya estábamos utilizando el *framework* Django, y Celery puede usarla para gestionar las colas, ésta debía ser una opción a considerar.

La principal ventaja era que no se requería de *software* adicional, ya que nuestro sistema ya requería de una base de datos. Sin embargo, no es una solución diseñada específicamente para los propósitos requeridos. Esta opción es más indicada para entornos de desarrollo más que de producción, ya que tiene algunas carencias, entre las que destacan dos que han ocasionado su descarte [59]:

- El soporte de Celery al uso de la base de datos de Django es experimental. Hay errores conocidos y el equipo de Celery no tiene recursos para solucionarlos.
- Soporta muy pocos ejecutores de tareas simultáneamente, pudiendo llegar a la repetición de tareas.

3.4.3.2 La opción de Redis

Redis es uno de los dos gestores de colas cuyo soporte por parte de Celery se considera estable. Sin embargo, no es la opción más apropiada ya que este gestor, como ya se analizó en el apartado 2.4.1.4, se trata de un almacén de datos volátil, sacrificando así fiabilidad por rendimiento, ya que no garantiza la entrega de los mensajes [60].

3.4.3.3 La opción de RabbitMQ

Finalmente, RabbitMQ ofrece un soporte estable por parte de Celery y la garantía de que las peticiones llegarán a su destino, por lo que fue la opción finalmente escogida.

3.5 Arquitectura del *software*

Para el diseño del sistema, se ha optado por una arquitectura modular siguiendo el patrón MVC que separa los datos (modelo), la interfaz de usuario (vista) y la lógica del sistema (controlador).

Seguir un modelo modular facilita que el sistema sea escalable, ya que nos permite desplegar los módulos de manera independiente en un entorno de computación distribuido, pudiendo, por ejemplo, tener máquinas dedicadas en exclusiva a ejecutar simulaciones y otras a servirle la interfaz web al usuario.

La Figura 13 muestra un esquema de la arquitectura software del sistema, indicando las principales tecnologías utilizadas para implementar cada módulo.

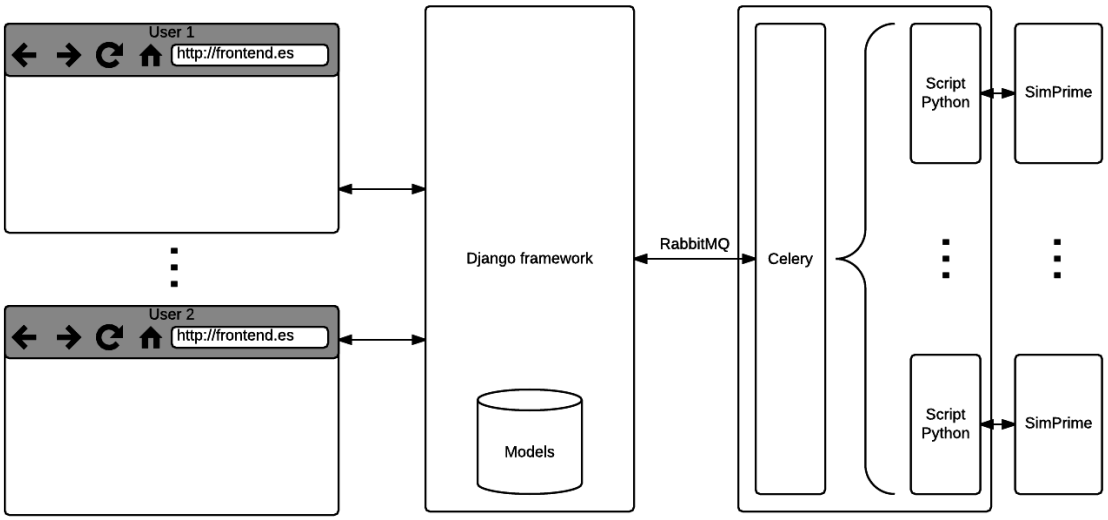


Figura 13: Esquema general de la arquitectura del sistema.

3.5.1 Módulos

Para definir el mapa de módulos que tendría el sistema finalmente, se hizo un análisis de las principales funcionalidades requeridas para el mismo. Éstas son:

- Servirle al usuario una interfaz a través de una página web (*FUN01, FUN09, FUN10, FUN11, FUN12, FUN13* y *NFUN05*).
- Analizar el fichero S11 y generar los ficheros de datos de entrada al simulador (*FUN03, FUN04* y *FUN05).*
- Generar un fichero de configuración .ini de entrada al simulador (*FUN02*).
- Ejecutar la simulación (*FUN06* y *NFUN03*).
- Procesar y almacenar los resultados (*FUN07* y *FUN08*).
- Limpiar el entorno del simulador de ficheros no necesarios (*FUN09*).
- Enviar un correo electrónico de aviso al usuario que solicitó la simulación (*NFUN04*).

El requisito no funcional *NFUN01 (El sistema debe ser escalable)*, es transversal a toda la implementación y por tanto no se corresponde con ninguna funcionalidad concreta, aunque debe ser tenido en cuenta a la hora de diseñar y desarrollar su implantación.

Una vez claras las funcionalidades, se tomó la decisión de crear tres módulos de Django y un módulo independiente para las ejecuciones del motor tal y como se explica en los siguientes apartados.

3.5.1.1 Interfaz (Frontend)

Este módulo está programado utilizando el *framework* Django y es el responsable de toda la comunicación entre el sistema y el usuario, implementando las siguientes funcionalidades:

- Servirle al usuario una interfaz a través de una página web.
- Enviar un correo electrónico de aviso al usuario que solicitó la simulación.

Como se ha visto en la sección 3.5.1, estas funcionalidades cubren los siguientes requisitos:

- FUN01: El sistema deberá proporcionar una interfaz gráfica para el usuario en la que pueda introducir los detalles de la simulación solicitada.
- FUN09: El sistema deberá proporcionar una interfaz gráfica de salida que provea al usuario con los detalles de la simulación en cada momento.
- FUN10: La interfaz de salida con los detalles de la simulación deberá incluir todos los parámetros que el usuario introdujo en la interfaz de entrada.
- FUN11: Los ficheros generados deberán ser accesibles desde la interfaz de salida.
- FUN12: La interfaz de salida con los detalles de la simulación deberá proporcionar una tabla de auditoría que incluya datos sobre el estado de procesamiento de la simulación.
- FUN13: Los resultados TTRAll del simulador deben visualizarse gráficamente en la interfaz de salida con los detalles de la simulación.
- NFUN04: El usuario debe ser avisado por correo electrónico al finalizar la simulación.
- NFUN05: Sólo los usuarios autorizados podrán acceder al sistema.

3.5.1.2 Analizador de ficheros S11

Este módulo está programado utilizando el *framework* Django y es el responsable de analizar el fichero S11 y los valores BER proporcionados por el usuario, para a continuación generar los siguientes ficheros:

- Matriz de atenuaciones
- Fichero de topología
- Árbol descriptivo de topología

Como se ha visto en la sección 3.5.1, esta funcionalidad cubre los siguientes requisitos:

- FUN03: El sistema deberá generar un fichero de topología de red con el formato requerido por el simulador.
- FUN04: El sistema deberá generar la matriz de atenuaciones de los pares de nodos con el formato requerido por el simulador.
- FUN05: El sistema deberá generar un fichero descriptivo de la topología de la red.

3.5.1.3 Generador de ficheros de configuración (.ini)

Este módulo está programado utilizando el *framework* Django y es el responsable de generar el fichero de configuración .ini requerido por el simulador.

Como se ha visto en la sección 3.5.1, esta funcionalidad cubre los siguientes requisitos:

- FUN02: El sistema deberá generar el fichero de configuración (.ini) del simulador.

3.5.1.4 Motor (Backend)

Este módulo implementa las siguientes funcionalidades:

- Ejecutar la simulación.
- Procesar y almacenar los resultados.
- Limpiar el entorno del simulador de ficheros no necesarios.

Como se ha visto en la sección 3.5.1, estas funcionalidades cubren los siguientes requisitos:

- FUN06: El sistema deberá ejecutar el simulador proporcionado usando como entrada de datos los ficheros proporcionados.
- FUN07: El sistema deberá preprocesar los datos de salida del simulador.
- FUN08: El sistema deberá almacenar los resultados de la simulación para poder proporcionárselos al usuario.
- FUN09: El sistema deberá proporcionar una interfaz gráfica de salida que provea al usuario con los detalles de la simulación en cada momento.
- NFUN03: La simulación debe poder ejecutarse en segundo plano.

El motor es el módulo con mayor carga de trabajo (en consumo de recursos) y debe estar ubicado junto al componente de simulación proporcionado. Además, salvo el acceso a la base de datos, este módulo no requiere de ninguna de las funcionalidades proporcionadas por Django, por lo que no parecía adecuado implementarlo con este *framework* junto con el resto de módulos, ya que lo convertiría en una solución más compleja innecesariamente y dificultaría el despliegue conjunto del módulo y del simulador.

Al no usar Django, se necesita de una solución alternativa y eficaz para el acceso a la base de datos. Por ello este módulo está programado utilizando el *framework* SQLAlchemy que proporciona un acceso transparente a la misma con una configuración mínima.

3.5.2 Comunicación entre módulos

Para comunicarse entre sí, los módulos deben implementar tareas de Celery que disparen la ejecución de los procesos y proporcionen la información contextual suficiente para llevarlos a cabo. Todos los módulos tienen acceso a la base de datos por lo que la información persistente no debe comunicarse en las solicitudes de ejecución de tareas que se realicen.

Las tareas expuestas a Celery son las siguientes:

- Analizar S11 (por el módulo analizador de ficheros S11).
- Generar fichero de configuración .ini (por el módulo generador de ficheros de configuración .ini).

- Ejecutar motor (por el motor).
- Enviar correo electrónico de aviso al usuario (por el módulo interfaz).

Al ser la ejecución de todas las tareas dependiente de la finalización de todas las tareas de módulos anteriores, se sigue un esquema de ejecución lineal como se describe en la Figura 14.

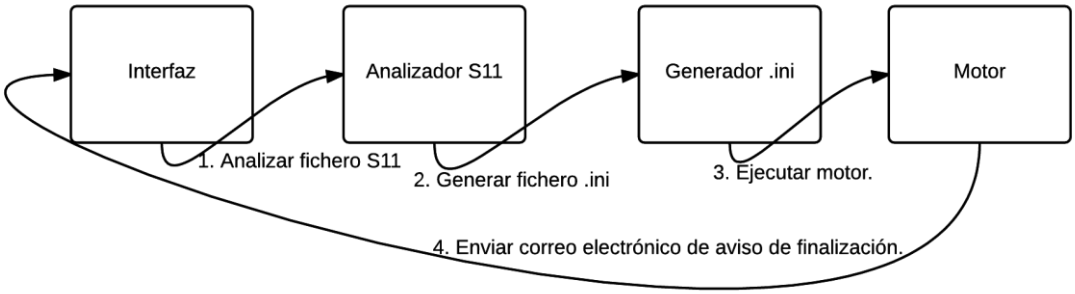


Figura 14: Representación gráfica de la comunicación entre módulos para la ejecución de una simulación.

Las tareas no se ejecutan inmediatamente después de ser solicitadas, sino que Celery las envía al gestor de colas RabbitMQ. Una vez en la cola, la tarea se envía al primer ejecutor disponible capaz de llevar a cabo esa tarea en concreto.

3.5.3 Modelo de datos

El modelo de datos es una pieza clave del desarrollo, ya que además de asegurar la persistencia de la información, permite a los distintos módulos el acceso a la información no contextual (por ejemplo, los datos introducidos por el usuario en la solicitud de simulación) a partir de la información contextual que se les proporciona en su inicio (por ejemplo, el identificador de solicitud de simulación sobre la que se debe ejecutar la tarea).

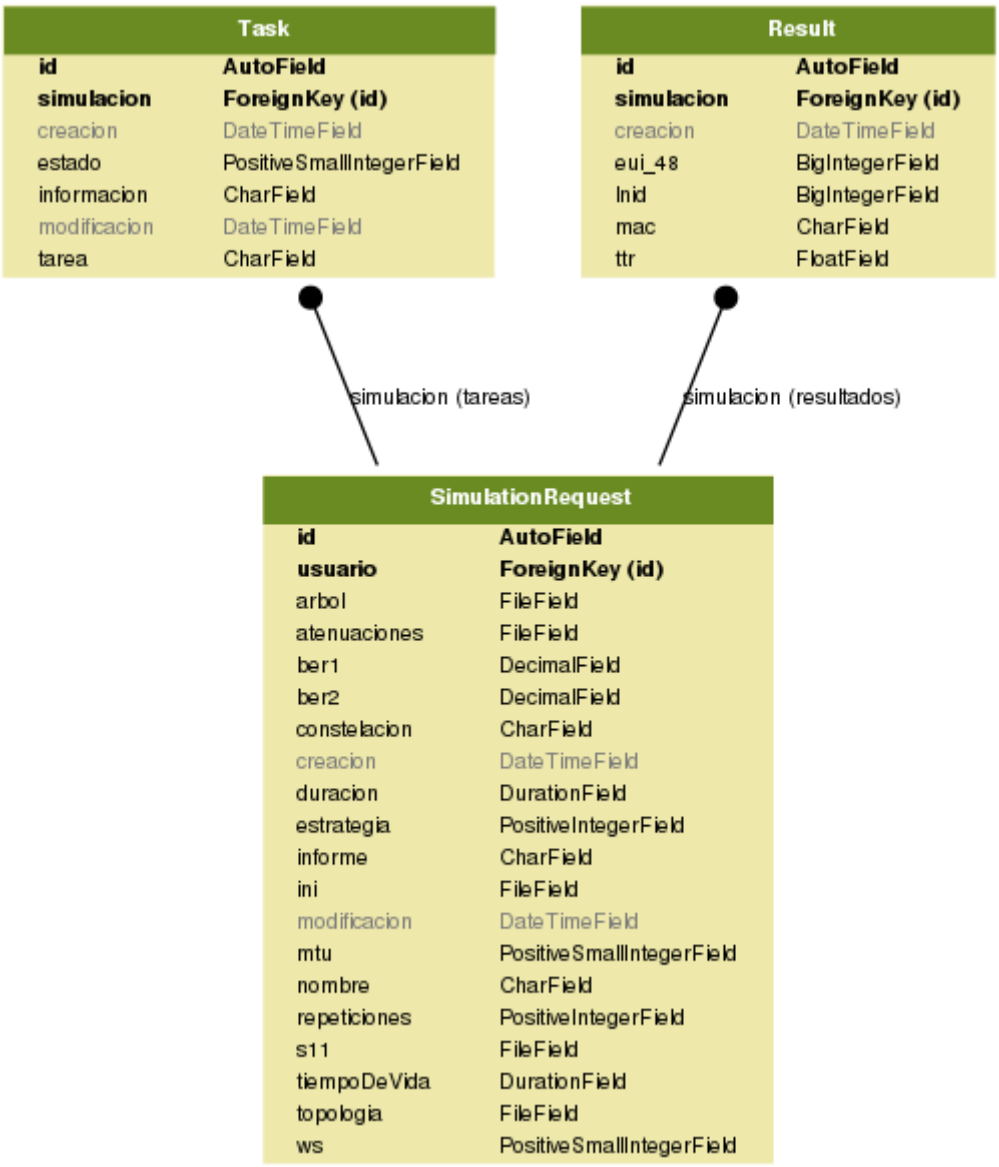


Figura 15: Modelo de datos.

Para implementar este modelo de datos, representado en la Figura 15, se crean tres tablas en una base de datos:

- **SimulationRequest:** Contiene todos los datos introducidos por el usuario al solicitar la simulación.
- **Task:** Contiene detalles sobre el proceso de ejecución de las tareas para cada simulación solicitada.
- **Result:** Contiene los resultados de la simulación solicitada.

Además, debido a la escalabilidad requerida (*NFUN01*), el sistema puede ejecutarse de forma distribuida, por lo que toda la información persistente debe ser incluída en un repositorio de datos accesible por todas las máquinas y módulos que conformen el sistema. Por ello los ficheros S11 proporcionados por los usuarios y los ficheros requeridos por el sistema son almacenados en la misma base de datos.

Como se puede ver en la Figura 16, para este fin se precisan 3 tablas:

- **S11File:** Almacena los ficheros S11 proporcionados por los usuarios.
- **IniFile:** Almacena los ficheros de configuración .ini generados.
- **ParserFile:** Almacena los ficheros generados a raíz del análisis del fichero S11 proporcionado por el usuario:
    - Árbol de topología en texto plano y lenguaje natural.
    - Matriz de atenuaciones.
    - Fichero de topología.

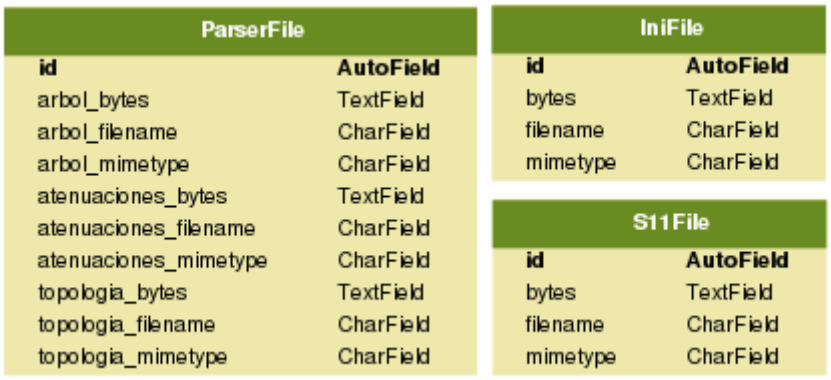


Figura 16: Tablas de almacenamiento de ficheros.

La tabla *SimulationRequest* vista anteriormente incluye referencias a estas tablas para que los archivos relacionados con la solicitud de simulación puedan ser encontrados.

3.6 Flujo de trabajo

Esta sección incluye una serie de figuras que describen gráficamente y en detalle el funcionamiento que debe tener el sistema desde que un usuario accede al mismo para solicitar una simulación hasta que este recibe un correo electrónico advirtiéndole de que la ejecución de la misma ha finalizado.

Como se puede comprobar en la leyenda mostrada en la Figura 17, se han utilizado una serie de pictogramas para representar los distintos elementos del diagrama. Estos son:

- **Actor:** Es la representación del usuario realizando acciones sobre el sistema.
- **Modelo:** Es la representación del componente *modelo* de la arquitectura MVC. Representa la información con la que trabaja el sistema.
- **Vista:** Es la representación del componente *vista* de la arquitectura MVC. En este caso en concreto, se refiere a las plantillas implementadas en el *framework* Django. Es el componente con el que interactúa el usuario (interfaz de usuario).
- **Controlador:** Es la representación del componente *controlador* de la arquitectura MVC. Implementa la lógica necesaria.
- **Proceso:** Representa el tiempo de vida del componente al que hace referencia.
- **Acción/Solicitud:** Representa una acción o solicitud del componente o elemento origen al componente o elemento destino.
- **Respuesta:** Representa la devolución de datos de un elemento a otro.
- **Ejecuciones alternativas:** Establece un procedimiento distinto según la condición que se cumpla.
- **Bucle de repetición:** Repite el procedimiento mientras se cumpla la condición.
- **Referencia a otro diagrama:** Representa el encaje de otro diagrama en el punto en el que se encuentre.
- **Diagrama referenciado:** Si un diagrama está referenciado en otro, estará incluido dentro de este recuadro.

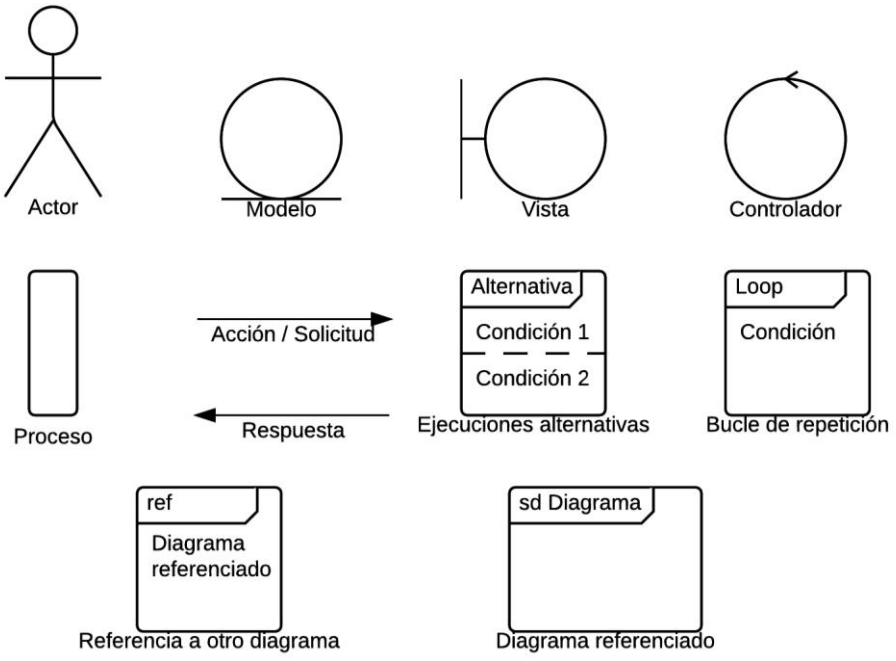


Figura 17: Leyenda de pictogramas.

En la Figura 18 se puede comprobar cómo el usuario entra en el sistema, identificándose ante el mismo. Una vez el usuario tiene una sesión activa, accede a la interfaz de solicitud de simulación y rellena el formulario correspondiente.

Una vez rellenado ese formulario, se comprueba que todo esté correcto. Si no es así, se le indica al usuario solicitándole la subsanación de los errores.

Si el formulario enviado es correcto, se crea una nueva entrada de solicitud de simulación en el modelo y se procede al análisis del fichero S11 proporcionado por el usuario.

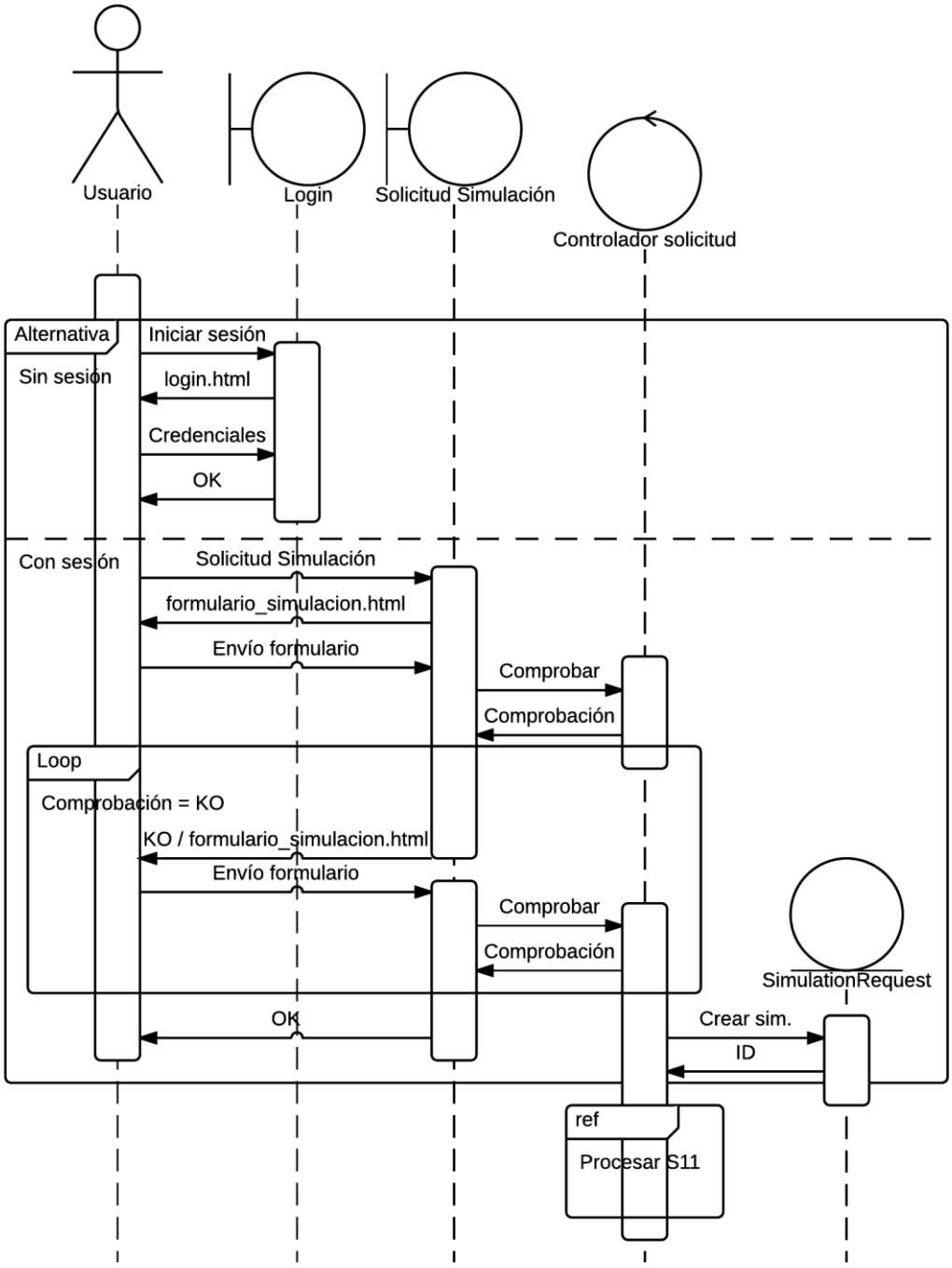


Figura 18: Flujo de solicitud de simulación.

La Figura 19 describe el proceso de análisis del fichero S11. Como se trata de un módulo diferente al que slo le llega información contextual, lo primero que hace es recuperar los detalles persistentes de la base de datos.

Posteriormente lee el fichero S11 y genera los tres ficheros requeridos en consecuencia. Una vez generados los ficheros, el sistema llama al módulo de generación de ficheros .ini.

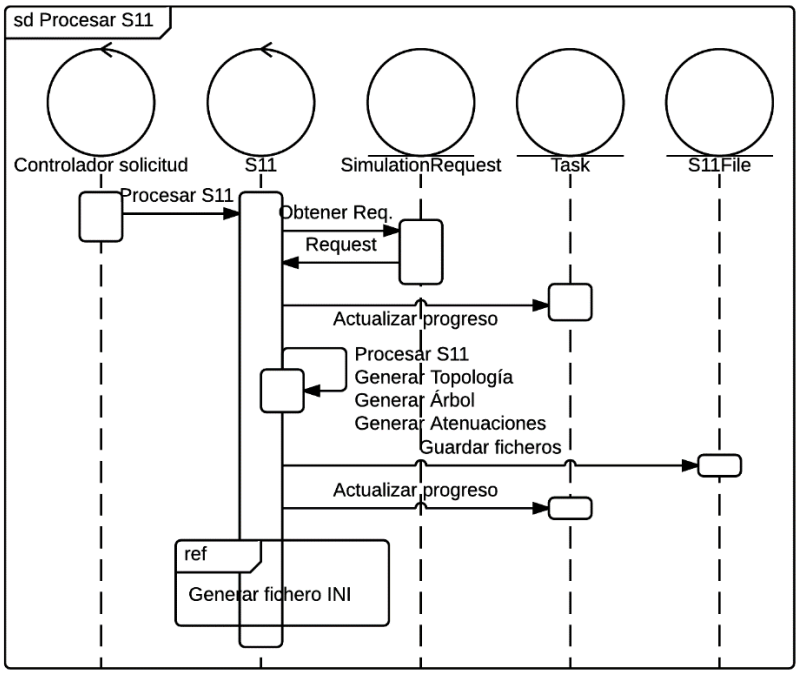


Figura 19: Flujo de análisis y proceso del fichero S11.

La Figura 20 describe el proceso de generación del fichero .ini. Al igual que en el análisis del fichero S11, la primera acción corresponde a la recuperación de los detalles de la simulación a partir de la información contextual proporcionada.

Durante este proceso se genera el fichero .ini y se solicita la ejecución de la tarea de simulación.

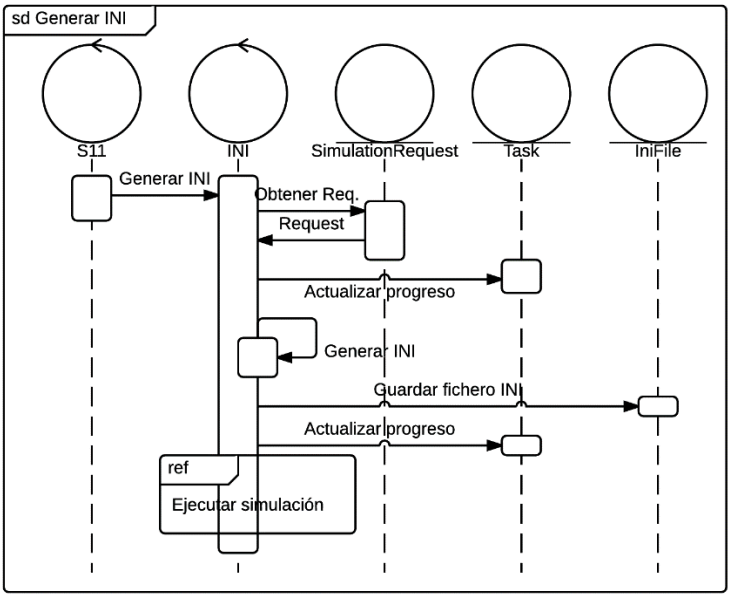


Figura 20: Flujo de generación del archivo .ini.

La Figura 21 describe el proceso de ejecución de la simulación, donde, al igual que en los módulos anteriores, el primer paso es recuperar los detalles de la simulación a partir de la información contextual proporcionada.

Posteriormente, el motor debe obtener del modelo los ficheros generados por el analizador del fichero S11 y el fichero de configuración .ini.

Una vez descargados los ficheros de la base de datos, deben ubicarse en las rutas adecuadas para que el simulador, ejecutado a continuación, los encuentre y los tome en cuenta a la hora de realizar la simulación.

Una vez finalizada la ejecución del simulador, se requiere un posprocesado de los resultados generados por este. Los datos resultantes de este posprocesado se envían para su almacenamiento en el proceso de datos.

Finalmente se le requiere al módulo interfaz que envíe un correo electrónico al usuario informándole de la finalización de la simulación.

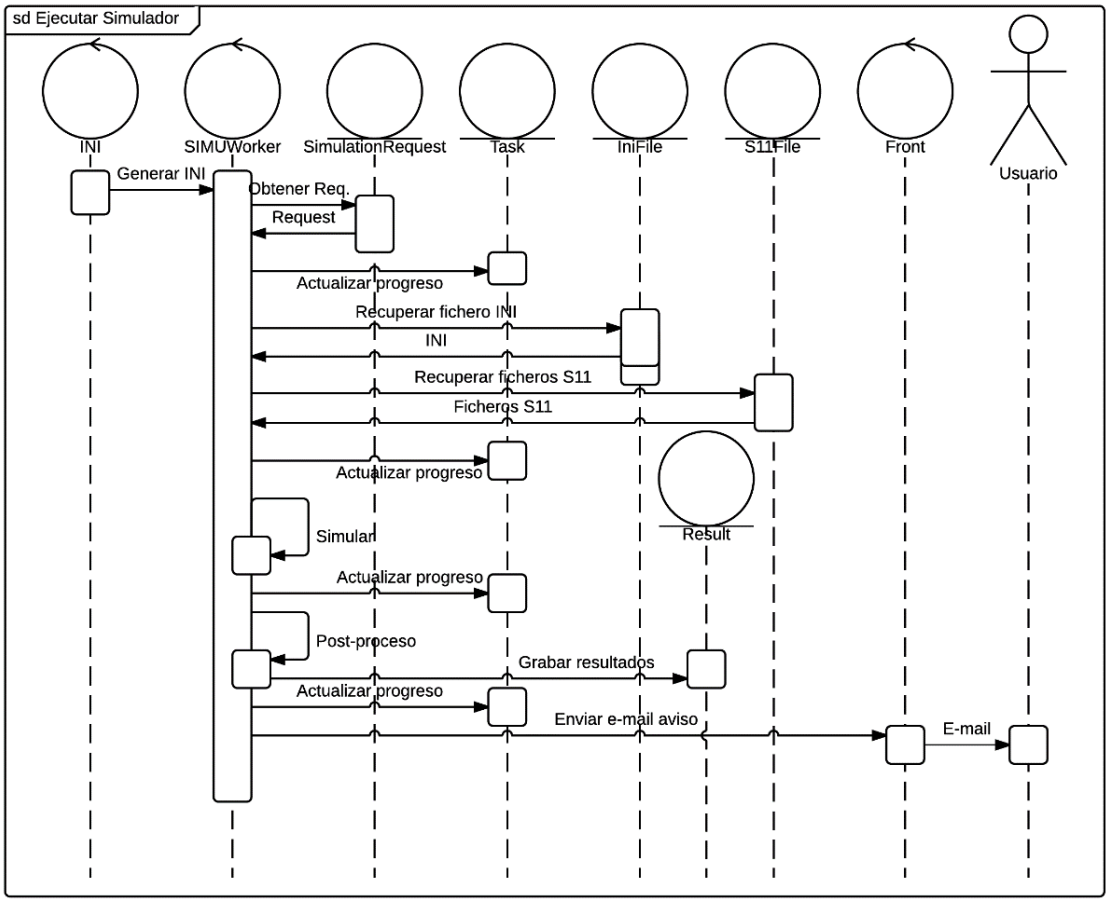


Figura 21: Diagrama que muestra el funcionamiento del motor que ejecuta las simulaciones.

Durante todos los procesos se introducen en el modelo referencias al estado de ejecución de las tareas que se estén llevando a cabo.

Capítulo 4

Desarrollo y Puesta en marcha

El objetivo de este capítulo es detallar la implementación de cada uno de los módulos que conforman el sistema, siguiendo los principios de diseño definidos en el capítulo anterior. También incluye detalles sobre el procedimiento para la puesta en marcha del sistema.

4.1 Interfaz (*Frontend*)

Tal y como se define en el apartado 3.5.1.1, este módulo, implementado con el *framework* Django, es el que interactúa directamente con el usuario, cumpliendo las siguientes funciones:

- Servirle al usuario una interfaz a través de una página web.
- Enviar un correo electrónico de aviso al usuario que solicitó la simulación.

En los siguientes apartados, se explicará la implementación de ambas funciones.

4.1.1 Interfaz de usuario a través de una página web

Uno de los requisitos de la aplicación (*NFUN05*) era que el sistema diera acceso únicamente a los usuarios a los que previamente se les hubiera otorgado el acceso. Django provee de un sistema de autenticación de usuarios que cumple con esta función, proporcionando una interfaz para que el administrador pueda añadir y eliminar usuarios [61].

Las pantallas son el pilar básico de interactuación con el usuario del sistema. El sistema de plantillas de Django proporciona una forma sencilla de generar pantallas en HTML de forma dinámica, correspondiéndose con los componentes vista del patrón MVC [62].

Como base para la implementación de plantillas se han utilizado los estándares HTML5 y CSS3.

Al ser un sistema pensado para incrementar la productividad de usuarios en su puesto de trabajo, se ha implementado pensando en un ordenador como dispositivo de acceso. Sin embargo, toda la funcionalidad del sistema está disponible si este es accedido desde una tableta o teléfono inteligente.

Todas las plantillas se han escrito teniendo en mente el sistema de internacionalización y localización proporcionado por Django [63]. Así, si en un futuro se requiere la implementación del sistema en otro idioma, sería sencillo implementarlo incluyendo un fichero de textos traducidos.

En los siguientes apartados se describe la implementación de todas las plantillas necesarias para la interactuación del usuario con el sistema.

4.1.1.1 base.html

La plantilla base implementa el aspecto básico de todas las pantallas, incluyendo el encabezado, el pie de página y el menú de navegación. Esta plantilla básica es *heredada* por otras plantillas específicas a la información que se va a mostrar en cada momento.

De esta forma, en el caso de que una entidad quisiera implementar este sistema, sólo tendría que cambiar esta plantilla y el fichero de estilos *estilo.css* para adaptarlo a su imagen corporativa.

Puede verse un ejemplo gráfico del resultado de la generación de pantallas con el uso de esta plantilla en la Figura 22.

Figura 22: Muestra de la plantilla base

4.1.1.2 login.html

Esta plantilla implementa el formulario de inicio de sesión necesario para utilizar el sistema de autenticación de Django, mostrando al usuario la pantalla que puede verse en la Figura 23.

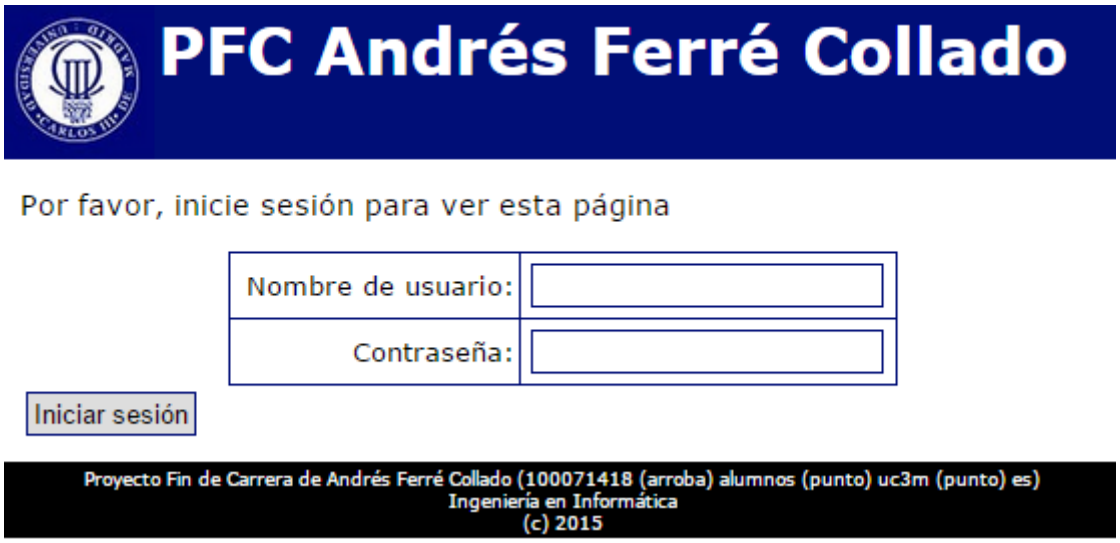


Figura 23: Muestra de la pantalla de inicio de sesión.

4.1.1.3 formulario_simulacion.html

Para solicitar una simulación, el usuario debe rellenar un formulario indicando las opciones deseadas para su ejecución (*FUN01*). Como se puede ver en la Figura 24, el formulario consta de los siguientes campos, que ya se analizaron en la sección 3.2.1:

- Nombre
- Duración
- Repeticiones
- Constelación
- S11
- BER1
- BER2
- Informe
- MTU
- WS
- Estrategia
- Tiempo de Vida

Los valores por defecto aparecen prerrellenos al cargar la pantalla, los campos numéricos sólo permiten la introducción de cifras y los campos BER (decimales), permiten caracteres relacionados con la notación decimal convencional y la notación científica.

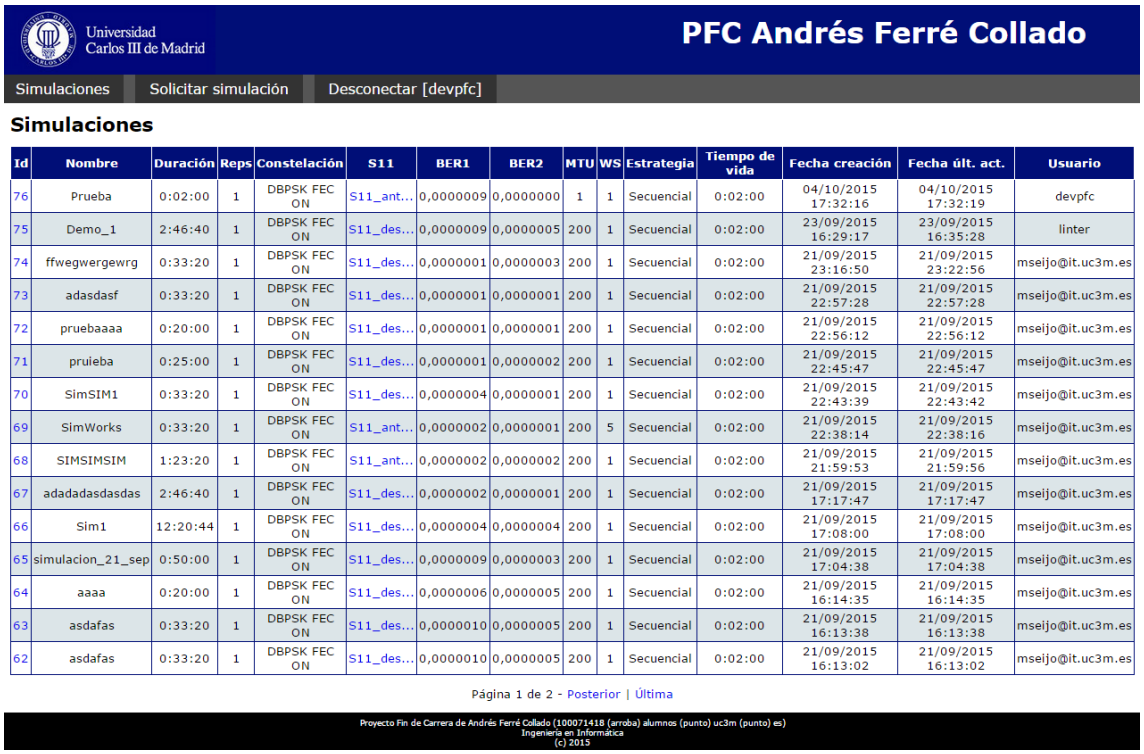
Cuando se solicita la simulación, el módulo genera una tarea de Celery destinada al módulo analizador de ficheros S11.

Figura 24: Muestra del formulario de solicitud de simulación

4.1.1.4 simulationrequest_list.html

Esta plantilla, que genera pantallas como la que puede verse en la Figura 25, proporciona una lista de simulaciones solicitadas. Muestra para cada simulación los detalles que fueron introducidos por el usuario al solicitar la simulación, además de:

- Fecha de creación de la simulación
- Fecha de última actualización de la simulación
- Usuario que solicitó la simulación.



Id	Nombre	Duración	Reps	Constelación	S11	BER1	BER2	MTU	WS	Estrategia	Tiempo de vida	Fecha creación	Fecha últ. act.	Usuario
76	Prueba	0:02:00	1	DBPSK FEC ON	S11_ant...	0,0000009	0,0000000	1	1	Secuencial	0:02:00	04/10/2015 17:32:16	04/10/2015 17:32:19	devpfc
75	Demo_1	2:46:40	1	DBPSK FEC ON	S11_des...	0,0000009	0,0000005	200	1	Secuencial	0:02:00	23/09/2015 16:29:17	23/09/2015 16:35:28	linter
74	ffwegwergewrg	0:33:20	1	DBPSK FEC ON	S11_des...	0,0000001	0,0000003	200	1	Secuencial	0:02:00	21/09/2015 23:16:50	21/09/2015 23:22:56	mseijo@it.uc3m.es
73	adasdasf	0:33:20	1	DBPSK FEC ON	S11_des...	0,0000001	0,0000001	200	1	Secuencial	0:02:00	21/09/2015 22:57:28	21/09/2015 22:57:28	mseijo@it.uc3m.es
72	pruebaaaa	0:20:00	1	DBPSK FEC ON	S11_des...	0,0000001	0,0000001	200	1	Secuencial	0:02:00	21/09/2015 22:56:12	21/09/2015 22:56:12	mseijo@it.uc3m.es
71	prueba	0:25:00	1	DBPSK FEC ON	S11_des...	0,0000001	0,0000002	200	1	Secuencial	0:02:00	21/09/2015 22:45:47	21/09/2015 22:45:47	mseijo@it.uc3m.es
70	SimSIM1	0:33:20	1	DBPSK FEC ON	S11_des...	0,0000004	0,0000001	200	1	Secuencial	0:02:00	21/09/2015 22:43:39	21/09/2015 22:43:42	mseijo@it.uc3m.es
69	SimWorks	0:33:20	1	DBPSK FEC ON	S11_ant...	0,0000002	0,0000001	200	5	Secuencial	0:02:00	21/09/2015 22:38:14	21/09/2015 22:38:16	mseijo@it.uc3m.es
68	SIMSIMSIM	1:23:20	1	DBPSK FEC ON	S11_ant...	0,0000002	0,0000002	200	1	Secuencial	0:02:00	21/09/2015 21:59:53	21/09/2015 21:59:56	mseijo@it.uc3m.es
67	adadadasdasdas	2:46:40	1	DBPSK FEC ON	S11_des...	0,0000002	0,0000001	200	1	Secuencial	0:02:00	21/09/2015 17:17:47	21/09/2015 17:17:47	mseijo@it.uc3m.es
66	Sim1	12:20:44	1	DBPSK FEC ON	S11_des...	0,0000004	0,0000004	200	1	Secuencial	0:02:00	21/09/2015 17:08:00	21/09/2015 17:08:00	mseijo@it.uc3m.es
65	simulacion_21_sep	0:50:00	1	DBPSK FEC ON	S11_des...	0,0000009	0,0000003	200	1	Secuencial	0:02:00	21/09/2015 17:04:38	21/09/2015 17:04:38	mseijo@it.uc3m.es
64	aaaa	0:20:00	1	DBPSK FEC ON	S11_des...	0,0000006	0,0000005	200	1	Secuencial	0:02:00	21/09/2015 16:14:35	21/09/2015 16:14:35	mseijo@it.uc3m.es
63	asdafas	0:33:20	1	DBPSK FEC ON	S11_des...	0,0000010	0,0000005	200	1	Secuencial	0:02:00	21/09/2015 16:13:38	21/09/2015 16:13:38	mseijo@it.uc3m.es
62	asdafas	0:33:20	1	DBPSK FEC ON	S11_des...	0,0000010	0,0000005	200	1	Secuencial	0:02:00	21/09/2015 16:13:02	21/09/2015 16:13:02	mseijo@it.uc3m.es

Figura 25: Muestra de la lista de simulaciones solicitadas.

Sobre esta plantilla se pueden acometer pequeños cambios que pueden permitir, por ejemplo, una selección de columnas más restrictiva o que los usuarios sólo puedan ver las simulaciones solicitadas por ellos mismos.

En el caso de que haya más de quince simulaciones mostradas en esta pantalla, se mostrará un sistema de paginado que permite visualizar la lista de quince en quince elementos. Este sistema está ejemplificado en la Figura 26.

Figura 26: Sistema de paginado para la lista de simulaciones.

El número de elementos mostrados por defecto en cada página de esta pantalla puede ser modificado en el controlador asociado a esta plantilla (*SimulacionesView* en el fichero *front/views.py*).

4.1.1.5 simulationrequest_detail.html

Finalmente, esta plantilla permite la visualización de los detalles de la simulación, modificando su estructura dinámicamente desde que fue solicitada hasta que los resultados están disponibles según se solicita en el requisito *FUN09*.

Puede verse un ejemplo de generación de una pantalla para una simulación por esta plantilla en la Figura 27.

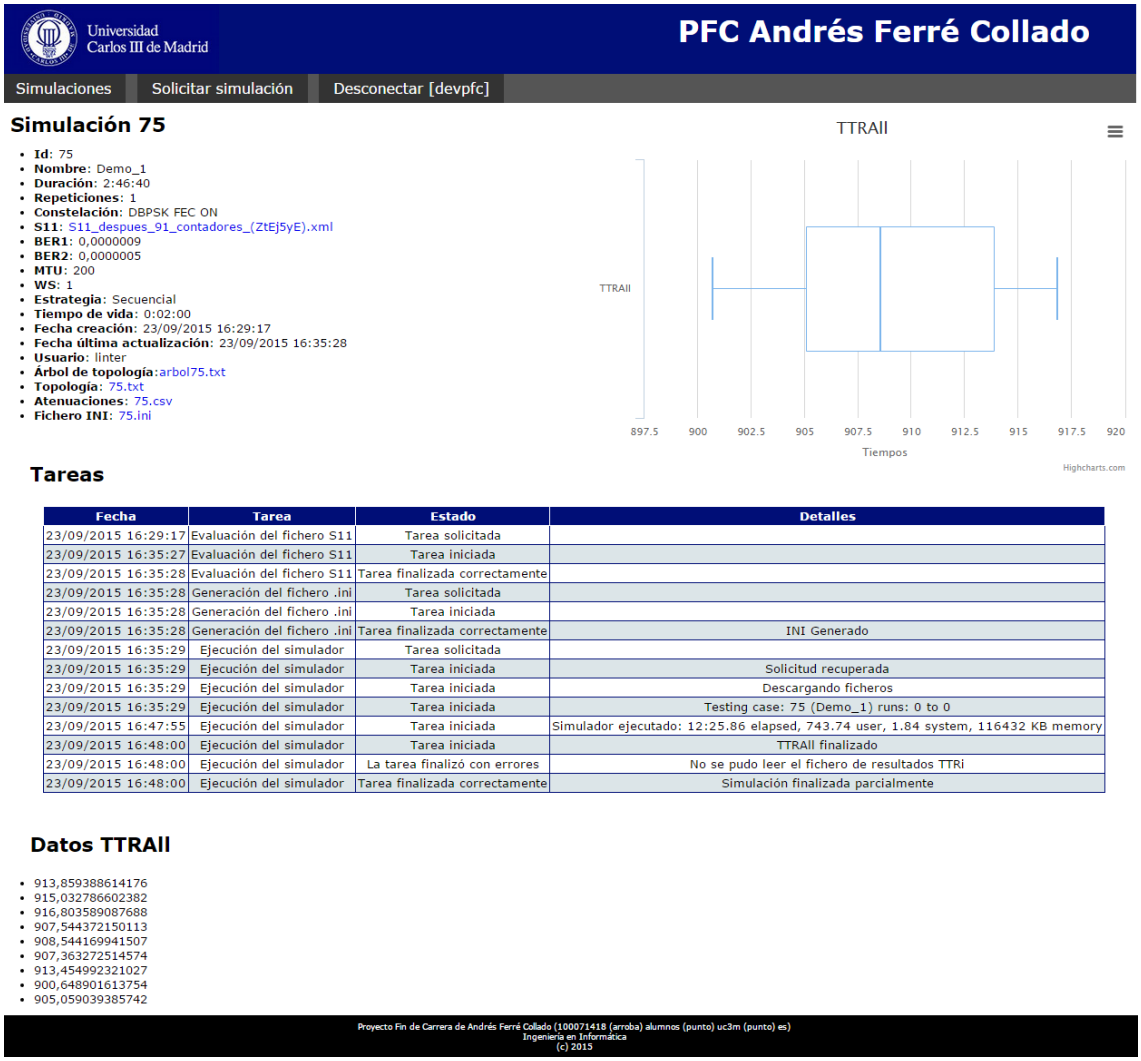


Figura 27: Muestra de la pantalla de detalles de una simulación, con la simulación terminada correctamente.

Dos de las secciones, la de detalles y la de tareas, se muestran sea cual sea el estado de ejecución de la simulación. El resto de apartados únicamente se mostrarán si tienen información disponible que aportar.

DETALLES DE LA SIMULACIÓN

En la sección de detalles, ejemplificada en la Figura 28, se pueden observar todos los datos que se introdujeron en el formulario de solicitud de la simulación. Además, en cuanto están disponibles, se muestran los ficheros que se han ido generando durante el procesamiento de los detalles de la solicitud, tal y como se solicita en el requisito *FUN11*:

- Árbol de topología.
- Topología.

- Atenuaciones.
- Fichero INI.

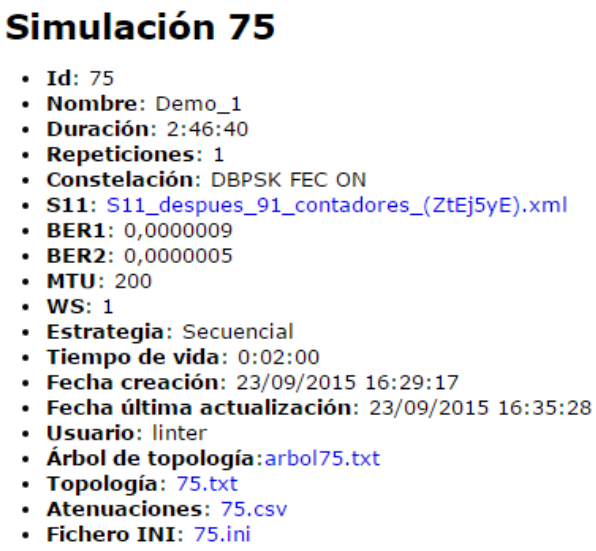
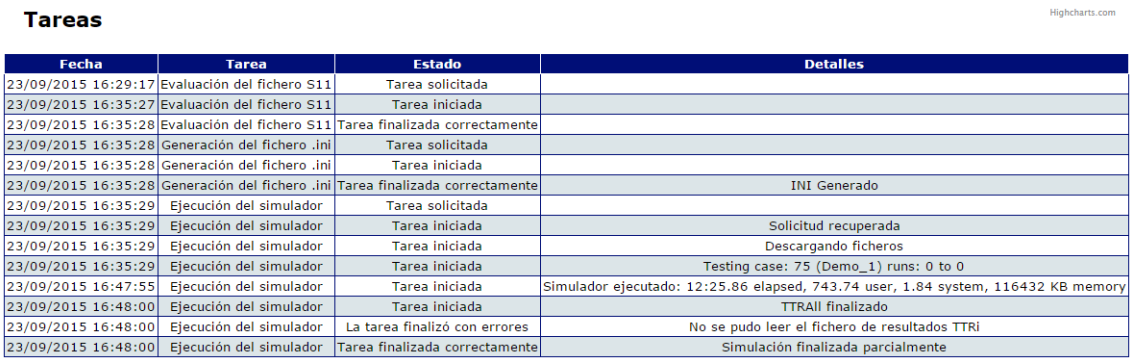


Figura 28: Muestra de la sección de detalles de la simulación.

TAREAS

Esta sección, ejemplificada en la Figura 29, permite visualizar el estado de ejecución de la simulación y algunos detalles sobre la misma, tal y como se solicita en el requisito *FUN12*.

- Fecha: Momento en el que se ha añadido la entrada al registro de tareas.
- Tarea: Tarea a la que se refiere la entrada en el registro. Puede contener los siguientes valores:
  - Evaluación del fichero S11.
  - Generación del fichero .ini.
  - Ejecución del simulador.
- Estado: Indica el estado de ejecución de la tareas. Puede tener los siguientes valores:
  - Tarea solicitada: Se ha lanzado la tarea pero aún no se ha iniciado.
  - Tarea iniciada: La tarea se ha iniciado, pero aún no se ha completado.
  - Tarea finalizada correctamente: La tarea finalizó correctamente.
  - La tarea finalizó con errores: Ha ocurrido algún error durante la ejecución de la tarea. Si después puede verse un mensaje indicando que la tarea se finalizó correctamente, es porque la tarea pudo recuperarse del error.



Tareas

Highcharts.com

Fecha	Tarea	Estado	Detalles
23/09/2015 16:29:17	Evaluación del fichero S11	Tarea solicitada	
23/09/2015 16:35:27	Evaluación del fichero S11	Tarea iniciada	
23/09/2015 16:35:28	Evaluación del fichero S11	Tarea finalizada correctamente	
23/09/2015 16:35:28	Generación del fichero .ini	Tarea solicitada	
23/09/2015 16:35:28	Generación del fichero .ini	Tarea iniciada	
23/09/2015 16:35:28	Generación del fichero .ini	Tarea finalizada correctamente	INI Generado
23/09/2015 16:35:29	Ejecución del simulador	Tarea solicitada	
23/09/2015 16:35:29	Ejecución del simulador	Tarea iniciada	Solicitud recuperada
23/09/2015 16:35:29	Ejecución del simulador	Tarea iniciada	Descargando ficheros
23/09/2015 16:35:29	Ejecución del simulador	Tarea iniciada	Testing case: 75 (Demo_1) runs: 0 to 0
23/09/2015 16:47:55	Ejecución del simulador	Tarea iniciada	Simulador ejecutado: 12:25.86 elapsed, 743.74 user, 1.84 system, 116432 KB memory
23/09/2015 16:48:00	Ejecución del simulador	Tarea iniciada	TTRAll finalizado
23/09/2015 16:48:00	Ejecución del simulador	La tarea finalizó con errores	No se pudo leer el fichero de resultados TTRi
23/09/2015 16:48:00	Ejecución del simulador	Tarea finalizada correctamente	Simulación finalizada parcialmente

Figura 29: Muestra de la sección de tareas de la simulación.

DATOS TTRALL

Esta sección, ejemplificada en la Figura 30, muestra todos los TTRAll de la simulación en bruto, tal y como se solicita en el requisito *FUN13*

Figura 30: Muestra de la sección Datos TTRAll.

TTRALL EN FORMATO GRÁFICO

Esta sección, ejemplificada en la Figura 31, representa, mediante un diagrama de caja y bigotes, los TTR resultantes de la simulación tal y como se solicita en el requisito *FUN13*. Este gráfico ha sido implementado utilizando el API de JavaScript Highcharts.

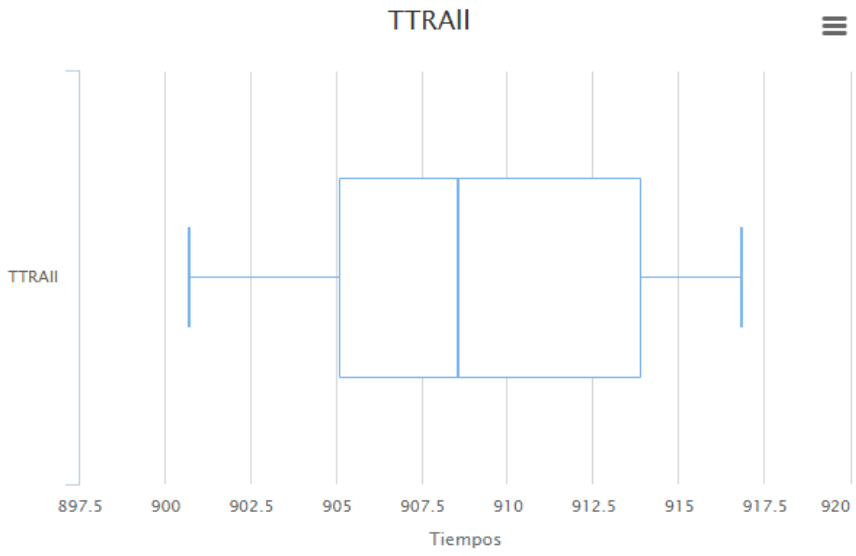


Figura 31: Muestra de la sección TTRAll

Los datos representados son los siguientes:

- TTRAll Máximo
- Cuartil superior
- Mediana
- Cuartil inferior
- TTRAll Mínimo

Puede verse la representación numérica de los datos pasando el ratón por encima como se muestra en la Figura 32.

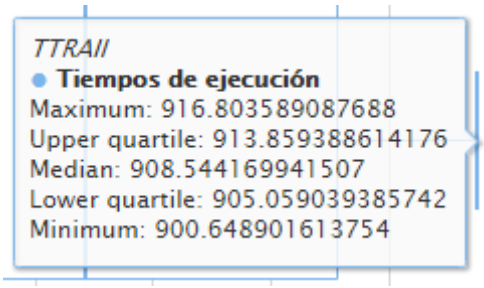


Figura 32: Datos representados en la gráfica TTRAll.

También hay un botón en la esquina superior derecha del gráfico (☰) que muestra un menú de opciones que permite imprimir o descargar la imagen en diversos formatos tal y como se muestra en la Figura 33.

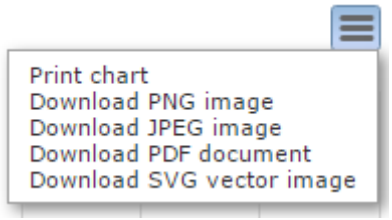


Figura 33: Opciones de la gráfica TTRAll.

4.1.2 Comunicación por correo electrónico al usuario

Tal y como se establece en el requisito *NFUN04*, el sistema deberá avisar al usuario de que una simulación que ha solicitado ha finalizado correctamente. Para ello, cuando el motor finaliza una simulación, encola mediante el uso de Celery una tarea de solicitud de envío de correo electrónico, incluyendo en los parámetros:

- El identificador de la solicitud de simulación.
- Si la simulación ha finalizado correctamente o con errores.
- Un texto adicional opcional que se quiera incluir en el correo electrónico.

El módulo interfaz implementa una función Celery llamada *enviacorreo,* que consume tareas de la cola de envío de mensajes y, mediante la funcionalidad provista por Django para el envío de correo electrónico ([64]), le transmite la información al usuario tal y como se puede ver en la Figura 34.

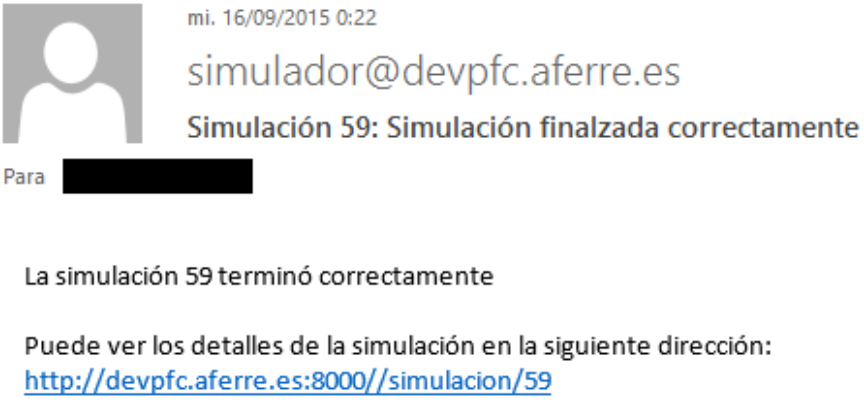


Figura 34: Ejemplo del correo electrónico enviado por el sistema.

4.2 Analizador de ficheros S11

Tal y como se define en el apartado 3.5.1.2, este módulo, implementado con el *framework* Django, se encarga de generar los siguientes ficheros, utilizando para ello un fichero en XML (fichero S11) y los valores BER proporcionados por el usuario:

- Matriz de atenuaciones
- Fichero de topología
- Árbol descriptivo de topología

Para la implementación de este analizador, se ha utilizado como base una implementación en Java del mismo que ha sido proporcionada previamente (ver Anexo III). Por tanto, el trabajo efectuado sobre este motor ha consistido en la optimización y traducción del código a Python 3, la adaptación al modelo de datos utilizado por el sistema y la integración del mismo en el flujo de tareas de Celery.

Este módulo implementa una tarea de Celery para ser ejecutado, que es llamada desde el módulo de la interfaz, con el objeto del modelo *SimulationRequest* correspondiente a la solicitud de simulación como parámetro.

Como pasos previos a la generación de los ficheros, el módulo realiza las siguientes tareas:

- Abrir en memoria tres abstracciones de ficheros (una por cada uno de los ficheros generados).
  El motivo por el cual no se abren ficheros convencionales es que esos ficheros no se guardarán en el sistema de ficheros de la máquina que ejecuta el módulo, sino en la base de datos del sistema.
- Calcular las atenuaciones máxima, media y mínima, tal y como se establece en el requisito *FUN04*.
- Analiza el fichero XML de entrada S11, cargando una lista de nodos, identificados en el fichero por la etiqueta "*macListRegDevices*", utilizando para ello la implementación "*xml.dom.minidom*" de Python.
  Esta implementación permite abstraerse del fichero y del código XML y recuperar los datos del fichero buscándolos por sus etiquetas y nombres de atributo.
- Crea un objeto "Nodo" por cada nodo presente en la lista, incluyendo los siguientes atributos que provee el fichero:
  - *regEntryID*: identificador unívoco y universal del nodo (dirección MAC)
  - *regEntryLNID:* identificador local del nodo. Se trata de un identificador temporal que el concentrador le asigna a un nodo al registrarse en la red
  - *regEntryState:* estado en el que se encuentra el nodo, es decir, si está desregistrado, si está registrado como contador o si ha promocionado a *switch*
  - *regEntryLSID:* identificador local de *switch.* Toma el valor 255 si el nodo no funciona como *switch*
  - *regEntrySID:* identificador del *switch* al que se conecta el nodo en cuestión. Toma el valor '0' si el nodo se comunica directamente con el concentrador

    - *regEntryLevel*: nivel de la topología lógica en el que se encuentra el nodo en cuestión (p.ej., '0' si el nodo se comunica directamente con el concentrador)
- Inserta todos los nodos en una matriz. El nivel en el que se insertarán en la matriz viene definido por el valor del atributo "*regEntryLevel*".

Cuando finaliza la generación de los ficheros, estos se cierran y se guardan en la base de datos, generándose una tarea de Celery para el módulo generador de ficheros de configuración .ini.

A continuación se describe con más detalle la generación de cada uno de los ficheros.

4.2.1 Ficheros de topología

Según los requisitos *FUN03* y *FUN05*, es necesario que este módulo genere dos ficheros de topología distintos. Uno pensado para su lectura por parte del usuario (en lenguaje natural) y otro pensado para su lectura por parte del simulador (con valores separados por plecas y saltos de línea).

Como los datos incluidos en ambos ficheros son prácticamente los mismos, siendo la diferencia el formato en el que se escriben, pueden generarse simultáneamente aprovechando los mismos procesos.

Lo primero que hace este generador es recorrer la matriz de niveles de nodos, escribiendo cada uno que lea en una línea del fichero pensado para su lectura por el simulador, con el siguiente formato (sic):

regEntryLevel|id|regEntryLNID|regEntryLSID|regEntrySID

Después comprueba si es un nodo huérfano, en ese caso lo añade en una cadena de caracteres con el siguiente formato (sic):

Nodo huerfano: id

En caso de no ser un nodo huérfano, se modifica el objeto "*Nodo*" para indicar cuál es su *nodo padre*.

Una vez recorridos todos los nodos, se empieza a generar el fichero de topología en texto natural, empezando por la siguiente cadena de caracteres (sic):

The network logical topology is :
I am node with SID=0

Después, se introducen todos los nodos, anidando los nodos hijos bajo los nodos padre con el uso de tabuladores y plecas para mostrar las relaciones. Excluyendo estos caracteres, cada nodo se representaría con la siguiente cadena (sic):

-> I am node with SID = regEntrySID and LNID = regEntryLNID

Finalmente se introduce en el fichero la cadena de nodos huérfanos que se ha ido generando anteriormente.

4.2.2 Matriz de atenuaciones

Según el requisito *FUN04*, es necesario que este módulo genere una matriz de atenuaciones incluyendo la atenuación correspondiente entre cada pareja de nodos.

Para ello, se recorre la lista de nodos, introduciendo en cada línea la atenuación entre sí mismo y cada uno de los nodos que están en la lista (tal y como se explica en el apartado 3.2.4), según el número de niveles que los distancie, separando cada valor con plecas.

4.3 Generador de ficheros de configuración (.ini)

Tal y como se define en el apartado 3.5.1.3, este módulo, implementado con el *framework* Django, se encarga de generar el fichero .ini de configuración del simulador.

Este módulo implementa una tarea de Celery para ser ejecutado, que es llamada desde el módulo analizador de ficheros S11, con el objeto del modelo *SimulationRequest* correspondiente a la solicitud de simulación como parámetro.

El requisito que define cómo debe realizarse esta tarea es el *FUN02*. A nivel técnico se ha resuelto implementando una plantilla (ver ANEXO II) en la que se incluyen los valores estáticos que deberán incluirse en el fichero resultante (que son la mayoría) y los campos dinámicos con una cadena representando el valor que deberán contener.

Los campos dinámicos y las cadenas identificativas de los valores que deberán contener son los siguientes:

- [Config ID$iden]
- sim-time-limit = $simTime s
- **.endSimTime = $simTime
- **.initialIndComMode = $indComMode
- **.nHostsPerBranch = $topologiaLines
- **.attFileAddress = "$attFile"
- **.archFileAddress = "$topologiaFile"
- **.AppReqMsgSize_Bytes = $AppReqMsgSize
- **.AppResMsgSize_Bytes = $AppResMsgSize
- **.llc.CIMTUSizeBits = $mtu
- **.llc.defaultWindowSize = $ws
- **.appBehavior = $appBehavior
- **.PLC_BN.mac.DEFAULT_ALV_TIME = $keepalive

Por tanto, la tarea consiste en establecer los valores de cada una de las variables mencionadas en la plantilla para, posteriormente, cargar el fichero de la plantilla (*plantilla.ini*) en una cadena de caracteres y, gracias al método *substitute* de la clase *Template* del módulo nativo de Python 3 *string* ([65]), obtener una cadena de caracteres con el contenido final del fichero .ini, incluyendo los valores ya sustituidos.

Finalmente esa cadena se guarda en un fichero, que como en caso de los ficheros generados por el módulo analizador de ficheros S11, no se guarda en el sistema de ficheros de la máquina que ejecuta el módulo sino en la base de datos del sistema.

Hecho esto, se genera una tarea Celery para la ejecución del módulo motor.

4.4 Motor (*Backend)*

Tal y como se define en el apartado 3.5.1.4, este módulo, implementado con el *framework* SQLAlchemy, es el que interactúa directamente con el software de simulación, cumpliendo las siguientes funciones:

- Ejecutar la simulación.
- Procesar y almacenar los resultados.
- Limpiar el entorno del simulador de ficheros no necesarios.

Este módulo implementa una tarea de Celery para ser ejecutado, que es llamada desde el módulo de generación de ficheros .ini, con el número de identificador de la solicitud de simulación como parámetro.

Para llevar a cabo esas funciones, cuya implementación se explica en los siguientes apartados, el módulo requiere la descarga de los siguientes archivos de la base de datos:

- Fichero de configuración .ini.
- Matriz de atenuaciones.
- Fichero de definición de la topología.

Al ser este módulo independiente de los demás y no estar implementado con Django, se ha configurado el *framework* SQLAlchemy indicándole el nombre de las tablas en la base de datos con instrucciones de cargar su estructura automáticamente.

Al finalizar la ejecución del simulador y el almacenamiento de los resultados, se le envía un correo electrónico al usuario mediante una tarea Celery destinada al módulo interfaz, informándole de si la ejecución fue o no exitosa e incluyendo un mensaje de error si es necesario.

4.4.1 Ejecutar la simulación

El módulo deberá ejecutar el simulador tal y como se establece en el requisito *FUN06*.

Teniendo ya disponibles los ficheros necesarios, esta funcionalidad tan sólo tiene que ejecutar el siguiente comando y esperar a su finalización:

*/usr/bin/time -o "results/**IDSIMULACIÓN**-time.txt" -f "%E elapsed, \n%U user, \n%S system, \n%M KB memory" opp_runall -j4 ../src/simPRIME -r 0..**NÚMERO_REPETICIONES**' -f **RUTA_A_FICHERO_INI** -c ID**IDSIMULACIÓN** -G -u Cmdenv -n .:../src*

Después, se comprueba el código de retorno y, sólo en caso de no ser 0, se termina en ese momento la ejecución enviándole al usuario un correo electrónico de error mediante el procedimiento mencionado en el punto anterior.

4.4.2 Procesar y almacenar los resultados

El módulo deberá procesar los resultados del simulador tal y como se establece en el requisito *FUN07*, y almacenarlos tal y como se establece en el requisito *FUN08*.

Teniendo disponibles los datos de salida del simulador, debe ejecutarse el siguiente comando para su procesamiento y posterior almacenamiento:

*scavetool vector -p "name(DLMSCOSEMAppTTRAllMeters)" -O "results/**IDSIMULACIÓN**-**NÚMERO_REPETICIÓN**-TTRAll" -F csv "results/ID**IDSIMULACIÓN**-**NÚMERO_REPETICIÓN**.vec*

Este comando genera un fichero separado por comas que contendrá todos los tiempos de inicio y finalización calculados por el simulador.

El fichero deberá ser leído para calcular todos los valores TTR (según se explica en el apartado 3.1.2) y almacenarlos en la base de datos.

En este momento, tanto si la ejecución es correcta como si no, deberá mandarse un correo electrónico avisando al usuario de la finalización de la simulación y de si ha finalizado correctamente o no. Este correo electrónico se enviará según el procedimiento indicado en el apartado 4.4.

4.4.3 Limpiar el entorno del simulador de ficheros no deseados

Finalmente se ejecutará una limpieza de todos los ficheros utilizados durante la simulación (tanto los descargados como los generados posteriormente). Para ello se ha debido ir creando una lista de todos estos ficheros, según se han ido guardando en el sistema. Al finalizar la ejecución se recorre esa lista de ficheros y se van eliminando uno a uno.

4.5 Puesta en marcha

En este apartado se describe el procedimiento de puesta en marcha del sistema, incluyendo los sistemas adicionales o de apoyo que pueda requerir y la ejecución de las tareas Celery.

El *software* desarrollado para este proyecto viene separado en dos subsistemas:

- Los módulos implementados mediante el *framework* Django:
  - Módulo interfaz.
  - Módulo analizador de ficheros S11.
  - Módulo generador de ficheros de configuración .ini.
- El módulo independiente del motor de ejecución del simulador implementado mediante el *framework* SQLAlchemy.

En los siguientes apartados se describen los componentes necesarios para la ejecución del software y los ajustes necesarios a la configuración del sistema.

Nótese que el subsistema del módulo independiente del motor de ejecución del simulador debe desplegarse en máquinas en las que el simulador de redes PRIME esté instalado y sea operativo. Se puede configurar la ruta en la que está instalado modificando la variable "*RUTA_SIMPRIME*" del fichero de configuración "*config.py*".

4.5.1 Intérprete de Python 3

Para la ejecución del *software* se requiere la instalación de un intérprete de Python 3.X en todas las máquinas.

Para ello se puede acudir a la página oficial de descargas de Python en [66], aunque la mayor parte de distribuciones Linux incluyen un paquete de instalación en sus repositorios. De hecho, las últimas versiones de Ubuntu ya llevan la versión 3 de Python preinstalada.

Según como su sistema operativo haya resuelto el mantenimiento simultáneo de dos ramas de Python (2.7 y 3.X), es posible que los comandos mencionados en este capítulo no funcionen tal cual. Por ejemplo, para ejecutar la herramienta de administración de Django ([67]), según el sistema operativo pueden ser válidos, entre otros, los siguientes comandos:

- ./manage.py <comando> [opciones]
- python manage.py <comando> [opciones]
- python3 manage.py <comando> [opciones]
- python3.X manage.py <comando> [opciones]

Nota: El módulo requerido por los módulos implementados mediante el *framework* Django "*django-db-file-storage*" tiene algunos problemas de compatibilidad con Windows debido a la diferente notación para separar los directorios en los ficheros. Este fallo podría corregirse en futuras versiones, pero hasta ese momento el software no se considera estable en este sistema operativo.

4.5.1.1 Instalación de dependencias

Ambos subsistemas establecen una serie de dependencias de Python que deben ser instaladas. Éstas se incluyen en un fichero llamado *requirements.txt* y puede utilizarse la herramienta *pip* provista por Python para instalarlas. De nuevo, la ejecución de este comando puede ser distinta según el sistema operativo utilizado, siendo los comandos más comunes:

- pip install –r requirements.txt[3]
- pip3 install –r requirements.txt
- pip3.X install –r requirements.txt
- python –m pip install –r requirements.txt
- python3 –m pip install –r requirements.txt
- python3.X –m pip install –r requirements.txt

Asimismo, según el sistema operativo utilizado, puede ser necesaria la instalación de dependencias adicionales del sistema operativo mediante su sistema de repositorios. Se recomienda consultar la documentación si se recibe algún error durante la instalación de dependencias o durante el arranque del *software*.

4.5.2 Servidor de aplicaciones Python o Servidor web

En las máquinas donde se desee ejecutar la interfaz de usuario, deberá estar instalado un servidor web compatible con la especificación *Web Server Gateway* Interface (WSGI).

Pueden seguirse las instrucciones para el despliegue en [68] junto con el manual del servidor web elegido. Django proporciona un manual de despliegue para Apache en [69].

El código que se proporciona es el de la versión de desarrollo. Se deben tener en cuenta algunas cosas antes de desplegar el software en un entorno de producción. La documentación de Django proporciona una lista de comprobaciones para esto en [70].

Únicamente si el sistema va a ser ejecutado en un entorno no productivo, puede ejecutarse tal y como es provisto, utilizando el servidor de aplicaciones de Django:

- ./manage.py runserver [puerto o ip:puerto]

Esto ejecutará un servidor web de desarrollo en la dirección IP y puerto especificados. Si no se especifica ninguna dirección IP, el sistema sólo será accesible desde la interfaz *lookback* (*localhost* o 127.0.0.1). El puerto por defecto es el 8000.

No existe limitación en el número de servidores web o de servidores de aplicaciones Python que puedan ejecutar simultáneamente el *software*, mientras todos ellos tengan acceso a la misma base de datos y al mismo servidor de colas.

[3] En algunos sistemas operativos podría parecer que este comando funciona correctamente. Sin embargo, instalará las dependencias en el entorno de Python 2.7 en lugar de en el de Python 3.X.

El módulo interfaz requiere el envío de correos electrónicos, por ello el servidor SMTP local debe aceptar las peticiones de envío (*relay*) de correo electrónico de Python. Alternativamente, puede configurarse otro servidor SMPT añadiendo las variables indicadas en la documentación en [71] en el fichero "*pfc/settings.py*".

4.5.3 Base de Datos

Deberá haber una máquina que provea un SGDB para el sistema. El modelo de datos fue implementado en el servidor de desarrollo sobre una base de datos PostgreSQL ya que es un SGBD gratuito, transaccional por defecto (MySQL requiere que las bases de datos sean de tipo InnoDB para ello) y los tipos de datos proporcionados por Django tienen una correspondencia casi exacta con los proporcionados por PostgreSQL.

Sin embargo, aunque no todos los sistemas han sido probados, Django proporciona una capa de abstracción que permite cambiar de SGBD de forma transparente para el desarrollo. Así, Django es compatible con los siguientes SSGGBBDD ([72]):

- PostgreSQL
- MySQL
- Oracle
- SQLite
  Nota: Por las características especiales de SQLite, no se recomienda el uso del mismo con esta aplicación ya que los datos se almacenan en un fichero de datos en el disco duro local, limitando esto las posibilidades de escalabilidad del desarrollo, ya que todas las instancias deberían ejecutarse en la misma máquina. La posibilidad de compartir el recurso por NFS no está recomendada por los desarrolladores ya que, según ellos, los bloqueos no funcionan correctamente en muchas implementaciones[4].

El motor utiliza el *framework* SQLAlchemy para conectarse con la base de datos, pero esto no plantea ningún problema de incompatibilidad, ya que también soporta todos los SSGGBBDD anteriores ([73]).

Para que el *software* utilice el SGBD deseado, debe modificarse el fichero de configuración de ambos subsistemas para incluir los detalles necesarios, incluyendo el *host* o la dirección IP de la máquina que ejecute el servicio, el nombre de la base de datos, el usuario y la contraseña.

El fichero de configuración del subsistema que implementa los módulos implementados en Django está en la ruta "*pfc/settings.py*". Deberá modificarse la variable "*DATABASES*".

El fichero de configuración del módulo independiente del motor de ejecución del simulador es "*config.py*". Deberá modificarse la variable "*db*".

[4] https://www.sqlite.org/faq.html

Una vez configurado el SGBD, el modelo debe ser instalado en la base de datos ejecutando el siguiente comando:

- ./manage.py syncdb --all

Si la base de datos va a ser accedida utilizando una red no segura como Internet, se recomienda utilizar conexiones cifradas mediante SSL ([74]) o túneles SSH para las conexiones con la base de datos ([75]). Se recomienda consultar la documentación de su SGBD para ver cómo se hace esto.

4.5.4 Servidor de colas

Deberá haber una máquina que provea un servidor de colas para el funcionamiento del sistema. Aunque técnicamente es posible utilizar otro servidor de colas soportado por Celery, sólo se recomienda el uso de RabbitMQ para la ejecución de este *software*. Se recomienda consultar las guías oficiales de instalación y los enlaces de descarga de RabbitMQ disponibles en [76].

Si el servidor de colas va a ser accedido utilizando una red no segura como Internet, se recomienda utilizar conexiones cifradas mediante SSL ([77]) o túneles SSH para las conexiones con el servidor de colas ([78]). En cualquier caso se recomienda deshabilitar el usuario *guest* y crear otro protegido por contraseña siguiendo el procedimiento de la documentación oficial en [79].

Para que el *software* utilice el servidor de colas deseado, debe modificarse el fichero de configuración de ambos subsistemas para incluir los detalles necesarios, incluyendo el *host* o la dirección IP de la máquina que ejecute el servicio, y en su caso, el usuario y la contraseña.

El fichero de configuración del subsistema que implementa los módulos desarrollados en Django está en la ruta "*pfc/settings.py*". Deberá modificarse la variable "*BROKER_URL*".

El fichero de configuración del módulo independiente del motor de ejecución del simulador es "*config.py*". Deberá modificarse la variable "*BROKER_URL*".

4.5.5 Arranque de los ejecutores Celery

Los ejecutores o *workers* Celery son los procesos encargados de consumir lar tareas que se han enviado al servidor de colas. Tal y como viene configurado el *software* por defecto, se utilizan dos colas: la cola *celery* (nombre por defecto), para los módulos implementados mediante el *framework* Django, y la cola *simulador* para la tarea correspondiente al módulo independiente del motor. De esta forma, un mismo ejecutor es capaz de consumir las tareas de los tres módulos implementados mediante el *framework* Django.

Si se desean tener colas independientes para los módulos implementados mediante el *framework* Django[5], puede seguir la documentación en [80]. Bajo este escenario, es posible desactivar los módulos no necesarios comentando las líneas correspondientes en la variable "*INSTALLED_APPS*" del fichero de configuración "*pfc/settings.py*".

Para iniciar un ejecutor de la cola *celery* en el subsistema implementado en Django, debe ejecutarse este comando:

- celery -A pfc worker

Para iniciar un ejecutor de la cola *simulador* en el subsistema del motor, debe ejecutarse este comando:

- celery -A run worker -Q simulador

Si el comando celery no funciona, debe buscarse el comando adecuado para el sistema operativo, siendo los siguientes los más comunes:

- python3 –m celery
- python3.X –m celery

Si los ejecutores no realizan bien las tareas, es probable que se estén ejecutando sobre un intérprete de Python 2.7. Por lo tanto, se recomienda comprobar que se ejecutan utilizando una versión posterior de Python.

Los comandos proporcionados no hacen que los ejecutores sean iniciados como servicios (lo que es necesario en un entorno productivo). Para hacer que éstos se ejecuten como servicios se puede consultar la documentación de Celery en [81].

No existe limitación en el número de *workers* Celery que se puedan ejecutar simultáneamente, tanto en máquinas distintas como en la misma máquina para aprovechar mejor el *hardware*, mientras todos ellos tengan acceso a la misma base de datos y al mismo servidor de colas.

4.5.6 Usuario administrador y creación de usuarios

Para acceder al sistema se necesitan cuentas de usuario que pueden ser creadas por un administrador. Los administradores también pueden acceder al sistema. Para crear la primera cuenta de administrador se debe ejecutar el siguiente comando y seguir las instrucciones en pantalla:

- manage.py createsuperuser

El resto de usuarios pueden ser creados desde la interfaz de administración de Django, disponible en la siguiente dirección: http://servidor_web:puerto/admin

[5] No debe utilizarse otro nombre que no sea "celery" para la cola de envío de correos electrónicos. De lo contrario, el módulo motor no le haría llegar correctamente las peticiones al módulo interfaz.

Por el mismo motivo, tampoco debe cambiarse el nombre de la cola que consume el módulo motor.

Capítulo 5

Validación

Este capítulo muestra las validaciones realizadas para comprobar el correcto funcionamiento del *software* y las validaciones incorporadas en el formulario de solicitud de simulaciones para verificar que los datos introducidos por el usuario son correctos.

5.1 Validación de los datos introducidos en el formulario de la aplicación

El formulario de solicitud de simulaciones, como es necesario en cualquier sistema de entrada de datos por parte de un usuario o un tercer sistema, contiene una serie de validaciones de los datos para comprobar si todos los campos han sido rellenados y si se ha hecho con el tipo de dato correcto.

La plantilla es capaz de informar al usuario de los contratiempos que ha habido al realizar dichas validaciones, como se muestra en la Figura 35.

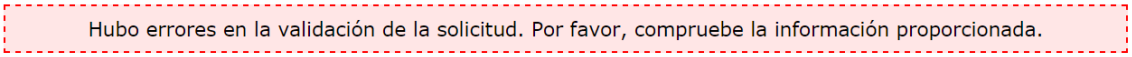


Figura 35: Muestra del aviso de errores de validación

Al ocurrir un error de validación, también se especifica en qué campo del formulario ha sido y el motivo por el que ha fallado como se muestra en la Figura 36.

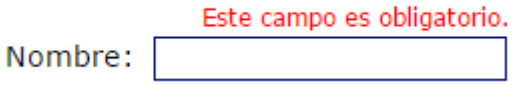


Figura 36: Error de validación. Campo obligatorio.

Duración y Tiempo de Vida son dos campos que requieren una duración de tiempo. Si no se especifica nada, el sistema entenderá que la cifra indicada está expresada en segundos, pero también se puede expresar esta duración de la siguiente forma:

- 01:02:03 (una hora, dos minutos y tres segundos)

Si se introduce un valor que el sistema no reconoce como una cantidad de tiempo, la validación fallará mostrando el mensaje que aparece en la Figura 37.

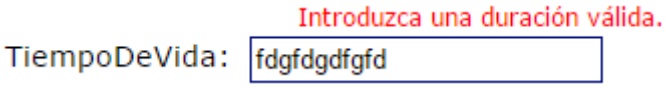


Figura 37: Error de validación. Duración no válida.

Los campos BER sólo admiten números decimales con ocho cifras de precisión (un entero y siete decimales). Sólo se permite la introducción de cifras y los caracteres coma (,) o punto (.), negativo (-) y la letra e (ya que se permite la notación científica, como se muestra en la Figura 38).

Figura 38: Campo con número decimal en notación científica.

Cuando se selecciona el campo, aparecen dos flechas a la derecha que permiten incrementar y reducir el número en 0,0000001 por cada pulsación.

Al permitirse la introducción de caracteres que no son cifras, puede que el usuario introduzca un valor que no sea un número. La validación de este campo se realiza en el

navegador del usuario al presionar el botón de solicitar simulación como se muestra en la Figura 39.

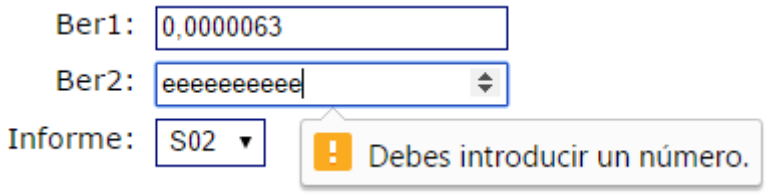


Figura 39: Error de validación. Número no válido.

Cuando los datos introducidos por el usuario pasen la validación con éxito, se mostrará la pantalla con los detalles de la simulación (descrita a continuación) incluyendo el siguiente mensaje indicando que la simulación se ha solicitado correctamente como se muestra en la Figura 40.

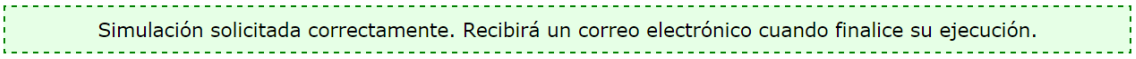


Figura 40: Muestra del mensaje indicando que la simulación se ha solicitado correctamente.

5.2 Validaciones de los módulos

5.2.1 Interfaz (Frontend)

Este módulo es el encargado de proporcionar las siguientes funcionalidades:

- Servirle al usuario una interfaz a través de una página web.
- Enviar un correo electrónico de aviso al usuario que solicitó la simulación.

5.2.1.1 Servir la interfaz de usuario a través de una página web

Para validar esta funcionalidad se ha ejecutado el módulo a través del servidor web de desarrollo que incluye Django y se ha accedido a todas las páginas comprobando que se cumplían los siguientes requisitos:

- FUN01: El sistema deberá proporcionar una interfaz gráfica para el usuario en la que pueda introducir los detalles de la simulación solicitada.
- FUN09: El sistema deberá proporcionar una interfaz gráfica de salida que provea al usuario con los detalles de la simulación en cada momento.
- FUN10: La interfaz de salida con los detalles de la simulación deberá incluir todos los parámetros que el usuario introdujo en la interfaz de entrada.
- FUN11: Los ficheros generados deberán ser accesibles desde la interfaz de salida.
- FUN12: La interfaz de salida con los detalles de la simulación deberá proporcionar una tabla de auditoría que incluya datos sobre el estado de procesamiento de la simulación.
- FUN13: Los resultados TTRAll del simulador deben visualizarse gráficamente en la interfaz de salida con los detalles de la simulación.
- NFUN05: Sólo los usuarios autorizados podrán acceder al sistema.

5.2.1.2 Enviar un correo electrónico de aviso al usuario de la finalización de la simulación.

Esta funcionalidad es requerida por el siguiente requisito:

- NFUN04: El usuario debe ser avisado por correo electrónico al finalizar la simulación.

Para poder validarla, es necesario ejecutar la consola de Django mediante el siguiente comando:

- ./manage.py Shell

En la consola que aparecerá, debe ejecutarse lo siguiente:

- >>> from front.tasks import enviacorreo
- >>> enviacorreo(1, 2, 'Simulación finalizada correctamente')

La simulación con identificador 1 debe existir previamente. El usuario que la creó debería recibir un correo electrónico notificando que la simulación finalizó correctamente.

En caso de error, el problema más frecuente es la configuración errónea del servidor SMTP.

5.2.2 Analizador de ficheros S11

Este módulo es el responsable de analizar el fichero S11 y los valores BER proporcionados por el usuario, para a continuación generar los siguientes ficheros:

- Matriz de atenuaciones
- Fichero de topología
- Árbol descriptivo de topología

Como este código es una traducción de una implementación hecha en Java, la forma más sencilla de validar su funcionamiento es ejecutar la implementación en Java y la implementación en Python con los mismos valores de entrada. Para ello se ha implementado en el módulo de Django un "*método main*" cuyo uso debería ser equivalente. Debe compilarse la implantación de Java y ejecutarse los siguientes comandos:

- ./parseador.py S11 antes de S02 89 contadores.xml sal1 sal2 sal3 1 2 3
- Java ProcessS11 S11 antes de S02 89 contadores.xml jal1 jal2 jal3 1 2 3

Si los ficheros "sal1", "sal2" y "sal3" generados por la implementación del módulo son idénticos a los ficheros "jal1", "jal2" y "jal3" generados por la implementación en Java, la validación es correcta. En el entorno resultante de este PFC, e ha incluido una carpeta llamada "*S11/ejemplos/*" con los resultados de esta validación.

5.2.3 Generador de ficheros de configuración (.ini)

Este módulo es el responsable de generar el fichero de configuración .ini requerido por el simulador. Para su validación se ha rellenado el formulario de solicitud de simulaciones a partir de un fichero .ini de referencia y, una vez ejecutada la tarea, se ha descargado el fichero .ini generado y se ha comparado con el de referencia utilizando para ello la herramienta diff, disponible en sistemas operativos Linux.

5.2.4 Motor (Backend)

Este módulo implementa las siguientes funcionalidades:

- Ejecutar la simulación.
- Procesar y almacenar los resultados.
- Limpiar el entorno del simulador de ficheros no necesarios.

Para su validación se ha implementado un "*método main*" de Python que ejecuta la solicitud de simulación número 59. Para ejecutarlo, debe ejecutarse el siguiente comando:

- ./run.py

Puede cambiarse el número de solicitud de simulación ejecutada en la línea 179 del fichero *run.py*.

Pueden cambiarse las funcionalidades ejecutadas comentando o descomentando las líneas de código 160, 162, 164 y 168.

5.3 Otras validaciones

5.3.1 Validación de la interacción entre componentes

Para validar la interacción entre todos los componentes debe ponerse en marcha el sistema de forma completa tal y como se describe en el apartado 4.5.

El siguiente paso es enviar una solicitud de simulación al sistema a través de la interfaz de usuario. Paralelamente debe ejecutarse una simulación equivalente de forma manual sin usar ninguno de los componentes proporcionados por el sistema.

Una vez finalizadas ambas simulaciones, la validación será correcta si se cumplen las siguientes condiciones:

- Los resultados de ambas simulaciones son equivalentes.
- Los ficheros de configuración .ini son equivalentes.
- Los ficheros de matriz de atenuaciones son equivalentes.
- Los ficheros de topología, tanto en lenguaje natural como formateados para el simulador, son equivalentes.
- Se ha recibido el correo electrónico informando de que la simulación ha finalizado correctamente.
- El enlace incluido en el correo electrónico funciona correctamente y en él se muestran los resultados de la simulación.

Esta prueba se ha realizado con diferentes simulaciones obteniéndose siempre resultados exitosos.

5.3.2 Validación de la escalabilidad

Para validar la escalabilidad, deben seguirse las instrucciones de puesta en marcha del punto 4.5, incluyendo el despliegue de varios servidores web y múltiples ejecutores Celery para cada cola (puede usarse el flag "*-l info*" para ver más detalles sobre las tareas realizadas por cada ejecutor).

Deben solicitarse múltiples simulaciones de forma más o menos simultánea, utilizando para ello al menos dos servidores web distintos. Esto valida que el sistema puede trabajar con múltiples servidores web simultáneamente.

Debe comprobarse la distribución de las tareas en los ejecutores Celery. Todas las tareas deberían ser entregadas si hay ejecutores Celery disponibles escuchando en la cola correspondiente. Todas las tareas deberían poder ejecutarse independientemente de lo que estén haciendo el resto de ejecutores Celery. Esto valida que el sistema puede trabajar ejecutando multitud de tareas de forma simultánea y distribuida.

Actualmente sólo se han llegado a ejecutar dos simulaciones simultáneamente en la misma máquina, por lo que las pruebas de escalabilidad masivas quedan como trabajo futuro una vez se haya migrado la herramienta a un entorno *cloud*.

Capítulo 6

Conclusiones y Trabajos Futuros

Este capítulo resume las principales conclusiones obtenidas tras la ejecución del presente Proyecto Fin de Carrera, además de una serie de posibles mejoras que se podrían realizar en el sistema en un futuro.

6.1 Conclusiones

En el presente Proyecto Fin de Carrera se ha desarrollado una solución de *software* que implementa una interfaz más amigable para el simulador de redes PRIME SimPRIME, facilitando de esta forma su uso por personas sin conocimientos específicos sobre la herramienta e incrementando la productividad de aquellas que sí sabían utilizarlo.

El trabajo ha sido desarrollado dentro del ámbito del proyecto de investigación nacional OSIRIS (Optimización de la Supervisión Inteligente de la Red de Distribución), financiado por el Ministerio de Economía y Competitividad y liderado por Unión Fenosa Distribución (tercera distribuidora eléctrica a nivel nacional). La interfaz web desarrollada ya ha sido presentada a los miembros del consorcio, despertando el interés especialmente de Unión Fenosa Distribución.

El Proyecto Fin de Carrera ha cubierto desde la fase de diseño de la aplicación y selección de plataforma hasta la fase de validación, incluyendo el desarrollo de la misma así como la documentación de su puesta en marcha.

SimPRIME es un sistema de *software* ya existente, basado en Matlab y OMNeT++, dedicado a la simulación de redes PRIME cuyo objetivo es facilitar la planificación, despliegue y operación de este tipo de redes. Este *software* fue provisto sin código fuente y con una serie de ejemplos de uso que debían ser permitidos por la herramienta.

Desde el punto de vista del diseño y de la implementación de la interfaz, no es necesario conocer el funcionamiento interno del sistema sobre el que se está creando dicha interfaz. En ese sentido, se trata de una caja negra. Sin embargo sí es recomendable adquirir una serie de conocimientos técnicos, sobre cómo poner en marcha y cómo utilizar el sistema, así como funcionales, sobre cuál es el uso de la herramienta y una contextualización de los datos de entrada y de salida.

La adquisición de esos conocimientos técnicos y funcionales del sistema resultó especialmente importante para una correcta interpretación de los requisitos del sistema, ya que un malentendido en este punto podría haber llevado al desarrollo de un sistema que se aleje de lo esperado inicialmente.

Esta adquisición de conocimientos puede suponer un reto para cualquier proyecto que, como éste, requiera la creación de una abstracción sobre un sistema de *software* creado por terceras partes, más aún cuando se encuadra en un área de conocimiento completamente nueva para el proyectista.

Hoy en día existen multitud de tecnologías que facilitan la implementación de este tipo de sistemas de *software,* por lo que fue preciso estudiar las alternativas existentes y sus principales características. Una vez conocidas, se tuvieron que comparar para decidir cuáles se ajustaban en mayor medida a las necesidades del proyecto.

Una vez claras las herramientas que se iban a utilizar, y mediante un análisis de los requisitos del sistema, se elaboró un diseño del mismo, en el que se establecían varios componentes modulares, qué tareas y requisitos debían cumplir y cómo debía ser la comunicación entre ellos. Empezar la implementación habiendo hecho un diseño claro, facilitó

mucho la tarea, ya que se partió de una visión global del sistema, y ahorró horas de trabajo *a posteriori*.

Una vez se definió el sistema, el siguiente paso fue desarrollarlo siguiendo lo más fielmente posible dicha definición. A continuación, el sistema desarrollado se validó adecuadamente para comprobar que cumplía con los requisitos solicitados. Asimismo, como parte de este Proyecto Fin de Carrera también se ha documentado la puesta en marcha del sistema en estrecha colaboración con el resto del grupo de investigación donde se ha llevado a cabo el proyecto.

6.2 Trabajos futuros

6.2.1 Implementación de TTRi

Además del TTRAll, que se refiere al tiempo que se tarda en recolectar los consumos de todos los contadores de una red PRIME después de haber realizado una solicitud a tal efecto, el simulador es capaz de proporcionar otros datos como el TTRi, que se refiere al tiempo que se tarda en obtener el dato de consumo de un contador individual.

Actualmente el simulador ya proporciona estos datos. Por lo tanto, la mejora pasaría por realizar el posprocesado también para ellos y, una vez obtenidos en un fichero plano, transferirlos a la base de datos. También habría que modificar la interfaz para mostrar los datos recopilados de forma adecuada.

6.2.2 Mejoras en la validación del formulario

Actualmente el formulario de solicitud de simulaciones sólo valida que los campos introducidos sean correctos en el sentido de que sean del mismo tipo de datos que se esperaba. Sin embargo, se pueden introducir datos que, aunque puedan parecer válidos a primera vista, no producen ninguna salida por parte del simulador, detectándose tan sólo tras la creación de la petición y el preproceso de todos los datos necesarios.

Por ello, además de las validaciones de tipo de datos, sería interesante incorporar algún tipo de validación que comprobara la coherencia de los datos antes de crear la solicitud de información y, en caso de que los datos introducidos no sean coherentes, rechazara la simulación con una breve explicación de los motivos.

6.2.3 Sistema de automatización de la escalabilidad

El sistema desarrollado es escalable en el sentido de que, si es necesario, se pueden desplegar más componentes para procesar una mayor carga de trabajo. Sin embargo, este proceso de añadir recursos (y eliminarlos cuando ya no son necesarios) es completamente manual.

Existen ciertas soluciones en el mercado para controlar el proceso de escalabilidad de forma automática, ampliando y reduciendo los recursos del sistema de forma dinámica. Destacan sobre todo aquellas proporcionadas por proveedores de *clouds*. Estos servicios proporcionan APIs y herramientas para facilitar que, desde el mismo *software,* se detecten picos y valles de demanda que pueden desencadenar en la asignación de los recursos estrictamente necesarios en cada momento.

En este sentido se plantea extender el presente Proyecto Fin de Carrera migrando la aplicación desarrollada a la plataforma FIWARE [82] mediante el desarrollo de un módulo de *software*, situado entre el *frontend* y el simulador, que se encargue de monitorizar los recursos

que se están consumiendo, reservando más recursos conforme el número de simulaciones vaya aumentando y liberándolos conforme vaya disminuyendo.

Referencias

[1] G. López López, «Contribution to Machine-to-Machine Architectures for Smart Grids,» Tesis doctoral, Leganés, 01 2014.

[2] H. Farhangi, «The Path of the Smart Grid,» *IEEE Power and Energy Magazine,* vol. 8, nº 1, pp. 18-28, Enero-Febrero 2010. ISSN: 1540-7977.

[3] G. López, J. I. Moreno, H. Amaris y F. Salazar, «Paving the Road toward Smart Grid through Large-Scale Advanced Metering Infrastructures,» *Electric Power Systems Research,* 2014. DOI: 10.1016/j.epsr.2014.05.006.

[4] «Meters And More |,» [En línea]. Available: http://www.metersandmore.com/. [Último acceso: 17 10 2015].

[5] «PRIME Alliance | Advanced Meter Reading & Smart Grid Standard,» [En línea]. Available: http://www.prime-alliance.org/. [Último acceso: 17 10 2015].

[6] «Narrowband orthogonal frecuency division multiplexing power line comunication transceivers for PRIME networks,» ITU-T standard G.9904, 2012.

[7] M. Seijo, G. López, J. I. Moreno, J. Matanza, F. Martín y C. Martínez, «Herramientas TIC para la mejora del rendimiento y la planificación de redes PLC para Smart Grids,» JITEL 2015.

[8] J. Matanza, «Improvements in the PLC Systems for Smart Grids Environments,» Tesis doctoral, 10 2013.

[9] «OSIRIS: Optimización de la Supervisión Inteligente de la Red de Distribución,» [En línea]. Available: http://www.unionfenosadistribucion.com/es/redes+inteligentes/investigacion+y+desarroll

o/nacionales/1297262412251/osiris.html. [Último acceso: 17 10 2015].

[10] G. Deconinck, «An evaluation of two-way communication means for advanced metering in Flanders (Belgium),» de *IEEE Int. Conf. on Instrumentation and Measurement Technology*, Victoria, Vancouver Island, Canada, 2008.

[11] KEMA Consulting, «Smart Meter Requirements, Dutch Smart Meter specification and tender dossier.,» 02 2008.

[12] A. Ankou, G. Romero y G. Mauri, «Design of the overall system architecture, Tech. Rep. 3.1,» de *OPEN meter Consortium*, 02 02 2010.

[13] T. Schaub, «Powerline Carrier - The basis for advanced metering,» de *19th International Conference on Electricity Distribution*, Viena, 2007.

[14] «Malaga smartcity,» [En línea]. Available: http://www.smartcitymalaga.es/. [Último acceso: 17 10 2015].

[15] A. Aidine, A. Tabone y J. Muller, «Deployment of Power Line Communication by European Utilities in Advanced Metering Infrastructure,» de *IEEE ISPLC 2013*, Johannesburg, South Africa, 2013.

[16] V. Oksman y J. Zhang, «G.hnem: the new itu-t standard on narrowband plc technology,» *Communications Magazine, IEEE,* vol. 49, nº 12, pp. 36-44, 09 2011.

[17] E. Alonso, J. Matanza, C. Rodriguez-Morcillo y S. Alexandres, «A switch promotion algorithm for improving PRIME PLC network latency,» de *18th IEEE International Symposium on Power Line Communications*, 2014 doi:10.1109/ISPLC.2014.6812350.

[18] A. Sendin, I. Urrutia, M. Garai, T. Arzuaga y N. Uribe, «Narrowband PLC for LV Smart Grid Services,» de *18th IEEE International Symposium on Power Line Communications*, 2014 doi:10.1109/ISPLC.2014.6812376.

[19] DLMS User Association, «Excerpt from Compation Specification for Energy Metering - Architecture and Protocols,» 2009.

[20] DLMS User Association, «Excerpt from COSEM - Identification System and Interface Classes,» 2010.

[21] «Riverbed modeler,» [En línea]. Available: http://www.riverbed.com/products/steelcentral/steelcentral-riverbed-modeler.html. [Último acceso: 06 10 2015].

[22] «NS-3,» [En línea]. Available: https://www.nsnam.org/overview/what-is-ns-3/. [Último acceso: 2015 10 06].

[23] A. Varga y R. Horning, «An overview of the OMNeT++ simulation environment,» de *Proceedings of the Simutools08 1st International Conference on Simulation tools and Techniques for Communications, Networks and Systems & Workshops*, Bruselas, 2008.

[24] «Simulador PLC basado en NS3,» [En línea]. Available: http://www.ece.ubc.ca/~faribaa/User_Guide.pdf. [Último acceso: 06 10 2015].

[25] O. G. Hooijen, «A channel model for the residential power circuit used as a digital communications medium,» *IEEE Transactions on Electromagnetic Compatibility,* vol. 40, nº 4, pp. 331-336, 1998.

[26] Oracle, «Differences between Java EE and Java SE - Your First Cup: An Introduction to the Java EE Platform.,» 01 04 2012. [En línea]. Available: https://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html. [Último acceso: 17 10 2015].

[27] Oracle, «Java™ SE Development Kit 7 Update 6 Release Notes,» 13 11 2012. [En línea]. Available: http://www.oracle.com/technetwork/java/javase/7u6-relnotes-1729681.html. [Último acceso: 17 10 2015].

[28] T. Lindholm, F. Yellin, G. Bracha y A. Buckley, «The Java® Virtual Machine Specification. Java

SE 8 Edition.,» 13 02 2015. [En línea]. Available: https://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf. [Último acceso: 17 10 2015].

[29] J. Gosling, B. Joy, G. Steele, G. Bracha y A. Buckley, «The Java® Language Specification. Java SE 8 Edition.,» 13 02 2015. [En línea]. Available: https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf. [Último acceso: 19 10 2015].

[30] «Scala Language Specification,» [En línea]. Available: http://www.scala-lang.org/files/archive/spec/2.11/. [Último acceso: 17 10 2015].

[31] «The Groovy programming language - Groovy reference documentation,» [En línea]. Available: http://www.groovy-lang.org/single-page-documentation.html. [Último acceso: 17 10 2015].

[32] «34. Dynamic language support,» [En línea]. Available: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/dynamic-language.html#dynamic-language-scenarios. [Último acceso: 17 10 2015].

[33] «zeroturnaround.com >> Top 4 Java Web Frameworks Revealed: Real Life Usage Data of Spring MVC, Vaadin, GWT and JSF,» [En línea]. Available: http://zeroturnaround.com/rebellabs/top-4-java-web-frameworks-revealed-real-life-usage-data-of-spring-mvc-vaadin-gwt-and-jsf/. [Último acceso: 17 10 2015].

[34] «21. Web MVC framework,» [En línea]. Available: http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/mvc.html. [Último acceso: 17 10 2015].

[35] «1. Spring Batch Introduction,» [En línea]. Available: http://docs.spring.io/spring-batch/reference/html/spring-batch-intro.html. [Último acceso: 17 10 2015].

[36] «Philosophy,» [En línea]. Available: https://www.playframework.com/documentation/2.4.x/Philosophy. [Último acceso: 17 10 2015].

[37] «What is Python? Executive Summary | Python.org,» [En línea]. Available: https://www.python.org/doc/essays/blurb/. [Último acceso: 17 10 2015].

[38] «Python vs Java | Computer Language Benchmarks Game,» [En línea]. Available: http://benchmarksgame.alioth.debian.org/u64q/python.html. [Último acceso: 17 10 2015].

[39] «Comparing Python to Other Languages | Python.org,» [En línea]. Available: https://www.python.org/doc/essays/comparisons/. [Último acceso: 15 10 2015].

[40] «Python2orPython3 - Python Wiki,» [En línea]. Available: https://wiki.python.org/moin/Python2orPython3. [Último acceso: 17 10 2015].

[41] B. Peterson, «PEP 0373 -- Python 2.7 Release Schedule | Python.org,» 03 11 2008. [En línea]. Available: https://www.python.org/dev/peps/pep-0373/. [Último acceso: 17 10 2015].

[42] «Picking an Interpreter - The Hitchhiker's Guide to Python,» [En línea]. Available: http://docs.python-guide.org/en/latest/starting/which-python/. [Último acceso: 17 10 2015].

[43] «About the Django Software Foundation | Django,» [En línea]. Available: https://www.djangoproject.com/foundation/. [Último acceso: 17 10 2015].

[44] «Django Overview | Django,» [En línea]. Available: https://www.djangoproject.com/start/overview/. [Último acceso: 17 10 2015].

[45] «Django appears to be a MVC framework, but you call the Controller the “view”, and the View the “template”. How come you don’t use the standard names? | FAQ: General | Django documentation | Django,» [En línea]. Available: https://docs.djangoproject.com/en/1.8/faq/general/#django-appears-to-be-a-mvc-

framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names. [Último acceso: 17 10 2015].

[46] «Celery - Distributed Task Queue - Celery 3.1.18 documentation,» [En línea]. Available: http://docs.celeryproject.org/en/latest/. [Último acceso: 17 10 2015].

[47] «Introduction to Celery - Celery 3.1.18 documentation,» [En línea]. Available: http://docs.celeryproject.org/en/latest/getting-started/introduction.html. [Último acceso: 17 10 2015].

[48] «Lenguaje de Programación Ruby,» [En línea]. Available: https://www.ruby-lang.org/es/. [Último acceso: 17 10 2015].

[49] «Getting started with Rails - Ruby on Rails Guides,» [En línea]. Available: http://guides.rubyonrails.org/getting_started.html. [Último acceso: 17 10 2015].

[50] «The Go Project - The Go Programming Language,» [En línea]. Available: https://golang.org/project/. [Último acceso: 17 10 2015].

[51] «Documentation - The Go Programming Language,» [En línea]. Available: https://golang.org/doc/. [Último acceso: 17 10 2015].

[52] «PHP: What is PHP? - Manual,» [En línea]. Available: http://php.net/manual/en/intro-whatis.php. [Último acceso: 17 10 2015].

[53] «MySQL :: MySQL 5.1 Reference Manual :: 1.3.1 What is MySQL?,» [En línea]. Available: https://dev.mysql.com/doc/refman/5.1/en/what-is-mysql.html. [Último acceso: 17 10 2015].

[54] «PostgreSQL: About,» [En línea]. Available: http://www.postgresql.org/about/. [Último acceso: 17 10 2015].

[55] «Introduction to the Oracle Database,» [En línea]. Available: http://docs.oracle.com/cd/B19306_01/server.102/b14220/intro.htm. [Último acceso: 17 10 2015].

[56] «Introduction to Redis – Redis,» [En línea]. Available: http://redis.io/topics/introduction. [Último acceso: 17 10 2015].

[57] «RabbitMQ - Messaging that just works,» [En línea]. Available: http://www.rabbitmq.com/. [Último acceso: 13 10 2015].

[58] «Interactive JavaScript charts for your webpage | Highcharts,» [En línea]. Available: http://www.highcharts.com/. [Último acceso: 17 10 2015].

[59] «Using the Django Database — Celery 3.1.18 documentation,» [En línea]. Available: http://docs.celeryproject.org/en/latest/getting-started/brokers/django.html. [Último acceso: 17 10 2015].

[60] L. Chandnani, «Django Celery: Redis Vs RabbitMQ message broker,» 13 07 2013. [En línea]. Available: http://blog.langoor.mobi/django-celery-redis-vs-rabbitmq-message-broker/. [Último acceso: 17 10 2015].

[61] «User authentication in Django | Django documentation | Django,» [En línea]. Available: https://docs.djangoproject.com/en/1.8/topics/auth/. [Último acceso: 17 10 2015].

[62] «Templates | Django documentation | Django,» [En línea]. Available: https://docs.djangoproject.com/en/1.8/topics/templates/. [Último acceso: 17 10 2015].

[63] «Internationalization and localization | Django documentation | Django,» [En línea]. Available: https://docs.djangoproject.com/en/1.8/topics/i18n/. [Último acceso: 17 10 2015].

[64] «Sending email | Django documentation | Django,» [En línea]. Available: https://docs.djangoproject.com/en/1.8/topics/email/. [Último acceso: 17 10 2015].

[65] «6.1.4 Template strings | 6.1. string — Common string operations — Python 3.5.0

documentation,» [En línea]. Available: https://docs.python.org/3/library/string.html#template-strings. [Último acceso: 17 10 2015].

[66] «Download Python | Python.org,» [En línea]. Available: https://www.python.org/downloads/. [Último acceso: 17 10 2015].

[67] «django-admin and manage.py | Django documentation | Django,» [En línea]. Available: https://docs.djangoproject.com/en/1.8/ref/django-admin/. [Último acceso: 17 10 2015].

[68] «How to deploy with WSGI | Django documentation | Django,» [En línea]. Available: https://docs.djangoproject.com/en/1.8/howto/deployment/wsgi/. [Último acceso: 17 10 2015].

[69] «How to use Django with Apache and mod_wsgi | Django documentation | Django,» [En línea]. Available: https://docs.djangoproject.com/en/1.8/howto/deployment/wsgi/modwsgi/. [Último acceso: 17 10 2015].

[70] «Deployment checklist | Django documentation | Django,» [En línea]. Available: https://docs.djangoproject.com/en/1.8/howto/deployment/checklist/. [Último acceso: 17 10 2015].

[71] «SMTP backend | Sending email | Django documentation | Django,» [En línea]. Available: https://docs.djangoproject.com/en/1.8/topics/email/#smtp-backend. [Último acceso: 17 10 2015].

[72] «Databases | Django documentation | Django,» [En línea]. Available: https://docs.djangoproject.com/en/1.8/ref/databases/. [Último acceso: 17 10 2015].

[73] «Dialects - SQLAlchemy 1.0 Documentation,» [En línea]. Available: http://docs.sqlalchemy.org/en/rel_1_0/dialects/index.html. [Último acceso: 17 10 2015].

[74] «PostgreSQL: Documentation: 9.1: Secure TCP/IP Connections with SSL,» [En línea]. Available: http://www.postgresql.org/docs/9.1/static/ssl-tcp.html. [Último acceso: 17 10 2015].

[75] «PostgreSQL: Documentation: 9.1: Secure TCP/IP Connections with SSH Tunnels,» [En línea]. Available: http://www.postgresql.org/docs/9.1/static/ssh-tunnels.html. [Último acceso: 17 10 2015].

[76] «RabbitMQ - Downloading and Installing RabbitMQ,» [En línea]. Available: https://www.rabbitmq.com/download.html. [Último acceso: 17 10 2015].

[77] «RabbitMQ - SSL Support,» [En línea]. Available: https://www.rabbitmq.com/ssl.html. [Último acceso: 17 10 2015].

[78] B. Chamith, «SSH Tunneling Explained | Source Open,» 21 04 2012. [En línea]. Available: https://chamibuddhika.wordpress.com/2012/03/21/ssh-tunnelling-explained/. [Último acceso: 17 10 2015].

[79] «RabbitMQ - Access Control,» [En línea]. Available: https://www.rabbitmq.com/access-control.html. [Último acceso: 17 10 2015].

[80] «Routing Tasks - Celery 3.1.18 documentation,» [En línea]. Available: http://docs.celeryproject.org/en/latest/userguide/routing.html. [Último acceso: 17 10 2015].

[81] «Running the worker as a daemon - Celery 3.1.18 documentation,» [En línea]. Available: http://docs.celeryproject.org/en/latest/tutorials/daemonizing.html. [Último acceso: 17 10 2015].

[82] «FIWARE,» [En línea]. Available: https://www.fiware.org/. [Último acceso: 17 10 2015].

[83] N. Coghlan, «Python 3 Q & A - Nick Coghlan's Python Notes 1.0 documentation,» 29 06

2012. [En línea]. Available: http://python-notes.curiousefficiency.org/en/latest/python3/questions_and_answers.html. [Último acceso: 17 10 2015].

ANEXO I

Ejemplo de fichero de configuración (.ini) proporcionado

Este anexo pretende mostrar el fichero de configuración para el simulador PRIME que se proporcionó como ejemplo al inicio del proyecto.

```
[General]
## TOPOLOGÍAS
# Boolean to choose between uniform or typical distributions:
**.uniform_d=false
# EAR. variables stored in DLMSCOSEMAppBN/SN.cpp
**.recordDLMSCOSEMAppMsgRoundtripLatency=true
**.recordDLMSCOSEMAppMsgLatency=false
**.recordDLMSCOSEMAppTTRAllMeters=true
# EAR. 04/07/2013. Included to store the info received by
the BN in the PRO_REQ_S message
**.record_PRO_Num_Switches=true
**.record_REG=true
# EAR. 19/09/2013
**.printLog = false
**.BER_SIM= -1.0  # usa los params de las curvas de BER
**.PLC_SN[*].mac.RANDOM_REGISTRATION_NUMBER = 10
**.PLC_SN[*].mac.RANDOM_REGISTRATION = false
**.PLC_SN[*].mac.ALV_OFFSET = 60
**.PLC_SN[*].mac.SWITCH_CAPABILITY = true  # use of SWITCH capability in
SN
#**.PLC_SN[*].mac.PRN_WAIT_TIME = 30 # tiempo de espera para enviar un
mensaje de PRN en caso de no haber recibido ningun beacon
**.mac.PRN_WAIT_TIME = 30 # tiempo de espera para enviar un mensaje de
PRN en caso de no haber recibido ningun beacon
**.interAppMsgTime = 10.0
**.firstAppMsgAt = 1000.0
**.PLC_BN.mac.promotionNode_EUI48 = 10 #
**.PLC_BN.mac.promotionNode_2ndLevel_EUI48 = 25
**.ctrlIndComMode = 0
**.interPromotionTime = 200
**.llc.llcReTxTimer = 10 # Number of seconds for which a LLC entity waits
for acknowledgement of a DATA_PRIM  message
**.PLC_SN[*].mac.allowSwitchIfDBPSK_FEC = true
**.PLC_SN[*].mac.refreshSwitchPower = false
**.PLC_SN[*].mac.numDisconnectionsBeforePRN = 5
**.alvNChances = 3
**.plcBus[*].impulsive = false
**.plcBus[*].nullAttenuation = false
**.sequentialPollingInterval =  9
**.useSequentialPolling = true
**.sendALV = true
**.appTimeOut = 50
**.txQueueLimit = 10000
warmup-period = 5s
repeat = 500
network = PLC_netDLMSCOSEM
**.nBranches = 1

[Config ID]
#~ Simulation time
sim-time-limit = xxx s
**.endSimTime = xxx
```

```
#~ Constellation
**.usePRM = true/false
**.initialIndComMode = x
**.fixedArch = true  # para simular arquitecturas fijas o aleatorias
(Para UC3M)
**.busLength = 400
**.nBranches = 1
**.nHostsPerBranch = x
**.attFileAddress = "ruta/att.txt"
**.archFileAddress = "ruta/topologia.txt"
#~ Requested report (S02/S05)
**.AppReqMsgSize_Bytes = xx
**.AppResMsgSize_Bytes = xxxx
#~ MTU and WS
**.llc.CIMTUSizeBits = xxxx
**.llc.defaultWindowSize = x
#~ Polling strategy Sequential -> 2001 ; Simultaneous -> 2002
**.appBehavior = 2001/2002
#~ Keepalive time
**.PLC_BN.mac.DEFAULT_ALV_TIME = xxx
**.positions = "25 38 37"
```

ANEXO II

Plantilla de fichero de configuración (.ini) generada

Este anexo muestra la plantilla implementada para generar ficheros de configuración .ini, obtenida una vez depurado el fichero .ini proporcionado inicialmente (eliminando líneas vacías y con comentarios e identificando las variables a modificar).

```
[General]
**.uniform_d=false
**.recordDLMSCOSEMAppMsgRoundtripLatency=true
**.recordDLMSCOSEMAppMsgLatency=false
**.recordDLMSCOSEMAppTTRAllMeters=true
**.record_PRO_Num_Switches=true
**.record_REG=true
**.printLog = false
**.BER_SIM= -1.0
**.PLC_SN[*].mac.RANDOM_REGISTRATION_NUMBER = 10
**.PLC_SN[*].mac.RANDOM_REGISTRATION = false
**.PLC_SN[*].mac.ALV_OFFSET = 60
**.PLC_SN[*].mac.SWITCH_CAPABILITY = true
**.mac.PRN_WAIT_TIME = 30
**.interAppMsgTime = 10.0
**.firstAppMsgAt = 1000.0
**.PLC_BN.mac.promotionNode_EUI48 = 10
**.PLC_BN.mac.promotionNode_2ndLevel_EUI48 = 25
**.ctrlIndComMode = 0
**.interPromotionTime = 200
**.llc.llcReTxTimer = 10
**.PLC_SN[*].mac.allowSwitchIfDBPSK_FEC = true
**.PLC_SN[*].mac.refreshSwitchPower = false
**.PLC_SN[*].mac.numDisconnectionsBeforePRN = 5
**.alvNChances = 3
**.plcBus[*].impulsive = false
**.plcBus[*].nullAttenuation = false
**.sequentialPollingInterval =  9
**.useSequentialPolling = true
**.sendALV = true
**.appTimeOut = 50
**.txQueueLimit = 10000
warmup-period = 5s
repeat = 500
network = PLC_netDLMSCOSEM
**.appBehavior = 2001
**.nBranches = 1

[Config ID$iden]
sim-time-limit = $simTime s
**.endSimTime = $simTime
**.usePRM = $usePRM
**.initialIndComMode = $indComMode
**.fixedArch = true
**.busLength = 400
**.nBranches = 1
**.nHostsPerBranch = $topologiaLines
**.attFileAddress = "$attFile"
**.archFileAddress = "$topologiaFile"
**.AppReqMsgSize_Bytes = $AppReqMsgSize
**.AppResMsgSize_Bytes = $AppResMsgSize
**.llc.CIMTUSizeBits = $mtu
**.llc.defaultWindowSize = $ws
**.appBehavior = $appBehavior
**.PLC_BN.mac.DEFAULT_ALV_TIME = $keepalive
**.positions = "25 38 37"
```

ANEXO III

Implementación en Java del analizador del fichero S11

Junto con los requisitos del proyecto, se hizo entrega del código fuente de una implementación en Java del analizador de ficheros S11. Esta implementación consta de dos clases, *ProcessS11* y *Nodo*, que se adjuntan a continuación.

III.1 Nodo

```
import java.util.Vector;
public class Nodo{
    int id, regEntryLNID, regEntryState, regEntryLSID, regEntrySID,
regEntryLevel;
    String regEntryID;
    boolean orphan;
    Vector children;

public Nodo(){
    this.id = 0;
    this.regEntryID = "";
    this.regEntryLNID = 0;
    this.regEntryState = 0;
    this.regEntryLSID = 0;
    this.regEntrySID = 0;
    this.regEntryLevel = 0;
    this.children = new Vector();
    this.orphan =true;
}
public Nodo(int ID, String regEntryID, int regEntryLNID, int
regEntryState, int regEntryLSID, int regEntrySID, int
regEntryLevel){
    this.id = ID;
    this.regEntryID = regEntryID;
    this.regEntryLNID = regEntryLNID;
    this.regEntryState = regEntryState;
    this.regEntryLSID = regEntryLSID;
    this.regEntrySID = regEntrySID;
    this.regEntryLevel = regEntryLevel;
    this.children = new Vector();
    this.orphan = true;
}
public void addChild(Nodo child){
    this.children.add(child);
}
public String printChildren(){
    String nuevalinea = System.getProperty("line.separator");
    String tree;
    String tabulado="\t";
    for(int i=0;i<this.regEntryLevel;i++){
        tabulado = tabulado + "\t";
    }
    tree = tabulado + "|" + nuevalinea;
    tree = tree + tabulado + "|" + nuevalinea;
    tree = tree + tabulado + "-> I am node with SID = " +
this.regEntrySID + " and LNID = " + this.regEntryLNID + nuevalinea;
    if(children.size()!=0){
      for (int i = 0; i < children.size(); i++){
            tree = tree +
((Nodo)this.children.get(i)).printChildren();
        }
    }
    return tree;
}
}
```

III.2 ProcessS11

```
// Imports para el parser
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import java.io.File;
import java.util.Vector;
import java.lang.Math;

// Para escribir el fichero de salida
import java.io.BufferedWriter;
import java.io.FileWriter;

public class ProcessS11 {

  public static void main(String argv[]) {

    try {

    //System.out.println(argv[0]);
    //1st argv -> Input file (XML)
    File fXmlFile = new File(argv[0]);
    DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
    DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
    Document doc = dBuilder.parse(fXmlFile);
    //opcional
    //http://stackoverflow.com/questions/13786607/normalization-in-
dom-parsing-with-java-how-does-it-work
    doc.getDocumentElement().normalize();
    //System.out.println("Root element :" +
doc.getDocumentElement().getNodeName());

    //2nd argv -> Output file 1 (topology file)
    String out_file = argv[1];
    BufferedWriter bw = new BufferedWriter(new
FileWriter(out_file));

    //3rd argv -> Output file 2 (topology tree)
    String out_file2 = argv[2];
    BufferedWriter bw2 = new BufferedWriter(new
FileWriter(out_file2));

    //4rd argv -> Output file 3 (attenuation matrix file)
    String out_file3 = argv[3];
    BufferedWriter bw3 = new BufferedWriter(new
FileWriter(out_file3));
```

```
        //5th, 6th, 7th args -> attenuations
        //5th = the highest (with ourselves and with nodes which are
    two levels away from our level)
        //6th = the lowest (with nodes which are in our level)
        //7th = medium (with nodes which are one level away from our
    level)

        NodeList nList = doc.getElementsByTagName("macListRegDevices");

        //System.out.println("----------------------------");

        for (int temp = 0; temp < nList.getLength(); temp++) {

            Node nNode = nList.item(temp);

            //System.out.println("\nCurrent Element: " +
    nNode.getNodeName());

            if (nNode.getNodeType() == Node.ELEMENT_NODE) {

                Element eElement = (Element) nNode;

                /*System.out.println("MAC: " +
    eElement.getAttribute("regEntryID"));
                System.out.println("Nodo de nivel: " +
    eElement.getAttribute("regEntryLevel"));
                System.out.println("LNID: " +
    eElement.getAttribute("regEntryLNID"));
                System.out.println("LSID: " +
    eElement.getAttribute("regEntryLSID"));
                System.out.println("SID: " +
    eElement.getAttribute("regEntrySID"));*/

            }
        }

        int niveles = 5;
        Vector nivel[] = new Vector[niveles];
        for (int i=0; i<niveles; i++){
            nivel[i] = new Vector();
        }

        for (int temp = 0; temp < nList.getLength(); temp++) {

            Node nNode = nList.item(temp);

            if (nNode.getNodeType() == Node.ELEMENT_NODE) {

                Element eElement = (Element) nNode;

    nivel[Integer.parseInt(eElement.getAttribute("regEntryLevel"))].add
    (new Nodo(0, eElement.getAttribute("regEntryID"),
    Integer.parseInt(eElement.getAttribute("regEntryLNID")),
    Integer.parseInt(eElement.getAttribute("regEntryState")),
    Integer.parseInt(eElement.getAttribute("regEntryLSID")),
    Integer.parseInt(eElement.getAttribute("regEntrySID")),
    Integer.parseInt(eElement.getAttribute("regEntryLevel"))));
            }
        }
```

```
    //Genera fichero de topologia
    int id_nodo=1;
    for (int i=0; i < niveles; i++){
        for (int ii = 0; ii < nivel[i].size(); ii++){
            Nodo nodo_temporal = (Nodo)nivel[i].get(ii);
            nodo_temporal.id = id_nodo++;
            nivel[i].setElementAt(nodo_temporal,ii);
            bw.write(nodo_temporal.regEntryLevel + "|" +
nodo_temporal.id + "|" + nodo_temporal.regEntryLNID + "|" +
nodo_temporal.regEntryLSID + "|" + nodo_temporal.regEntrySID
+"|\n");
        }
    }
    bw.close();

    //Genera fichero de arbol de topologia
        //Asigna nodos hijo a los nodos padre.
    for (int i=1; i < niveles; i++){
        for (int index_child = 0; index_child < nivel[i].size();
index_child++){
            Nodo node_child = (Nodo)nivel[i].get(index_child);

            Nodo node_father;
            for(int index_father = 0; index_father < nivel[i-
1].size(); index_father++){
                node_father = (Nodo)nivel[i-1].get(index_father);

if(node_father.regEntryLSID==node_child.regEntrySID){
                    node_child.orphan=false;
                    nivel[i].setElementAt(node_child,index_child);
                    node_father.addChild(node_child);
                    nivel[i-
1].setElementAt(node_father,index_father);
                }
            }
        }
    }
        //Imprime el arbol de topologia para todos los nodos raiz.
    String nuevalinea = System.getProperty("line.separator");
    bw2.write("The network logical topology is :"+nuevalinea+"I am
node with SID=0"+nuevalinea);
    for(int index_root = 0; index_root < nivel[0].size();
index_root++){
        Nodo node_root = (Nodo)nivel[0].get(index_root);
        bw2.write(node_root.printChildren());
    }

        //Busca e imprime nodos marcados como huérfanos
    for(int i=0;i < niveles;i++){
        for(int j=0;j < nivel[i].size();j++){
            if(((Nodo)nivel[i].get(j)).orphan){
                bw2.write("Nodo huerfano: " +
((Nodo)nivel[i].get(j)).id + "\n");
            }
        }
    }
```

```
        bw2.close();

        nivel[0].add(new Nodo());
        //Genera fichero de matriz de atenuaciones
        for (int level = 0; level < niveles ; level++){
            //System.out.println("Pasando por nivel "+level+"\n");
            for (int pos = 0; pos < nivel[level].size(); pos++){
                //System.out.println("Pasando por la posicion "+pos+"
    del nivel "+level+"\n");
                for (int i=0; i < niveles; i++){
                    //System.out.println("Recorriendo nivel "+i+"\n");
                    for (int j = 0; j < nivel[i].size(); j++){
                        if(level==i){
                            if (j==pos){
                                //System.out.println("Soy yo mismo
    "+argv[4]+"\n");
                                bw3.write(argv[4]+"|");
                            }
                            else{
                                //System.out.println("Es de mi nivel
    "+argv[5]+"\n");
                                bw3.write(argv[5]+"|");
                            }
                        }
                        else if (Math.abs(level-i)==1){
                                //System.out.println("Esta a un nivel
    de distancia "+argv[6]+"\n");
                                bw3.write(argv[6]+"|");
                        }
                        else{
                            //System.out.println("Esta a mas de un
    nivel de distancia "+argv[4]+"\n");
                            bw3.write(argv[4]+"|");
                        }
                    }
                }
                bw3.write("\n");
            }
        }
        bw3.close();

        } catch (Exception e) {
        e.printStackTrace();
        }
      }
    }
```

ANEXO IV

Presupuesto

Este anexo presenta la planificación y fases de desarrollo del Proyecto Fin de Carrera y el presupuesto del mismo, desglosándolo en costes tangibles e intangibles y en costes de personal.

IV.1 Planificación

El presente Proyecto Fin de Carrera se ha dividido en las siguientes tareas:

- Análisis.
- Diseño.
- Desarrollo.
- Validación.
- Documentación.

La Figura 38 ilustra la ejecución del mismo.

Nombre de tarea	Duración	Comienzo	Fin
Aplicación WEB para Automatización y Visualización de Redes PRIME	**72,31 días**	**jue 25/06/15**	**mié 14/10/15**
Análisis	30 horas	jue 25/06/15	jue 09/07/15
Diseño	40 horas	jue 09/07/15	jue 16/07/15
Desarrollo	200 horas	jue 16/07/15	jue 20/08/15
Validación	25 horas	jue 20/08/15	mar 25/08/15
Documentación	250 horas	mar 01/09/15	mié 14/10/15

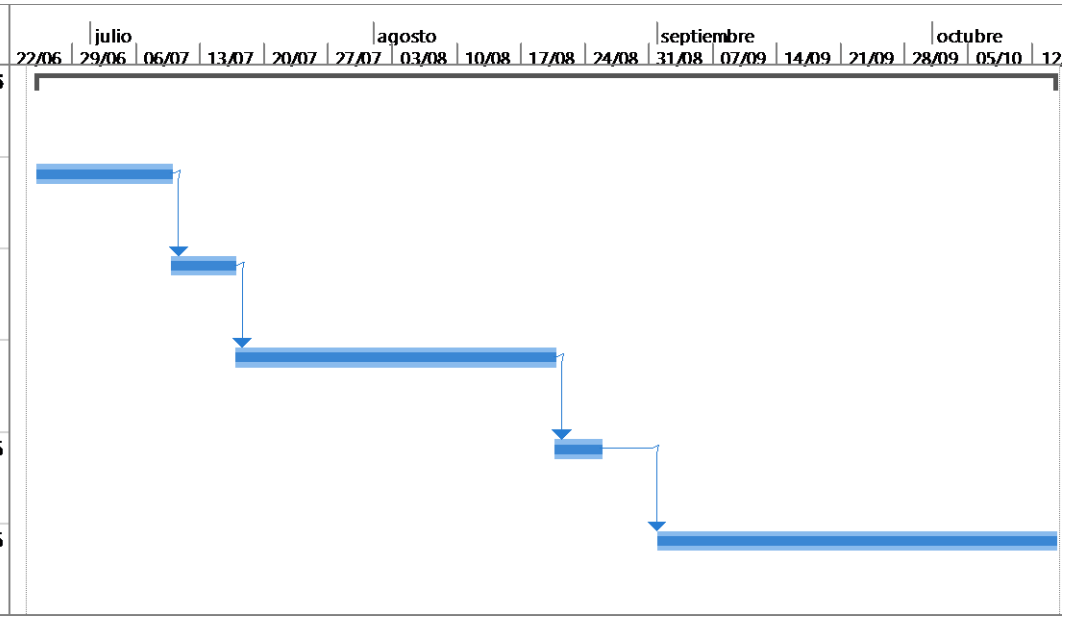


Figura 41: Diagrama de Gantt

IV.2 Recursos y Costes

IV.2.1 Recursos tangibles e intangibles

Para la realización del presente Proyecto Fin de Carrera, se ha requerido del uso de los siguientes recursos:

- Ordenador clónico con un coste de 1000€ (incluye monitor y equipamiento de red).
- Servidor privado virtual con un coste de 3,62€ mensuales.

Como el ordenador tiene un tiempo de amortización esperado de 5 años, y sólo se ha utilizado para el proyecto durante una fracción de ese periodo, su coste debe ser amortizado. También debe tenerse en cuenta que durante la realización del proyecto, el ordenador no estaba dedicado en exclusiva al mismo, por lo que debe imputársele sólo el 100% del tiempo.

$$Amortización = \frac{\frac{1000\ €}{5}}{12\ meses} x\ 4\ meses \cdot 0.5 = 33.33\ €$$

El servidor de desarrollo tiene un coste mensual de 3.67€, que es imputablemente al proyecto durante todo el periodo.

Elemento	Coste unitario	Cantidad	Coste total
Ordenador (meses)	8,33 €	4	33,33 €
Servidor (meses)	3,67 €	4	14,68 €
TOTAL			**48,01 €**

Figura 42: Coste de los recursos tangibles e intangibles.

Las siguientes licencias académicas de *software* han sido proporcionadas de forma gratuita por los distintos fabricantes, siendo el resto del *software* utilizado libre y gratuito:

- Windows 8.1 (y, posteriormente, Windows 10)
- Office 2013
- Lucidchart

Por tanto el coste total de los recursos tangibles e intangibles ha sido de 48,01€ (cuarenta y ocho euros con un céntimo).

IV.2.2 Recursos humanos

A efectos presupuestarios, la cantidad de trabajo se ha medido en horas, sumando un total de 545 horas. El coste por persona y hora establecido por la Universidad Carlos III de Madrid para un ingeniero es de 30€ a la hora, siendo este un “coste de empresa” (el salario bruto del empleado más un 33% aproximadamente en impuestos y otros gastos).

Por tanto, el coste total de los recursos humanos ha sido de dieciséis mil trescientos cincuenta euros.

Tarea	Coste / Trabajo	Trabajo	Coste
Análisis	30 € / hora	30 horas	900,00 €
Diseño		40 horas	1.200,00 €
Desarrollo		200 horas	6.000,00 €
Validación		25 horas	750,00 €
Documentación		250 horas	7.500,00 €
TOTAL		**545 horas**	16.350,00 €

Figura 43: Coste de recursos humanos.

IV.2.3 Total presupuesto

El presupuesto final del proyecto incluye ambos tipos de gastos, por lo que el coste total asciende a dieciseis mil trescientos noventa y ocho euros con un céntimo.

$$Coste\ total = 48{,}01\ € + 16.350{,}00\ € = 16398{,}01\ €$$