UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FINAL DE GRADO

Estudio de arquitecturas software para servicios de Internet de las Cosas

SAMUEL MORENO SAIZ

2015

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS TELEMÁTICOS

TÍTULO: "Estudio de arquitecturas software para servicios de Internet de las Cosas"

Autor: D. Samuel Moreno Saiz
Tutor: D. Juan Carlos Yelmo García
Departamento: Departamento de Ingeniería de Sistemas Telemáticos

Tribunal Calificador:

- Presidente: D. Juan Carlos Yelmo García
- Vocal: D. Miguel Ángel de Miguel Cabello
- Secretario: D. José María del Álamo Ramiro
- Suplente: D. Juan Carlos Dueñas López

Fecha de lectura:

Calificación:

Resumen

A medida que la sociedad avanza, la cantidad de datos almacenados en sistemas de información y procesados por las aplicaciones y servidores software se eleva exponencialmente. Además, las nuevas tecnologías han confiado su desarrollo en la red internacionalmente conectada: Internet. En consecuencia, se han aprovechado las conexiones máquina a máquina (M2M) mediante Internet y se ha desarrollado el concepto de "Internet de las Cosas", red de dispositivos y terminales donde cualquier objeto cotidiano puede establecer conexiones con otros objetos o con un teléfono inteligente mediante los servicios desplegados en dicha red. Sin embargo, estos nuevos datos y eventos se deben procesar en tiempo real y de forma eficaz, para reaccionar ante cualquier situación. Así, las arquitecturas orientadas a eventos solventan la comprensión del intercambio de mensajes en tiempo real. De esta forma, una EDA (*Event-Driven Architecture*) brinda la posibilidad de implementar una arquitectura software con una definición exhaustiva de los mensajes, notificándole al usuario los hechos que han ocurrido a su alrededor y las acciones tomadas al respecto.

Este Trabajo Final de Grado se centra en el estudio de las arquitecturas orientadas a eventos, contrastándolas con el resto de los principales patrones arquitectónicos. Esta comparación se ha efectuado atendiendo a los requisitos no funcionales de cada uno, como, por ejemplo, la seguridad frente a amenazas externas. Asimismo, el objetivo principal es el estudio de las arquitecturas EDA (*Event-Driven Architecture*) y su relación con la red de Internet de las Cosas, que permite a cualquier dispositivo acceder a los servicios desplegados en esa red mediante Internet. El objeto del TFG es observar y verificar las ventajas de esta arquitectura, debido a su carácter de tipo inmediato, mediante el envío y recepción de mensajes en tiempo real y de forma asíncrona. También se ha realizado un estudio del estado del arte de estos patrones de arquitectura software, así como de la red de IoT (*Internet of Things*) y sus servicios.

Por otro lado, junto con este TFG se ha desarrollado una simulación de una EDA completa, con todos sus elementos: productores, consumidores y procesador de eventos complejo, además de la visualización de los datos. Para ensalzar los servicios prestados por la red de IoT y su relación con una arquitectura EDA, se ha implementado una simulación de un servicio personalizado de Tele-asistencia. Esta prueba de concepto ha ayudado a reforzar el aprendizaje y entender con más precisión todo el conocimiento adquirido mediante el estudio teórico de una EDA. Se ha implementado en el lenguaje de programación Java, mediante las soluciones de código abierto RabbitMQ y Esper, ayudando a su unión el estándar AMQP, para completar correctamente la transferencia.

Palabras clave: arquitectura software, patrón, arquitectura dirigida por eventos, EDA, middleware, MOM, , SOA, CEP, procesador de eventos complejo, RabbitMQ, Esper, EPL, AMQP, cola de mensajería, broker de mensajería, Internet de las Cosas, productor, consumidor, Tele-asistencia.

Abstract

As society progresses, the amount of data stored in information systems and processed by different software applications and servers rises exponentially. In addiction, new technologies have trusted their development to the international worldwide network: Internet. In consequence, it has been taken advantage from the Machine-to-Machine connection (M2M) and it has been developed the Internet of Things concept, which means that this network of terminals and devices can allow any everyday object to connect with other objects or a smartphone by the services that have been deployed on this IoT network. However, this new data and events must be processed in real time and effectively, in order to react to any situation. Moreover, event-driven architectures learned comprehension exchange messages in real time. As a matter of fact, an EDA (Event-Driven Architecture) provides the possibility of implementing a software architecture with a comprehensive definition of the messages to notify the user about the group of events that have happened around him/her and tha actions taken with that information.

This Final Project focuses on the study of different architectural patterns, based on non-functional requirements of each one, for example, the security against external threats. Moreover, the main objective is the study of event-driven architectures and its relationship to Internet of Things, that means that any everyday object can access to the IoT network services by the Internet network connecting. The object of this project is to observe and verify the advantages of this type of architecture, because of its immediate character and its capacity to send and receive messages in real time and asynchronously. It has also been included a study of the art state of this type of architectural patterns and Internet of Things (IoT) network and its services.

From another side, with this project it has been developed a simulation of a complete event-driven architecture, with all its elements: producers, consumers and complex processor events, and also how to display the results. To praise the importance of the services provided by the IoT network and its relationship with an EDA architecture, it has been implemented a simulation personalized Telecare service. This proof of concept has helped to learn and understand more accurately and precisely all the concepts acquired by the theoretical study of an EDA. It has been developed with the Java programming language, using the open source code from RabbitMQ and Esper. The standard AMQP has helped to their union, in order to transfer the events.

Keywords: software architectures, standard, event-driven architecture, EDA, middleware, MOM, SOA, CEP, event processor complex, RabbitMQ, Esper, EPL, AMQP, messaging queue, messaging broker, Internet of Things, producer, consumer, Telecare service.

ÍNDICE

CAPÍTULO 1. Introducción 1

1.1. Motivación y descripción del proyecto 1

1.2. Objetivos del TFG 3

1.3. Estructura del documento 5

1.4. Plan de trabajo y metodología seguida 6

CAPÍTULO 2. Estudio general de patrones o estilos de arquitectura del software 8

2.1. Características principales de los patrones generales de arquitecturas software 8

2.2. Dominios de los patrones de arquitectura software 9

2.3. Patrones de arquitectura software más comunes 9

2.3.1. Modelo cliente-servidor 10

2.3.2. Arquitectura orientada a servicios (SOA) 10

2.3.3. Modelo cliente-servidor multinivel 10

2.3.4. Arquitectura orientada a eventos (EDA) 11

2.4. Requisitos no funcionales más característicos 12

2.5. Relación de los requisitos no funcionales con los diferentes patrones arquitectónicos . 12

CAPÍTULO 3. Estudio del arte sobre Internet de las Cosas. Características técnicas 14

3.1. Concepto de Internet de las Cosas: evolución 14

3.2. Características técnicas principales 15

3.3. Servicios desplegados sobre la red de dispositivos de IoT. Ejemplos 16

3.4. Relación de una EDA e Internet de las Cosas 18

CAPÍTULO 4. Estudio del arte de un EDA y su relación con Internet de las Cosas. Elementos principales 19

4.1. Estado del arte sobre tecnologías, herramientas para implementación y despliegue de aplicaciones EDA 19

4.2. Comparativa de una EDA con el modelo cliente-servidor 22

4.3. Capas de una EDA genérica 23

4.4. Generadores y publicadores de eventos 25

4.4.1. Funcionalidad 25

4.4.2. Implementación: RabbitMQ 25

4.4.3. Características principales de RabbitMQ 26

4.5. Canales 27

4.5.1. Funcionalidad 27

4.5.2. Implementación: AMQP 28
4.6. Procesadores de eventos CEP 28
4.6.1. Tipos de procesamiento 29
4.6.2. Implementación: Esper 29
4.6.3. Características de Esper 30
4.7. Visualización de los datos 32
CAPÍTULO 5. Caso de estudio 33
5.1. Principales requisitos 33
5.2. Validación 36
5.3. Principales resultados 38
CAPÍTULO 6. Conclusiones y trabajos futuros 39
6.1. Conclusiones 39
6.2. Trabajos futuros 40
CAPÍTULO 7. Bibliografía y recursos web consultados 41

GLOSARIO

AMQP Advanced Message Queueing Protocol

CEP Complex Event Proccessing

EDA Event-Driven Architecture

EPL Event-Processing Language

HTTP Hypertext Transfer Protool

IoT Internet of Things

JAR Java Archive

JEE Java Platform, Enterprise Edition

JMS Java Message Service

MOM Messaging-Oriented Middleware

M2M Máquina a Máquina (Machine to Machine)

OPT Open Telecom Platform

POJO Plain Old Java Object

S.O. Sistema Operativo

SOA Service-Oriented Architecture

SQL Structure Query Language

TFG Trabajo Final de Grado

CAPÍTULO 1. Introducción

En la actualidad, las formas de representar los objetos de la realidad y cada uno de sus parámetros ha tenido un crecimiento exponencial respecto a unos años atrás. En consecuencia, el patrón de arquitectura software más común para prestar servicios específicos a cada cliente es aquel denominado SOA (*Service Oriented Architecture*). Sin embargo, para cierto tipo de sistemas es conveniente tener en cuenta la información sobre eventos en tiempo real y tomar decisiones y acciones concretas leyendo esa información. Para este tipo de sistemas se ha definido la arquitectura orientada a eventos EDA, objeto de estudio de este trabajo.

Este TFG se ha desarrollado bajo el estudio de los principales patrones arquitectónicos software y su relación con los principales requisitos no funcionales que rige la sociedad, como, por ejemplo, el rendimiento del sistema y la seguridad frente a amenazas externas del mismo. Además, dentro de estos patrones se encuentra el patrón EDA (*Event-Driven Architecture*), el cual procesa de manera lógica y con reglas preestablecidas los eventos proporcionados por dispositivos externos, ofreciendo comunicación en tiempo real y notificando de los principales acontecimientos.

Asimismo, en este proyecto se ha realizado un estudio del estado del arte de este tipo de arquitectura software, además de presentar la relación con el concepto innovador de Internet de las Cosas. También se han detallado las principales características de este nuevo concepto, para encontrar así la relación con una arquitectura orientada a eventos.

Por último, para realizar un estudio más exhaustivo se ha desarrollado una cadena sencilla de una EDA, definiendo cada uno de sus elementos. Éstos se han implementado con RabbitMQ y Esper, detallando posteriormente cada una de las partes de la EDA correspondiente a cada solución de código abierto, junto con sus características más importantes por las que se tomo la decisión de usarlos. Los datos a analizar serán implementados mediante una simulación de objetos con acceso a los servicios desplegados sobre la red de "la Internet de las Cosas", proporcionando un servicio personalizado de Tele-asistencia.

1.1. Motivación y descripción del proyecto

En el ámbito de las arquitecturas software, son múltiples los patrones arquitectónicos definidos para poder solventar diferentes requisitos no funcionales de los sistemas. Sin embargo, uno de los más importantes es aquel referido a la Arquitectura Orientada a Servicios (SOA), en el cual una entidad central proporciona servicios a otros servidores externos para proporcionárselos al usuario, teniendo en cuenta requisitos respecto a implementación, innovación y rendimiento, entre otros.

En este trabajo se ha estudiado la implementación del análisis de la información en tiempo real. Para ello, la EDA es aquella bajo estudio en este trabajo, incorporada junto con la arquitectura SOA en la definición del estándar SOA 2.0. [1] Esta arquitectura surge como ampliación de la SOA, enfocando su implementación en el análisis de información en tiempo real, además de su envío y recepción. Incluso se puede considerar una sucesora de la arquitectura orientada a servicios, ya que los elementos propios de la red de dispositivos de IoT deben poder enviar y recibir su información inmediatamente y de forma asíncrona. Asimismo, también cobra especial importancia el procesamiento inteligente de los eventos que generen los objetos conectados a la red de IoT. Por otro lado, al implementar una EDA y relacionarla con dichos objetos, es importante realizar un estudio detallado, debido a su progresivo crecimiento en la sociedad y la implantación de Internet en objetos de la vida cotidiana.

Para poder respaldar la importancia del patrón de arquitectura orientada a eventos y sus aplicaciones en la actualidad, se describen a continuación las principales ventajas frente a los modelos tradicionales y las diferentes arquitecturas. [2] En el Capítulo 2 se realiza un estudio de los diferentes patrones y estilos de arquitectura software, incluyendo una pequeña comparación con una EDA.

- **Eventos asíncronos**: no es necesario que el receptor, en este caso el consumidor, deba estar en constante percepción de la recepción de un evento. Es decir, el transmisor o generador de eventos no se queda bloqueado una vez manda un evento, no es necesario que le llegue una respuesta por parte del receptor. Este generador sigue enviando eventos sin necesidad de llegar o no al consumidor. Si el consumidor está "activado" los recibirá, si no, se guardarán en una cola de mensajería para ser recibidos por el mismo a posteriori, o, en última instancia, se descarta dicho evento o mensaje. El patrón SOA se diferencia en este ámbito en su utilización de eventos síncronos, cuya respuesta por parte del receptor es necesaria para poder seguir enviando mensajes desde el transmisor.
- **Posibilidad de establecer comunicaciones en tiempo real**, debido a la fluidez de la cadena del sistema completo.
- Se pueden abarcar muchos **eventos simultáneos** con un solo procesador de eventos y un solo canal. Asimismo, también se pueden tener un flujo concreto de eventos del mismo tipo, diferentes tipos o múltiples combinaciones, donde se pueden tener diferentes receptores para poder procesar los diferentes mensajes.
- **Posibilidad de poder reutilizar eventos**, ya que se definen de forma genérica y se pueden llegar a especificar dependiendo del contexto en el que se utilicen. Por ejemplo, si se quiere enviar un mensaje de error, se puede añadir directamente el parámetro de error a dicho mensaje y el procesador lo tratará como tal.

Por otro lado, la arquitectura orientada a eventos tiene una serie de elementos que se van a detallar a lo largo de esta memoria. Estos elementos hacen posible la comunicación de tiempo real entre transmisor y receptor, además de la detección de eventos de elevada importancia. Asimismo, atendiendo a cada uno de los mismos, se

puede advertir como la cadena entera del sistema completo no puede llevar a cabo la visualización, función propia del navegador o aplicación, si alguno de los componentes de dicho sistema falla. En consecuencia, se detallarán los parámetros, en su caso, necesarios para cada elemento del sistema. Los principales componentes son: [3]

- **Productores o publicadores de eventos**: es aquel elemento o elementos que se encargan de generar o recoger los eventos que encuentra en el entorno. Puede ser cualquier aplicación, sensor, dispositivo externo, etc.
- **Canal**: será el medio por el que se transmitan los eventos. Puede ser, por ejemplo, una conexión TCP/IP, entre otros. En el caso del TFG, se simulará un canal por el que los dispositivos externos y sensores, conectados a la red de Internet de las Cosas, enviarán sus datos al procesador alojado en la nube.
- **Consumidores y procesador de evento**: este elemento se encarga de la consumición del evento. Es decir, se trata de un procesador de eventos que lleva a cabo la acción con un evento o eventos concretos. Sin embargo, se debe prestar especial atención a dicho procesador, el cual se encarga de diferenciar entre las acciones a tener en cuenta para cada uno de los eventos que lleguen. De esta forma, se trata de un procesador inteligente, que ejecuta una acción determinada siempre que le llegue un evento concreto o una combinación de los mismos, pudiendo enviar eventos a unos consumidores o consumirlo él mismo.
- **Representación de los datos** y de las acciones tomadas para cada evento: en este último elemento se lleva a cabo la representación de los datos y acciones.

1.2. Objetivos del TFG

Dentro del marco de los diferentes patrones que existen, y, en concreto, la elección del estudio de la arquitectura orientada a eventos y su relación con el concepto de Internet de las Cosas, se han establecido para este TFG una serie de objetivos específicos a cumplir:

- **Estudio general de patrones o estilos de arquitectura del software y su relación con el cumplimiento de requisitos no funcionales.**
  En este epígrafe se detallan los diferentes patrones arquitectónicos de software que se pueden encontrar en la actualidad. Además, se puntualizan las características principales de un arquitectura de software básica, especificando también los diferentes dominios y ámbitos en los que actúan y tienen importancia cada una de estas arquitecturas software.
- **Elaboración de un estudio del arte sobre los servicios desplegados en la red de Internet de las Cosas y sus características técnicas.**
  En este segundo apartado se realizará un estudio exhaustivo de los servicios que proporciona la red de dispositivos y terminales de Internet de las Cosas, detallando sus características principales. Asimismo, también se especifica y define esta nueva

red, aportando múltiples ejemplos que ilustren mucho mejor su aplicación, utilidad y beneficios tangibles para sus usuarios.

- **Estudio del patrón de arquitectura dirigida por eventos (EDA) y su aplicabilidad a servicios de Internet de las Cosas.**
  Una arquitectura orientada a eventos tiene unos elementos concretos que hacen posible el envío, recepción y detección de eventos y anomalías en sus parámetros. Sin embargo, lo que se muestra en este epígrafe es la descomposición en sus diferentes componentes de una EDA completa, atendiendo a sus parámetros más importantes y la funcionalidad de cada elemento. También se ha llevado a cabo una ilustración de los diferentes lenguajes, entornos y herramientas usados para la implementación de dicha cadena.
- **Estudio del estado del arte sobre herramientas, componentes y tecnologías para la implementación y despliegue de aplicaciones EDA.**
  Este objetivo complementa al anterior. Esto es debido a que la elección de las herramientas y tecnologías usadas para la implementación y despliegue de la cadena EDA reside en el estudio exhaustivo de sus diferentes componentes, junto con los lenguajes y herramientas usadas para la implementación. Con este objetivo se abordan y especifican las herramientas, componentes y tecnologías más importantes y desarrolladas hasta la fecha, además de detallar la relación con la implementación y despliegue de una aplicación EDA.
- **Selección y despliegue de un conjunto básico de herramientas para desarrollo y despliegue de aplicaciones EDA.**
  Una vez definida la arquitectura EDA y cada uno de sus elementos, se ha procedido a la selección minuciosa de las herramientas necesarias para poder elaborar una aplicación EDA sencilla. Dicha aplicación cuenta con varios productores o publicadores y un canal por el que llegan los eventos. Además, también tiene un procesador de eventos complejo que decide el envío y las acciones a tomar según la información que le llegue del canal, y varios consumidores o receptores, que reciben los eventos según las reglas preestablecidas del procesador.
- **Validación de la plataforma de desarrollo y despliegue mediante una prueba de concepto básica en el contexto de Internet de las Cosas.**
  Los productores de eventos serán aquellos objetos que tengan acceso a la red de Internet de las Cosas y a los servicios que presta la misma. Para este TFG, se ha llevado a cabo una simulación de dichos objetos, donde se podrá visualizar la secuencia entera y el resultado final de dicho evento. Asimismo, se han implementado diversos escenarios, como, por ejemplo, mostrar el nivel de glucosa en sangre. Concretamente, el sistema proporciona un servicio personalizado de Tele-asistencia, notificando las anomalías en los parámetros medidos del usuario.
- **Redacción de una memoria que cumpla con la normativa de la ETSIT-UPM** e incluya: estado del arte, metodología, principales resultados, validación, conclusiones, trabajo futuro y bibliografía.

1.3. Estructura del documento

La memoria que aquí se presenta tiene unos puntos importantes respecto al conjunto del Trabajo de Fin de Grado, que son los que especifican los diferentes capítulos de la misma. A continuación, se muestra un breve índice y esquema de los puntos a tratar a lo largo de la memoria:

Capítulo 1: Introducción. En este apartado se detalla una visión general del TFG y los puntos más importantes a tratar dentro del mismo, además de sus objetivos y de los diferentes puntos de la estructura del documento.

Capítulo 2: Estudio general de los patrones de arquitectura software. En este epígrafe se especifican los patrones arquitectónicos más relevantes actualmente, junto con sus principales características y funcionamiento. Además, también se expone su relación con los diferentes requisitos no funcionales a los que hacen referencia.

Capítulo 3: Estudio del arte sobre los servicios que proporciona la red de Internet de las Cosas y sus características técnicas. Se especifica y explica el nuevo concepto de Internet de las Cosas y sus principales características, además de su origen y el objetivo de esta red de dispositivos. También se detallan algunos ejemplos que aportan una explicación visual de este concepto que cobra fuerza en la sociedad.

Capítulo 4: Estudio del arte de una EDA y su relación con Internet de las Cosas. En este apartado se expone la información sobre las arquitecturas orientadas a eventos y sus principales cambios a lo largo de su evolución, además de su relación con algunas de las arquitecturas software tradicionales (por ejemplo, el modelo cliente-servidor). Asimismo, se puntualizan las tecnologías, componentes y herramientas usadas y estudiadas para la implementación y despliegue de una EDA. También se detallan los diferentes elementos que componen una EDA, complementándolos con las diferentes tecnologías usadas para este TFG. También se ilustra la relación que tienen este tipo de patrones arquitectónicos y la red de dispositivos y sensores de Internet de las Cosas.

Capítulo 5: Caso de estudio. En este epígrafe se detalla el caso de estudio completo, que presenta una cadena completa de una arquitectura EDA. Se trata de un servicio personalizado de Tele-asistencia, donde unos sensores y dispositivos externos recogen los valores de los parámetros de la salud del paciente. En consecuencia, la EDA consta de varios productores, un canal, un procesador de eventos y varios consumidores.

Capítulo 6: Conclusiones y trabajos futuros. Se ilustran las principales conclusiones una vez terminado el trabajo. También se muestran las posteriores mejoras que se puedan implementar y los trabajos futuros que puede aportar este TFG o sus variaciones dentro del estudio de arquitectura orientada a eventos.

Capítulo 7: Bibliografía y recursos web consultados. Principales páginas web y documentos de referencia para realizar este Trabajo de Fin de Grado.

1.4. Plan de trabajo y metodología seguida

Este Trabajo de Fin de Grado se ha realizado siguiendo una serie de pasos o etapas para cumplir con todos sus objetivos. De esta forma, acorde con estos objetivos propuestos para el mismo, se puede dividir en dos vertientes, una teórica y otra práctica. Se procede en este epígrafe a describir la elaboración tanto de la memoria del trabajo como de la parte práctica y de la implementación del sistema completo bajo estudio.

En primer lugar, se buscó información sobre los diferentes patrones arquitectónicos de software, centrándose en la arquitectura orientada a eventos. Además, también se llevó a cabo un estudio del arte de la red de dispositivitos y objetos físicos de Internet de las Cosas y sus múltiples servicios, teniendo en cuenta la recogida de información de los mismos y del tiempo que conlleva la contrastación de dicha información. Este punto de partida fue crucial para poder aprovechar al máximo las primeras semanas.

En segundo lugar, se llevó a cabo la búsqueda de información del estado del arte de las arquitecturas orientadas a eventos. En consecuencia, este estudio supuso la búsqueda de la relación entre el concepto de Internet de las Cosas con este tipo de arquitecturas. También se tuvo en cuenta la importancia de conocer los principales requisitos no funcionales en los que este tipo de relación es esencial, como, por ejemplo, la posibilidad de enviar varios eventos y que el procesador de eventos complejo sea posible de procesarlos y tomar decisiones al respecto (concurrencia). En esta etapa se especificaron los principales elementos y partes de la cadena de una EDA, pudiendo llevar a cabo el estudio de las mismas en la siguiente.

Una vez definidas las etapas de una EDA, se procedió a detallarlas en profundidad, definiendo sus características concretas. Además, en esta tercera etapa se buscó información sobre las tecnologías y entornos de implementación de los productores, consumidores y procesador de eventos, para poder llevar a cabo la implementación de la cadena entera de una arquitectura orientada a eventos. En consecuencia, se eligieron las soluciones de código abierto RabbitMQ para los productores y Esper para el procesador de eventos y consumidores, además del protocolo AMQP para canal y la unión de los dos anteriores.

La siguiente etapa se centró en el aprendizaje del lenguaje de las tecnologías elegidas anteriormente. Asimismo, una vez se realizaron los diferentes tutoriales de cada una de las soluciones de código abierto [34][35] y se tuvo un conocimiento robusto y suficiente para este TFG, se llevó a cabo la definición de las diferentes partes de una arquitectura orientada a eventos. De esta forma, se definieron los productores según el concepto de Internet de las Cosas y el tipo de objetos, sensores y dispositivos que se pueden beneficiar de los servicios prestados por esta red. Con esto, se adecuó el contexto de este TFG: un servicio personalizado de Tele-asistencia. También se realizaron las diferentes sentencias EPL para poder llevar a cabo acciones específicas

para cada evento. Al mismo tiempo, también se implementó la unión entre ambos, mediante el protocolo de intercambio complejo de eventos AMQP.

Por último, se llevó a cabo la recopilación de los diferentes resultados de la implementación de toda la EDA, teniendo en cuenta cada uno de los eventos simulados y la salida obtenida con cada combinación de eventos. Por otro lado, también se realizó una pequeña valoración de algunos de los trabajos futuros para este TFG y sus variantes respecto al prototipo original y sus características. Para terminar, se redactó la memoria completa según las indicaciones y especificaciones de la Escuela.

A continuación, se muestra un diagrama de Gantt resumen de la metodología seguida y la implementación, búsqueda de información y duración de cada uno de los elementos del plan de trabajo seguido.

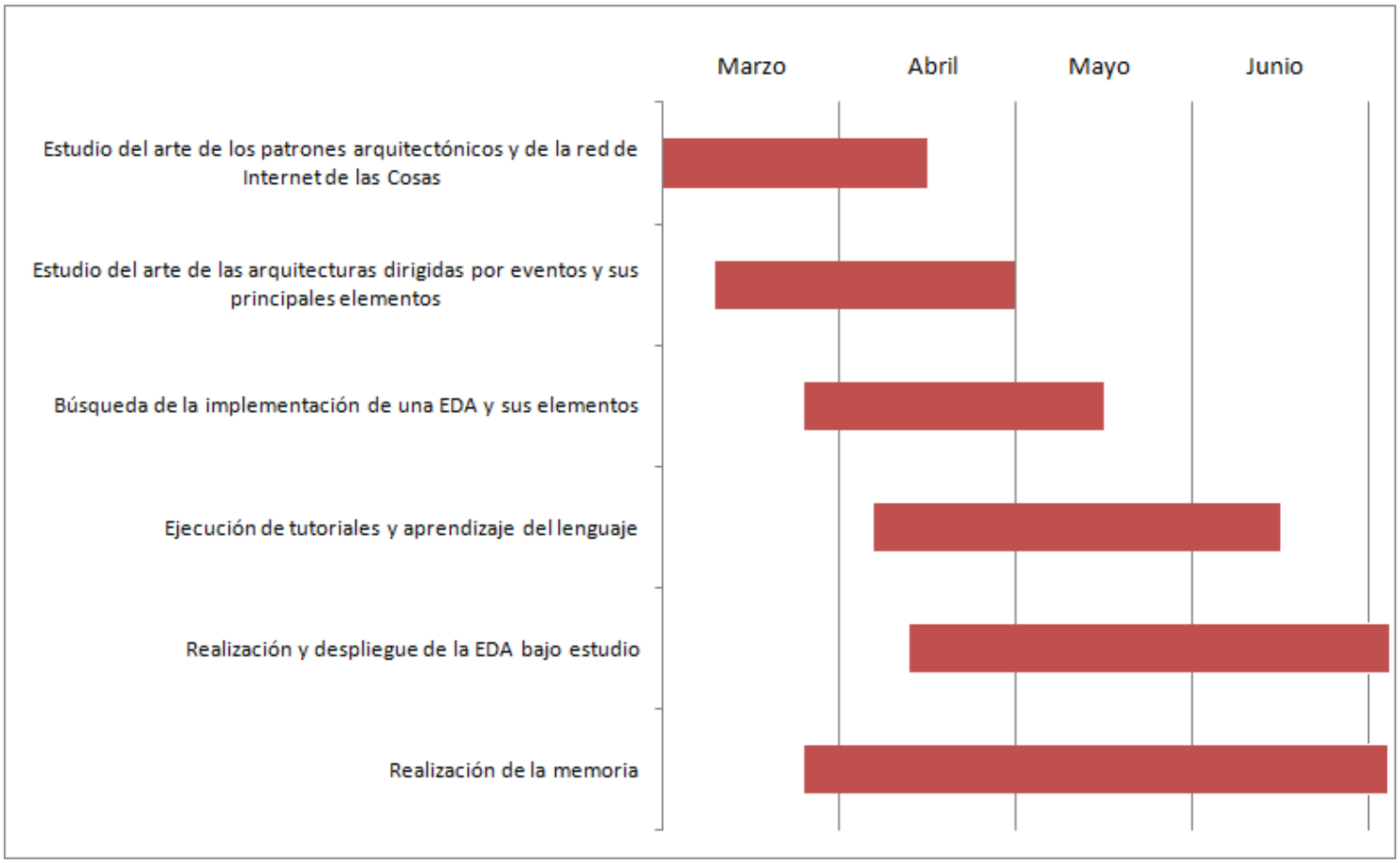


Diagrama de Gantt de la realización del TFG

CAPÍTULO 2. Estudio general de patrones o estilos de arquitectura del software

En la actualidad, muchas aplicaciones software tienen numerosos problemas o dificultades para implementar ciertos requisitos no funcionales, tales como la planificación y repartición de recursos. Esto es debido, por ejemplo, al incremento de la demanda imprevista en poco tiempo de los servicios del sistema o aplicación. Además, igual que en un edificio se necesitan planos y especificaciones para poder empezar a construirlo, en una servicio software es necesario, en su mayoría, un patrón arquitectónico para poder sustentar el proyecto y solventar los problemas derivados de la escalabilidad, entre otros. En consecuencia, los patrones de arquitectura software permiten implementar ciertas pautas de diseño para el sistema en cuestión. De esta forma, tenemos unas guías generales para realizar la arquitectura del sistema con una abstracción mayor que la propia implementación del código y su diseño. Estos patrones proporcionan una descripción de sus elementos y la adecuación de unos requisitos no funcionales concretos, además de la relación entre cada uno de dichos elementos.

2.1. Características principales de los patrones generales de arquitecturas software

Para visualizar mejor estos patrones, se van a detallar las características principales de las arquitecturas software, donde se observa cómo la definición de las mismas es esencial para diseñar e implementar cualquier sistema software: [4]

- La arquitectura software es el **diseño al más alto nivel de la estructura de un sistema**. En consecuencia, los elementos y la relación entre los mismos deben estar definidos en ella, además de los requisitos funcionales y no funcionales, lo que proporciona una visión completa y específica del sistema en su conjunto.
- Sin embargo, estos **requisitos** son los que también ayudan a definir la arquitectura en cuestión. Es decir, por ejemplo, si se quiere implementar una aplicación Android en la que se pida autenticación cada vez que se acceda a ella, la arquitectura estará marcada por esos requisitos de interfaz y de seguridad y confidencialidad. Por tanto, las restricciones y requisitos serán también los que definan los tipos de tecnologías a usar y sus interacciones.
- En la arquitectura software también se especifica la arquitectura física a tener en cuenta en el despliegue de la aplicación o servicio software, ya que el sistema debe tener tareas asignadas en un espacio computacional reservado para tal uso.
- Por último, este tipo de arquitecturas se basan en un conjunto de patrones y abstracciones concretas, proporcionando un "marco" para el sistema.

2.2. Dominios de los patrones de arquitectura software

Los patrones de diseño arquitectónicos proporcionan un marco al sistema, sobre el cual se pueda trabajar y desarrollar cada uno de los elementos del mismo, teniendo en cuenta los requisitos y restricciones que deben tener implementados. Se trata de una captura de los elementos esenciales de la arquitectura software. Un patrón de arquitectura software ayuda a solventar los problemas de calidad de un sistema o arquitectura, pudiendo ser algunos de estos responsables del rendimiento o, incluso, utilizados para tener éxito en la disponibilidad.

En consecuencia, se van a detallar los patrones arquitectónicos más importantes y el factor de calidad al que está encaminado. Sin embargo, antes de llevar a cabo esa descripción, se deben puntualizar los dominios en los que se mueven los diseños de los patrones arquitectónicos de software. Estos dominios son cuatro: [5]

- **Control de acceso.** El control de las personas que acceden al sistema, al software, a las características y/o a la funcionalidad del mismo debe ser restrictivo y preciso a la hora de permitir dicho acceso.
- **Concurrencia.** En muchas aplicaciones y arquitecturas software es esencial manejar múltiples tareas de forma paralela. Cada tarea puede ser representada de muchas maneras, incluyendo la forma de interactuar entre ellas.
- **Distribución.** Para desplegar la aplicación en sistemas distribuidos es necesario definir la forma en la que estos sistemas se comunican entre sí. La forma o patrón más común es el *broker*, donde este elemento se encarga de establecer la conexión. Este *broker* permite el intercambio de mensajes en un sistema donde se tienen diferentes lenguajes, mediante la traducción de dichos mensajes.
- **Persistencia.** Estos datos persistentes son almacenados en bases de datos o archivos en los que el usuario puede modificar o leer.

2.3. Patrones de arquitectura software más comunes

Una vez detallados los dominios de los patrones arquitectónicos, se van a enumerar las principales características de los patrones más importantes, además de su relación con los requisitos no funcionales en los que hace énfasis. [5]

2.3.1. Modelo cliente-servidor

El modelo cliente-servidor es aquel que tiene una parte de cliente y otra de servidor, basando su funcionamiento en el tradicional proceso de petición-respuesta. El primero de ellos manda una petición para la obtención de información en forma de texto HTML, imágenes u otros elementos. Dicha información se aloja en el servidor, cuya misión es valorar dicha petición y decidir qué tipo de información se manda acorde con la petición. El usuario interactúa con la parte cliente, donde éste construye una solicitud del servicio y la envía por la capa del modelo OSI de transporte, normalmente por TCP. El servidor recibe esa solicitud, realiza el servicio requerido y lo devuelve según los formatos ya mencionados. Habitualmente, el servidor puede soportar múltiples conexiones simultáneas de diferentes clientes. [6] [7] [10]

2.3.2. Arquitectura orientada a servicios (SOA)

En el caso de una arquitectura orientada a servicios se crea o realiza un marco para el desarrollo del software y para la implementación del mismo. Este tipo de patrón dictamina unas pautas a seguir por el proveedor o desarrollador del software para proporcionar las restricciones del sistema, con respuestas rápidas y adaptativas a posibles cambios en los requisitos. El proceso reside en un enfoque de composición por orquestación, donde un servidor central reparte los servicios entre diferentes servidores, y éstos a su vez lo llevan hacia sus clientes respectivos. Las conexiones entre el servidor central y el resto de servidores se realiza mediante un proceso de petición-respuesta. al igual que en el modelo cliente-servidor, siendo este la base de una SOA. [8] [9] [10]

2.3.3. Modelo cliente-servidor multinivel

Este patrón es un variante del modelo cliente-servidor, donde el cliente realiza una petición y el servidor responde a dicha petición con la información correspondiente. Sin embargo, la diferencia en este tipo de arquitectura reside en la distribución de una serie de capas, que hacen que sea escalable de forma mucho más eficiente. Estas capas son: **capa de presentación**, aquella que funciona como interfaz gráfica al usuario, capturando sus movimientos y mostrando los resultados de las operaciones correspondientes; **capa de negocio**, donde se alojan los programas ejecutables que llevan a cabo la lógica de toda la aplicación, comunicándose con la capa de presentación para capturar las peticiones del usuario y con la capa de datos para poder acceder e interactuar con los ficheros de la base de datos; y la **capa de datos**, donde se encuentran las bases de datos y sus correspondientes gestores. En este caso, también se lleva a cabo un enfoque de composición por orquestación, siendo la capa de negocio la encargada de coordinar todo el proceso de conexión entre capas y con el cliente. [10] [11]

2.3.4. Arquitectura orientada a eventos (EDA)

Este patrón permite la monitorización, visualización, producción y reacción a un evento. Un evento es el cambio de estado de alguno de los parámetros de un elemento, y en este sistema se realiza una cadena que pueda visualizar este tipo especial de notificaciones. Tenemos cuatro elementos, los cuales se van a detallar en el epígrafe correspondiente a ello: **productores, canal, consumidores y actividad.** El proceso es sencillo: un productor genera un evento y se transmite por el canal hasta llegar al consumidor, el cual puede llevar a cabo alguna lógica especial (procesador de eventos) o puede enviárselo al navegador directamente, para su visualización o descarga. Este tipo de patrón arquitectónico es esencial en los sistemas con necesidad de un alto grado de reacción ante cualquier evento no ordinario. Además, esta arquitectura está muy normalizada para eventos de tipo asíncrono y no predecibles. Por otro lado, el enfoque de composición en este caso reside en la coreografía, donde todos los elementos del sistema recíprocamente con los demás de forma independiente según unas reglas comunes de interacción, gracias al middleware correspondiente. [10] [12] [16]

A continuación se muestra una imagen del modelo cliente-servidor multinivel para Java EE, donde se ilustran todos los niveles mencionados anteriormente.

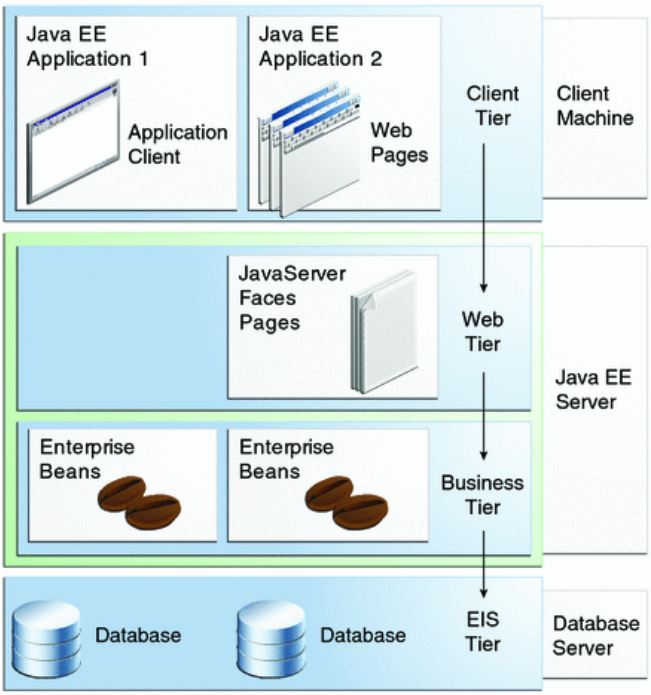


Diagrama del modelo cliente-servidor multinivel. Fuente: [11]

2.4. Requisitos no funcionales más característicos

A continuación, se especifica el ámbito en el que se enfocan cada uno de los requisitos no funcionales más relevantes, además de la importancia para cada elemento del sistema o dispositivo. Asimismo, se muestra una pequeña lista de dichos requisitos no funcionales, aquellos más representativos para los sistemas actuales. [13] [14] [15]

- **Escalabilidad y mantenimiento**. Este requisito es el responsable de poder mantener las características de rendimiento, concurrencia y capacidad del sistema aún cuando éste tenga un número mayor de usuarios o evolucione en cuanto a su infraestructura. Es vital cuando se presenta la situación de una demanda masiva imprevista para proveer con un servicio concreto a los usuarios.
- **Seguridad**. Este requisito trata de evitar o resolver amenazas al sistema que pongan en riesgo la confidencialidad, integridad, autenticación y autorización de la información y datos sensibles (contraseñas...).
- **Pruebas de aceptación y verificación**. Se trata de un requisito importante para conocer el estado de verificación y los principales resultados del sistema. Define los diferentes criterios para verificar el resultado del sistema según los datos o combinación de datos en la entrada del mismo.
- **Documentación**. Este requisito dictamina la incorporación de documentación en el sistema para su posterior ejecución o comprensión por parte del usuario, además de incorporar información técnica relevante.
- **Recursos**. Este requisito se centra en la definición de los límites en cuanto a recursos de almacenamiento, computación y comunicación, como puede ser la memoria en disco o la memoria RAM del sistema.
- **Interoperabilidad.** Cualquier programa o arquitectura software debe poder ser implementada en cualquier sistema operativo.
- **Salvaguarda y replicación de datos**. Para no tener un punto único de fallo, este requisito obliga a tener una réplica de la información en otra base de datos o a tener una replicación del sistema entero en otro entorno.

2.5. Relación de los requisitos no funcionales con los diferentes patrones arquitectónicos

Una vez vistos los requisitos nos funcionales principales, se va a analizar más en profundidad cada uno de ellos según se hayan implementado en los patrones arquitectónicos del software indicados anteriormente.

- **Modelo cliente-servidor.** En el caso del modelo cliente-servidor, el requisito no funcional predominante es la concurrencia, debido a los múltiples usuarios a los que tiene que dar servicio si se produce un crecimiento masivo en la demanda. En consecuencia, la escalabilidad es también un punto muy importante para este modelo, ya que se puede dar tanto en vertical, mejorando las prestaciones de los servicios proporcionados, y horizontal, haciendo que se agreguen múltiples usuarios sin llegar a poner en peligro el rendimiento del sistema. Sin embargo, este modelo tiene problemas en: la replicación del servidor, ya que, si queda inhabilitado, los usuarios se quedan sin servicio, y en la seguridad, porque se deben hacer verificaciones tanto en el cliente como en el servidor.
- **Arquitectura Orientada a Servicios (SOA).** El requisito que más se ajusta a esta arquitectura es la escalabilidad, ya que se trata del aprovisionamiento de un servicio de calidad y concreto al usuario. Por tanto, debe estar preparada para un posible crecimiento de la demanda. Por otro lado, la réplica de datos y su mantenimiento se hacen notables frente al modelo cliente-servidor, ya que, si es necesario, se pueden replicar los servidores ante una posible caída del sistema. Es decir, la prevención sobre proveer a todos los usuarios con el servicio oportuno sin tener en cuenta la posibilidad de una inhabilitación de alguno de los servidores, hace que esta arquitectura tenga un punto importante respecto al mantenimiento del sistema.
- **Modelo cliente-servidor multinivel.** Este modelo se fundamenta en el clásico modelo cliente-servidor, pero aumenta su eficiencia respecto a su escalabilidad y eficiencia, donde puede llevar a cabo el proceso de expansión de forma mucho más óptima que su predecesor. Por tanto, para este patrón el requisito más significativo es la escalabilidad y mantenimiento, debido a su carácter multiusuario y la separación en capas. Sin embargo, también se deben tener en cuenta la centralización de la información, reforzando la seguridad en la capa de datos. Este punto es muy importante, ya que la seguridad de un sistema es vital para su correcto funcionamiento. En consecuencia, mejora las prestaciones del modelo cliente-servidor, pudiendo aislar cualquiera de los módulos en caso de amenaza externa, enfocándose en la protección del nivel de datos e información.
- **Arquitectura Orientada a Eventos (EDA).** Los requisitos no funcionales que se esperan de esta arquitectura se encuentra en la escalabilidad de la cadena de productores, procesador de eventos y consumidores. En consecuencia, se tienen varios eventos procedentes de diferentes productores, por lo que el sistema debe ser capaz de interactuar con múltiples eventos al mismo tiempo. También se incluye la concurrencia de eventos, ya que es primordial para una arquitectura EDA. Es importante destacar que este patrón se caracteriza por enviar y recibir eventos de tipo asíncrono, objeto diferenciador en el resto de patrones. De esta forma, la escalabilidad y concurrencia del sistema será vital, además de las pruebas de verificación y validación del resultado del envío y recepción de eventos, y de la documentación de este tipo de eventos, para informar al usuario de todo lo acontecido en el sistema, junto con su funcionamiento básico.

CAPÍTULO 3. Estudio del arte sobre Internet de las Cosas. Características técnicas

En la actualidad, los teléfonos móviles inteligentes, los televisores de plasma y los terminales táctiles (tablets), cuentan con conexión a Internet. Este suceso ha hecho que hayan aumentado su número de unidades rápidamente, debido a la inversión en innovación y en la forma de presentación de la información, además del auge de las redes sociales. Sin embargo, estas últimas, por ejemplo, no aportan un beneficio real al usuario, no generan un bienestar social en su rutina, por lo que lo que se necesita son la prestación de servicios convencionales unida a los continuos avances de la tecnología. En consecuencia, la investigación ha dado un paso más allá y ha decidido invertir sus esfuerzos en el desarrollo de una nueva red que preste servicios de Internet a objetos de la vida cotidiana, como puede ser la mayor parte de los objetos que usamos en nuestra rutina, como, por ejemplo, una cafetera, el frigorífico o, incluso, las camas de un hospital. A continuación, se va a describir el proceso, aún en investigación, para poder implementar Internet en objetos cotidianos, además de una breve lista de ejemplos y sus beneficios para las personas, que son el eslabón de la cadena sobre la que recae el peso de dictaminar las tendencias de su propia sociedad. [17]

3.1. Concepto de Internet de las Cosas: evolución

El proceso consiste en tener acceso a Internet para comunicar objetos entre sí. A priori, puede resultar sencillo implementar ese tipo de conexión en un frigorífico, por ejemplo, pero no resultó así, ya que se tuvo que definir un nuevo nivel o capa física de Internet. La red de IoT está basada en la comunicación M2M (Máquina a Máquina), donde se establecen protocolos y estándares para poder comunicar máquinas cercanas entre ellas mediante Internet, además de desplegar una serie de servicios en esa red interconexionada entre los diferentes dispositivos. Esa capa de baja velocidad descrita, que se desarrolló en el Centro de Bits y Átomos del MIT (Instituto Tecnológica de Massachusetts) alrededor del año 2000, proporciona una transmisión lenta de datos, pero manteniendo una conexión a Internet casi de forma constante. De esta manera, ese nivel se definió para poder asignar direcciones IP a cualquier objeto, que es justo lo que se intenta desarrollar con este nuevo servicio. La nueva capa definida pasó a llamarse Internet 0, para poder distinguirla de internet de alta velocidad o Internet 2. Internet 0, en su desarrollo, debió lidiar con algunos requisitos esenciales, además de tener en cuenta temas de seguridad y confidencialidad de datos, así como los destinatarios a los que llega la conexión. Esta nueva capa tiene una función clave: proporcionar conexión a Internet a baja velocidad a cualquier dispositivo, casi constantemente, función que resulta esencial en los servicios proporcionados por la red de IoT. [18] [19]

3.2. Características técnicas principales

- **Características de implementación.** Internet 0 se diseñó como un puerto serie transmitiendo y recibiendo a una velocidad de 9.600 símbolos por segundo, mandando los datos según el sistema binario. Este sistema se basa en la composición de "uno" y "cero" mediante un conjunto de bytes, formados por la combinación de los diferentes bits del sistema. Con esto, se forman una serie de códigos concretos, que se transmiten con la modulación pulso-posición. Sin embargo, para separar cada uno de los paquetes enviados y sincronizar cada dispositivo se usa el doble-pulso, por lo que la velocidad se reduce y eso permite que se envíen esos mensajes a través del puerto serie del dispositivo con el protocolo TCP/IP. Las direcciones IP de los mensajes se leen en cada dispositivo y se descartan los que nos correspondan a ese envío concreto.
- **Requisitos en los protocolos.** Los protocolos de Internet IP (*Internet Protocol*) y SLIP (*Serial Line Internet Protocol*), junto con los protocolos de transporte TCP (*Transmission Control Protocol*) y UDP (*User Datagram Protocol*), son necesarios para la implementación de este servicio en los diferentes dispositivos. Se necesita tener un mínimo de códigos para mantener una baja velocidad de la conexión, por lo que en la red local Internet 0 se aloja un pequeño traductor de estos códigos, llegando así el mensaje al puerto serie del dispositivo, donde éste actúa como cortafuegos y los manda a cada uno de los protocolos descritos. En consecuencia, estos protocolos deberán tener bien definida el acceso a la red y el descarte o no de los diferentes mensajes.
- **Difusión y objetos destinatarios: alcance.** La red Internet 0 se diseñó con una baja velocidad para poder conectar cualquier dispositivo a internet. Asimismo, se trata de una red de espacio abierto, donde cualquier dispositivo se puede conectar a ella, sincronizándose y configurando su puerto serie según proceda. Además, también se debe tener en cuenta que estos dispositivos se auto-organizan de manera independiente, por lo que son ellos mismos los que eligen el alcance y a que objetos quieren llegar, aunque la red les proporcione la posibilidad de llegar a todos los terminales dentro de su rango.
- **Seguridad y encriptación.** Un punto importante en este tipo de conexiones es la seguridad de las mismas. Internet 0 utiliza el sistema de encriptación tradicional, que resulta eficaz de forma teórica, pero se pueden encontrar algunos fallos que deben ser solventados. La asignación de direcciones IP y dispositivo y la inicialización de las claves cifradas deben ser algo estrictamente confidencial de cada dispositivo. Es decir, por ejemplo, si se tiene la puerta de un edificio informatizada para abrirla y cerrarla con este servicio, un hacker puede llegar a tomar control de la misma si las medidas de encriptación son descubiertas por no ser lo suficientemente robustas. Este tipo de situaciones son bastante sensibles a los usuarios porque no somos todavía conscientes de lo que Internet supondría en los objetos cotidianos, por lo que constituye un punto importante de investigación y desarrollo. [15] [21]

Con todos esos puntos especificados, se debe centrar la atención en los **servicios** que ofrece la red de dispositivos y sensores de Internet de las Cosas. Asimismo, también se detallan los puntos comunes con la arquitectura orientada a eventos, eje de este TFG.

3.3. Servicios desplegados sobre la red de dispositivos de IoT. Ejemplos

Internet de las Cosas fue desarrollado para generar un beneficio práctico y empírico para los usuarios, mediante la interconexión de los objetos cotidianos con una red proveedora de servicios, como, por ejemplo, la realización de la compra online mediante un frigorífico "inteligente". Estos beneficios desembocan en una revolución tecnológica, donde todo dispositivo tendría conexión a Internet y se podría controlar cualquiera de sus parámetros desde nuestro teléfono inteligente o tablet.

Además, la arquitectura de este tipo de conexión está muy ligado a la arquitectura orientada a eventos. Concretamente, las similitudes se pueden encontrar en el envío de mensajes como notificaciones, es decir, si desde un Smartphone se quiere disminuir la intensidad de la lámpara del salón, se traduce como un cambio de estado de ese parámetro y se trata como un evento. Se puede observar cómo se tendría un productor o publicador del evento (el usuario) y un consumidor (la lámpara).

Por otro lado, este tipo de conexiones están diseñadas para llegar a cualquier dispositivo, por lo que su alcance es muy amplio. A continuación, se muestran algunos ejemplos de dispositivos que usarán esta red o ya tienen acceso a sus servicios. [20]

- Pulseras que cuando se salga a correr se informe de las constantes vitales del usuario, algo muy importante para deportistas profesionales o para personas que necesiten saber sus parámetros médicos en cada momento.
- Conocer en tiempo real tus facultades médicas para que las conozca el servicio de sanidad y emergencias. Este tipo de pulseras o dispositivos externos (sensores, por ejemplo) son muy útiles para personas mayores, ya que pueden sufrir algún tipo de anomalía propia de su edad, y para personas enfermas que pueden sufrir algún tipo de desmayo, como, por ejemplo, personas con diabetes.
- Poder pagar con el teléfono móvil inteligente, mediante los dispositivos iBeacon. Este tipo de dispositivos detectan a otros mediante la detección de su presencia por proximidad. Por tanto, se puede pagar, por ejemplo, la compra realizada en el supermercado sin llegar a tener que sacar el dinero en efectivo ni la tarjeta. Esto genera un conflicto de confidencialidad de datos y de permisos de cuánto dinero debe sustraer el supermercado de la cuenta, pero son temas todavía en vías de desarrollo e investigación.
- Encender un electrodoméstico antes de llegar a casa, ya que enciende el horno unos minutos antes de llegar y solamente se introduciría la comida en el mismo.

- Un cepillo que detecte caries y pida cita automáticamente al dentista, con la base del cepillo como un sensor de anomalías en los dientes, dentro de un rango predeterminado, y la extensión de los filamentos del cepillo para detectarlas.
- Conocer en tiempo real las necesidades de las plantas o del animal en el hogar. Se instalan sensores en la maceta de las plantas para verificar los nutrientes que tienen, mientras que para los animales se puede poner un sensor en el collar.
- Frigorífico "inteligente". En este tipo de frigorífico está diseñado para poder realizar dos funciones: por un lado, el recuento del número de alimentos o productos, además de su marca y preferencia, para poder realizar una compra rápida a través de Internet; por otro lado, el producto puede avisarnos de su caducidad inmediata o dentro de un corto periodo de tiempo.

En este TFG se han implementado una serie de objetos con acceso a Internet, realizando una simulación de los mismos. Concretamente se ha implementado un servicio personalizado de Tele-asistencia, midiendo los parámetros médicos más relevantes de un paciente, como la tensión arterial y el ritmo cardíaco, además de eventos especiales como la caída del paciente por un nivel bajo de glucosa en sangre. Esos eventos se mandan al procesador de eventos CEP de Esper, junto con la hora y la fecha en ese momento concreto del envío.

Fuente de la imagen: [22]

3.4. Relación de una EDA e Internet de las Cosas

En primer lugar, es necesario poner de manifiesto la importancia de una EDA dentro de los servicios desplegados sobre la red de Internet de las Cosas y los cambios en los parámetros de cada uno de los sensores o dispositivos conectados a esta red.

La red de dispositivos y sensores de IoT se basa en la conexión mediante la capa física de baja velocidad denominada Internet 0, donde todos los elementos y objetos tengan la posibilidad de tener conexión a Internet pueden transmitir información a baja velocidad. De esta forma, los objetos con acceso a Internet, como, por ejemplo, un frigorífico, una lavadora o cualquier sensor externo (temperatura, humedad, gases nocivos...), podrán acceder a esta red para el intercambio mutuo de información.

Para poder realizar esos intercambios de datos se deben establecer unos criterios concretos para poder procesar los eventos que se generen. Esos eventos se tomarán como cambios de estado del dispositivo conectado a esa red de Internet, donde cada dispositivo generará sus propios mensajes de forma regular. De esta forma, cuando un dispositivo o elemento mande un mensaje o cambie de estado al contrario (de encendido a apagado, por ejemplo), el procesador de eventos mandará una alerta al consumidor, para su visualización o para llevar a cabo una actividad concreta.

En este TFG se han implementado unos dispositivos y sensores concretos con la posibilidad de tener conexión a los servicios prestados por la red de terminales y dispositivos de Internet de las Cosas incorporado. En consecuencia, los sensores usados se enfocan en un ámbito importante para el usuario: la salud, por lo que se ha propuesto un caso de estudio de un servicio personalizado de Tele-asistencia. Así, se tiene una simulación de una arquitectura EDA, con sus productores, el procesador de eventos y sus consumidores. En el apartado de Caso de Estudio se especifica con mucho detalle cada uno de los productores y sus posibles valores y estados, para así obtener una respuesta concreta del sistema global. [22]

CAPÍTULO 4. Estudio del arte de un EDA y su relación con Internet de las Cosas. Elementos principales

Una arquitectura orientada a eventos o comúnmente denominada EDA (*Event-Driven Architecture)*, es un patrón de arquitectura software de intercambio de mensajes, permitiendo su producción, publicación, transmisión y visualización. Este tipo de patrón arquitectónico procesa los mensajes de manera asíncrona, por lo que no se produce un bloqueo de los transmisores a la espera de una respuesta por parte del receptor.

Una EDA necesita eventos para poder realizar sus funciones. Un evento es el cambio de estado de un parámetro asociado y la notificación de dicho cambio. De esta forma, cuando alguno de los parámetros sobre los que se monitorizan cambian, se notifica el cambio y se lleva a cabo la acción correspondiente, o la visualización del resultado. Los productores y publicadores son los encargados de notificar estos cambios, enviándolos a través del canal correspondiente hasta los consumidores.

La relación entre la red de Internet de las Cosas y una cadena EDA, aunque ya especificada en el epígrafe anterior, se irá explicando según sea necesario, teniendo en cuenta que son los transmisores y receptores son los que se beneficiarán de los servicios desplegados en la red de IoT, y, en el caso del TFG, del servicio de Tele-asistencia. [23]

4.1. Estado del arte sobre tecnologías, herramientas para implementación y despliegue de aplicaciones EDA

Son muchos los lenguajes, herramientas y entornos de programación sobre los que se basan los servicios fundamentados en software. Esto es debido a la cantidad de aplicaciones desplegadas, ya que éstas pueden usar un patrón similar pero el lenguaje cambia en cada caso, o viceversa, y se despliegue sobre un sistema operativo diferente. De esta forma, también se pueden encontrar diferentes formas de definir y programar las arquitecturas orientadas a eventos, teniendo en cuenta tanto el transmisor como el receptor, además del despliegue y la aplicación a la que va destinada. En este epígrafe se va a llevar a cabo un estudio de las herramientas, entornos y lenguajes de programación para las arquitecturas orientadas a eventos, donde finalmente se elegirá uno de esos patrones y herramientas para poder formalizar la demostración de una arquitectura orientada a eventos completa.

Antes de proceder a explicar cada uno de los entornos y lenguajes estudiados, se debe hacer hincapié en la diferencia entre la programación tradicional y la programación dirigida a eventos. En un programa secuencial es el programador o desarrollador del producto el que decide el orden y flujos del sistema sin tener en cuenta al usuario, simplemente se diseña un flujo de datos fijos, donde se puede pedir un número limitado de parámetros para inicializar el sistema. Sin embargo, en la programación dirigida por eventos, el usuario puede intervenir en el proceso de determinación del flujo de datos, es decir, puede enviar en cualquier momento un evento o flujo de eventos para poder obtener una respuesta diferente en cada caso. De esta forma, se obtienen diferentes flujos en el sistema dependiendo de los eventos que mande el usuario al procesador. [2]

Otra diferencia significativa que se debe tener en cuenta es la implementación de la programación asíncrona en una arquitectura orientada a eventos. Un evento asíncrono no necesita respuesta por parte del receptor o servidor, ya que ese evento o mensaje se enviará al canal, sin importar si llega al receptor o no. Además, el transmisor del mensaje no se queda bloqueado, puede seguir enviando mensajes sin llegar a conocer el estado al que llegan al receptor o que mensaje es el que se entrega finalmente al mismo, ya que puede que el procesador de eventos envíe una notificación concreta según la combinación de eventos que le lleguen. Por otro lado, en la programación secuencial, el transmisor del mensaje se queda bloqueado, esperando una respuesta por parte del receptor o servidor, lo que se denomina un evento síncrono. Es decir, se necesita una sincronización entre transmisor y receptor. No obstante, nótese que también en la programación orientada a eventos igualmente se puede elaborar un receptor que mande una respuesta al transmisor, aún tratándose de una comunicación asíncrona, pero no por ello bloquear o dejar de mandar eventos en dirección transmisor-receptor. [3]

El middleware MOM (*Message-Oriented Middleware*) permite a aplicaciones distribuidas comunicar e intercambiar información a través del envío y recepción de mensajes. Dentro de los middleware se puede observar que tenemos dos tipos: [24] [25]

- **Espera**. Este tipo de middleware cuenta con un mensaje a transmitir y una cola de mensajería, usando el MOM del cliente para realizar dicha transferencia a la cola y a los consumidores posteriormente. El middleware recoge las peticiones de envío en un orden de espera preestablecido.
- **Publicación/subscripción**. Este middleware está algo más orientado a eventos, debido al registro de estos. Si un cliente llega a este tipo de middleware se une al bus de información y espera a que el MOM le envíe la notificación de envío de evento, registrado previamente por el publicador. El servidor MOM envía esa noticia al subscriptor cuando la información del evento esté disponible.

Algunos de los middleware que se comercializan son MQSeries, ActiveMQ y RabbitMQ. El middleware RabbitMQ se irá especificando a lo largo de la memoria.

MQSeries es un middleware de intercambio de mensajería entre transmisor y receptor, facilitando el desarrollo de aplicaciones para cualquier sector. Fue desarrollado por IBM y se ha renombrado a IBM-MQ en la actualidad. Este middleware ofrece

intercambio de eventos en muchos ámbitos y de forma fiable, segura, rápida y supervisada para que llegue a la cola de mensajería correspondiente, teniendo como emisores aquellos objetos interconectados en la red de IoT. Sin embargo, este middleware exige una compensación económica a medio plazo, por lo que se decidió utilizar RabbitMQ, que se trata de una solución de código abierto y asequible para unirlo con Esper, ya que tienen definido un canal común entre ellos (AMQP). [26]

ActiveMQ también es un middleware de intercambio de mensajería, donde tiene puntos de similitud importantes con RabbitMQ. Se trata de una solución de código abierto desarrollada por Apache, bajo su licencia de Apache 2.0, y lleva a cabo todas las tareas de transmisión y recepción, obteniendo así un broker de mensajería completo, con capacidad para ofrecer intercambio de mensajes empresariales y para implementar la solución JMS. Además, AMQP es un plugin que tiene tanto ActiveMQ como RabbitMQ para poder conectarse con Esper. Aunque ActiveMQ tiene muchas similitudes en cuanto a funcionamiento con RabbitMQ, este segundo tiene una sintaxis más asequible para poder entrar en el aprendizaje para enviar y recibir eventos. [27]

Asimismo, se han implementado los productores de eventos mediante RabbitMQ y su plugin AMQP, que tienen varias configuraciones respecto al intercambio de información mediante las colas de mensajería. [28]

En cuanto a las implementaciones del lenguaje orientado a eventos más conocidas podemos tener Visual Basic, bastante generalizado y conocido. Este lenguaje fue desarrollado por Alan Cooper para Microsoft, con la primera versión en 1991, para solucionar los problemas de implementación de aplicaciones simplificando la programación permitiendo la creación de interfaces gráficas. Visual Basic es un lenguaje de programación orientado a objetos, implementándolos en las bases de datos oportunas (por ejemplo, Data Access Objects (DAO) es un componente software que suministra una interfaz gráfica). Visual Basic dispone de un entorno de desarrollo integrado (IDE), que dispone de muchos elementos de edición de textos, código fuente e interfaces gráficas. Visual Basic utiliza objetos con propiedades y métodos, pero carece de las propiedades de herencia y polimorfismo, puntos que Java solventa. [30] [33]

Sin embargo, se ha decidido por utilizar Esper y RabbitMQ para la implementación de la EDA del TFG. Esto es debido al lenguaje de programación usado, Java, que resulta más sencillo para el desarrollo de la EDA, ya que dispongo un buen conocimiento de este lenguaje. Además, Esper cuenta con las sentencias EPL, cuya sintaxis es muy similar a las sentencias de las bases de datos SQL. La cantidad de plugins y posibilidades de aplicación se ajustan perfectamente al objetivo de desarrollo de una arquitectura orientada a eventos en relación con objetos que se benefician de los servicios desplegados en la red de Internet de las Cosas. [29]

Para la realización del prototipo del TFG se ha utilizado el entorno de desarrollo Eclipse, usando las librerías oportunas, debido al amplio conocimiento de este entorno.

4.2. Comparativa de una EDA con el modelo cliente-servidor

Para ilustrar mejor una EDA, se ha realizado una comparación con el modelo cliente-servidor, donde se establecen diferencias y similitudes entre ambos patrones:

- En cuanto a las **similitudes**, se verifica que el sistema completo de una EDA puede funcionar como un sistema petición-respuesta, es decir, el servidor se corresponde con el procesador de eventos complejo, donde se recibe un evento por parte de los generadores. El procesador resuelve esa petición y genera una respuesta correspondiente a ese evento concreto. Asimismo, ambos tienen un carácter multiusuario, donde el sistema es esencial para tener múltiples usuarios a la vez. Se refuerza también la idea de concurrencia y disponibilidad.
- Las **diferencias** son claramente el eje de este patrón. Se establecen dos:
  - Siguiendo con el ejemplo anterior del proceso petición-respuesta, se puede advertir cómo la respuesta no es redireccionada hacia el transmisor. En una EDA, esa respuesta se encamina a los consumidores, que tienen la potestad de descargarse el evento y visualizarlo mediante los procesos oportunos. De esta forma, se observa la diferencia frente al modelo cliente-servidor: la respuesta del supuesto servidor se envía al siguiente elemento de la cadena, y no al generador de la petición.
  - Por otro lado, se puede tener un evento sin necesidad de la producción por parte de los generadores. Por ejemplo, el procesador de eventos puede tener programado el envío de un evento a los consumidores. De esta forma, la diferencia con el modelo cliente-servidor se hace notable, siendo vital distinguir la producción del evento y su posterior consumo.

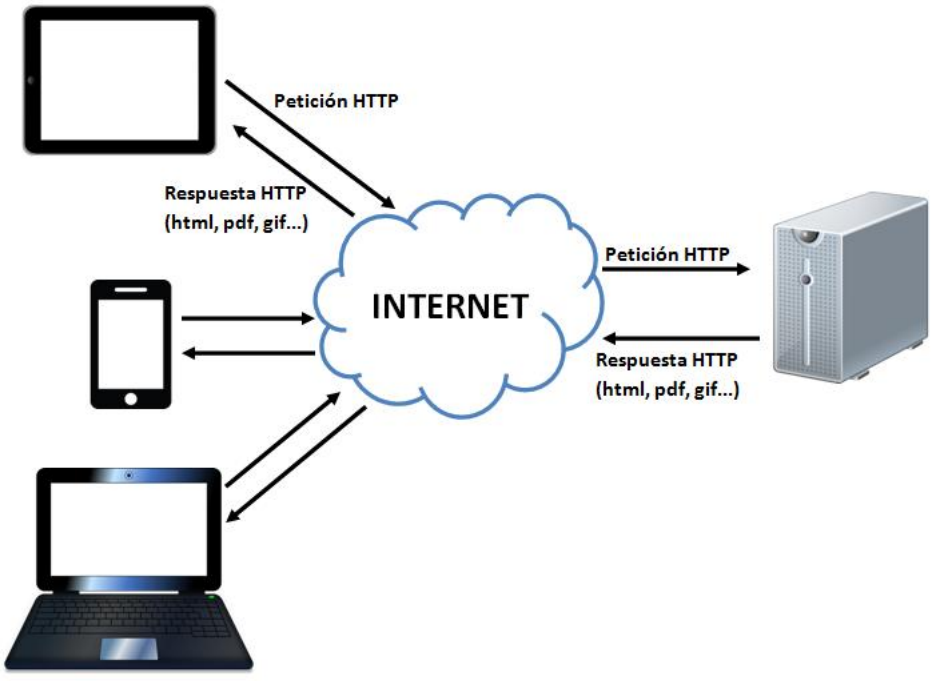


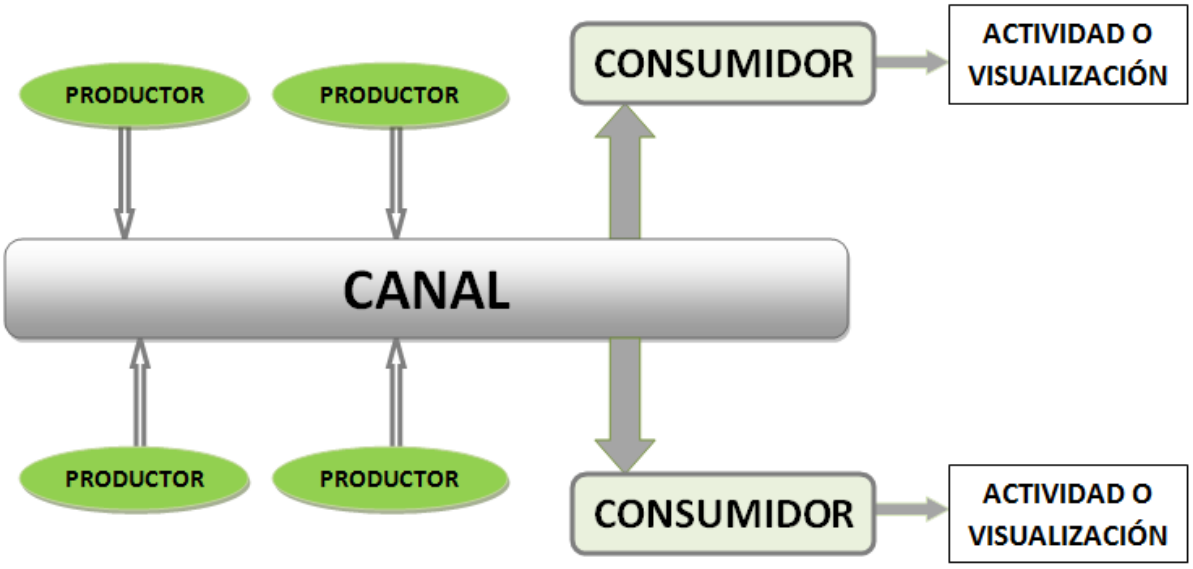
Diagrama del modelo cliente-servidor

4.3. Capas de una EDA genérica

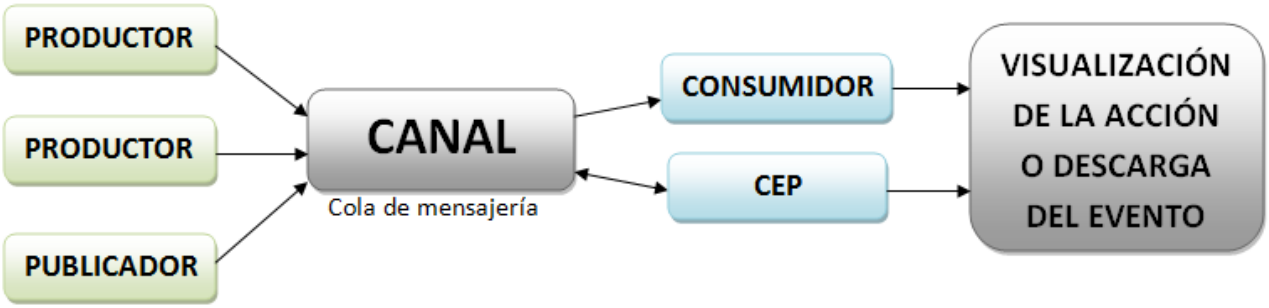
Una arquitectura orientada a eventos tiene varios elementos que se van a analizar detalladamente a lo largo de esta memoria. Además, atendiendo a cada uno de los mismos, se puede advertir como la cadena entera del sistema completo no puede llevar a cabo todo el proceso si alguno de los componentes de dicho sistema falla. En consecuencia, se especifican los parámetros, en su caso, necesarios para que cada elemento funcione correctamente. Los principales componentes son: [23] [28] [29]

- **Productores o publicadores de eventos:** es aquel componente o componentes encargado(s) de generar o recoger los eventos que encuentra en el entorno. Puede ser cualquier aplicación, sensor, dispositivo externo... En el caso concreto del TFG, se llevará a cabo la definición de unos sensores publicadores de información de importancia para el sistema completo, referidos a los parámetros de salud del usuario dentro del entorno de servicio de Tele-asistencia.
- **Canal:** será el medio por el que se transmitan los eventos. Para el Trabajo Final de Grado se unirán ambos elementos del sistema (productores y consumidores) mediante los plugins oportunos para ello, detallados más adelante en el epígrafe de la implementación del canal, para comunicarse con el procesador de eventos.
- **Consumidores y procesador de evento:** este elemento se encarga de la consumición y/o toma de decisiones respecto al evento. Sin embargo, se debe prestar especial atención al dicho procesador de eventos, ya que se encarga de diferenciar entre las acciones a tener en cuenta para cada uno de los eventos que lleguen, decidiendo el envío de alarma por anomalía en los valores de algún parámetro de los eventos que recibe. De esta forma, se trata de un procesador inteligente, que ejecuta una acción determinada siempre que le llegue un evento concreto o una combinación de los mismos, pudiendo enviar eventos a unos consumidores o consumir él mismo dicho evento.
- **Representación de los datos** y de las acciones tomadas para cada evento: en este último elemento se lleva a cabo la representación de los datos y acciones.

A continuación, se muestra una imagen de todos los elementos interconexionados entre sí, donde el productor o publicador manda un evento al canal, y dicho mensaje es recogido por los consumidores y/o procesador de eventos, donde posteriormente se lleva a cabo una visualización de los resultados.



Sin embargo, para que este esquema sea mucho más ilustrativo, se ha procedido a dimensionarlo de manera secuencial, donde se pueda ver todo el proceso de envío y recibimiento de eventos. Los productores y publicadores mandan unos eventos o mensajes al canal, donde se almacenan en una cola de mensajería mientras que el receptor no se encuentre disponible, o se descartan. Los productores y publicadores pueden seguir mandando dichos mensajes, ya que el canal los enviará cuando el sistema esté preparado. Al otro lado del canal se tiene al consumidor, el cual procesa el evento o "consume" y pasa a la visualización o descarga del mismo. Sin embargo, normalmente se tiene un procesador de eventos complejos (CEP), donde se alberga una lógicas o reglas preestablecidas para poder ejecutar una acción o enviar un mensaje dependiendo de los mensajes procedentes del transmisor. De esta forma, esos mensajes se devuelven al canal y un consumidor (o varios) llevan a cabo la tarea de visualización de la acción determinada por el CEP y/o la descarga del evento.



4.4. Generadores y publicadores de eventos

4.4.1. Funcionalidad

Los generadores o publicadores de eventos se encargan del envío de los mensajes al canal. Estos mensajes serán los indicadores del cambio de estado de un objeto o de un parámetro del mismo, es decir, de un evento. Se encargarán de recoger los datos de sensores externos, como, por ejemplo, sensores de medida de los parámetros médicos de un paciente, como puede ser un sensor del nivel de glucosa en sangre o la medida de la tensión arterial y ritmo cardíaco de dicho paciente. Asimismo, también pueden estar incorporados en el terminal móvil inteligente del usuario, recogiendo sus datos de ubicación y acelerómetro.

En consecuencia, se usan dispositivos y sensores con acceso a los servicios proporcionados por la red de IoT para poder mandar esos mensajes y eventos al canal. Estos mensajes llegarán al procesador de eventos, que tomará las decisiones oportunas según unas reglas predeterminadas por el propio desarrollador. Así, será los productores y publicadores los que deban tener acceso a la red de IoT para mandar los datos de sus parámetros a través de Internet. Además, los consumidores también deberán disponer de acceso a esta red en alguno de los casos. En este TFG se realizará una simulación de dichos generadores mediante unos sensores de los parámetros médicos de un usuario, para poder verificar el correcto funcionamiento de la arquitectura dirigida por eventos.

4.4.2. Implementación: RabbitMQ

En este apartado se especifica de forma rigurosa la elección de RabbitMQ para el envío de los datos al procesador de eventos, teniendo en cuenta la relación entre RabbitMQ y Esper, usado este último para implementar el procesador de eventos, mediante el lenguaje EPL, cuyas características se especificarán más adelante.

El servidor de RabbitMQ está escrito en Erlang, lenguaje muy extendido por sus características de programación concurrente. Además, RabbitMQ es una solución de envío de mensajes de código abierto, donde el lenguaje en el que se programa depende el desarrollador de la aplicación. En consecuencia, actúa como middleware de mensajería, es decir, se trata del software que hace posible el intercambio de mensajes entre un transmisor y un receptor, teniendo en cuenta la lógica del sistema en su conjunto. Se consigue, por tanto, definir en términos sencillos los elementos correspondientes a una EDA. Asimismo, RabbitMQ utiliza el framework OPT (Open Telecom Platform), servicio de código abierto y de aplicaciones para construir sus capacidades de comunicación y de conmutación de errores, escrito en Erlang. [28]

RabbitMQ también tiene diferentes pasarelas o plugins para los protocolos HTTP y XMP. Además, almacena bibliotecas de Java y el framework .NET en línea,

aunque también dispone de las bibliotecas necesarias para cada uno de los lenguajes en los que se puede elaborar el sistema. Un plugin de mención especial es el AMQP (*Advanced Mesagge Queueing Protocol*), que se detallará más adelante cuando se especifique el canal sobre el que se mandan los mensajes y eventos.

Como ya se ha comentado, RabbitMQ es una solución de código abierto para implementar productores y consumidores según las especificaciones que sean necesarias en el sistema. Además, dichas soluciones se pueden implementar según el lenguaje que más cómodo sea para el programador o diseñador del proyecto (Java, Python, Ruby, PHP o C#). Con cada uno de estos lenguajes se proporcionan las librerías concretas del cliente. Para desarrollar la cadena EDA se ha elegida Java de este TFG, ya que para el procesador de eventos CEP es mucho más sencillo encajar ambas piezas, debido a que Esper usa Java también.

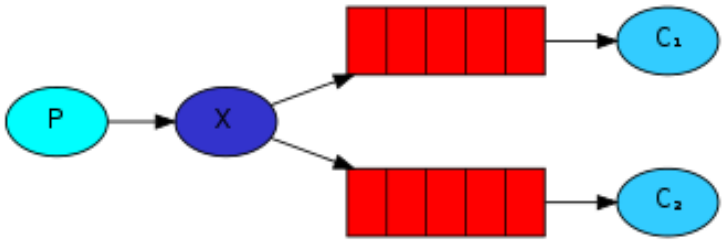
4.4.3. Características principales de RabbitMQ

RabbitMQ permite tener diferentes configuraciones, además de las creadas por el propio desarrollador. En los casos propuestos por la página oficial de RabbitMQ [34], se puede observar cómo se tienen tres tipos de intercambio de mensajes dependiendo del tipo de comunicación que se quiera tener:

- **Direct**: se trata de una comunicación punto a punto, donde se copia el mensaje en la cola cuando coincide exactamente la clave del mensaje y la de la cola.
- **Fanout**: este tipo de comunicación es el usado para la configuración publicación/subscripción, copiando el mensaje en todas las colas que tenga conectadas.
- **Topic**: es este último tipo el mensaje se manda según las especificaciones sobre el mensaje que se concretan en la implementación del código. Así, si se tiene un productor que manda mensajes de error, información y warning, tendremos varios receptores configurados para que les lleguen los mensajes de error, información y warning por separado respectivamente. También se pueden configurar para que un receptor le lleguen los mensajes warning y de error. De la misma forma, los publicadores o productores pueden ser más de uno, especificando qué tipo de mensajes manda cada uno.

Para implementar el EDA del TFG se ha empleado la configuración de RabbitMQ propia del esquema de publicación/subscripción, donde se emite un mensaje por parte de los productores y se envía a todos los receptores y consumidores del sistema. Sin embargo, se debe prestar especial atención a las colas de mensajería de este tipo de configuración, ya que cada receptor tendrá su propia cola de espera, para así no interferir unos mensajes con otros si se da el caso de congestión o descarte. No obstante,

se debe tener en cuenta que no se usan exactamente colas de mensajería en esta configuración, si no que se utiliza lo denominado **intercambios**. Estos intercambios pueden tener la función de preprocesador, donde se mandan hacia las diferentes colas de mensajería los eventos oportunos procedentes de los productores asociados a dicho intercambiador de mensajes. A continuación, se muestra un diagrama sencillo de la configuración publicación/subscripción, donde el intercambio de mensajes está marcado por un X en azul oscuro. En el caso concreto del TFG, es el propio middleware de RabbitMQ, mediante esta configuración de publicación/subscripción quien decide qué eventos se mandan a cada consumidor, ya que este subscriptor o consumidor ha notificado anteriormente el tipo de eventos que quiere recoger. Los productores mandan todos sus eventos al middleware para que lleguen a los consumidores oportunos.



Esquema publicación/subscripción. Fuente: [34]

4.5. Canales

4.5.1. Funcionalidad

Una vez se tienen definidos los generadores y publicadores de eventos, se debe especificar el canal por el que mandar estos datos. Al tener el procesador alojado en Internet y los datos provenientes de los generadores también se reciben en este protocolo, se aprovechará este escenario para implementar el canal en Internet. En el caso de este TFG, será un plugin de RabbitMQ para poder ensamblar con los consumidores y procesador de eventos en Esper. Ese plugin es AMQP.

Sin embargo, se debe tener en cuenta que no siempre se trata de un canal alojado en Internet, también se puede dar la situación de hacer que llegue la información hacia el consumidor mediante conexiones a nivel de transporte, con intercambio de mensajes en los puertos TCP del sistema. Al tener relación directa con los servicios desplegados en la red de dispositivos de Internet de las Cosas, los productores y consumidores envían los eventos con este sistema, por lo que resulta conveniente y fundamental implementar el canal mediante, en este caso, un plugin de RabbitMQ.

4.5.2. Implementación: AMQP

Además de la implementación de los productores mediante clases de Java y el middleware de RabbitMQ, se han implementado los consumidores y la lógica del procesador CEP mediante Esper, cuyas características son fundamentales para una EDA. Sin embargo, para poder unir estos dos elementos se ha usado el protocolo estándar AMQP (*Advanced Message Queueing Protocol*). Este protocolo se caracteriza por su definición respecto a la orientación a mensajes, enrutamiento de los mismos, su encolamiento y la exactitud a la hora de enviar y/o recibir los eventos. [28] [29] [32]

El estándar AMQP surgió con la necesidad de implementar arquitecturas orientadas a eventos para desarrollarlas en cualquier S.O. Tradicionalmente, se debía contactar con el proveedor para poder llevar a cabo la portabilidad de ese software a otro dispositivo o sistema operativo. Además, en algunas ocasiones el software no aceptaba ciertas plataformas de despliegue, por lo que era necesario definir la aplicación en un lenguaje completamente distinto. Sin embargo, para paliar estos problemas, se desarrolló un middleware de intercambio de mensajes abierto para cualquier sistema operativo, denominado AMQP (*Advanced Message Queueing Protocol*), con su origen en JP Morgan Chase (empresa de servicios financieros). Asimismo, este protocolo de mensajes de estándar abierto permite implementar cualquier EDA, además de las aplicaciones en diferentes lenguajes de programación.

El objetivo del estándar es simple: definir la mecánica de la transferencia segura, confiable y eficaz de mensajes entre emisor y receptor. En consecuencia, algunas de sus características más importantes son: eficacia, debido a su orientación a la conexión de ambas partes del sistema y la elaboración de esquemas de monitorización de la red; fiable, debido a sus configuraciones de envío y olvido o las de entrega confirmada; flexible, debido a las diferentes formas de conexión que puede albergar (cliente-cliente, cliente-servidor, etc.); y la independencia con los elementos del sistema, ya que no impone ninguna restricción en cuanto a dichos componentes. Estas características y los lazos en común con RabbitMQ y Esper han determinado usar este protocolo como middleware o canal de intercambio de mensajes y eventos.

4.6. Procesadores de eventos CEP

El procesador de eventos es el eje principal de una arquitectura orientada a eventos. Es aquel que alberga la lógica y las reglas de envío, descarte, modificación y detección de eventos y anomalías en los parámetros de los mismos. De esta forma, el procesador de eventos se encargará de alojar las diferentes combinaciones de detecciones de eventos, enviando un mensaje o alarma cuando se exponga el escenario concreto que tiene configurado. Asimismo, también llevará a cabo la acción determinada que tenga configurada y enviará el evento a uno o varios consumidores.

Además, se debe tener en cuenta que también puede llegar a recoger datos de varios productores y ponderar una media y enviar ese dato concreto a un solo consumidor.

Como se puede observar, se pueden tener múltiples combinaciones si se dispone del número de productores y consumidores suficientes. En este TFG, se tendrán productores suficientes para poder implementar una lógica compleja al procesador de eventos. Para la implementación del procesador se ha escogido el lenguaje EPL incluido en las sentencias de *data flow* del software de código libre Esper, cuyas características se muestran más adelante.

4.6.1. Tipos de procesamiento

En una arquitectura orientada a eventos se pueden observar tres tipos de eventos o mensajes generados en los productores o recogidos por los publicadores: [23]

- **Evento simple.** Solamente se detecta el envío de un solo evento, el cual se procesa y se toma la decisión oportuna respecto al mismo, procesándolo, por tanto, en tiempo real.
- **Flujo de datos.** En este caso, al procesador de eventos le llegan múltiples eventos y debe tener reglas de decisión sobre la importancia de ser tratados dichos datos. Así, el procesador irá tomando decisiones sobre los eventos uno a uno, teniendo en cuenta el orden de prioridad de cada uno de ellos.
- **Eventos complejos**. Los productores mandan varios eventos al procesador y éste debe tomar una combinación de los mismos para poder tomar una decisión según esas combinaciones. El procesador lleva a cabo un análisis avanzado y puede anticipar resultados según una correlación y análisis de los datos inyectados al mismo. Este tipo de eventos está muy normalizado para empresas que necesitan grandes cálculos económicos o para anticipar ciertos riesgos predecibles, gracias a su análisis avanzado y modelado de un patrón general de los datos y de la información.

4.6.2. Implementación: Esper

Una vez que ya se tiene especificado la elección de la estructura de los productores (clases de Java: POJOs) y del canal o middleware (RabbitMQ) que implementa la relación entre los productores y el procesador de eventos. En consecuencia, como ya se comentó, se ha elegido para implementar la lógica del procesador de eventos la solución de código abierto Esper, ya que su lenguaje es una variante de Java junto con el lenguaje EPL (*Event Proccessing Language*), variante a su vez de las sentencias y sintaxis SQL y que será el que decida qué tipo de eventos tengan

una lógica u otra dependiendo de los valores de sus distintos parámetros. Es decir, Esper implementa un CEP (*Complex Event Proccessing*) mediante el lenguaje EPL, para poder enviar eventos concretos a los consumidores y/o tomar decisiones sobre los que le lleguen sobre unas reglas y pautas de comportamiento preestablecidas.

Como ya se ha especificado, Esper es una solución de código abierto que se basa en la implementación del procesamiento complejo de eventos (CEP). Esper se distribuye mediante la licencia GPL (*GNU General Public License*). El proyecto comenzó en el año 2004, pero la primera versión se creó y comercializó en 2006 por EsperTech Inc. A lo largo de esto últimos casi 10 años, se ha ido desarrollando y perfeccionando para obtener una versión robusta para implementarla en plataformas de desarrollo en el lenguaje Java, como, por ejemplo, en el entorno de desarrollo Eclipse, entre otros. También cuenta con un versión comercial que ofrece un valor añadido respecto a formación y soporte extra, proporcionado por la empresa EsperTech Inc. [28]

4.6.3. Características de Esper

La estructura de Esper es sencilla y tiene muchas posibilidades de implementación en diferentes expresiones, teniendo en cuenta que es Java en lo que se basa este tipo de procesador de eventos complejo. Es decir, Esper puede implementar diferentes eventos en distintos lenguajes, por ejemplo en XML, pero la forma de procesarlos y decidir las reglas para enviarlos se realizan en Java, junto con sentencias EPL. A continuación, se detallan los diferentes elementos de Esper, para estudiarlos de forma más exhaustiva. Además, más adelante se especifica cómo se ha implementado en la cadena EDA del TFG este procesador de eventos complejo. [28] [31] [35]

- **Eventos**. Los eventos que procesa Esper pueden estar descritos en cualquier lenguaje de representación de datos. Sin embargo, es más común encontrarse con eventos descritos con sentencias Java, debido a la facilidad posterior de procesarlos con el mismo tipo lenguaje. Aún así, también es frecuente encontrar eventos descritos en el lenguaje de marcas XML (*eXtensible Markup Language*: lenguaje de marcas extensible), debido a la facilidad de representar objetos y mensajes mediante este tipo de lenguaje. En el caso de Java, para poder representar los eventos y mensajes se elaboran mediante los POJOs o beans, es decir, se tratan de clases de Java "normales y corrientes", aquellas clases que solamente tienen las variables propias de las mismas, su correspondiente constructor y los métodos accesores y modificadores, además del método toString() para poder imprimir por pantalla dicho evento.
- **Sentencias o statements**. Las sentencias son las que albergan la lógica del sistema. Estos statements se describen en lenguaje EPL, extensión de la sintaxis de SQL, además de sus métodos de consulta y extracción de datos. Esta consulta de datos proporciona un resultado concreto, que se traslada al siguiente eslabón

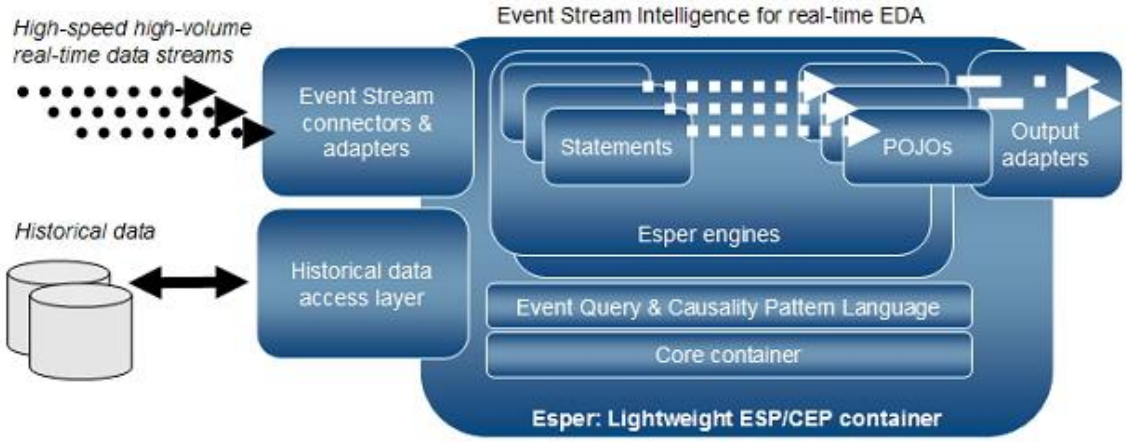
de la cadena: el consumidor o listener en Esper. A continuación, se muestra un ejemplo de sentencia EPL, cuyo resultado se "consumiría" en un listener.

```
select avg(price) from StockTick having avg(price) <= 2.0
```

En este ejemplo, tenemos un Tick u objeto a comprar con un precio y un nombre. De esta forma, la sentencia, en este caso, funciona de forma similar a las sentencias SQL: extrae el precio del objeto que se ha enviado o le ha proporcionado el productor, especificando solamente aquel objeto que tenga su precio menor o igual a 2.0 euros. Por tanto, si tenemos varios objetos que ha mandado el productor al CEP, solamente se imprimirá por pantalla aquellos que tengan un precio menor o igual a dos euros. Asimismo, se pueden implementar múltiples sentencias EPL según la lógica seguida por el CEP, ya que son muchos los productores que se pueden tener y muchos los consumidores, por lo que cada uno puede estar relacionado entre sí de una forma concreta.

- **Listeners**. Este elemento es el encargado de recibir el resultado de las sentencia EPL correspondiente. En el caso del EDA general, este elemento actúa como consumidor de los eventos, ya que será donde se podrá observar el resultado específico de la sentencia EPL concreta.
- **Configuración y envío de eventos**. Aunque no es necesario implementar una configuración específica para poder ejecutar las sentencias en Esper, es recomendable elaborar una configuración propia para cada tipo de sentencia o evento, especificando de clase provienen los eventos y el número de sentencias EPL y consumidores asociados a cada evento. Asimismo, el envío de eventos de debe implementar de forma obligatoria, concretando sus parámetros y buscando la relación con la configuración propuesta anteriormente. Los eventos se pueden mandar de forma "manual" (creando directamente un objeto de la clase del evento) o de forma iterativa, con bucles for y sentencias if, entre otros.

A continuación se muestra una imagen que esquematiza estos elementos:



Arquitectura software de Esper. Fuente: [22]

En relación a la EDA del TFG, los productores y el canal por el que mandar esos eventos se han realizado con RabbitMQ con sus funciones de middleware y el protocolo estándar AMQP, respectivamente, siendo este último un puente entre RabbitMQ y Esper. En consecuencia, Esper actuará como el procesador de eventos complejo, gracias a la sentencias EPL y la gran capacidad para procesar eventos de tipo Java. Por tanto, la implementación de Esper en la EDA será esencial para el procesamiento y consumición del evento, teniendo en cuenta la posterior visualización por parte del navegador.

4.7. Visualización de los datos

La visualización de los datos puede llegar a ser una parte importante del sistema global, ya que en ella reside la correcta comprensión de los datos de salida. En consecuencia, se tienen muchas herramientas para poder visualizar el resultado de la combinación de los diferentes eventos y su posterior consumición.

En primer lugar, una de las posibilidades es la tecnología push de servidor. Se opone al funcionamiento habitual del modelo cliente-servidor, aunque sus bases conceptuales son similares. Así, el servidor es quien manda la información en tiempo real sin necesidad de una petición previa por parte del cliente. El servidor mantiene la conexión abierta para la actualización de los datos cada cierto tiempo. En el caso de una EDA se necesitaría la actualización de los datos de salida del procesador de eventos complejo para comprender sus valores y tomar decisiones al respecto. Dentro de esta tecnología se encuentran múltiples frameworks para implementar dicha tecnología push de servidor, como, por ejemplo, Comet o Atmosphere, entre otros. [36]

En segundo lugar, otra posibilidad es aquella implementada mediante los plugins que proporciona Esper. Dichos plugins se han estudiado atendiendo a las salidas del procesador de eventos. En consecuencia, se trata de aquellos plugins correspondientes a conexiones HTTP y Sockets, donde las sentencias de *data flow* son vitales para implementar dichos plugins y sus funciones. [29]

En tercer y último lugar, se puede considerar la posibilidad de realizar una monitorización en tiempo real denominada BAM (*Business Activity Monitoring*), que está ligada a los procesos de negocio y económicos, debido a la cantidad de datos almacenados y a su correcta interpretación. En otros ámbitos, se puede implementar de esta forma, teniendo en cuenta que se debe realizar un análisis exhaustivo de la información, mostrando una interpretación personalizada y objetiva de los datos. [37]

Finalmente, la implementación de la visualización de la arquitectura dirigida por eventos y sus correspondientes alarmas del TFG se ha llevado a cabo mediante su implementación con recursos web. Estos recursos detectan los cambios concretos de la salida del procesador de eventos complejo, notificando de ello a un centro de Tele-asistencia. Este centro se hace cargo de la posible emergencia. En el caso del TFG, se mostrarán las alarmas o datos de la anomalía en el navegador.

CAPÍTULO 5. Caso de estudio

El caso de estudio que se presenta en este TFG tiene por propósito la validación de la arquitectura EDA propuesta y del prototipo implementado mediante el estudio de un caso realista y verosímil en el contexto de los servicios desplegados sobre la red de Internet de las Cosas. Esa EDA se compone de varios elementos importantes para el envío y recepción de eventos. Se detallarán estos componente según su función específica en la EDA, aunque ya se han especificado las herramientas a usar. Dichos elementos son: los productores, el canal de envío de datos, el procesador de eventos, y los consumidores, además de añadir la visualización de los datos para su comprensión.

El caso de estudio al que se ha aplicado el TFG ha sido a un servicio personalizado de Tele-asistencia, enfocándolo hacia una aplicación para smartphones. En esta prestación del servicio se hace uso de la red de dispositivos de IoT, centrándose en terminales externos y sensores de detección de anomalías de la salud de un paciente. Concretamente, se realizarán medidas sobre la tensión, nivel de glucosa y ritmo cardíaco del paciente, además de proporcionar la opción de la pulsación del botón del pánico para alertar de cualquier emergencia, similar al número de emergencias habitual. Además, también se realizarán medidas sobre el estado del lugar donde se encuentre dicho usuario, para alertar de un posible incendio, siendo éste una correlación de datos entre la detección de gas nocivo y la detección de una elevación brusca de la temperatura. En consecuencia, este servicio personalizado de Tele-asistencia genera un beneficio concreto al usuario, atendiendo sus problemas de salud al instante y alertando a los servicios de emergencias oportunos de la situación actual del paciente.

Para implementar la arquitectura se han estudiado los diferentes lenguajes de programación de los elementos del sistema. En consecuencia, se han trabajado las características de RabbitMQ, Esper y AMQP, realizando los tutoriales correspondientes para el correcto aprendizaje de este tipo de tecnologías y herramientas. [34] [35]

5.1. Principales requisitos

La arquitectura de orientada a eventos realizada en este TFG deberá tener y especificar una serie de requisitos primordiales para la implementación de la misma.

En primer lugar, se procederá a concretar los distintos productores y publicadores de la información relacionada con los eventos. Es decir, se llevará a cabo una definición de los generadores de las anomalías que puedan surgir respecto a la información habitual de un evento. El servicio personalizado de Tele-asistencia se basará en la recogida de datos de los objetos conectados a la red de IoT. En consecuencia, estos elementos, consistirán en una simulación de unos sensores que recojan medidas concretas del usuario, para satisfacer este servicio. Esos objetos son:

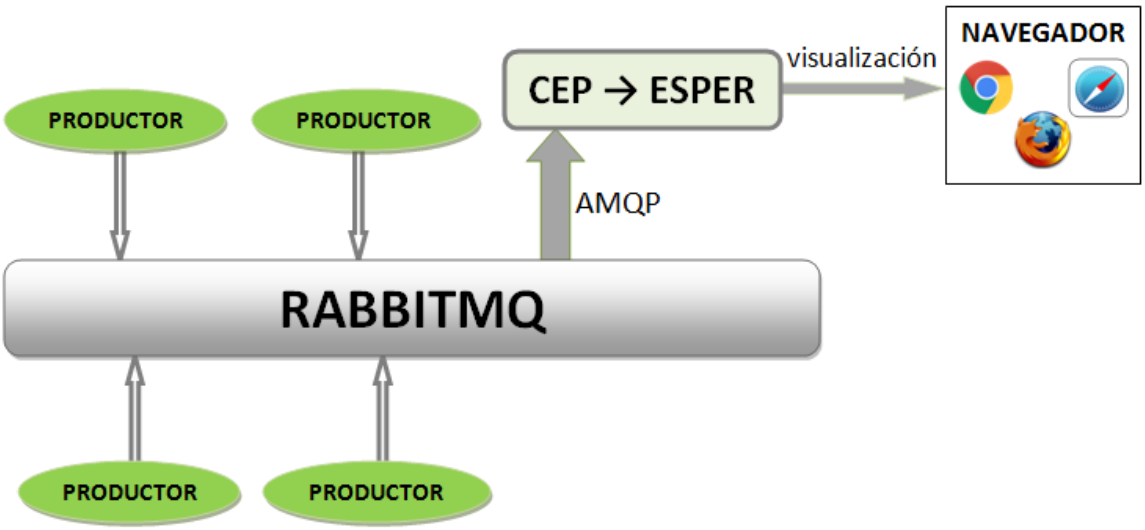
- Sensor de **nivel de glucosa**, que mide la glucosa o azúcar en sangre del usuario.
- Sensor de **ritmo cardiaco y tensión arterial**. Mide los parámetros principales del sistema nervioso del usuario, como el ritmo cardiaco y la tensión arterial.
- Sensor de pulsación del **botón del pánico**. El sistema tendrá un botón que el usuario pulsará en caso de emergencia, donde se tomará como una alarma de emergencia directamente.
- Sensor de **incremento brusco de la temperatura interna de una habitación**. Mide el incremento brusco de la temperatura, para detectar un posible incendio en la sala o lugar donde se encuentre el usuario, según proceda en cada caso.
- Sensores del **nivel de gas y/o monóxido de carbono**, detectando posibles fugas de gas y avisando a los servicios de emergencias oportunos, además de a los usuarios de la vivienda.
- Sensores del **acelerómetro y GPS** propios del teléfono inteligente, para detectar posibles caídas y conocer el lugar donde se encuentra el paciente ante alguna anomalía grave y que pueda ser atendido personalmente.

Se puede observar que el conjunto de objetos con el que se va a trabajar, o a simular su proceso, se pueden englobar en un ámbito: salud. También se puede considerar la detección de incendios dentro de un segundo ámbito sobre el estado del hogar. Por tanto, al tener esos principales productores ya definidos, como clases de Java (POJOs o beans), se deben enviar los datos a un middleware que soporte el lenguaje usado. En consecuencia, se ha elegido RabbitMQ, un middleware de envío y recepción de mensajes. RabbitMQ tendrá los datos de los productores preparados para enviarlos a través de AMQP. Nótese que cada productor corresponde con una máquina diferente que manda sus datos al middleware, aunque en el prototipo sea una simulación.

Con los datos de los productores y publicadores ya definidos y ubicados de una manera precisa y concisa, se necesita un canal para poder mandarlos. En este TFG se ha utilizado la arquitectura inteligente de Esper para implementar el procesador de eventos complejos. De esta forma, el canal, y, en consecuencia, la cola de mensajería, se implementará mediante un plugin de RabbitMQ para poder encajarlo con Esper. Este plugin se denomina AMQP, que usa *data flow* para poder implementar todo el proceso.

El procesador de eventos complejos (CEP) es uno de los puntos importantes en una EDA. Se ha decidido implementarlo mediante Esper, debido a su compatibilidad con RabbitMQ, por su lenguaje de programación, y por sus sentencias de consultas, muy similares a las usadas con bases de datos SQL. Además, Esper proporciona una configuración con la que tener diferentes combinaciones de los eventos que se detecten. Por ejemplo, se puede tener un nivel alto de gas y un incremento brusco de la temperatura, y Esper puede realizar una combinación de ambos eventos y determinar que hay un incendio. Para representar estos datos, se han desplegado en el navegador.

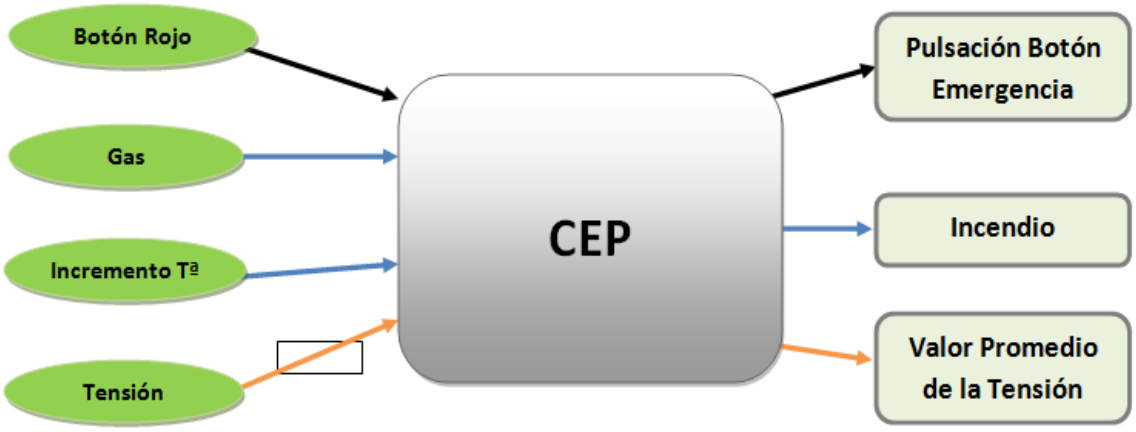
A continuación se muestra una imagen que engloba el sistema en su conjunto. Los productores, con sus datos y parámetros "almacenados" en el middleware de RabbitMQ, envían esos datos al CEP (implementado mediante Esper), mediante el plugin de AMQP. Además, se procede a la visualización mediante en el navegador.



Por otro lado, se especifica en la siguiente imagen los tres tipos de eventos que se pueden llegar a recibir en esta EDA:

- Evento simple: se trata de un evento simple donde se detecta su cambio de estado. Por ejemplo, la detección de la pulsación del botón de emergencia.
- Correlación de eventos: en este caso, dos o más eventos tienen que tener valores fuera de su rango para obtener una salida en el procesador de eventos. Por ejemplo, la detección de incendio sólo se detecta si llegan al procesador de eventos un evento de gas nocivo y de incremento brusco de temperatura.
- Ponderación de eventos: se lleva a cabo un promedio de los eventos ocurridos en una ventana de tiempo considerable para que el procesador realice las acciones oportunas, así como la detección y/o visualización de los datos.

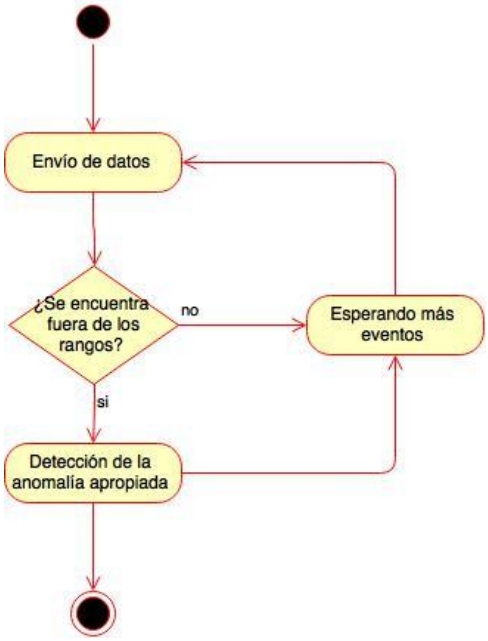
Se observan los tres casos de eventos que se pueden tener, donde cada flecha indica el resultado esperado a la salida del CEP. El rectángulo propio de la tensión determina que se han enviado un promedio de datos de los eventos.



5.2. Validación

Para obtener una validación acorde con todos los elementos del sistema completo, se deberán tener en cuenta los parámetros de entrada y los parámetros de salida correspondientes a los primeros. Es decir, cada sensor deberá obtener la respuesta adecuada en relación a su entrada. En consecuencia, se llevará a cabo una combinación lógica de eventos y mensajes para que el procesador de eventos complejos produzca una respuesta concreta cuando lleguen ciertos eventos específicos a cada cola de mensajería. Se muestra una tabla resumen de los datos de entrada y de sus posibles combinaciones.

Sin embargo, para poder tener una perspectiva más global del funcionamiento del sistema, se ha decidido adjuntar un diagrama de actividad del sistema.



Este diagrama muestra un estado inicial de recogida de datos, donde posteriormente se envían al procesador. Si esos datos o sus combinaciones oportunas ya preestablecidas para su lectura se encuentran fuera de los rangos definidos, se lleva a cabo la detección de la anomalía en los mismos y la comunicación al consumidor de ello. Si no se detecta anomalía o se ha terminado su detección se vuelve al estado inicial de escucha de recepción de datos. En el caso concreto del despliegue de la aplicación se ha decidido dejar un intervalo de tiempo de cinco segundos entre el envío de un evento y el siguiente.

DATOS DE ENTRADA	CONDICIÓN	DATOS DE SALIDA
- Incremento de Temperatura Interna - Gas	- Incremento Brusco de Tª Interna=true - Gas=true	Detección de incendio
- Botón del pánico	- Botón=true	Detección situación de emergencia mediante la pulsación del botón rojo
- Tensión Mínima - Tensión Máxima	- T Mínima > 8 - T Máxima > 13	Tensión Alta
- Tensión Mínima - Tensión Máxima	- T Mínima < 6.5 - T Máxima < 11.5	Tensión Baja
- Ritmo Cardiaco - Nivel de Glucosa	- Ritmo > 80 - Nivel Glucosa > 110	Ritmo Cardiaco Alto, Nivel de Azúcar Alto. Taquicardia

- Ritmo Cardiaco - Nivel de Glucosa	- Ritmo < 60 - Nivel Glucosa < 75	Ritmo Cardiaco Bajo, Nivel de Azúcar Bajo. Bradicardia
-Acelerómetro -Nivel de glucosa	- Nivel de glucosa < 75 -Ritmo < 60 - Acelerómetro=true	Caída del paciente
-Tensión Mínima -Tensión Máxima	-Promedio	Promedio de cada tensión con varios productores

A la vista de esta tabla, se deben tener algunas consideraciones.

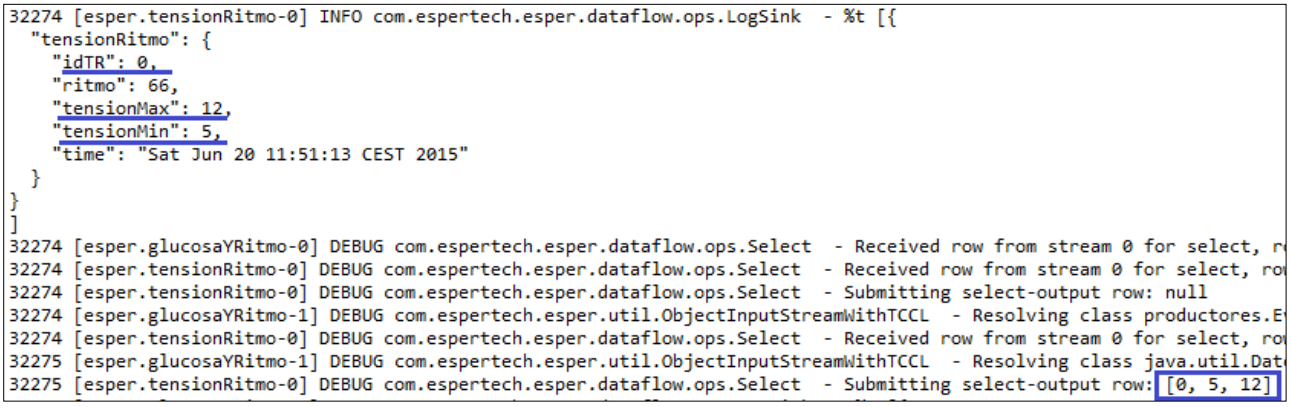
En primer lugar, para que se den esas condiciones y esos resultados a la salida del sistema no es necesario que solamente tengamos esos datos de entrada. Es decir, el procesador de eventos recibirá constantemente datos de todos los productores. Sin embargo, para obtener los datos de salida, se deben cumplir esas condiciones concretas en los datos especificados para ello. También se debe tener en cuenta el id de cada usuario, ya que las condiciones deben pertenecer a un solo usuario.

En segundo lugar, no todas las condiciones son necesarias para que se obtenga la información deseada a la salida. Así, se puede observar que hay dos vertientes:

- Condiciones que se deben cumplir de forma simultánea. Estas condiciones son las relacionadas con la temperatura, la temperatura y el gas, y el ritmo cardiaco y el nivel de glucosa. De esta forma, se deben dar ambas condiciones en cada caso simultáneamente, para poder llegar a obtener resultados coherentes y con una base lógica. Por ejemplo, para accionar el aire acondicionado, la temperatura externa debe ser mayor que la interna del edificio, porque en caso contrario no tiene mucho sentido activar dicho dispositivo. Otro ejemplo es el del ritmo cardiaco con el nivel de glucosa, ya que sus valores son dependientes entre sí.
- Condiciones que se deben dar por separado. En el caso de la medida de la tensión de un paciente, ésta tiene dos valores, mínima y máxima, y ambos son importantes para el análisis de resultados. Si cualquiera de estos valores presenta un dato anómalo el procesador de eventos dará noticia de ello. Por ejemplo, si una persona tiene la tensión mínima en 9 y la tensión máxima en 12, el procesador de eventos mandará un mensaje de alerta por tensión alta en el paciente. Nótese que solamente uno de los parámetros presenta valores fuera de su rango, lo que resulta bastante coherente en relación a la lectura de parámetros médicos. También pueden darse simultáneamente anomalías en ambas tensiones

Por otro lado, para verificar que el sistema completo funciona y que las condiciones se cumplen, se muestran una captura de pantalla realizada a la terminal de salida de datos del entorno de desarrollo Eclipse.

Concretamente, se ha elegido la detección de tensión baja. Tal y como se puede observar, no se cumplen las condiciones para un valor de los datos normalizado, por lo que se detecta como una anomalía. En la primera de ellas se puede observar como la tensión no está dentro de su rango definido y el procesador de eventos lo detecta como una anomalía del paciente por tensión baja.



Detección de tensión baja (5,12) en el paciente con identificador 0

5.3. Principales resultados

Los principales resultados de la arquitectura dirigida por eventos se reflejan a la salida del procesador de eventos, que ya ha tomado las decisiones oportunas respecto a los datos de entrada. Los productores y publicadores se encargan de recoger la información de los sensores externos del servicio de Tele-asistencia. En consecuencia, los resultados acordes a cada uno de los datos de entrada y sus combinaciones se rigen por las reglas preestablecidas del CEP, aquellas descritas en la tabla del apartado anterior. En definitiva, los principales resultados se deben ajustar a un marco lógico y coherente para obtener resultados que concuerden con el objetivo del sistema global.

Respecto a la validación del epígrafe anterior, es interesante comentar los resultados esperados de los datos de entrada. Teniendo en cuenta que los productores llevan a cabo una generación de números pseudoaleatorios (se tratan de números aleatorios dentro de un rango específico), los resultados a la salida del procesador de eventos no van a tener siempre un valor concreto, es decir, si se encuentran dentro de los rangos en los que los datos son lógicos y no representan una anomalía, el sistema devuelve *null* como resultado del sistema. Esa respuesta "vacía" se concibe como una respuesta del procesador a la ausencia de anomalías en el paciente, incluso teniendo en cuenta las combinaciones posibles del sistema. También se han implementado productores con valores fijos, para verificar la detección de las anomalías adecuadas.

CAPÍTULO 6. Conclusiones y trabajos futuros

6.1. Conclusiones

Como conclusión general se puede advertir que los patrones arquitectónicos de software solventan problemas de gran envergadura a los desarrolladores y ayudan a entender al usuario el esquema general de la arquitectura software. Asimismo, algunos parámetros necesitan ser procesados y definidos mediante un patrón concreto, debido a las necesidades específicas que requiera el sistema en cuestión. Es por ello que las arquitecturas software tienen unas configuraciones muy específicas para poder enfocarse en un requisito no funcional concreto. En consecuencia, su utilidad reside en la elección de un patrón según los requisitos no funcionales del sistema a implementar.

Por otro lado, la red de dispositivos de IoT se especializa en ofrecer servicios específicos a los objetos y dispositivos interconexionados en la misma, llevando a cabo la conexión a través de Internet. En consecuencia, este concepto ha supuesto una revolución en cuanto a las comunicaciones y sus tecnologías. Además, la cantidad de servicios desplegados en esta red es muy amplia, destacando el servicio personalizado de Tele-asistencia estudiado en este TFG, que brinda al usuario beneficios y facilidades para la monitorización de sus parámetros saludables. En consecuencia, el estudio de este nuevo concepto ha demostrado que Internet es muy útil para realizar todo tipo de conexiones en tiempo real, y es la arquitectura orientada a eventos la idónea para ello.

Con todo el estudio del arte y la versión de una arquitectura a eventos sencilla desarrollada mediante las herramientas descritas, se ha concluido que una EDA es aquella que representa de una forma exhaustiva y rigurosa todos los elementos del sistema de intercambio de mensajes, además de notificar los eventos en tiempo real. También realiza una lectura inteligente de los eventos, teniendo en cuenta una lógica preestablecida y una combinación de dichos eventos acorde con la respuesta esperada. En el caso de estudio de este TFG se ha verificado la importancia del tratamiento de información y la detección de posibles anomalías en los datos, debido a la importancia de, por ejemplo, detectar un desmayo de un paciente en plena calle. El servicio personalizado de Tele-asistencia es un claro ejemplo de la importancia de una EDA dentro de los servicios desplegados sobre la red de dispositivos y terminales de IoT.

En definitiva, una arquitectura orientada a eventos es una de las mejores formas de procesar los mensajes y acontecimientos que suceden a nuestro alrededor. Además, se trata de una opción bien fundamentada para poder abordar el nuevo concepto de Internet de las Cosas, ya que la sociedad empieza a tener un conocimiento del mismo, por la infinidad de aplicaciones software y beneficios tangibles para el usuario.

6.2. Trabajos futuros

Los trabajos futuros y proyecciones de este tipo de arquitecturas puede llegar a tener muchas más aplicaciones de las que realmente se puedan imaginar.

En relación a las proyecciones de este TFG, es importante destacar la implementación de la arquitectura desarrollada en un sistema operativo propio de los teléfonos inteligentes. En consecuencia, se desplegaría sobre el sistema operativo Android, muy comercializado en la actualidad y con características similares en el lenguaje de programación Java. Además, se procederían a realizar pruebas formales sobre el prototipo, usando Selenium como herramienta enfocada a verificar el sistema en Java. Por último, también se llevaría a cabo la extensión de la funcionalidad del prototipo, pudiendo abarcar muchas más medidas del paciente, como, por ejemplo, su estado de ánimo, con las nuevas aplicaciones sobre detección de sentimientos y emociones, y así evitar una futura depresión del usuario.

Una de las aplicaciones futuras de una EDA general puede ser la especialización del tipo de eventos y el público al que se dirige dicha aplicación. Tal y como se ha mostrado en este TFG, una parte importante de la búsqueda de aplicaciones se centran en aquellas dirigidas hacia la salud de los usuarios. Una arquitectura orientada a eventos esencial en la sociedad actual es aquella que ofrece un sistema de Tele-asistencia sanitaria personalizado, donde se informa y procesa la información precisa recogida por una serie de sensores y dispositivos externos. Otro ejemplo es el sistema bancario, donde el volumen de información económica es enorme y sigue aumentando, siendo complicado informar de los cambios realizados a los profesionales interesados. De esta forma, una EDA resulta idónea para poder procesar esos eventos, donde cualquier cambio de los parámetros económicos puede repercutir de forma drástica en las decisiones a tomar.

Otra futura aplicación es la definición de estándares de eventos de cualquier tipo de sistema, es decir, al igual que se definen patrones arquitectónicos de software para distintos sistemas, se pueden definir patrones de diferentes eventos para incorporar en las EDA que las requieran. Asimismo, si se tiene una entidad grande que incorpore una EDA para centralizar la información de los eventos que le lleguen puede agregar dichos patrones de definición de eventos para poder dar homogeneidad al sistema. Esto se puede transponer a diferentes áreas geográficas de actuación y a diferentes ámbitos, donde se disponga de la definición de eventos oportuna para ese sistema.

Finalmente, las arquitecturas dirigidas por eventos tienen múltiples ámbitos de actuación y su procesamiento de la información en tiempo real y de forma asíncrona. Esto hace de ellas un referente para el intercambio de información, y más aún con el nuevo concepto de Internet de las Cosas en pleno auge en la sociedad actual. El prototipo de este TFG se puede ampliar a un servicio de Tele-asistencia aún más personalizado, donde se detecten los sentimiento y emociones del usuario.

CAPÍTULO 7. Bibliografía y recursos web consultados

[1] "Arquitectura dirigida por eventos". Noviembre de 2014. Disponible en: http://arquitectura-software56.blogspot.com.es/2014/11/arquitectura-dirigida-por-eventos.html

[2] "Transparencias de Arquitectura dirigida a Eventos. Beneficios". Diciembre de 2012. Disponible en: http://es.slideshare.net/rehoscript/arquitectura-dirigida-a-eventos

[3] "Capas del flujo de eventos". "Estilo de procesamiento de Eventos". Disponible en: https://es.wikipedia.org/wiki/Arquitectura_dirigida_por_eventos

[4] "Patrones de arquitectura vs Patrones de diseño". Agosto de 2012. Disponible en: https://arlethparedes.wordpress.com/2012/08/27/patrones-de-arquitectura-vs-patrones-de-diseno/

[5] Búsqueda de las arquitecturas software más comunes. Disponible en: https://es.wikipedia.org/wiki/Arquitectura_de_software

[6] "El modelo cliente-servidor" Disponible en: http://neo.lcc.uma.es/evirtual/cdd/tutorial/aplicacion/cliente-servidor.html

[7] "Modelo cliente-servidor". Búsqueda de las ventajas e inconvenientes. Disponible en: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/marquez_a_bm/capitulo5.pdf

[8] "EDA versus CEP and SOA". Octubre de 2088. Disponible en: http://soa-eda.blogspot.com.es/2008/10/eda-versus-cep-and-soa.html

[9] "SOA, Arquitectura Orientada a Servicios". Búsqueda de las diferencias con EDA. Disponible en: http://antoniogalanvazquez.blogspot.com.es/2007/09/soa-arquitectura-orientada-servicios.html

[10] "Composición de Servicios". Búsqueda de las relaciones de orquestación y coreografía. Diapositivas proporcionadas del moodle del Departamento de Ingeniería de Sistemas Telemáticos. Asignatura: Ingeniería de Sistemas y Servicios Telemáticos.

[11] "Arquitectura cliente-servidor multinivel para Java EE" Disponible en: http://docs.oracle.com/javaee/6/tutorial/doc/bnaay.html

[12] "Patrones Arquitectónicos". Búsqueda de información del resto de patrones. Disponible en: https://ingeniods.wordpress.com/tag/arquitectura-dirigida-por-eventos/

[13] "Tema 2.1. Requisitos del software". Búsqueda de los principales requisitos no funcionales de los sistemas software. Disponible en: moodle de Ingeniería de Servicios y Sistemas Telemáticos (ISST): https://www.moodle.lab.dit.upm.es

[14] "Requisitos del software. Tipos". Curso 2005-2006. Disponible en: http://www.ie.inf.uc3m.es/grupo/docencia/reglada/psi/unidad6-DOC.pdf

[15] "La importancia de centralizar la información de seguridad a nivel corporativo". Disponible en: http://www.recursos-as400.com/tendenciesit004.shtml

[16] "Event Proccessing". Búsqueda del funcionamiento general de una EDA y sus elementos. Disponible en: http://www.manning.com/etzion/SampleCh-01.pdf

[17] "Internet de las Cosas, el próximo paso en la evolución de los sistemas inteligentes". Mayo 2015. Disponible en: http://rootear.com/web/internet-of-things

[18] "¿Qué es el Internet de las Cosas?". Búsqueda de su funcionamiento básico. http://searchdatacenter.techtarget.com/es/cronica/Explicacion-Que-es-la-internet-de-las-cosas

[19] Búsqueda de historia e Internet 0. Accesibilidad universal. Disponible en: https://es.wikipedia.org/wiki/Internet_de_las_cosas

[20] "Internet de las Cosas". Búsqueda de ejemplos en la sociedad moderna. Disponible en: http://www.areatecnologia.com/nuevas-tecnologias/internet-de-las-cosas.html

[21] "¿Qué es Internet de las Cosas?". Búsqueda de los problemas de seguridad. Disp. en:http://www.pcactual.com/articulo/actualidad/noticias/13647/que_internet_las_cosas.html

[22] Búsqueda de imagen que relacione EDA e Internet de las Cosas. Disponible en: www.nbdigitalmedia.com

[23] "Arquitecturas orientadas a eventos". Búsqueda de las principales herramientas para EDA. Disponible en: Transparencias proporcionadas por D. Juan Carlos Yelmo García del Máster Universitario de Ingeniería de Telecomunicación. Asignatura: APSV

[24] "Arquitectura MOM basada en el portafolio de servidores de aplicación Oracle". Disponible en: http://www.oracle.com/technetwork/es/articles/soa/arquitectura-mom-integracion-server-426198-esa.html

[25] Message-Oriented Middleware (MOM). Technical Overview. Disponible en: http://docs.oracle.com/cd/E18930_01/html/821-2443/aeraq.html

[26] IBM-MQ. Disponible en: http://www-03.ibm.com/software/products/es/ibm-mq

[27] ActiveMQ. Disponible en: http://activemq.apache.org/

[28] www.rabbitmq.com

[29] www.espertech.com

[30] "Programación orientada a eventos. Visual Basic". Disponible en: http://www.monografias.com/trabajos/progeventos/progeventos.shtml

[31] Búsqueda de tutorial de Esper. Complementación con la página oficial de Esper https://unpocodejava.wordpress.com/2012/01/22/un-poco-de-esper/

[32] Búsqueda de información de AMQP. Complementación a RabbitMQ y Esper. Disponible en: https://azure.microsoft.com/es-es/documentation/articles/service-bus-amqp-overview/

[33] "Lenguajes de programación orientada a eventos". Disponible en: http://www.larevistainformatica.com/Lenguajes-programacion-orientado-eventos.htm

[34] Tutoriales de RabbitMQ. Disponible en: http://www.rabbitmq.com/getstarted.html

[35] Tutorial de Esper. Disponible en: http://www.espertech.com/esper/quickstart.php

[36] "Tecnología Push. Así funciona" Disponible en: http://www.genbeta.com/a-fondo/tecnologia-push-asi-funciona

[37] "Monitorización del negocio en tiempo real (CEP)" Disponible en: http://www.tcpsi.com/servicios/Soluciones_Monitorizacion_del_negocio_en_tiempo_real.htm