Máster Universitario de Investigación en
Ingeniería de Software y Sistemas Informáticos

Implementación del protocolo CoAP en Xtend

Trabajo Fin de Máster
Itinerario de Ingeniería de Software (Código: 31105128)

AUTOR: César Estebas Gómez

DIRECTORA: Mª Magdalena Arcilla Cobián

Curso Académico 2013/2014
Convocatoria Septiembre

Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos

ITINERARIO: Ingeniería del Software

CÓDIGO DE ASIGNATURA: 31105128

TÍTULO DEL TRABAJO: Implementación del protocolo CoAP en Xtend

TIPO DE TRABAJO: Tipo A, proyecto específico propuesto por un profesor

AUTOR: César Estebas Gómez

DIRECTORA: Mª Magdalena Arcilla Cobián

Impreso TFdM05_Autor. Autorización de publicación

y difusión del TFdM para fines académicos

Autorización

Autorizo/amos a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del/los Autor/es

César Estebas Gómez

Juan del Rosal, 16
28040, Madrid

Tel: 91 398 89 10
Fax: 91 398 89 09

www.issi.uned.es

Resumen

En el ámbito del internet de las cosas y la comunicación máquina a máquina se hace necesario trabajar sobre protocolos diferentes a los utilizados convencionalmente en internet. El que se trata en este trabajo es el protocolo CoAP, Constrained Application Protocol, utilizado para comunicar y controlar pequeños dispositivos como sensores, interruptores o válvulas.

Se explican las características principales del protocolo y se comentan algunos ejemplos de implementaciones de este en diferentes lenguajes de programación.

Una vez tratadas las diferentes implementaciones existentes, se plantea una posible alternativa desarrollada en el lenguaje Xtend. Para ello se ha dividido el desarrollo en las fases clásicas de desarrollo de software como son las fases de análisis de requisitos, diseño del software, implementación, pruebas e implantación.

Una vez implementado el protocolo, se muestran una serie de ejemplos de uso y se comentan las conclusiones alcanzadas tras la realización del trabajo.

Palabras clave: Internet de las cosas, IoT, comunicación máquina a máquina. M2M, CoAP, Xtend.

Abstract

In the field of the internet of things and machine to machine communication is necessary to work on different protocols to those used conventionally in internet. Which is discussed in this paper is the CoAP protocol, Constrained Application Protocol, used to communicate and control small devices such as sensors, switches or valves.

There are commented the main features of the protocol and examples of implementations in different programming languages.

Once treated the different existing implementations, it is considered a possible alternative developed in the language Xtend. The development has been divided into the classic phases of software development that are the phases of requirements analysis, software design, implementation, testing and deployment.

Once the protocol is implemented, there are explained some examples of use and the conclusions reached following completion of the work are discussed.

Keywords: Internet of things, IoT, machine to machine communication. M2M, CoAP, Xtend.

Índice

Índice de ilustraciones

Índice de Tablas

1. Introducción

La información escrita en esta memoria corresponde al Trabajo Fin de Máster del Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos realizado por el alumno César Estebas Gómez en el curso académico 2013/2014.

El trabajo ha consistido en la investigación y desarrollo de una implementación del protocolo CoAP (Constrained Application Protocol) para comunicación M2M (Máquina a Máquina) en el lenguaje Xtend, un lenguaje desarrollado por Eclipse pensado para aumentar la potencia y facilidad de uso de Java. Para ello se han aplicado conocimientos adquiridos en asignaturas cursadas en el Máster.

En la primera parte de la memoria, se comenta la motivación inicial del trabajo y se habla de la importancia del protocolo CoAP en el ámbito del IoT (Internet de las cosas) y la comunicación M2M. Se exponen ejemplos de implementación del protocolo en diferentes lenguajes y se justifica el uso de Xtend para la implementación en este trabajo.

En la segunda parte, se explica la implementación del protocolo, realizando una planificación previa y planteando una serie de objetivos a cumplir tras el desarrollo. Tras fijar los objetivos, se procede al desarrollo del software, siguiendo las fases clásicas de la ingeniería de software.

Una vez implementado el protocolo en Xtend, se muestran ejemplos de uso del mismo en comunicaciones cliente/servidor y con proxys CoAP/HTTP y HTTP/CoAP.

En el apartado final de la memoria se comentan las conclusiones alcanzadas tras la realización del Trabajo Fin de Máster y las posibles vías de trabajo futuras.

1.1. Motivación del trabajo

La motivación surgió de la necesidad de realizar el Trabajo Fin de Máster a partir de conocimientos obtenidos por el alumno al cursar asignaturas del Máster. Para ello, el alumno solicitó realizar el trabajo a partir de una propuesta del profesor Ismael Abad Cardiel. La idea inicial del trabajo era realizar una aplicación Xtend basada en los estándares del OMA (Open Mobile Alliance) de comunicación M2M, pero debido a la amplitud del tema abarcado, se decidió centrar el trabajo en la el protocolo CoAP, parte superior del protocolo LWM2M (Lightweight Machine to Machine) del OMA, y realizar una implementación de este en Xtend.

El alumno ya realizó un trabajo sobre el uso del lenguaje de programación Xtend para la generación de código para la asignatura Generación Automática de Código, por lo que ya está familiarizado con el lenguaje. También cursó las asignaturas Computación Ubicua y Arquitecturas Orientadas a Servicios que le permitieron adquirir conocimientos sobre el IoT y las aplicaciones basadas en servicios.

Aplicando los conocimientos obtenidos tras cursar estas asignaturas, el alumno será capaz de desarrollar una implementación del protocolo CoAP en el lenguaje de programación Xtend, buscando la originalidad al acercar áreas tecnológicas en auge como la generación automática de código, el IoT y las arquitecturas orientadas a servicios.

1.2. El IoT y la comunicación M2M

Los avances en computación ubicua y las mejoras en las comunicaciones llevan a la sociedad hacia un mundo en el que todo esté comunicado. Desde las grandes computadoras a los aparatos más simples podrán conectarse a internet y comunicarse.

El concepto de IoT se refiere a esta interconexión entre objetos de uso cotidiano como pueden ser un libro o un frigorífico. Si estos objetos estuvieran conectados, podrían ser gestionados por otros equipos como lo haría un ser humano. Esto facilitaría la gestión de los datos y la información, ya que como comenta Ashton en [1], si existieran ordenadores que supieran todo lo que tuvieran que saber sobre las "cosas", mediante el uso de datos que ellos mismos pudieran recoger sin ayuda, se podría monitorizar, contar y localizar todo a nuestro alrededor, de esta manera se reducirían increíblemente gastos, pérdidas y costes. El Internet de las Cosas tiene el potencial para cambiar el mundo tal y como hizo la revolución digital hace unas décadas.

Como puede verse en la Ilustración 1, el IoT puede aplicarse en diferentes sectores como el de la salud, el transporte o la energía.

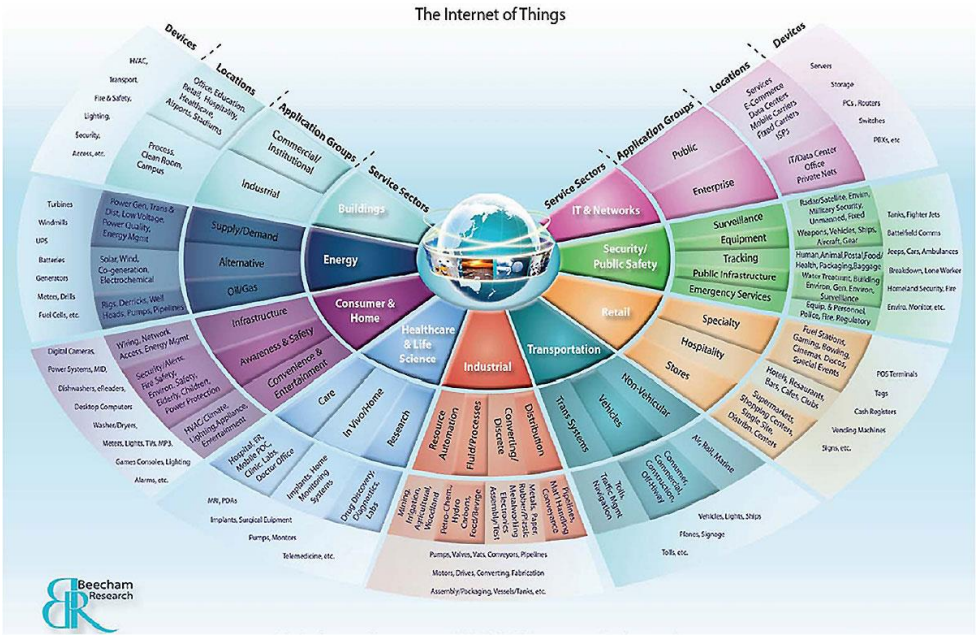


Ilustración 1 Diagrama del ecosistema IoT en diferentes sectores de la industria[1]

[1] Vía http://www.symplio.com/2011/09/4-infographics-about-internet-of-things/

Un elemento importante del IoT es la comunicación máquina a máquina. Para que todo esté correctamente interconectado, esta debe realizarse siguiendo una serie de normas y estándares. Aunque la investigación en este campo es relativamente joven, varias empresas y entidades han propuesto diferentes estándares para la comunicación.

Gracias al avance de las conexiones, los incluso pequeños dispositivos como sensores pueden tener dirección IP propia y pueden ofrecer servicios. Al necesitarse un rango mayor de IPs, no puede utilizarse IPV4, debe utilizarse IPV6. Para estandarizar las conexiones, el IETF (Internet Engineering Task Force) ha definido una alternativa en forma de estándar para que estos dispositivos estén realmente conectados a Internet, más concretamente a la Internet6. Se conoce con el nombre de "6LowPAN", el IPv6 de las redes LowPAN (Low power Wireless Personal Area Networks).

En dispositivos de cierta entidad y alimentados mediante red eléctrica es posible utilizar el modelo TCP/IP sobre redes WiFi o 3G, y que el dispositivo implemente los conocidos servicios WEB de tipo REST (Representational State Transfer) para su manejo. El problema surge al conectar estos dispositivos o sensores conectados a baterías o redes mayores que deben conectarse por redes de baja potencia como LowPAN. El problema de estas redes es que son propensas a perder gran cantidad de mensajes, por lo que el uso de HTTP (Hypertext Transfer Protocol) sobre TCP/IP se presenta ineficaz.

Como alternativa al modelo HTTP-REST sobre TCP/IP, aparece el modelo CoAP-REST sobre UDP (User Datagram Protocol). Entre las entidades que promueven su uso destaca el OMA y con su protocolo LWM2M [2], pensado para el control y la comunicación M2M de pequeños dispositivos. En la capa de aplicación utiliza el protocolo CoAP del que se habla en el siguiente apartado del trabajo.

1.3. El protocolo CoAP

El protocolo CoAP es un protocolo de la capa de aplicación pensado para trabajar en entornos restringidos. Utiliza UDP en la capa de transporte. El uso de este protocolo se hace necesario para conectar pequeños dispositivos a internet, ya que estos normalmente operan en redes LowPAN con mucha pérdida de mensajes y el uso del protocolo HTTP sería ineficiente.

CoAP mantiene el modelo REST y las operaciones básicas de HTTP (GET, PUT, POST y DELETE), para facilitar una conexión entre los dos protocolos.

En la Ilustración 2 puede verse una comparativa entre los modelos HTTP-REST sobre TCP/IP y CoAP sobre UDP.

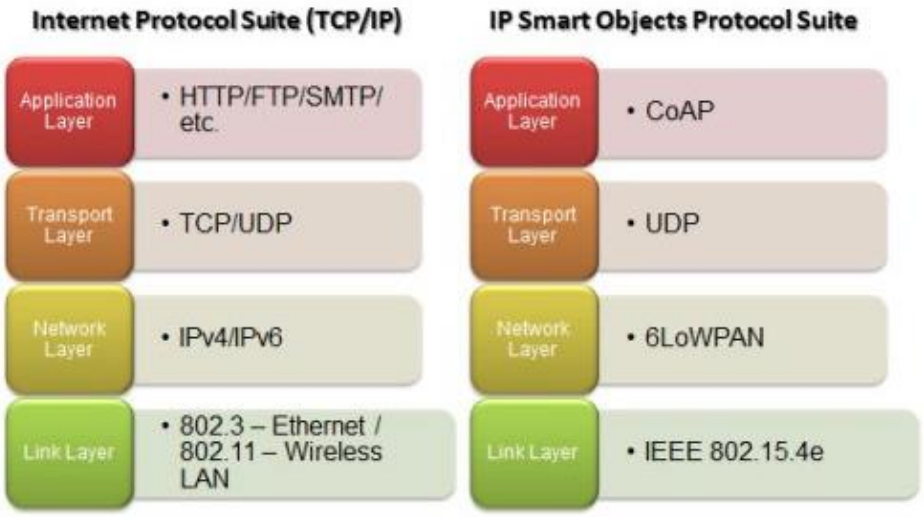
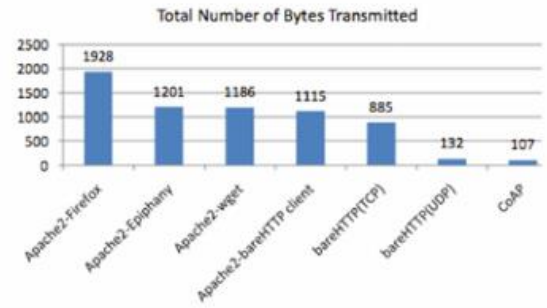


Ilustración 2 Comparativa entre modelos HTTP-REST y CoAP/UDP[2]

CoAP ofrece también una gran optimización del manejo de información enviada. En la ilustración 3 puede observarse una comparativa entre mensajes enviados en redes LowPAN sobre CoAP/UDP y REST-HTTP.

[2] Via Internet of Things Protocols & Standards - http://postscapes.com/internet-of-things-protocols



Access Method	Time (sec)
Apache2-Firefox	38.774
Apache2-Epiphany	31.972
Apache2-wget	2.660
Apache2-bareHTTP client	3.032
bareHTTP(TCP)	3.076
bareHTTP(UDP)	1.104
CoAP	1.029

Header	HTTP/TCP	HTTP/UDP	CoAP/UDP
Link Layer	160	32	32
IP	200	40	40
TCP/UDP	340	16	16
HTTP/CoAP	181	41	17
Data	4	4	2

Ilustración 3 Mensajes enviados mediante CoAP/UDP y HTTP-REST[3]

Como puede observarse, con el protocolo CoAP se retransmiten un número mucho menor de Bytes y en mucho menos tiempo, lo que supone una ventaja para su uso en pequeños dispositivos con poca memoria de almacenamiento.

En una red restringida, los sensores actúan como servidores CoAP que ofrecen servicios. A la información de estos servidores se podrá acceder mediante el protocolo CoAP o a través de un proxy HTTP-CoAP. De esta forma cualquier aplicación podrá acceder a la información del dispositivo.

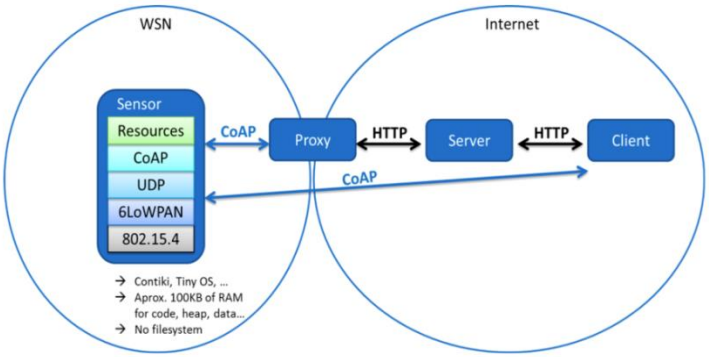


Ilustración 4 Conexión entre cliente HTTP y servidor CoAP

También es posible que estos sensores se conecten a recursos HTTP mediante un proxy CoAP-HTTP.

[3] Via http://blogthinkbig.com/ipv6-motor-internet-de-las-cosas-iot/

La especificación completa del protocolo CoAP puede consultarse en [3], y se apoya en [4], [5] y [6], pero a continuación se resaltan las partes más importantes.

Funcionalidades del protocolo CoAP

Las funcionalidades más importantes del protocolo CoAP son las siguientes:

- Diseño REST que facilita el mapeo con el protocolo HTTP.
- Soporta el uso de las opciones URI y Content-type.
- Soporte para el descubrimiento de los recursos proporcionados por los servicios de COAP conocidos. Uso del método DISCOVER.
- Suscripción simple para un recurso con recepción de mensajes cuando este cambia. Uso del método OBSEERVE.

Tipos de mensajes

CoAP hace uso de dos tipos de mensajes, peticiones y respuestas. Las cabeceras de los mensajes se codifican en binario, comenzando por un encabezado base y seguido por una serie de opciones. El resto del mensaje tras la cabecera, se considera el cuerpo del mensaje.

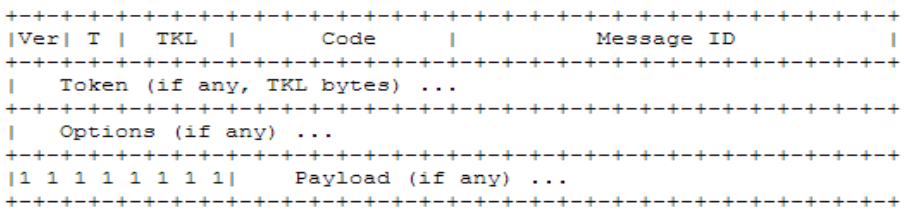


Ilustración 5 Mensaje CoAP

En la ilustración 5 se observa el modelo del mensaje CoAP. En él se observan la cabecera con los valores versión, tipo, longitud del token, código, ID del mensaje y Token. Tras esto aparecen las opciones y finalmente el cuerpo del mensaje.

Identificación de mensajes

Para identificar las peticiones con las respuestas se utilizan el Token y el identificador de mensaje.

El Token es una cadena de bytes generada en la petición. Si la respuesta se envía contestando al mensaje de la petición el Token debe ser el mismo. En caso de que la respuesta sea independiente de la petición, el Token será diferente.

No ocurre lo mismo con el identificador del mensaje, ya que una petición y su respuesta tendrán siempre el mismo identificador.

Códigos de respuesta

CoAP dispone de una lista de códigos de respuesta similar a la de HTTP. Los códigos pueden ser de tres tipos:

- 2.XX: Éxito.
- 4. XX: Error del cliente.
- 5. XX: Error del servidor.

La lista de códigos de respuesta es la siguiente:

Code	Description	Reference
2.01	Created	[RFC7252]
2.02	Deleted	[RFC7252]
2.03	Valid	[RFC7252]
2.04	Changed	[RFC7252]
2.05	Content	[RFC7252]
4.00	Bad Request	[RFC7252]
4.01	Unauthorized	[RFC7252]
4.02	Bad Option	[RFC7252]
4.03	Forbidden	[RFC7252]
4.04	Not Found	[RFC7252]
4.05	Method Not Allowed	[RFC7252]
4.06	Not Acceptable	[RFC7252]
4.12	Precondition Failed	[RFC7252]
4.13	Request Entity Too Large	[RFC7252]
4.15	Unsupported Content-Format	[RFC7252]
5.00	Internal Server Error	[RFC7252]
5.01	Not Implemented	[RFC7252]
5.02	Bad Gateway	[RFC7252]
5.03	Service Unavailable	[RFC7252]
5.04	Gateway Timeout	[RFC7252]
5.05	Proxying Not Supported	[RFC7252]

Ilustración 6 Códigos de respuesta CoAP

Ejemplos de uso de CoAP

A continuación se muestra un ejemplo básico, aunque bastante demostrativo, del uso del protocolo CoAP en una interacción entre elementos conectados al IoT.

Para realizar el ejemplo únicamente se necesitan un sensor que capte la temperatura de color y el nivel de luz ambiental y una lámpara. Ambos dispositivos podrán conectarse a internet, por lo que podrán comunicarse.

El sensor tendrá un recurso mediante el cual informará de los datos actuales de la luz. Mediante el protocolo CoAP, la lámpara podrá realizar una petición a este recurso y el sensor responderá con los datos almacenados.

Esta interacción es aún mejor si la lámpara "observa" el recurso con una petición OBSERVE y recibe un mensaje con la información sobre la luz cada vez que esta cambie.

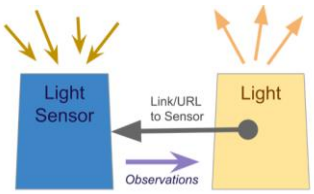


Ilustración 7 Interacción entre una lámpara y un sensor

El mismo ejemplo de la lámpara puede trasladarse a otros escenarios como por ejemplo sistemas de riego automáticos que comprueban el nivel de humedad o sistemas de aire acondicionado.

Un ejemplo de uso comercial del protocolo puede observarse en el servicio Zatar, desarrollado por Zebra Technologies[4], empresa pionera en el desarrollo de etiquetas inteligentes, para sus lectores de etiquetas de radio frecuencia. Los lectores no se comunican con las etiquetas mediante el protocolo HTTP, sino que lo hacen de forma más rápida y eficiente utilizando el protocolo CoAP.

[4] Página web de Zebra Technologies - http://www.zebra.com/us/en.html

1.4. Soluciones existentes

En la actualidad existen diversas implementaciones del protocolo CoAP que pueden clasificarse en función de las funcionalidades ofrecidas o del lenguaje utilizado para su implementación. A continuación se mostrarán las características de algunas de las más utilizadas.

- libcoap[5]:
  - Desarrollada en C, lo que supone posibilidades de uso en infinidad de dispositivos.
  - Permite el uso de cliente y servidor.
  - Licencia libre.
  - Implementa la opción OBSERVE.

- Californum[6]:
  - Desarrollado en Java y ampliamente extendido.
  - Permite el uso de cliente y servidor.
  - Licencia libre.
  - Implementa la opción OBSERVE.
  - Incluye una capa DTSL de seguridad.
  - En su última versión permite el uso de proxy.

- txThings[7]:
  - Desarrollado en Python.
  - Permite el uso de cliente y servidor.
  - Licencia libre.
  - Implementa la opción OBSERVE de forma parcial.

[5] Enlace al repositorio de libcoap - http://sourceforge.net/projects/libcoap/develop
[6] Enlace al repositorio de Californium - https://github.com/mkovatsc/Californium
[7] Enlace al repositorio de txThings - https://github.com/siskin/txThings/

- ETRI CoAP[8]:
  - Desarrollada en C.
  - Permite el uso de cliente y servidor.
  - Licencia comercial.
  - Implementa la opción OBSERVE.

De entre los ejemplos seleccionados, destaca Californium, ya que implementa la funcionalidad proxy aparte de la de cliente/servidor. Las conexiones serán más seguras ya que utiliza una capa de seguridad DTLS (Datagram Transport Layer Security).

1.5. La solución propuesta

La propuesta de este trabajo es realizar una implementación del protocolo en el lenguaje de programación Xtend.

1.5.1. El lenguaje de programación Xtend

Xtend[9] es un lenguaje de alto nivel para la máquina virtual de Java. Está basado en Java, aunque añade funcionalidades como la inferencia de tipos o la sobrecarga de operadores. El lenguaje Xtend y su IDE son desarrollados como proyecto de Eclipse[10], pero el lenguaje puede ser compilado y ejecutado independientemente de la plataforma Eclipse.

Java es uno de los lenguajes de programación con un uso más extendido. A lo largo de los años se ha desarrollado una gran de librerías y herramientas para el trabajo con el lenguaje. Sin embargo, su sintaxis es muy detallada y existen funcionalidades ofrecidas por otros lenguajes que no las ofrece Java o que son añadidas muy lentamente.

[8] Página web de ETRI CoAP - http://coap.or.kr/index_en.html
[9] Página web del proyecto Xtend - http://www.eclipse.org/xtend/
[10] Página web de Eclipse - https://www.eclipse.org/

Xtend nace con la finalidad de mantener lo mejor de Java, evitando el ruido sintáctico y añadiendo funcionalidades que permiten desarrollar un código más fácil de comprender y potente.

Compila directamente a código Java, lo que hace que sea muy adaptable a proyectos Java. Es compatible con la mayoría de las librerías Java y se puede integrar en cualquier tipo de proyecto, desde aplicaciones de escritorio a aplicaciones Android. Estas características lo convierten en una buena opción para programadores que vienen de Java y buscan nuevas funcionalidades sin perder parte de la funcionalidad de Java.

Xtend puede utilizarse para definir lenguajes de dominio y realizar tareas de generación de código ya que incluye las siguientes características:

- Elimina el ruido sintáctico de Java.
- Permite la extensión de métodos y el uso de expresiones lambda.
- Tiene funcionalidades para la creación de plantillas. Permite una manipulación de textos sencilla y limpia.
- Proporciona inferencia de tipos.
- Ofrece gestión inteligente de espacios en blanco. Esta característica es importante a la hora de generar código.
- Puede utilizarse junto con Xtext para realizar cualquier tarea de generación de código.

Aparte de estas ventajas con respecto a Java, Xtend ofrece otras funcionalidades a tener en cuenta:

- Puede integrarse con el desarrollo Android. Con una librería muy ligera y soporte avanzado de generación de código, ayuda a incrementar la productividad y crear aplicaciones ligeras.
- La integración con Eclipse asegura un buen soporte y amplia documentación.
- Se integra perfectamente con Java 8.

1.5.2. Justificación de la elección de Xtend

A continuación se comentan los motivos de la elección de Xtend para la implementación del protocolo:

- El primer motivo de elección de Xtend es que el alumno ya realizó un trabajo para la asignatura Generación Automática de Código basado en este lenguaje de programación.
- Como se ha comentado en el punto anterior, Xtend es una opción muy válida para la generación de código Java, ya que compila directamente a Java y los ficheros generados se pueden modificar.
- Simplifica el lenguaje, lo que supone una menor inversión de esfuerzos en el desarrollo.
- Permite crear varias clases en un solo fichero, lo que permite reducir el número de ficheros con que se trabaja.
- Una vez desarrollado el protocolo, este podría utilizarse tanto en proyectos Java como en proyectos Xtend, lo que ampliaría su uso.
- El alumno tiene amplia experiencia desarrollando con el IDE Eclipse, por lo que la integración de Xtend con el IDE no supondrá problemas.

1.5.3. Entregables del trabajo

Los entregables del trabajo son los siguientes:

- Código Xtend de la solución propuesta.
- Código Java generado para la solución.
- JavaDoc de la solución propuesta.
- Librería Java para importar en proyectos tanto Java como Xtend.

2. Implementación del protocolo

Una vez planteada la solución, se ha comenzado el desarrollo de la misma. Para ello se ha realizado una planificación previa para definir el alcance de la solución, indicando la funcionalidad que debe tener el software desarrollado. También se han indicado el método y los medios utilizados para alcanzar la solución.

Tras realizar la planificación, se han definido una serie de objetivos que se esperan cumplir tras el desarrollo de la solución.

Una vez realizada la planificación y definidos los objetivos se ha iniciado el desarrollo. Este se ha dividido en las diferentes fases clásicas del desarrollo de software. Primero se ha realizado un análisis de los requisitos del sistema y basado en estos requisitos se ha realizado el diseño del software. A partir de los modelos obtenidos en la fase de diseño se ha procedido a la implementación y una vez finalizada se han realizado pruebas unitarias de diferentes módulos. Finalmente se han realizado las tareas necesarias para implantar el software.

Para el desarrollo, se han consultado las referencias [7], [8]. [9] y [10].

2.1. Planificación

En este apartado se muestra el resultado de la planificación del trabajo indicando el método y los medios a utilizar, el alcance y el número de horas asignadas a cada tarea.

2.1.1. Método

Antes de comenzar con el desarrollo del software, se ha buscado información sobre el protocolo CoAP y las diferentes implementaciones que existen actualmente. A partir de estas y siguiendo la documentación oficial del protocolo CoAP se ha implementado la versión Xtend del protocolo.

Una vez realizada la implementación, se han desarrollado algunos ejemplos de uso para comprobar su funcionamiento.

2.1.2. Medios

Los medios utilizados para desarrollar el proyecto han sido los siguientes:

- Para el desarrollo del software, un portátil con procesador Intel® Core™ i7-2670QM CPU @ 2.20GHz 2.20 GHz, 8Gb de memoria RAM, conexión a internet y SO Windows 7 Professional N.
- Para los ejemplos, aparte del portátil utilizado para el desarrollo, se ha utilizado un ordenador con procesador Intel Core i7-3770 3.4Ghz Box, 8Gb de memoria RAM, y SO Ubuntu 14.04 LTS.
- Para realizar la memoria se ha utilizado el procesador de textos Microsoft Office Word 2010.
- Para realizar los diagramas de las fases de análisis y diseño se utilizará el software Cacoo[11], una herramienta online gratuita para la realización de diagramas.

[11] Sitio web de Cacoo - https://cacoo.com/

2.1.3. Alcance

Como se ha observado al analizar las soluciones existentes en el mercado, ninguna ofrece toda la funcionalidad que permite el protocolo CoAP. Esto puede deberse principalmente a que la mayoría de implementaciones tienen licencia de software libre y no tienen una funcionalidad completa. También es debido a que al ser un protocolo con documentación extensa y en constante evolución, la funcionalidad cambia, lo que a la larga complica el mantenimiento de un sistema con muchas funcionalidades.

Dado que este trabajo no deja de ser un trabajo académico de 15 créditos ECTS, no debe superar las 375 horas de duración. Este dato unido a que no tiene existe interés de comercialización por parte del alumno, han supuesto la decisión de acotar la funcionalidad implementada en el protocolo.

La funcionalidad del protocolo implementada es la siguiente:

- Se pueden desarrollar servidores CoAP que ofrezcan varios recursos organizado jerárquicamente.
- Se pueden desarrollar aplicaciones cliente CoAP capaces de conectarse a servidores CoAP.
- Un cliente puede utilizar el método OBSERVE para mantenerse informado de los cambios de un recurso.
- El servidor responde al método DISCOVER devolviendo la lista de recursos disponibles en el servidor.
- Se ofrecen implementaciones básicas de proxys CoAP a HTTP y HTTP a CoAP.

La funcionalidad no implementada es la siguiente:

- Posibilidad de enviar peticiones multicast.
- Implementación de una capa DTLS de seguridad.

- Envío de mensajes por bloques.
- Implementación de proxy CoAP a CoAP.

Aunque no se implemente la capa DTLS, se han tomado las medidas pertinentes de seguridad. Tanto esta funcionalidad como las otras no implementadas podrán ser añadidas en versiones futuras del protocolo.

2.1.4. Horas planificadas

Como ya se ha comentado anteriormente, la asignatura Trabajo Fin de Máster tiene un valor de 15 créditos ECTS. Si cada crédito corresponde a 25 horas de trabajo, la extensión máxima en horas del trabajo debe ser de unas 375 horas.

La fecha de inicio del trabajo es el 1 de Julio de 2014, una vez entregados finalizadas el resto de asignaturas del Máster. La fecha tope de entrega es el día 10 de Septiembre de 2014. El calendario de trabajo planificado es el siguiente:

JULIO

L	M	X	J	V	S	D
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

AGOSTO

L	M	X	J	V	S	D
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

SEPTIEMBRE

L	M	X	J	V	S	D
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Tabla 1 Calendario de trabajo

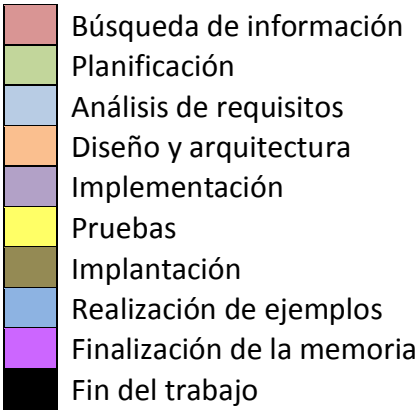


Tabla 2 Leyenda del calendario de trabajo

En la siguiente tabla se muestra la relación de horas planificadas para realizar cada tarea:

TAREA	DÍAS	HORAS AL DIA	TOTAL DE HORAS
Búsqueda de información	6	8	48
Planificación	1	7	7
Análisis de requisitos	8	8	64
Diseño y arquitectura	8	8	64
Implementación	16	8	128
Pruebas	2	8	16
Implantación	1	8	8
Realización de ejemplos	2	8	16
Finalización de la memoria	3	8	24
TOTAL DE HORAS DE TRABAJO			375

Tabla 3 Relación de horas asignadas a cada tarea

Como ha podido observarse, se ha planificado trabajar 8 horas al día de lunes a viernes, excepto el día que se realiza la planificación que se trabaja 7 horas. Se comenzará el día 1 de Julio y si se cumplen los plazos el trabajo estará terminado para el día 4 de Septiembre.

2.2. Objetivos

En este apartado se describen los objetivos que se desean cumplir con la realización del trabajo.

Objetivo principal

El objetivo principal del trabajo es realizar una implementación del protocolo CoAP en Xtend que pueda ser utilizada para desarrollar aplicaciones Cliente/Servidor comunicadas con este protocolo.

Objetivos secundarios

Otros objetivos importantes a cumplir son los siguientes:

- Realizar la funcionalidad para que puedan consultarse los servicios ofrecidos por el servidor mediante el método DISCOVER.
- Implementar la funcionalidad del método OBSERVE [4] para que un cliente pueda conocer los cambios que ocurran en el servicio que esté observando.
- Generar la documentación JavaDoc del software desarrollado. Esta documentación generada se adjuntará al código fuente del protocolo.
- El protocolo se podrá utilizar en otras aplicaciones añadiendo una librería JAVA.
- Desarrollar ejemplos de uso del protocolo para que los usuarios puedan ver ejemplos de clientes, servidores y proxys.

2.3. Fases del desarrollo

En este apartado se describen las diferentes fases en que se ha dividido el desarrollo del software.

2.3.1. Análisis de requisitos

En esta primera fase del desarrollo, se analizan y documentan las necesidades del sistema a desarrollar. Se analizan los requisitos tanto funcionales como no funcionales y los diferentes casos de uso de la aplicación.

Definiciones

Para no crear confusión, primero se van a definir una serie de términos empleados en el desarrollo del sistema.

Cliente: Cualquier aplicación que utilice el protocolo CoAP para conectarse a un servidor.

Servidor: Servidor CoAP que ofrece una serie de recursos.

Proxy CoAP-HTTP: Puerta de enlace entre el protocolo CoAP y el protocolo HTTP.

Proxy HTTP-CoAP: Puerta de enlace entre el protocolo HTTP y el protocolo CoAP.

Recurso: Similar a los recursos y servicios web HTTP. Pequeñas funcionalidades alojadas en el servidor.

Mensaje: Cualquier mensaje enviado entre cliente y servidor.

Petición: Mensaje enviado por un cliente o un proxy HTTP-CoAP a un servidor para interactuar con un recurso.

Respuesta: Mensaje enviado por un servidor a un cliente o proxy HTTP-CoAP con el resultado de la petición.

Token: Cadena de bytes utilizada para identificar las peticiones y sus respuestas.

Código de respuesta: Similar a los códigos de respuesta utilizados en el protocolo HTTP.

Método: Operación a realizar con un recurso. Puede ser de tipo GET, POST, PUT o DELETE. Para interacciones especiales con el servidor también pueden utilizarse los métodos OBSERVE y DISCOVER.

Observador: Cliente que realiza una petición a un recurso con el método OBSERVE. Cada vez que el estado del recurso cambie se le notificará al cliente.

Requisitos funcionales

Los requisitos funcionales son los siguientes:

- Los clientes podrán realizar peticiones a servidores CoAP. Los métodos permitidos en estas peticiones serán: GET, POST, PUT, DELETE, OBSERVE y DISCOVER.
- Los clientes podrán hacer peticiones a recursos HTTP a través de un proxy. El proxy recibirá la petición, generará la petición HTTP a partir de la petición CoAP, realizará la petición HTTP y devolverá el resultado en un mensaje CoAP.
- Podrán añadirse recursos al servidor. También podrán añadirse subrecursos a recursos existentes.
- El servidor permitirá las peticiones de tipo GET, POST, PUT, DELETE, OBSERVE y DISCOVER.

- Los proxys HTTP- CoAP recibirán peticiones HTTP y las convertirán en peticiones CoAP que realizarán al servidor CoAP. Una vez reciban la respuesta la codificarán en un mensaje HTTP y la devolverán.
- Las cabeceras de los mensajes deben seguir la estructura definida en la documentación oficial del protocolo [3].
- Los mensajes de respuesta tendrán un código de respuesta de entre los indicados en la documentación oficial del protocolo [3].
- Cada vez que se modifique un recurso, el servidor enviará un mensaje con el estado actual del recurso a los clientes que estén observando.

Requisitos no funcionales

Los requisitos no funcionales son los siguientes:

- Requisitos de seguridad:
    - Los Tokens generados tendrán una longitud mínima de 5 bytes, para suplir el no implementar una capa DTLS.
    - Las peticiones y respuestas estarán identificadas con el Token y el identificador de mensaje.
- Requisitos de rendimiento:
    - El protocolo desarrollado podrá añadirse a cualquier proyecto Xtend o Java.
    - La comunicación entre los diferentes elementos del sistema debe ser fluida.
    - Para que pueda funcionar una aplicación que utilice el protocolo, el dispositivo debe estar conectado a internet.
- Requisitos de fiabilidad:
    - La aplicación debe ser capaz de recuperarse de los fallos y no perder información.

  - Siempre que se produzca algún error debe ser notificado al usuario para que este conozca el estado en el que se encuentra la aplicación.
  - Si una petición no obtiene respuesta en un lapso de tiempo considerable, se volverá a enviar hasta un máximo de 4 veces. Si el cuarto mensaje enviado no obtiene respuesta, se comunicará al cliente que no se ha podido conectar.
- Facilidad de uso:
  - Existirá una documentación que podrá ser consultada por el desarrollador que utilice el protocolo en sus proyectos.
  - Se implementarán clases base para servidores y proxys con la funcionalidad básica de estos.
  - Se añadirán ejemplos de uso al código de la implementación.

Casos de uso

A partir de los requisitos identificados, se pueden obtener los actores del sistema y los diferentes casos de uso del protocolo.

Los actores identificados son los siguientes:

- Cliente: Actor que representa una aplicación cliente del protocolo CoAP. Podrá hacer peticiones a servidores CoAP, o a servidores HTTP a través de un proxy CoAP-HTTP.
- Servidor: Actor que representa un dispositivo que opera con el protocolo CoAP y ofrece una serie de servicios. Podrá realizar acciones como generar recursos o subrecursos y responder peticiones CoAP.

- Proxy HTTP-CoAP: Actor que representa un proxy que enlaza los protocolos HTTP y CoAP. Recibe peticiones HTTP y genera peticiones CoAP. Las respuestas CoAP recibidas las transforma en respuestas HTTP
- Proxy CoAP-HTTP: Actor que representa un proxy que enlaza los protocolos CoAP y HTTP. Recibe peticiones CoAP y genera peticiones HTTP. Las respuestas HTTP recibidas las transforma en respuestas CoAP.

A continuación pueden observarse los diferentes casos de uso identificados en el sistema:

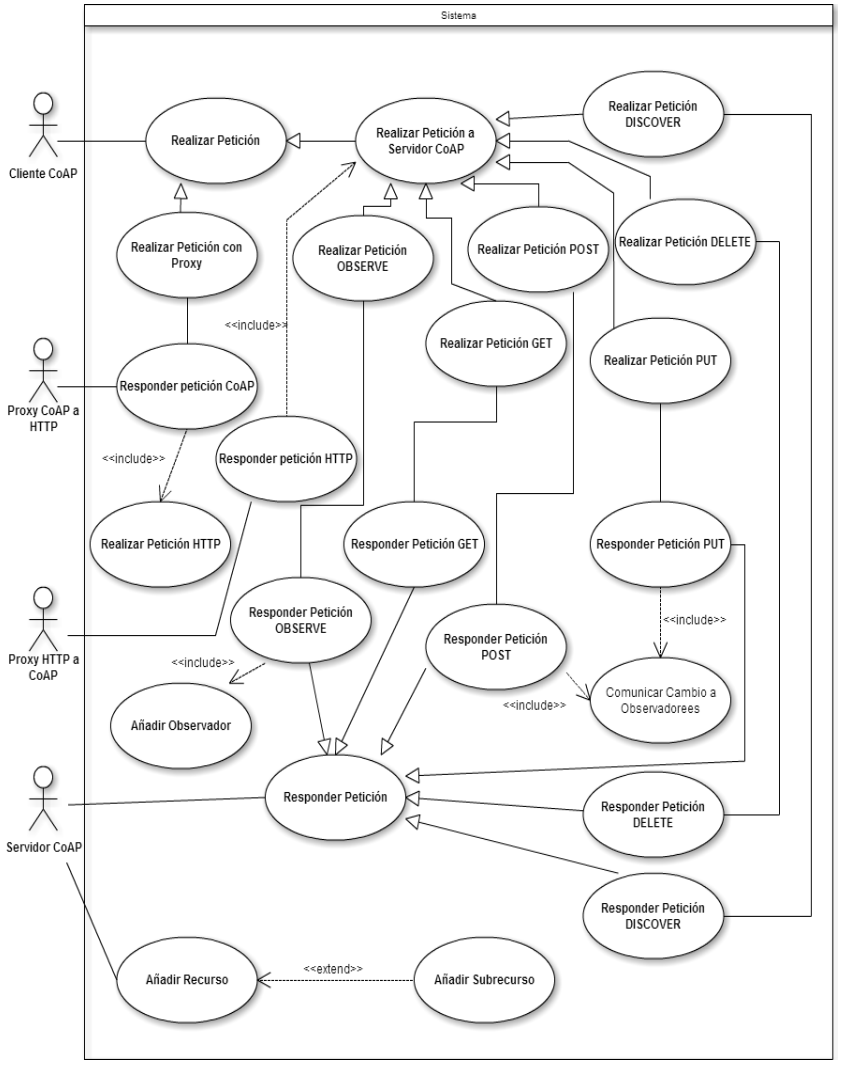


Ilustración 8 Casos de uso del sistema

A continuación se muestra la especificación de los casos de uso más relevantes, acompañados de un diagrama de actividad. La especificación del resto de los casos de uso puede verse en el Anexo 1 Casos de Uso.

Caso de Uso: Realizar Petición GET
Objetivo: Un cliente realiza una petición con el método POST a un recurso del servidor.
Actores: Cliente.
Precondiciones: No hay.
Pasos: 1. La aplicación cliente prepara los parámetros para realizar la petición. Los parámetros son la URL de destino (indicando el recurso con el que se quiere interactuar), el cuerpo del mensaje que se quiera enviar y el método en este caso GET. 2. Se validan los datos y se envía la petición. 3. El cliente recibe la respuesta del servidor.
Extensiones: 2.1. Si algún dato no es válido se muestra un mensaje de error indicándolo y no se realiza la petición. 3.1. Si la respuesta se demora, el cliente vuelve a enviar la petición hasta un máximo de cuatro veces. Si este número se supera, se muestra un mensaje indicando que no se ha podido conectar.

Tabla 4 Caso de uso Realizar Petición GET

Caso de Uso: Responder Petición GET
Objetivo: Un servidor responde una petición GET.
Actores: Servidor.
Precondiciones: No hay.
Pasos: 1. El servidor selecciona el recurso indicado. 2. Se realiza la operación asociada al método GET en el recurso. 3. El servidor responde al cliente con un mensaje con el código 2.05 Content y el resultado de la interacción con el recurso en el cuerpo del mensaje.
Extensiones: 1.1. Si el servidor no encuentra el recurso, responde con el código de error 4.04 Not Found. 2.1. Si el recurso no admite el método indicado, el servidor responde con el código de error 4.05 Method Not Allowed.

Tabla 5 Caso de uso Responder Petición GET

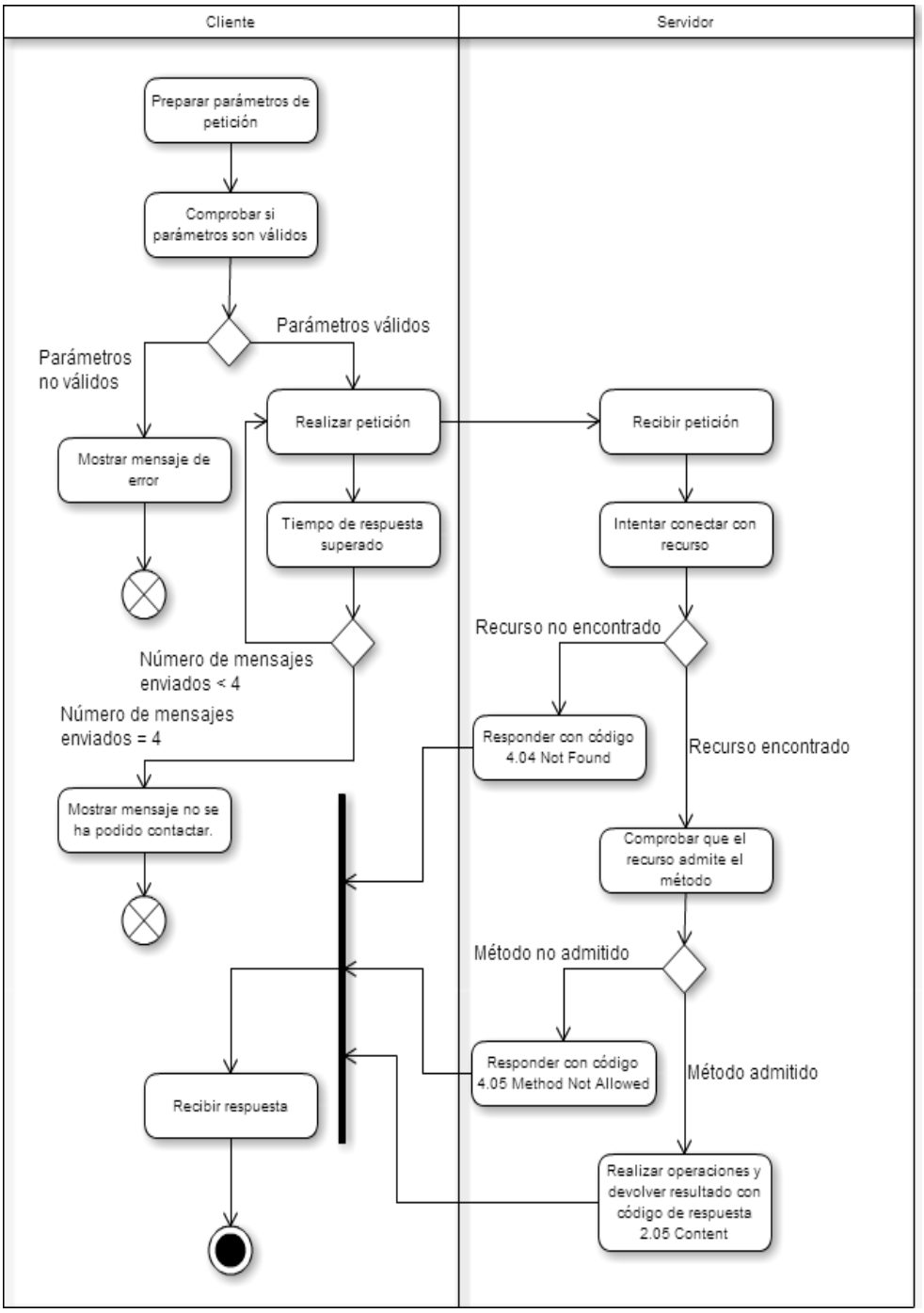


Ilustración 9 Diagrama de actividad Realizar Petición GET y Responder Petición GET

Caso de Uso: Realizar Petición POST
Objetivo: Un cliente realiza una petición con el método POST a un recurso del servidor.
Actores: Cliente.
Precondiciones: No hay.
Pasos: 1. La aplicación cliente prepara los parámetros para realizar la petición. Los parámetros son la URL de destino (indicando el recurso con el que se quiere interactuar), los parámetros en el cuerpo del mensaje y el método en este caso POST. 2. Se validan los datos y se envía la petición. 3. El cliente recibe la respuesta del servidor.
Extensiones: 2.1. Si algún dato no es válido se muestra un mensaje de error indicándolo y no se realiza la petición. 3.1. Si la respuesta se demora, el cliente vuelve a enviar la petición hasta un máximo de cuatro veces. Si este número se supera, se muestra un mensaje indicando que no se ha podido conectar.

Tabla 6 Caso de uso Realizar Petición POST

Caso de Uso: Responde Petición POST
Objetivo: Un servidor responde una petición POST.
Actores: Servidor.
Precondiciones: No hay.
Pasos: 1. El servidor selecciona el recurso indicado. 2. Se realiza la operación asociada al método POST en el recurso. 3. Se comunica a todos los clientes que estén observando el cambio en el recurso. 4. El servidor responde al cliente con un mensaje con el código 2.04 Changed.
Extensiones: 1.1. Si el servidor no encuentra el recurso, responde con el código de error 4.04 Not Found. 2.1. Si el recurso no admite el método indicado, el servidor responde con el código de error 4.05 Method Not Allowed.

Tabla 7 Caso de uso Responder Petición POST

Caso de Uso: Comunicar Cambios a Observadores
Objetivo: Un servidor comunica los cambios ocurridos en un recurso a los observadores del recurso.
Actores: Servidor.
Precondiciones: No hay.
Pasos: 1. El servidor accede a la lista de observadores del recurso. 2. El servidor envía un mensaje no confirmable a cada cliente que esté observando el recurso informado del cambio en el recurso.
Extensiones: No hay

Tabla 8 Caso de Uso Comunicar Cambios a Observadores

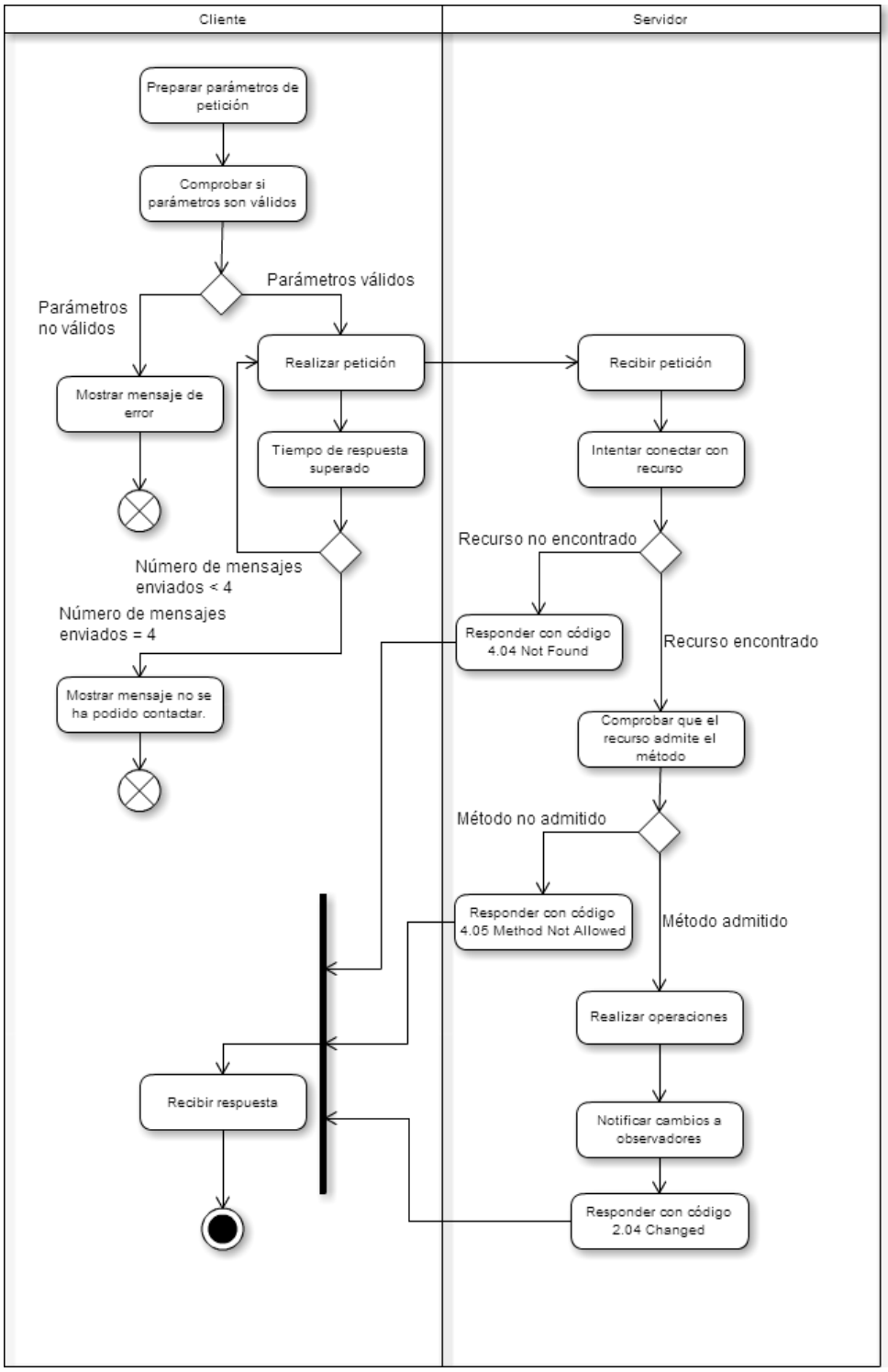


Ilustración 10 Diagrama de actividad Realizar Petición POST y Responder Petición POST

Caso de Uso: Realizar Petición OBSERVE
Objetivo: Un cliente realiza una petición con el método OBSERVE a un recurso del servidor.
Actores: Cliente.
Precondiciones: No hay.
Pasos: 1. La aplicación cliente prepara los parámetros para realizar la petición. Los parámetros son la URL de destino (indicando el recurso que se quiere observar) y el método en este caso OBSERVE. 2. Se validan los datos y se envía la petición. 3. El cliente recibe la respuesta del servidor.
Extensiones: 2.1. Si algún dato no es válido se muestra un mensaje de error indicándolo y no se realiza la petición. 3.1. Si la respuesta se demora, el cliente vuelve a enviar la petición hasta un máximo de cuatro veces. Si este número se supera, se muestra un mensaje indicando que no se ha podido conectar.

Tabla 9 Caso de Uso Realizar Petición OBSERVE

Caso de Uso: Responder Petición Observe
Objetivo: Un servidor responde una petición OBSERVE.
Actores: Servidor.
Precondiciones: No hay.
Pasos: 1. El servidor selecciona el recurso indicado. 2. Se comprueba que el recurso admite el método GET. 3. Se añade el cliente a la lista de observadores del recurso. 4. El servidor responde al cliente con un mensaje con el código 2.05 Content y la información almacenada en el recurso.
Extensiones: 1.1. Si el servidor no encuentra el recurso, responde con el código de error 4.04 Not Found. 2.1. Si el recurso no admite el método GET, el servidor responde con el código de error 4.05 Method Not Allowed.

Tabla 10 Caso de Uso Responder Petición OBSERVE

Caso de Uso: Añadir Observador
Objetivo: Un servidor añade un observador a la lista de observadores de un recurso..
Actores: Servidor.
Precondiciones: No hay.
Pasos: 1. El servidor accede a la lista de observadores del recurso. 2. El servidor añade el observador a la lista de observadores del recurso. 3. Se establece la conexión entre el cliente y el recurso observado.
Extensiones: No hay

Tabla 11 Caso de Uso Añadir Observador

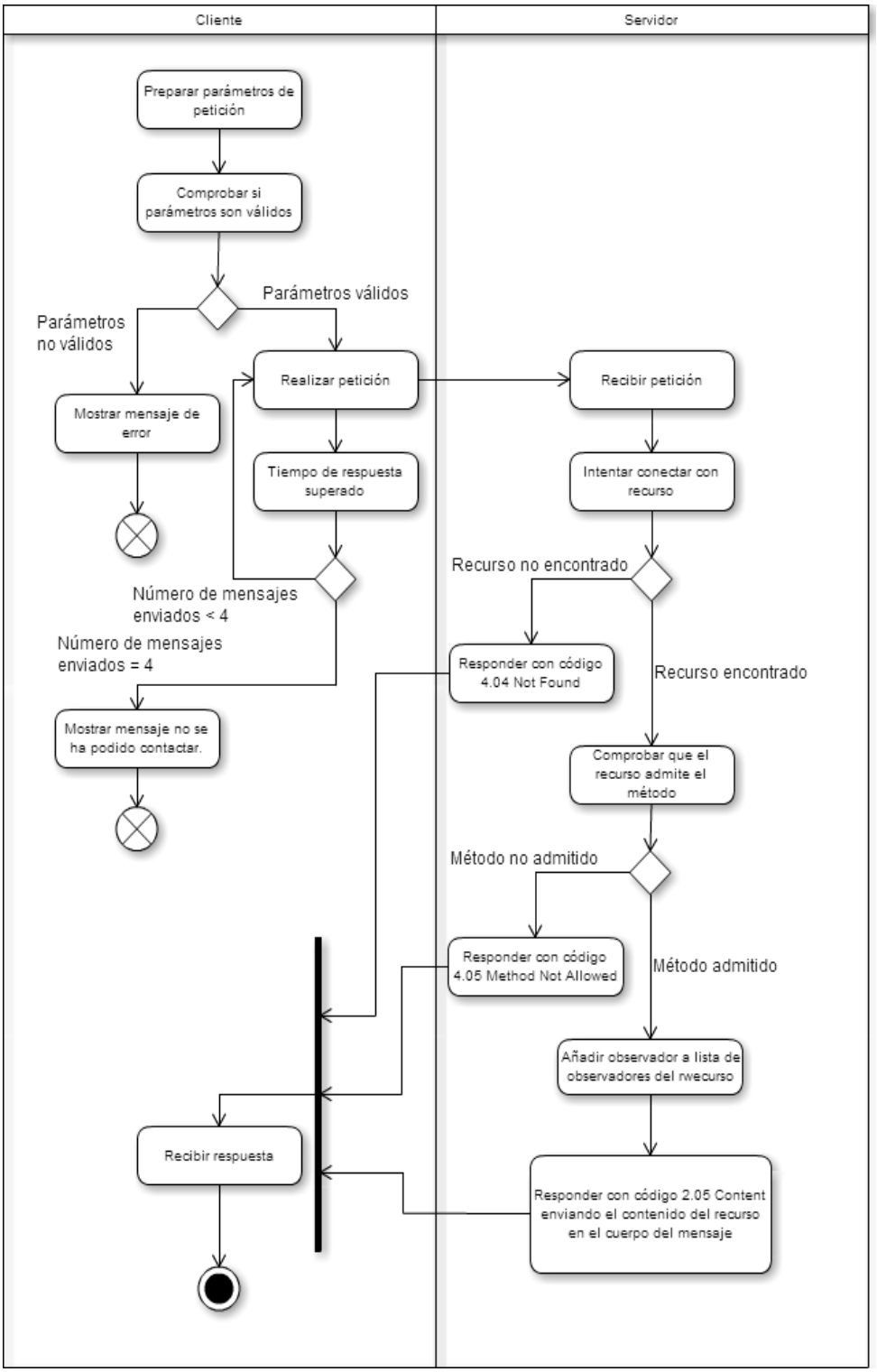


Ilustración 11 Diagrama de actividad Realizar Petición OBSERVE y Responder Petición OBSERVE

Caso de Uso: Realizar Petición con Proxy
Objetivo: Un cliente realiza una petición HTTP a través de un proxy CoAP-HTTP
Actores: Cliente.
Precondiciones: No hay.
Pasos: 1. La aplicación cliente prepara los parámetros para realizar la petición. Los parámetros son la URL de destino (indicando el recurso con el que se quiere interactuar), los parámetros si fueran necesarios, la uri del proxy, el puerto del proxy y el método de la petición. 2. Se validan los datos y se envía la petición. 3. El cliente recibe la respuesta del proxy.
Extensiones: 2.1. Si algún dato no es válido se muestra un mensaje de error indicándolo y no se realiza la petición. 3.1. Si la respuesta se demora, el cliente vuelve a enviar la petición hasta un máximo de cuatro veces. Si este número se supera, se muestra un mensaje indicando que no se ha podido conectar.

Tabla 11 Realizar Petición con Proxy

Caso de Uso: Responder Petición CoAP
Objetivo: Un proxy CoAP-HTTP responde una petición CoAP.
Actores: Proxy CoAP-HTTP.
Precondiciones: No hay.
Pasos: 1. El proxy obtiene los parámetros necesarios para realizar la petición HTTP. 2. Se realiza la petición HTTP. 3. El mensaje de respuesta HTTP se codifica en un mensaje de respuesta CoAP y se envía al cliente.
Extensiones: 2.1. Si el proxy no puede realizar la petición se envía un mensaje con código 5.05 Proxying Not Supported.

Tabla 12 Responder Petición CoAP

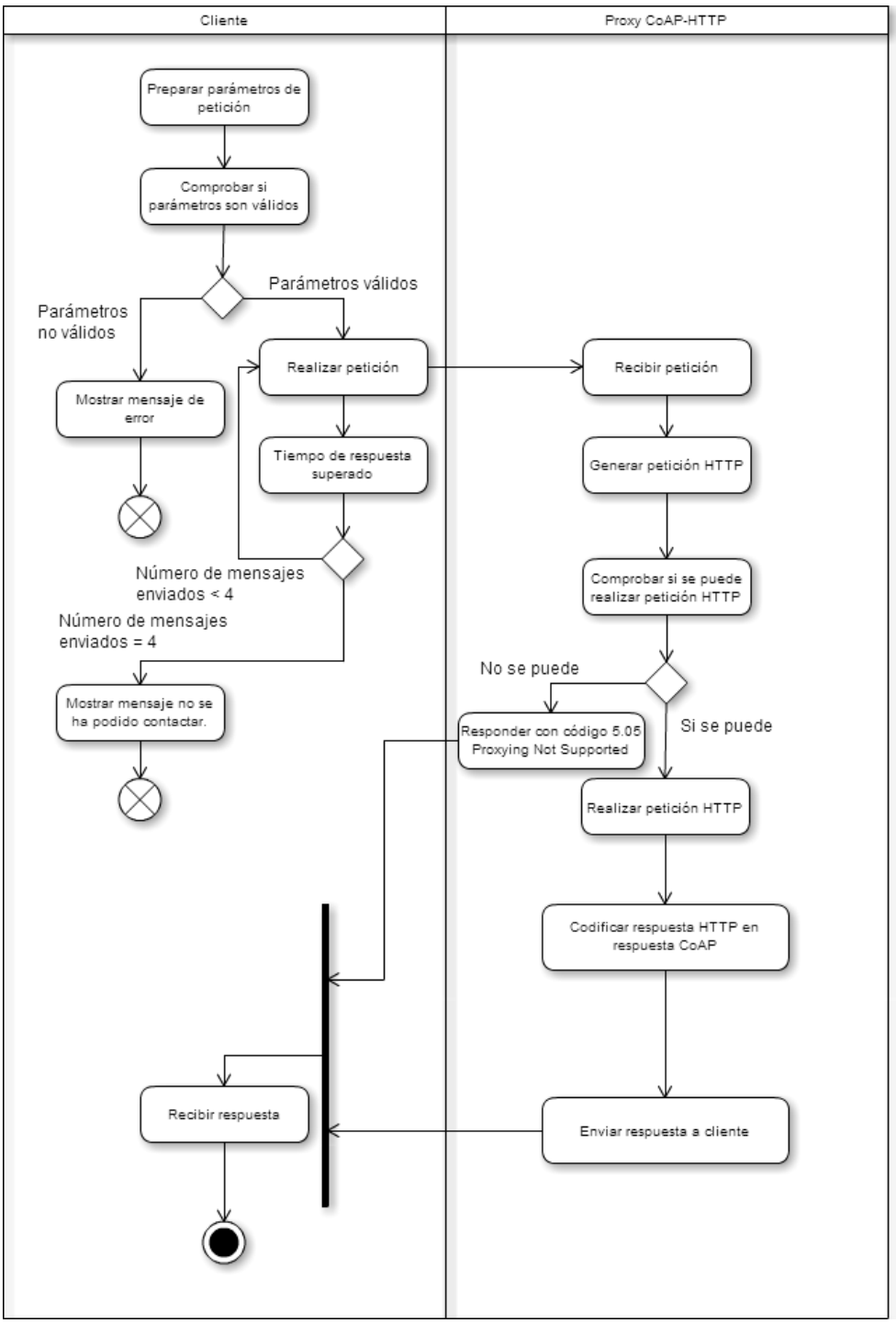


Ilustración 12 Casos de Uso Realizar Petición con Proxy y Responder Petición CoAP

2.3.2. Diseño del software

Diseño de clases

En este apartado se comenta el modelo de objetos diseñado para el protocolo. Se ha dividido el sistema en paquetes para agrupar las clases asociadas.

En primer lugar se han diseñado las clases para el manejo de mensajes. La clase principal es la clase Message, de la que heredan las clases Request y Response.

Aparecen dos interfaces, MessageHandler y MessageReceiver, en las que se definen métodos para el manejo y recepción de mensajes. También se ha diseñado una clase MessageSender, de la que heredarán objetos que envíen mensajes.

Dentro de los subpaquetes de las peticiones y las respuestas, se han añadido interfaces que definen métodos para el manejo de estas peticiones y respuestas. En el caso de la clase Request, se han definido clases hijas que hereden de esta para cada uno de los métodos de petición del protocolo.

En la Ilustración 12 puede observarse el diagrama de clases para las clases relacionadas con los mensajes.

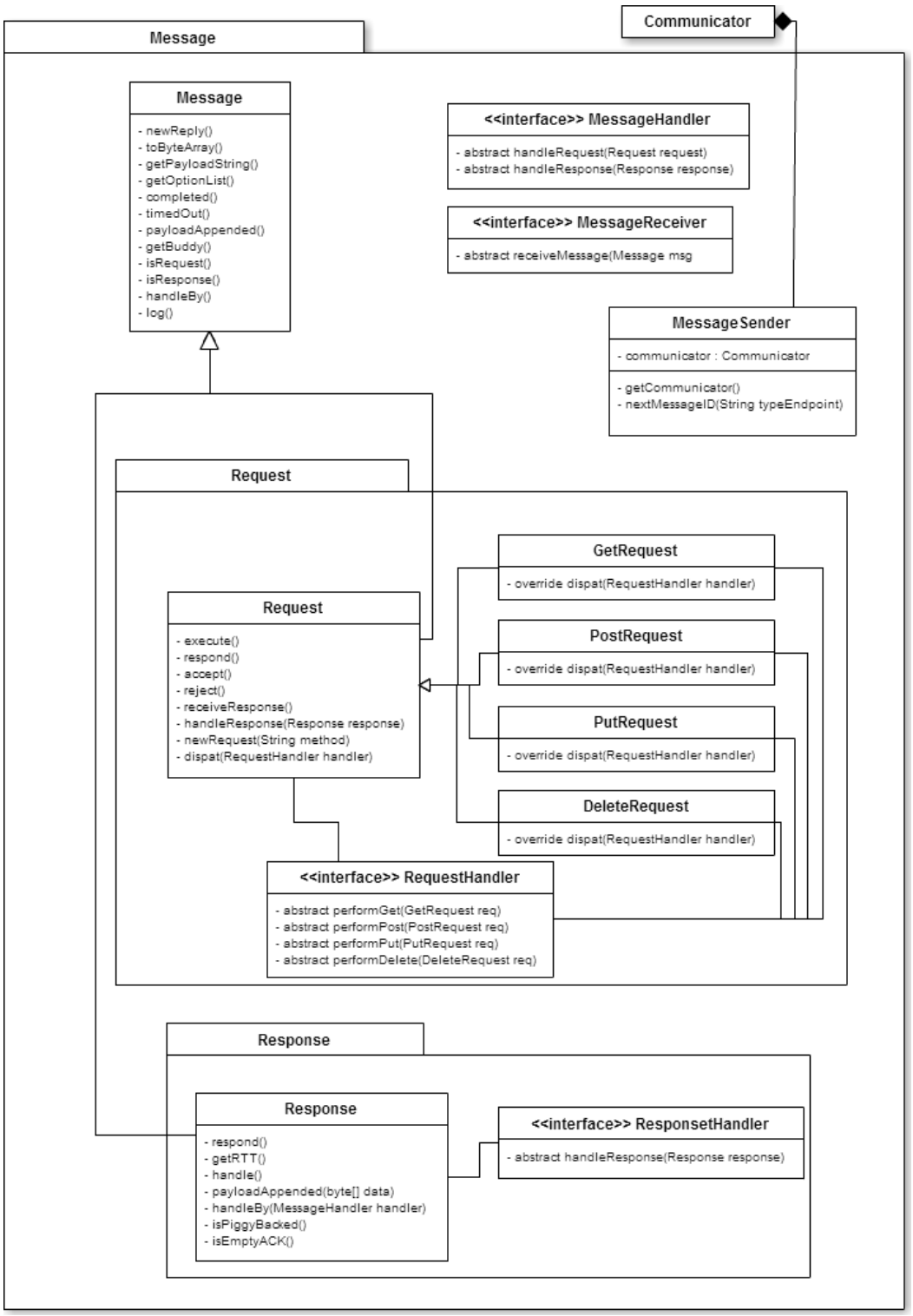


Ilustración 13 Diagrama de clases mensajes

A continuación se diseña el modelo de clases que implemente las diferentes capas del protocolo. Esta es una de las funcionalidades principales, ya que es la encargada de enviar los mensajes entre los elementos del sistema.

La clase principal es la clase Layer, que tendrá la funcionalidad básica asociada a una capa del protocolo. De ella heredan la clase UpperLayer, una capa que está sobre otra, y la clase UDPLayer, que será la capa más baja del protocolo.

La clase UDPLayer será la encargada de recibir los mensajes a través de un socket UDP y los analizará. Para realizar las operaciones, se desarrollará la clase ReceiverThread que trabajará en un hilo diferente y estará escuchando peticiones.

De la clase UpperLayer heredarán las clases TransactionLayer, capa destinada a la realización de transacciones, la clase MessageLayer para el intercambio de mensajes y la clase Communicator. En la clase MessageLayer se utilizarán dos clases anidadas para almacenar información sobre los mensajes y las transmisiones. La clase Communicator actuará como capa más alta y a partir de ella se construirá el modelo de capas.

En la Ilustración 13 se observa el diseño de clases de la estructura de capas del protocolo.

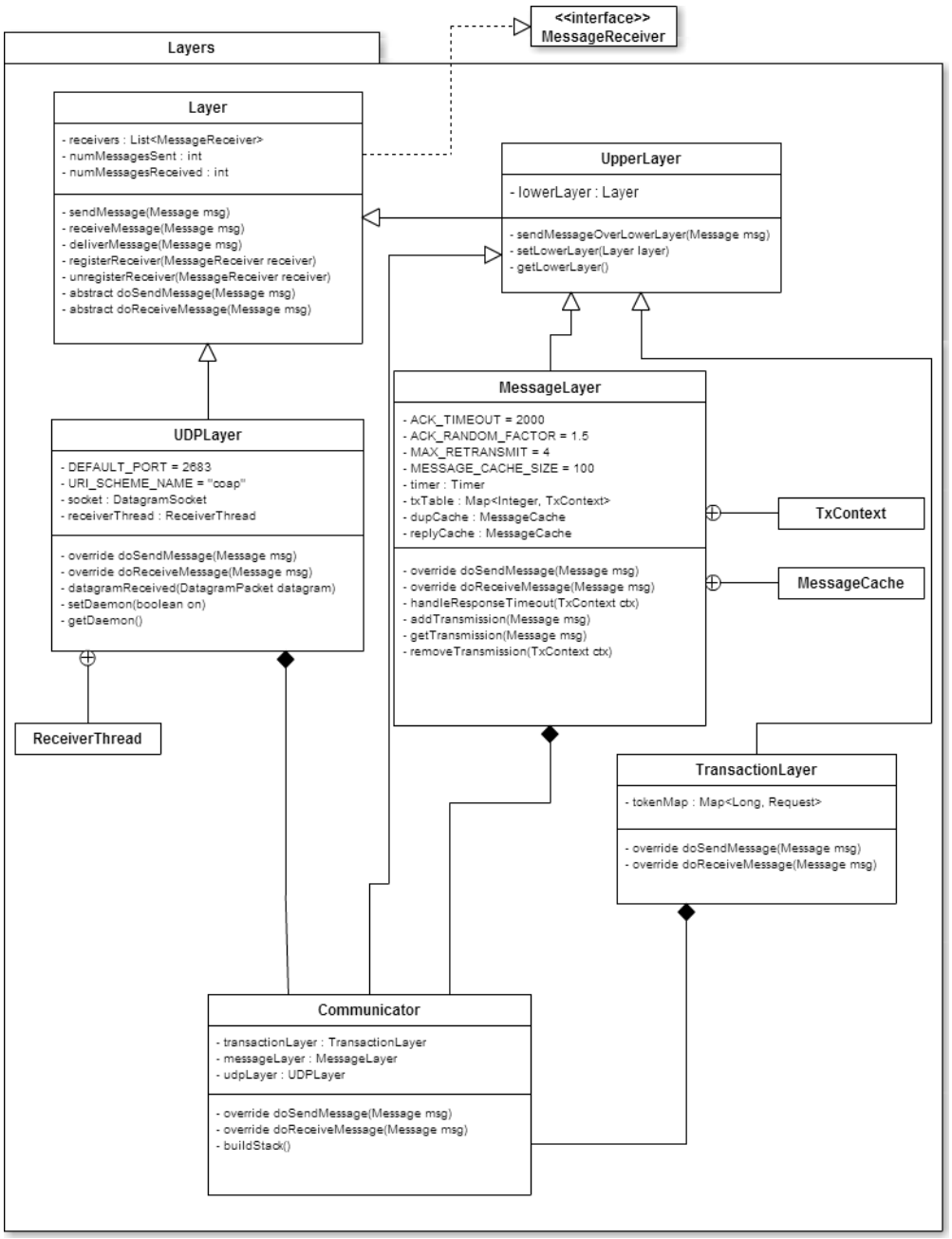


Ilustración 14 Diagrama de clases capas del protocolo

Para diseñar las clases relacionadas con los recursos, se crea el paquete Resource. La clase principal es la clase Resource que implementa la interfaz RequestHandler y contiene la funcionalidad básica de un recurso.

La clase LocalResource representa los recursos locales de un servidor CoAP. Esta hereda directamente de la clase Resource. Para los recursos de solo lectura se añade la clase ReadOnlyResource que hereda de LocalResource.

Finalmente para el recurso que informe de los recursos del servidor, se crea la clase DiscoveryResource que hereda de ReadOnlyResource.

En la Ilustración 14 se pueden ver las clases diseñadas para trabajar con recursos.

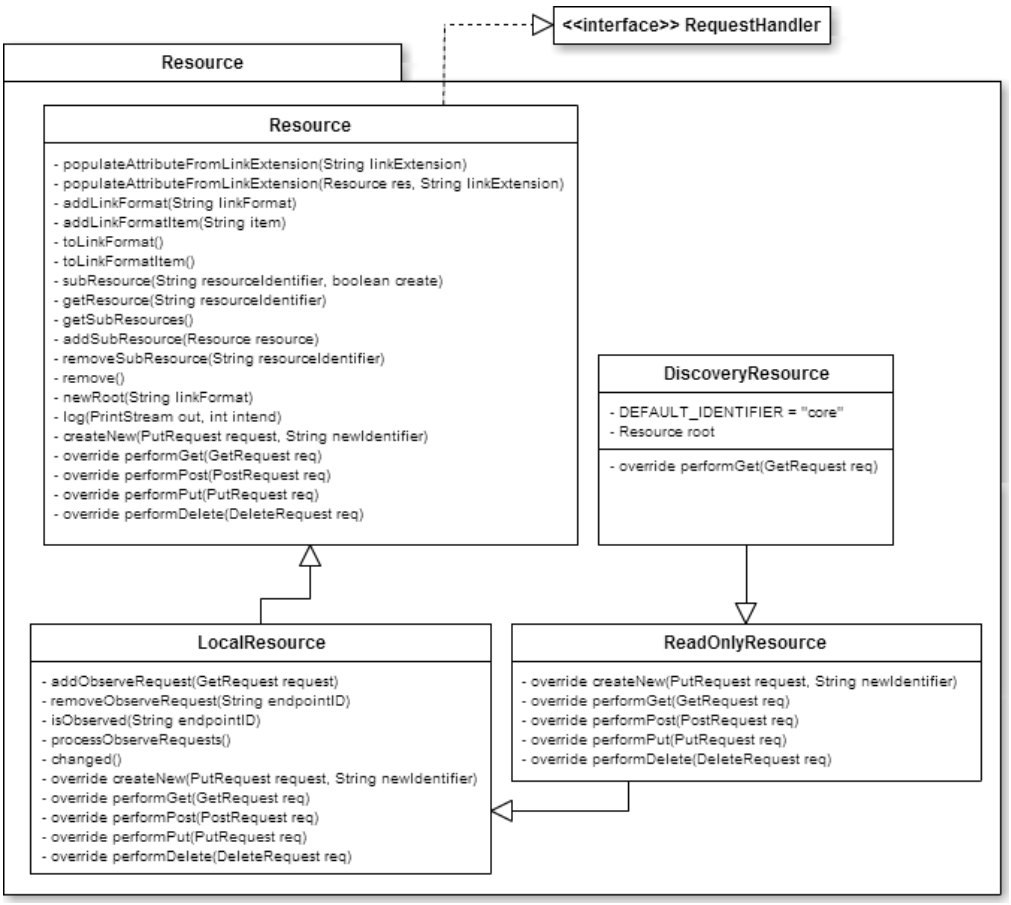


Ilustración 15 Diagrama de clases recurso

Una vez diseñadas los paquetes de mensajes, recursos y capas de la aplicación, se diseñan las clases base del servidor CoAP, el proxy HTTP-CoAP y el proxy CoAP-HTTP.

Las clases base heredarán de la clase EndPoint, que implementará las interfaces MessageReceiver y MessageHandler, además de heredar de MessageSender. Esto permitirá a los objetos de la clase EndPoint manejar, recibir y enviar mensajes CoAP.

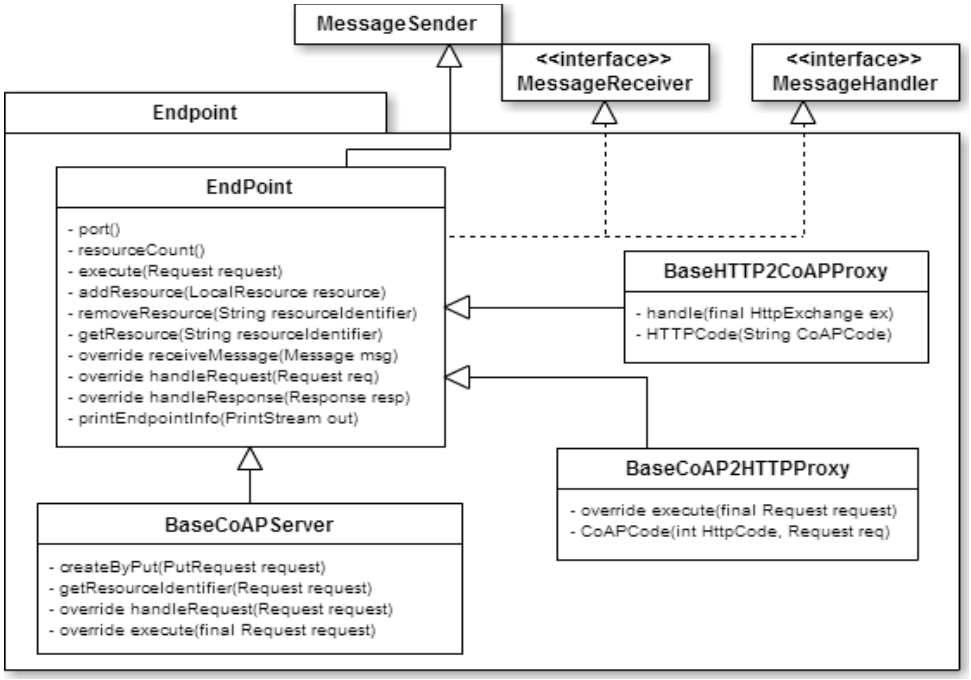


Ilustración 16 Diagrama de clases servidor y proxys

Finalmente se diseñan las clases de utilidades que servirán para trabajar con códigos CoAP, formatos, opciones de peticiones, tipos de mensajes, herramientas para trabajo con datagramas y operaciones hexadecimales.

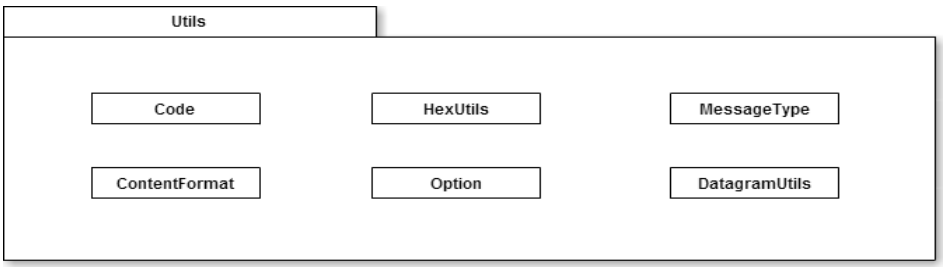


Ilustración 17 Diagrama de clases utilidades

Capas del protocolo

Como se observa en el diseño realizado, se van a desarrollar varias clases que representarán las diferentes capas del protocolo. Las capas aparecerán de la siguiente manera:

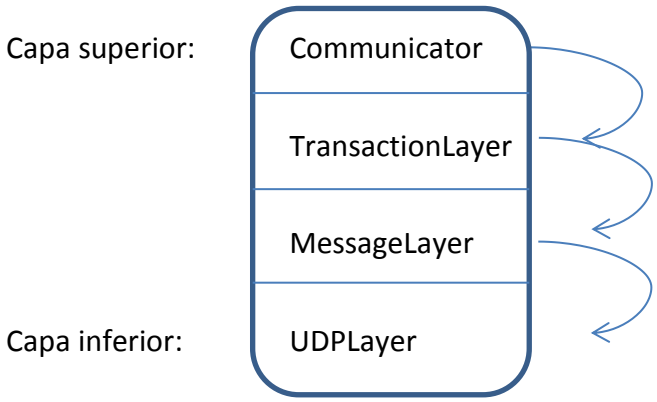


Ilustración 18 Capas del protocolo

Una vez se hayan realizado las tareas de una capa, se enviará el mensaje sobre la paca inferior. Cada elemento de la comunicación que pueda enviar mensajes dispondrá de un objeto de la clase Communicator.

Diseño de los recursos

Los recursos tendrán un nombre para identificarlos. El nombre por defecto del recurso de información (llamado con el método DISCOVER) es ./wel-known/core

Para que un recurso sea el recurso raíz del servidor debe modificarse el atributo rootResource del servidor asignando el valor del recurso.

Aplicaciones cliente, servidor y proxys

Se han diseñado clases base tanto para el servidor CoAP como para los dos tipos de proxy.

Para generar un servidor, este deberá heredar de BaseCoAPServer y añadir recursos. En el caso de los proxys, estos pueden utilizarse tal cual están diseñados o diseñar una clase que herede de ellos.

En el caso de aplicaciones cliente, estas deben heredar de la clase MessageSender.

Diseño del plan de pruebas

En esta sección se especificarán las pruebas que posteriormente se realizarán en la fase de pruebas. Esta definición va a servir como guía de testeo y medio de verificación de que los elementos del sistema cumplen los requisitos exigidos.

Se llevaran a cabo los siguientes tipos de pruebas:

- Pruebas unitarias: Aquellas que sirven para asegurar que cada módulo funciona de manera correcta.
- Pruebas de integración: Aquellas en las que se verifica la validez de los módulos en conjunto.
- Pruebas de aceptación: Aquellas que determinan si se cumplen o no los requisitos iniciales.

Las pruebas unitarias y de integración se realizarán con jUnit y las de aceptación serán los ejemplos de uso del protocolo.

2.3.3. Implementación

Esta fase tiene como objetivo poner en práctica todas las características y reglas que se han ido estableciendo en las fases de análisis y diseño. A continuación se describen las principales tecnologías utilizadas en el proyecto, las librerías externas empleadas, los problemas que han surgido en la implementación y como se han solventado.

Tecnologías utilizadas

Se han utilizado las siguientes tecnologías para el desarrollo del proyecto:

- Eclipse IDE: IDE basado en Java de licencia pública. Será el utilizado para desarrollar el código de la aplicación. Permite instalar plugins para ampliar la funcionalidad del IDE. El lenguaje de programación Xtend está desarrollado por Eclipse, lo que facilita su integración con el IDE.
- Git**:** Software de control de versiones que se utilizará en el proyecto. El proyecto se almacenará en GitHub, un repositorio público para el almacenado de proyectos. Mediante el plugin eGit se puede integrarlo en Eclipse.
- Maven: Herramienta de software para la gestión y construcción de proyectos Java. Sustituye el funcionamiento del IDE, por lo que es importante integrarlo en este. Permite descargar plugins del repositorio y trabajar con librerías en red.
- Javadoc: Utilidad de Oracle para la generación de documentación de APIs en formato HTML a partir de código fuente Java. Javadoc es el estándar de la industria para documentar clases de Java.

Librerías empleadas

Para facilitar el desarrollo de ciertas características del software, se han utilizado las siguientes librerías:

- Librerías Xtend para desarrollar el código Xtend:
  - Xtend-2.6.2
  - Xtext-xbase-2.6.2
  - Guava-14.0.1
- Librerías para conexión HTTP:
  - Httpclient-4.3.5
  - Httpcore-4.3.2
  - Commons-logging-1.1.3
  - Commons-codec-1.6
- Librerías para las pruebas:
  - Junit-4.10
  - Hamcrest-core-1.1

A la hora de trabajar con varios ordenadores en el proyecto, puede ser tedioso controlar que se importan bien todas las librerías necesarias. Para evitar problemas, Maven facilita la importación de estas librerías. Añadiendo una dependencia en el fichero pom.xml del proyecto ya no será necesario descargar la librería y añadirla al PATH del proyecto.

```
<dependency>
    <groupId>org.eclipse.xtend</groupId>
    <artifactId>org.eclipse.xtend.lib</artifactId>
    <version>${xtend.version}</version>
</dependency>
```

Ilustración 19 Dependencia para Xtend en el fichero pom.xml

Estructura del proyecto

Cuando se cree el proyecto en eclipse, el plugin de Maven se encargará de crear la estructura del proyecto. La estructura de directorios es la siguiente:

- src/main/java: Directorio en el que se alojará el código Xtend del protocolo. La organización del código Xtend será la decidida al diseñar las clases del sistema.
- src/test/java: Directorio en el que se alojarán las clases Xtend de pruebas. Estas clases utilizarán métodos de jUnit para realizar pruebas unitarias y de integración.
- src/main/generated-sources/xtend: Directorio en el que se generarán las clases Java del protocolo a partir de las clases Xtend desarrolladas.
- src/test/generated-sources/xtend: Directorio en el que se generarán las clases Java de las pruebas unitarias y de integración.
- Target: Directorio objetivo de la compilación.
- Src: directorio en el que se almacena todo el código generado.
- Fichero pom.xml: Fichero clon la información del proyecto. En él se indican las dependencias y la estructura del proyecto.

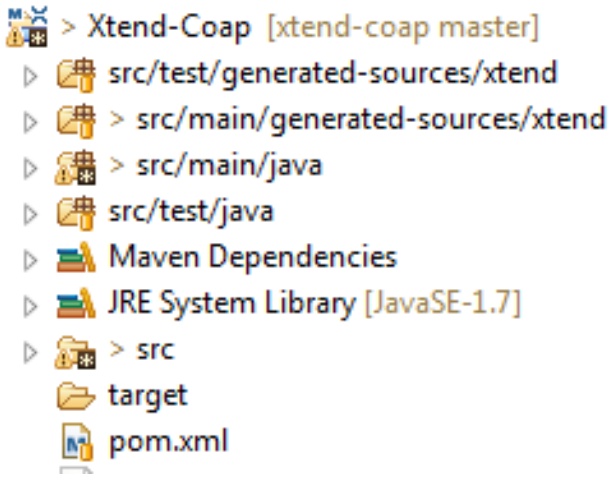


Ilustración 20 Estructura de directorios del proyecto

Creación del repositorio

Antes de comenzar a desarrollar código, se crea el repositorio del proyecto en GitHub. La url del repositorio es la siguiente:

https://github.com/ceesteg/xtend-coap

Mediante el plugin eGit se procede a la cargar el repositorio en Eclipse.

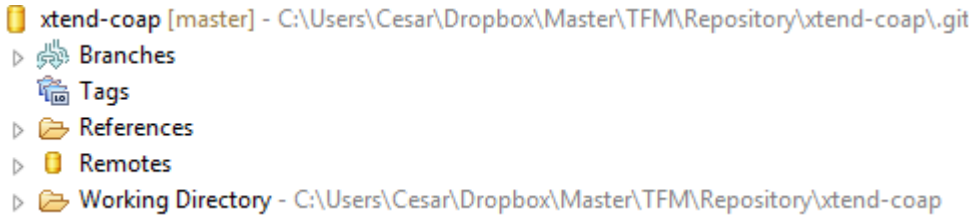


Ilustración 21 Repositorio del proyecto en Eclipse

Una vez se ha carga el repositorio, se añade la estructura de directorios al repositorio y se hace commit. De esta manera ya se puede empezar a desarrollar el código siguiendo los diseños realizados en la fase de diseño.

Implementación del código

A la hora de implementar el código se ha comenzado primero con las clases del paquete utils. El motivo de tomar esta decisión, ha sido que estas clases son utilizadas por el resto para realizar operaciones ya que representan códigos del protocolo o formatos de datos además de utilidades para el trabajo con los datagramas y operaciones hexadecimales.

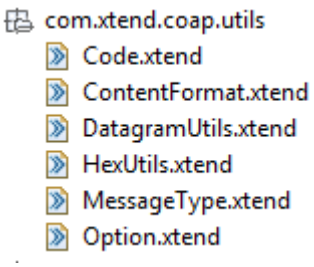


Ilustración 22 Paquete utils

Destacan las clases HexUtils y DatagramUtils. En la clase HexUtils se implementan métodos de conversión y representación de números hexadecimales como los siguientes:

```
def static byte[] longToBytes(long number, int numBytes) {
    var buffer = ByteBuffer.allocate(8)
    buffer.putLong(0, number)
    var array = buffer.array
    var byte[] aux = newByteArrayOfSize(numBytes)
    var n = numBytes - 1
    for (var i = 7; i > 8 - numBytes -1; i--) {
        aux.set(n, array.get(i))
        n--
    }
    return aux
}
```

Ilustración 23 Método para convertir un número de tipo long en un array de Bytes

```
def static long bytesToLong(byte[] bytes) {
    var buffer = ByteBuffer.allocate(8)
    for (var i=0; i < 8 - bytes.length; i++) {
        buffer.put(0.byteValue)
    }
    for (var i=0; i < bytes.length; i++) {
        buffer.put(bytes.get(i))
    }
    return buffer.getLong(0)
}
```

Ilustración 24 Método para convertir un array de Bytes en un número de tipo long

```
def static String hex(byte[] data) {
    val digits = "0123456789ABCDEF"
    if (data != null) {
        if (bytesToLong(data) == 0) {
            return "00"
        }
        var length = data.length
        var builder = new StringBuilder(length * 3);
        for (var i = 0; i < length; i++) {
            builder.append(digits.charAt(data.get(i).operator_doubleGreaterThan(4).bitwiseAnd(0xF)))
            builder.append(digits.charAt(data.get(i).bitwiseAnd(0xF)))
            if (i < length-1) {
                builder.append(' ')
            }
        }
        return builder.toString
    } else {
        return null
    }
}
```

Ilustración 25 Método para obtener la representación hexadecimal de un array de Bytes

La clase DatagramUtils tiene un objeto de la clase ByteArrayInputStream y otro de la clase ByteArrayOutputstream. Según se utilice la clase para lectura o escritura, se instanciará uno u otro.

```
def write(int data, int nBits) {
    for (var i = nBits-1; i >= 0; i--) {
        var validBit = data.operator_doubleGreaterThan(i).bitwiseAnd(1) != 0
        if (validBit) {
            cByte = cByte.bitwiseOr(1.operator_doubleLessThan(cBitIx)).byteValue
        }
        cBitIx--
        if (cBitIx < 0) {
            writeCurrentByte
        }
    }
}
```

Ilustración 26 Método para escribir una secuencia de bits

```
def read(int nBits) {
    var bRead = 0;
    for (var i = nBits-1; i >= 0; i--) {
        if (cBitIx < 0) {
            readCurrentByte
        }
        var validBit = cByte.operator_doubleGreaterThan(cBitIx).bitwiseAnd(1) != 0
        if (validBit) {
            bRead = bRead.bitwiseOr(1.operator_doubleLessThan(i))
        }
        cBitIx--;
    }
    return bRead;
}
```

Ilustración 27 Método para leer bits del buffer

Las siguientes clases a implementar son las del paquete message. Se crean los subpaquetes response y request para las clases de peticiones y respuestas.

Los métodos más cruciales para el manejo de mensajes son los relacionados con la serialización de los mensajes.

Se han desarrollado los métodos toByteArray que convierte un mensaje en un array de bytes para enviarlo por el protocolo y fromByteArray que obtiene la información sobre un mensaje de una cadena de bytes. Estos dos métodos pueden verse en el Anexo 2 Serialización de mensajes.

Una vez implementados los mensajes, es el turno del paquete layers. La clase más importante es la clase Layer de la que heredan las demás. Esta clase podrá permite mensajes a otras capas.

La clase UpperLayer hereda de Layer y permite enviar mensajes a través de una capa inferior.

```
def sendMessageOverLowerLayer(Message msg) throws IOException {
    if (lowerLayer != null) {
        lowerLayer.sendMessage(msg)
    } else {
        System.out.println("[" + getClass.getName + "] ERROR: No lower layer present")
    }
}

def setLowerLayer(Layer layer) {
    if (lowerLayer != null) {
        lowerLayer.unregisterReceiver(this as MessageReceiver)
    }
    lowerLayer = layer
    if (lowerLayer != null) {
        lowerLayer.registerReceiver(this as MessageReceiver)
    }
}
```

Ilustración 28 Métodos para añadir una capa inferior a la UpperLayer y para enviar un mensaje a través de la capa inferior

La clase UDPLayer es la última capa y es la que recibe los datagramas UDP con los mensajes.

```
def datagramReceived(DatagramPacket datagram) {
    var timestamp = System.currentTimeMillis()
    var data = Arrays.copyOfRange(datagram.getData, datagram.getOffset, datagram.getLength)
    var msg = Message.fromByteArray(data)

    msg.setTimestamp(timestamp)
    var scheme = URI_SCHEME_NAME
    var String  userInfo    = null
    var host = datagram.getAddress.getHostAddress
    var port = datagram.getPort
    var String path = null
    var String query = null
    var String fragment = null
    try {
        msg.setURI(new URI(scheme, userInfo, host, port, path, query, fragment))
    } catch (URISyntaxException e) {
        System.out.println("[" + getClass.getName + "] Failed to build URI for incoming message: " +  e.getMessage)
    }
    receiveMessage(msg)
}
```

Ilustración 29 Método de la clase UDPLayer para recibir un datagrama

Finalmente se implementan los paquetes resource y endpoint.

En el paquete de los recursos deben tenerse en cuenta varios elementos:

- La clase Resource tiene métodos para leer el formato link-format y generarlo. Estos métodos son útiles para enviar información sobre el servidor en peticiones con el método DISCOVER.

```
def String toLinkFormatItem() {
    var linkFormat = new StringBuilder
    linkFormat.append("<")
    linkFormat.append(getResourceIdentifier(true))
    linkFormat.append(">;")

    if (!this.getResourceName.isEmpty) {
        linkFormat.append("n=\"")
        linkFormat.append(this.getResourceName)
        linkFormat.append("\";")
    }
    if (!this.getInterfaceDescription.isEmpty) {
        linkFormat.append("d=\"")
        linkFormat.append(this.getInterfaceDescription)
        linkFormat.append("\";")
    }
    if (this.getContentTypeCode != -1) {
        linkFormat.append("ct=")
        linkFormat.append(this.getContentTypeCode)
        linkFormat.append(";")
    }
    if (this.getMaximumSizeEstimate != -1) {
        linkFormat.append("sz=")
        linkFormat.append(this.getMaximumSizeEstimate)
        linkFormat.append(";")
    }
    if (this.isObservable) {
        linkFormat.append("obs;")
    }
    //Remove last semicolon

    linkFormat.deleteCharAt(linkFormat.length-1)

    return linkFormat.toString
}
```

Ilustración 30 Método para obtener la representación en formato link del recurso

- La clase LocalResource permite añadir observadores al recurso. Estos observadores se almacenan en un objeto de la clase Map donde la clave es el identificador del observador y el valor la petición de observar.

```
def void addObserveRequest(GetRequest request) {
    if (request != null) {
        if (observeRequests == null) {
            observeRequests = new HashMap<String, GetRequest>
        }
        observeRequests.put(request.endpointID, request)
        System.out.println("Observation relationship between "
            + request.endpointID + " and " + getResourceIdentifier
            + " established."
        )
    }
}

def void removeObserveRequest(String endpointID) {

    if (observeRequests != null) {
        if (observeRequests.remove(endpointID) != null) {
            System.out.println("Observation relationship between "
                + endpointID + " and " + getResourceIdentifier
                + " terminated."
            )
        }
    }
}
```

Ilustración 31 Métodos para añadir y quitar un observador del Map de observadores

- Las peticiones del método DISCOVER, al igual que las del método OBSERVE se realizan sobre el método GET. Los recursos de información, método DISCOVER, tienen por defecto el identificador core y link-format [6] como formato de salida.

```
class DiscoveryResource extends ReadOnlyResource {

    val public static String DEFAULT_IDENTIFIER = "core"

    Resource root

    /*
     * Constructor for a new DiscoveryResource
     *
     * @param resources The resources used for the discovery
     */
    new (Resource root) {
        super(DEFAULT_IDENTIFIER)
        this.root = root
        setContentTypeCode(ContentFormat.LINK_FORMAT)
    }

    @Override
    override void performGet(GetRequest request) {
        var response = new Response(Code.RESP_CONTENT)
        response.setPayload(root.toLinkFormat, getContentTypeCode)
        request.respond(response)
    }
}
```

Ilustración 32 Clase DiscoveryResource

De la implementación del paquete endpoint hay que destacar que hay una clase EndPoint que implementa las interfaces MessageHandler y MessageReceiver y hereda de MessageReceiver.

Esta clase hace referencia a cualquier punto de una red CoAP que pueda recibir y enviar mensajes, como por ejemplo y dispositivo con servicios o un proxy. En el constructor de esta clase se inicializan:

- Un comunicador.
- Un recurso raíz.
- El recurso .well-known para el método DISCOVER.

```
new(int port, boolean daemon) throws SocketException {
    super(port, daemon)
    this.communicator.registerReceiver(this)
    this.rootResource = new EndPoint.RootResource(this)
    this.wellKnownResource = new LocalResource(".well-known", true)
    this.wellKnownResource.setResourceName("")
    this.discoveryResource = new DiscoveryResource(rootResource)
    rootResource.addSubResource(wellKnownResource)
    wellKnownResource.addSubResource(discoveryResource)
}
```

Ilustración 33 Constructor de la clase EndPoint

Se han implementado unas clases básicas para un servidor CoAP, un proxy HTTP-CoAP y un proxy CoAP-HTTP.

La idea de implementar las clases base es ayudar a los usuarios potenciales facilitando la creación de estos elementos. También pueden tomarse como ejemplo para el desarrollo de otros servidores o proxys.

Estas clases pueden verse en el Anexo 4 Servidor y Proxys Base

2.3.4. Pruebas

Como se indicó en el apartado diseño del plan de pruebas, se realizan tres tipos de pruebas. Las pruebas unitarias realizadas más destacables son las siguientes:

Prueba de lectura y escritura de datagrama

```
def testGETRequestHeader() {
    val versionIn = 1
    val versionSz = 2
    val typeIn = 0
    val typeSz = 2
    val optionCntIn = 1
    val optionCntSz = 4
    val codeIn = 1
    val codeSz = 8
    val msgIdIn = 0x1234
    val msgIdSz = 16
    var writer = new DatagramUtils(null)
    writer.write(versionIn, versionSz)
    writer.write(typeIn, typeSz)
    writer.write(optionCntIn, optionCntSz)
    writer.write(codeIn, codeSz)
    writer.write(msgIdIn, msgIdSz)
    val data = writer.toByteArray
    val dataRef = #[0x41.byteValue, 0x01.byteValue, 0x12.byteValue, 0x34.byteValue]
    var reader = new DatagramUtils(data)
    var versionOut = reader.read(versionSz)
    var typeOut = reader.read(typeSz)
    var optionCntOut = reader.read(optionCntSz)
    var codeOut = reader.read(codeSz)
    var msgIdOut = reader.read(msgIdSz)
    assertArrayEquals(dataRef, data)
    assertEquals(versionIn, versionOut)
    assertEquals(typeIn, typeOut)
    assertEquals(optionCntIn, optionCntOut)
    assertEquals(codeIn, codeOut)
    assertEquals(msgIdIn, msgIdOut)
}
```

Ilustración 34 Prueba lectura y escritura de cabecera

Prueba de comparación de mensajes

```
def void testMessage() {
    var msg = new Message
    msg.setCode(Code.METHOD_GET)
    msg.setType(MessageType.CONFIRMABLE)
    msg.setID(12345)
    msg.setPayload("some payload".getBytes)
    System.out.println(msg.toString)
    var data = msg.toByteArray
    var convMsg = Message::fromByteArray(data)
    assertEquals(msg.getCode, convMsg.getCode)
    assertEquals(msg.getType, convMsg.getType)
    assertEquals(msg.getID, convMsg.getID)
}
```

Ilustración 35 Prueba de comparación de mensajes

Prueba de comparación de opciones

```
def equalityTest() {
    var oneByteValue = 255
    var twoBytesValue = 256
    var nrRef = 1
    var optOneByte = new Option(oneByteValue, nrRef)
    var optTwoBytes = new Option(twoBytesValue, nrRef)
    var optTwoBytesRef = new Option(twoBytesValue, nrRef)
    assertTrue(optTwoBytes.equals(optTwoBytesRef))
    assertFalse(optTwoBytes.equals(optOneByte))
}
```

Ilustración 36 Prueba de comparación de opciones

Como prueba de integración destacan las siguientes:

Pruebas de petición y respuesta

```
def void testRespond() {
    var response = new Response
    var request = new GetRequest {
        @Override
        override void handleResponse(Response resp) {
            handledResponse = resp
        }
    }
    request.respond(response)
    assertSame(response, handledResponse)
}

def void testReceiveResponse() throws InterruptedException {
    var request = new GetRequest
    request.enableResponseQueue(true)
    var response = new Response
    timer.schedule(new RespondTask(request, response), 500)
    var receivedResponse = request.receiveResponse
    assertSame(response, receivedResponse)
}
```

Ilustración 37 Pruebas de petición y respuesta

Todas las pruebas unitarias y de integración han resultado satisfactorias. Las pruebas de implantación serán los ejemplos de uso del protocolo que se comentan en el tercer apartado de esta memoria. Ahí se verá si se cumplen los requisitos definidos en la fase de análisis.

2.3.5. Implantación

La última fase del desarrollo es la fase de implantación. En esta fase se realizan las acciones necesarias para obtener, a partir del código generado, la librería Java que se importará en los proyectos que utilicen el protocolo.

Para ello, primero se genera el JavaDoc del proyecto. Este se genera a partir de los comentarios que se han añadido a medida que se realizaba la implementación. Para obtener el JavaDoc simplemente hay que hacer clic con el botón derecho en el proyecto y elegir la opción Import. Se elige la opción Java/javadoc y se seleccionan los ficheros de los que obtener el JavaDoc, en este caso los ficheros Java generados. Se elige el directorio en el que se quiere almacenar la documentación y se indica el JavaDoc command, que es la ruta en la que está el fichero javadoc.exe.

En este caso, se ha creado la carpeta doc en la raíz del proyecto y ahí es donde se ha almacenado la documentación.

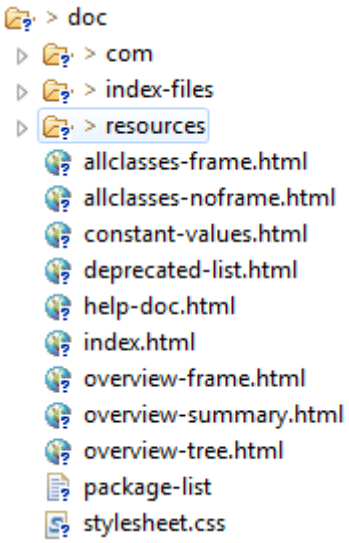


Ilustración 38 Contenido de la carpeta doc

Si se abre el fichero index.html se verá una página web como la de la ilustración 38 por la que se podrá navegar para ver los componentes del protocolo.

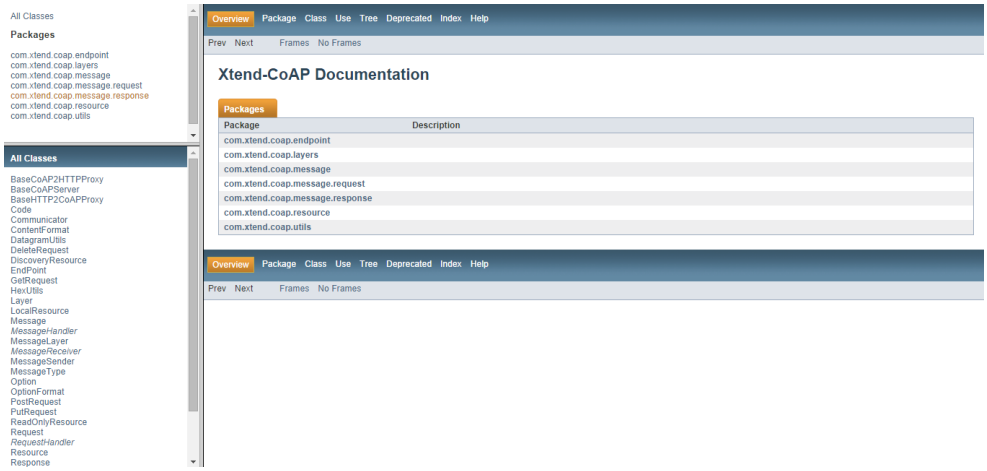


Ilustración 39 Vista previa de la documentación

Una vez generada la documentación, esta se añade al índice del proyecto.

Para generar la librería .jar, se utiliza el plugin Fat Jar que permite crear ficheros .jar incluyendo otros .jar dentro para evitar dependencias.

Se seleccionan los ficheros .class y se genera el fichero Xtend-CoAP_v1.0.jar.

Una vez creado el fichero .jar, se añade al proyecto, se hace un commit de todo y se actualiza el repositorio.

El siguiente trabajo a realizar es probar el protocolo.

3. Ejemplos de uso

En este apartado se mostrarán varios ejemplos de uso del protocolo implementado. Se realizarán pruebas del protocolo para cada escenario posible para comprobar que todo funciona correctamente. Estas pruebas se añadirán al repositorio, para que puedan utilizarse como ejemplo para posibles desarrollos. Todos los ejemplos funcionan, y los resultados de la ejecución de los mismos pueden verse en el Anexo 4 Ejecución de los ejemplos de uso.

3.1. Cliente CoAP/Servidor CoAP

Los primeros ejemplos a realizar son los relacionados con un servidor y un cliente CoAP.

3.1.1. Peticiones simples y método DISCOVER

El primer ejemplo desarrollado se compone de los siguientes elementos:

- Un programa servidor que tiene un recurso para almacenar datos. Pueden ocurrir los siguientes escenarios:
  - Si el recurso recibe una petición GET se devuelve el contenido almacenado.
  - Si recibe una petición PUT o POST se modifica el valor.
  - Si recibe una petición DELETE se borra el recurso
- Un programa cliente que realiza una petición DISCOVER para conocer los recursos del servidor. Después realiza una serie de peticiones al recurso de almacenamiento para comprobar que las operaciones funcionan.

3.1.2. Método OBSERVE

En este ejemplo se añade a la red del ejemplo anterior un nuevo cliente que realizará una petición OBSERVE al recurso de almacenamiento. Estará observando los cambios durante 20 segundos y terminará la conexión. Este ejemplo sirve para comprobar el buen funcionamiento del protocolo de observación.

3.2. Proxy HTTP-CoAP

Una vez comprobado el correcto funcionamiento del protocolo para conexiones cliente/servidor, se prueba la implementación de proxys. El primero a implementar es un proxy HTTP-CoAP.

En el ejemplo aparecen los siguientes elementos:

- El mismo servidor de los ejemplos anteriores.
- Un cliente HTTP.
- Un proxy HTTP-CoAP.

El cliente realizará una petición al servidor CoAP a través del proxy. Si todo sale bien, la aplicación cliente podrá interactuar sin problemas con el recrso de almacenamiento.

3.3. Proxy CoAP-HTTP

Por último se desarrolla un ejemplo de uso para un escenario en el que se necesite un proxy CoAP-HTTP.

Los elementos del ejemplo son:

- Un cliente CoAP que realiza una petición a un servidor HTTP a través de un proxy.
- El proxy CoAP-HTTP.
- Un servidor de aplicaciones web con una pequeña aplicación que proporciona un servicio de almacenamiento.

El cliente realizará la petición y el proxy la convertirá en una petición HTTP. Si todo funciona correctamente, el cliente podrá interactuar con el servidor HTTP sin problemas

4. Conclusiones y trabajos futuros

En este apartado final, se comprobará el cumplimiento de los objetivos planteados al iniciar el trabajo.

También se comentarán las conclusiones alcanzadas tras finalizar el trabajo y las posibles vías de trabajo futuras.

4.1. Cumplimiento de objetivos

Tanto el objetivo principal como los secundarios se han cumplido ya que:

- Se ha obtenido una implementación del protocolo CoAP que permite la conexión entre aplicaciones cliente y servidor.
- Se ha implementado la funcionalidad de método DISCOVER para conocer los recursos del servidor.
- Se ha implementado el protocolo OBSERVE mediante el cual un cliente puede observar los cambios que se produzcan en un recurso.
- Se ha generado la documentación de la documentación con JavaDoc y se ha añadido al proyecto.
- Se ha generado la librería Xtend-CoAP_v1.0.jar que puede ser utilizada en aplicaciones que usen el protocolo.
- Se han desarrollado una serie de ejemplos de uso del protocolo y se ha probado que funcionan.

Cumplimiento de horas establecidas

En la siguiente tabla puede observarse las horas que se planificaron para cada tarea, las horas realiza utilizadas y la desviación entre los dos valores.

TAREA	HORAS PLANIFICADAS	HORAS REALES	DESVIACIÓN %
Búsqueda de información	48	52	8,333333333
Planificación	7	7	0
Análisis de requisitos	64	60	-6,25
Diseño y arquitectura	64	71	10,9375
Implementación	128	132	3,125
Pruebas	16	13	-18,75
Implantación	8	4	-50
Realización de ejemplos	16	16	0
Finalización de la memoria	24	25	4,166666667
TOTAL	375	380	1,333333333

Tabla 13 Comparación de horas planificadas y horas reales

A continuación se explica el porqué de las desviaciones más altas:

- La desviación de casi un 11% en el diseño, se debe a que se plantearon varios modelos de paquetes antes de seleccionar el definitivo.
- La desviación de un -18% en las pruebas se debe a las facilidades que ofrece JUnit para realizar pruebas unitarias y a que al necesitar una clase que otra esté implementada, se realizaban pequeñas pruebas a lo largo del desarrollo.
- La desviación del -50% de horas en la fase de implantación, se debe a que en un principio no se contaba con Maven para realizar la planificación del proyecto, pero al utilizar la tecnología, las operaciones de creación de la librería, generación de la documentación y de control de dependencias se simplificaron.

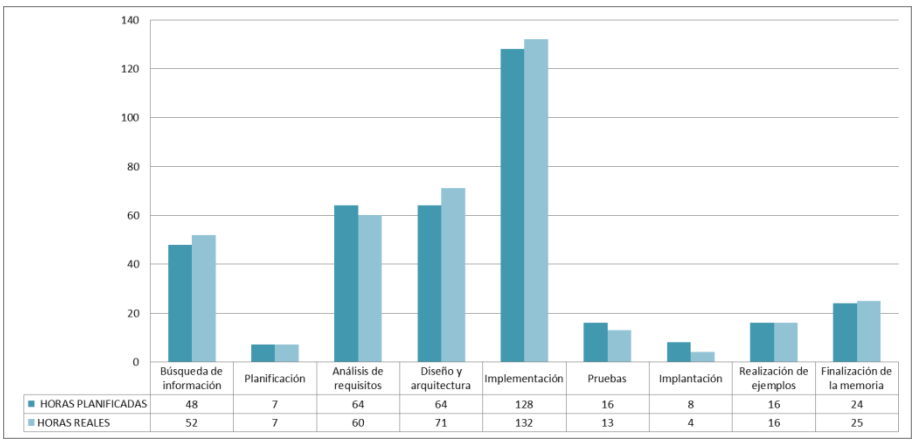


Ilustración 40 Gráfica comparativa horas planificadas-horas reales

Aunque las desviaciones horarias en algunos casos son bastante altas, se ha sabido compensar el trabajo y al final solamente se han trabajado 5 horas más de las planificadas, lo que supone una desviación de 1,33% que es casi despreciable.

4.2. Conclusiones

Una vez realizado el trabajo es hora de reflexionar sobre las conclusiones alcanzadas tras la finalización. Al tratarse de un trabajo de fin de Máster, este es extenso y ha abarcado desde tareas de investigación a ingeniería del software. En él, el alumno ha podido aplicar los conocimientos adquiridos en las asignaturas cursadas en el Máster, además de adquirir otros muchos conocimientos y competencias.

Entre los conocimientos adquiridos cabe destacar:

- Los conocimientos sobre protocolos para el IoT, en especial el protocolo CoAP en el que se ha basado el trabajo. Estos conocimientos le han permitido conocer un poco más de cerca los elementos de la comunicación en entornos ubicuos en donde hasta el más mínimo elemento está conectado.
- Uso del lenguaje Xtend para generación de código. El alumno ya tenía nociones de Xtend gracias a un trabajo realizado para la adignatura Generación Automática de Código, pero estos conocimientos se han afianzado tras la realización del trabajo. Puede concluirse que la elección del lenguaje ha sido un éxito, ya que además de ser una versión de Java simplificada, el número de líneas de código desarrolladas ha supuesto un 35% de las líneas Java generadas. Esto supone un gran ahorro de tiempo.
- El alumno nunca había trabajado con Maven para gestionar proyectos software y le ha parecido una herramienta fantástica ya que permite un control sencillo de las dependencias del proyecto, además de facilitar la generación de pruebas, documentación y ficheros .jar.

La realización del trabajo ha supuesto que el alumno adquiera competencias que le serán útiles en trabajos posteriores. El hecho de realizar un trabajo de esta magnitud en tan poco tiempo, apenas 2 meses y medio, ha supuesto la inversión de grandes esfuerzos para conseguir un buen resultado a pesar de las dificultades de tiempo. Pero este trabajo bajo presión ha servido para acostumbrarse a adaptarse a las circunstancias y vivir de cerca una situación parecida a la del mundo laboral donde las cosas normalmente se necesitan para ayer.

Por todo esto se puede concluir que la realización del trabajo ha supuesto un gran enriquecimiento a nivel personal y profesional y que a falta de la corrección final tras la defensa, se cree que el trabajo desarrollado y los resultados obtenidos han sido buenos.

4.3. Trabajos futuros

A partir del trabajo realizado se pueden abarcar diferentes proyectos:

- Pueden añadirse otras funcionalidades al protocolo que por motivos de falta de tiempo o limitaciones no se pudieron añadir en el trabajo. Es el caso de la capa de seguridad DTLS.
- También pueden desarrollarse versiones del protocolo para otros lenguajes de programación basados en Java.
- Puede utilizarse esta implementación del protocolo para, aprovechando la facilidad de integración de Xtend en las aplicaciones Android, conectar pequeños dispositivos con los teléfonos inteligentes.
- Por último, se abre la vía de la investigación. Seguir investigando en el área del IoT y la generación de código y ver cómo pueden unirse dos disciplinas que están en auge y que son parte del futuro de la informática.

5. Referencias bibliográficas

En este apartado se comentan las referencias utilizadas para la realización del trabajo. Se incluyen tanto referencias bibliográficas como recursos web y siempre que es posible, un enlace al texto original de la referencia.

5.1. Bibliografía escrita

[1] ASHTON, K., *That 'Internet of Things' Thing*. RFID Journal, 22 de Junio de 2009.

[2] Documentación del LWM2M Protocol, Open Mobile Alliance.

[3] SHELBY, Z., ARM, HARTKE, K., BORMANN, C., *The Constrained Application Protocol (CoAP).* Universidad Bremen TZI, Junio 2014.

[4] HARTKE, K., *Observing Resources in CoAP*, Universidad Bremen TZI, 30 de Junio de 2014.

[5] KLEINE, O., *CoAP Endpoint Identification,* University of Luebeck, ITM, 2 de Abril de 2014

[6] SHELBY, Z., *Constrained RESTful Environments (CoRE) Link Format,* Sensinode, 1 de Junio de 2012

[7] HERRINGTON, J., Code Generation in Action, Manning, 2003

[8] BETTINI, L., Implementing Domain-Specific Languages with Xtext and Xtend, PACKT PUBLISHING, 2013

[9] Documentación de Xtend.

[10] PRESSMAN, R., Ingeniería del Software, Un Enfoque Práctico 7ª Edición, Mc Graw Hill, 2010.

5.2. Recursos web

- Página web del proyecto Xtend - http://www.eclipse.org/xtend/index.html
- Generación de código con Xtend - http://xtextcasts.org/
- Generar JavaDoc con eclipse - http://www.myjavazone.com/2013/06/generar-javadoc-usando-eclipse-ide.html
- Jornada "Retos en el internet de las cosas - http://fundaciontecsos.es/noticias/jornada-retos-del-internet-de-las-cosas
- Internet of Things Protocols & Standards - http://postscapes.com/internet-of-things-protocols

- 4 infografías sobre el internet de las cosas - http://www.symplio.com/2011/09/4-infographics-about-internet-of-things/
- IPV6 motor del internet de las cosas - http://blogthinkbig.com/ipv6-motor-internet-de-las-cosas-iot/
- Enlace al repositorio de libcoap - http://sourceforge.net/projects/libcoap/develop
- Enlace al repositorio de Californium - https://github.com/mkovatsc/Californium
- Enlace al repositorio de txThings - https://github.com/siskin/txThings/
- Página web de ETRI CoAP - http://coap.or.kr/index_en.html

6. Siglas y acrónimos

Sección dedicada a la transcripción de las siglas y acrónimos utilizados en la memoria. Se incluyen desde protocolos como HTTP o UDP a convenios aceptados globalmente como IoT y M2M.

IoT: Internet of Things.

CoAP: Constrained Applicaction Protocol.

HTTP: Hypertext Transfer Protocol.

OMA: Open Mobile Alliance.

IETF: Internet Engineering Task Force.

M2M: Machine to Machine.

REST: Representational State Transfer.

LWM2M: Lightweight Machine to Machine.

UDP: User Datagram Protocol.

DTLS: Datagram Transport Layer Security.

LowPAN: Low power Wireless Personal Area Networks.

7. Anexos

En esta sección se incluyen los anexos del trabajo que complementan la memoria. Aparecen desde definiciones de casos de uso secundarios a implementaciones relevantes de métodos. También se incluye un manual para descargar el protocolo y crear un proyecto en eclipse.

Anexo 1 Casos de uso

En este anexo se definen los casos de uso no definidos en la memoria.

Caso de Uso: Realizar Petición PUT
Objetivo: Un cliente realiza una petición con el método PUT a un recurso del servidor.
Actores: Cliente.
Precondiciones: No hay.
Pasos: 1. La aplicación cliente prepara los parámetros para realizar la petición. Los parámetros son la URL de destino (indicando el recurso con el que se quiere interactuar), los parámetros en el cuerpo del mensaje y el método en este caso PUT. 2. Se validan los datos y se envía la petición. 3. El cliente recibe la respuesta del servidor.
Extensiones: 2.1. Si algún dato no es válido se muestra un mensaje de error indicándolo y no se realiza la petición. 3.1. Si la respuesta se demora, el cliente vuelve a enviar la petición hasta un máximo de cuatro veces. Si este número se supera, se muestra un mensaje indicando que no se ha podido conectar.

Caso de Uso: Añadir Recurso
Objetivo: Un servidor añade un recurso a su lista de recursos.
Actores: Servidor.
Precondiciones: No hay.
Pasos: 1. El Servidor indica el nombre del recurso y si es de tipo raíz 2. El servidor añade el recurso a la lista de recursos.
Extensiones:

Caso de Uso: Añadir SubRecurso
Objetivo: Un servidor añade unsub recurso a otre recurso.
Actores: Servidor.
Precondiciones: No hay.
Pasos: 1. El Servidor indica el nombre del recurso. 2. El servidor añade el recurso a la lista de recursos del recurso padre.
Extensiones: 2.1. Si el recurso padre no es raíz muestra un mensaje de error.

Caso de Uso: Realizar Petición DIACOVER
Objetivo: Un cliente realiza una petición con el método DISCOVER a un recurso del servidor.
Actores: Cliente.
Precondiciones: No hay.
Pasos: 1. La aplicación cliente prepara los parámetros para realizar la petición. Los parámetros son la URL de destino (indicando el recurso que se quiere observar) y el método en este caso DISCOVER. 2. Se validan los datos y se envía la petición. 3. El cliente recibe la respuesta del servidor.
Extensiones: 2.1. Si algún dato no es válido se muestra un mensaje de error indicándolo y no se realiza la petición. 3.1. Si la respuesta se demora, el cliente vuelve a enviar la petición hasta un máximo de cuatro veces. Si este número se supera, se muestra un mensaje indicando que no se ha podido conectar.

Caso de Uso: Responde Petición PUT
Objetivo: Un servidor responde una petición PUT.
Actores: Servidor.
Precondiciones: No hay.
Pasos: 1. El servidor selecciona el recurso indicado. 2. Se realiza la operación asociada al método PUT en el recurso. 3. Se comunica a todos los clientes que estén observando el cambio en el recurso. 4. El servidor responde al cliente con un mensaje con el código 2.04 Changed.
Extensiones: 1.1. Si el servidor no encuentra el recurso, responde con el código de error 4.04 Not Found. 2.1. Si el recurso no admite el método indicado, el servidor responde con el código de error 4.05 Method Not Allowed.

Caso de Uso: Responder Petición DISCOVER
Objetivo: Un servidor responde una petición OBSERVE.
Actores: Servidor.
Precondiciones: No hay.
Pasos: 1. El servidor selecciona el recurso 2. Se añade el cliente a la lista de observadores del recurso. 3. El servidor responde al cliente con un mensaje con el código 2.05 Content y la información sobre los recursos del servidor con el formato link-format.
Extensiones: 1.1. Si el servidor no encuentra el recurso, responde con el código de error 4.04 Not Found.

Caso de Uso: Realizar Petición HTTP
Objetivo: Un proxy CoAP-HTTP realiza una petición HTTP.
Actores: Proxy CoAP-HTTP.
Precondiciones: El proxy ha recibido una petición CoAP.
Pasos: 1. El proxy genera una petición HTTP con la información del mensaje CoAP. 2. El proxy envía la petición. 3. El proxy recibe el resultado de la petición.
Extensiones: 2.1. Si el servidor no encuentra el recurso, responde con el código de error 404 Not Found. 2.1. Si el recurso no admite el método indicado, el servidor responde con el código de error 405 Method Not Allowed.

Caso de Uso: Responder Petición HTTP
Objetivo: Un proxy HTTP-CoAP realiza una petición HTTP.
Actores: Proxy HTTP-CoAP.
Precondiciones: El proxy ha recibido una petición HTTP.
Pasos: 1. El proxy genera una petición CoAP con la información del mensaje HTTP. 2. El proxy envía la petición. 3. El proxy recibe el resultado de la petición y lo transforma en un mensaje de respuesta CoAP que envía al cliente
Extensiones: 2.1. Si el servidor no encuentra el recurso, responde con el código de error 404 Not Found. 2.1. Si el recurso no admite el método indicado, el servidor responde con el código de error 405 Method Not Allowed.

Anexo 2 Serialización de Mensajes

A continuación se muestran los dos métodos desarrollados para la serialización de los mensajes CoAP.

toByteArray()

Método para convertir un mensaje en un array de bytes que sea enviado por el protocolo.

```
def toByteArray() {
	var optWriter = new DatagramUtils(null)
	var optionCount = 0
	var lastOptionNumber = 0
	for (Option opt : getOptionList) {
		var optionDeltaExtended = -1
		var optionDelta = opt.getOptionNumber –
					lastOptionNumber
		if (optionDelta >= 0xD && optionDelta < 0x10D) {
			optionDeltaExtended = optionDelta - 0xD
			optionDelta = 0xD
		} else if (optionDelta >= 0x10D &&
				optionDelta <= 0xFFFF) {
			optionDeltaExtended = optionDelta - 0x10D
			optionDelta = 0xE
		} else if (optionDelta > 0xFFFF) {
			System.err.println("ERROR: Option number
					error.")
			return null
		}
		var optionLengthExtended = -1
		var optionLength = opt.getLength
		if (optionLength >= 0xD && optionLength < 0x10D) {
			optionLengthExtended = optionLength - 0xD
			optionLength = 0xD
		} else if (optionLength >= 0x10D &&
				optionLength <=   0xFFFF) {
			optionLengthExtended = optionLength - 0x10D
			optionLength = 0xE
		} else if (optionLength > 0xFFFF) {
			System.err.println("ERROR: Option length
					error.")
			return null
		}
		optWriter.write(optionDelta, OPTION_DELTA)
		optWriter.write(optionLength, OPTION_LENGTH)
```

```
                if (optionDeltaExtended != -1) {
                        if (optionDelta == 0xD) {
                                optWriter.write(optionDeltaExtended,
                                                OPTION_EXT_13)
                        } else if (optionDelta == 0xE) {
                                optWriter.write(optionDeltaExtended,
                                                OPTION_EXT_14)
                        }
                }
                if (optionLengthExtended != -1) {
                        if (optionLength == 0xD) {
                                optWriter.write(optionLengthExtended,
                                                OPTION_EXT_13)
                        } else if (optionLength == 0xE) {
                                optWriter.write(optionLengthExtended,
                                                OPTION_EXT_14)
                        }
                }
                optWriter.writeBytes(opt.getRawValue)
                optionCount++
                lastOptionNumber = opt.getOptionNumber
        }
        var writer = new DatagramUtils(null)
        writer.write(version, VER)
        writer.write(type.ordinal, T)
        writer.write(tokenLength, TKL)
        writer.write(Code.codeClass(code), CODE_CLASS)
        writer.write(Code.codeDetail(code), CODE_DETAIL)
        writer.write(messageID, ID)
        if (code == Code.EMPTY_MESSAGE) {
                return writer.toByteArray
        }
        writer.writeBytes(HexUtils.longToBytes(token,
        tokenLength))
        writer.writeBytes(optWriter.toByteArray)
        if (payload != null && payload.length > 0) {
                if (optionCount > 0) {
                        writer.writeBytes(payload_marker)
                }
                writer.writeBytes(payload)
        }
        return writer.toByteArray
}
```

fromByteArray()

Método para obtener la información de un mensaje de un array de bytes recibido. Devuelve el mensaje recibido.

```
def static fromByteArray(byte[] byteArray) {
		var datagram = new DatagramUtils(byteArray)
		var version = datagram.read(VER)
		var type = getTypeByID(datagram.read(T))
		var tokLen = datagram.read(TKL)
		var codeClass = datagram.read(CODE_CLASS)
		var codeDetail = datagram.read(CODE_DETAIL)
		var code = Code.genCode(codeClass, codeDetail)
		if (!Code.isValid(code)) {
			System.err.println("ERROR: Invalid message code: "
					+ code)
			return null
		}
		var Message msg
		try {
			msg = Code.getMessageClass(code).newInstance
		} catch (InstantiationException e) {
			e.printStackTrace
			return null
		} catch (IllegalAccessException e) {
			e.printStackTrace
			return null
		}
		msg.version = version
		msg.type = type
		msg.tokenLength = tokLen
		msg.code = code
		msg.messageID = datagram.read(ID)
		msg.token = HexUtils.bytesToLong(datagram.readBytes(
				msg.tokenLength)) // Read the token
		if (code == Code.EMPTY_MESSAGE) {
			if (msg.token != 0x0) {
				System.err.println("ERROR: Message format
						error.")
				return null
			}
		}
		var currentOption = 0
		var end = false
		var bytesLeft = datagram.readBytesLeft
		var aux = new DatagramUtils(bytesLeft)
		datagram = new DatagramUtils(bytesLeft)
		while (!end){
			var optionDelta = datagram.read(OPTION_DELTA)
			var optionLength = datagram.read(OPTION_LENGTH)
			if (optionDelta == 0xD) {
				optionDelta = datagram.read(OPTION_EXT_13) +
						0xD
			} else if (optionDelta == 0xE) {
				optionDelta = datagram.read(OPTION_EXT_14) +
						0x10D
			} else if(optionDelta == 0xF) {
				// Reserverd for the payload_marker
```

```
                        if(optionDelta != optionLength) {
                                System.err.println("ERROR: Message
                                                format error.")
                                return null
                        }
                }
                if (optionLength == 0xD) {
                        optionLength = datagram.read(OPTION_EXT_13) +
                                        0xD
                } else if (optionDelta == 0xE) {
                        optionLength = datagram.read(OPTION_EXT_14) +
                                        0x10D
                } else if (optionLength == 0xF) {
                         // Reserverd for future use
                        System.err.println("ERROR: Message format
                                        error.")
                        return null
                }
                currentOption += optionDelta
                var rB = datagram.readBytes(optionLength)
                var opt = new Option (rB, currentOption)
                msg.addOption(opt)
                bytesLeft = datagram.readBytesLeft
                if (bytesLeft.length == 0) {
                        end = true
                } else {
                        aux = new DatagramUtils(bytesLeft)
                        datagram = new DatagramUtils(bytesLeft)
                        if (Arrays.equals(aux.readBytes(
                            msg.payload_marker.length),
                            msg.payload_marker)) {
                               datagram.readBytes(
                                        msg.payload_marker.length
                                  )
                               msg.payload = datagram.readBytesLeft
                               if (msg.payload.length == 0) {
                                      System.err.println("ERROR:
                                              Message format error.")
                                      return null
                               }
                        }
                }
        }
        return msg
}
```

Anexo 3 Servidor y Proxys Base

A continuación se muestran las implementaciones de las clases base para el servidor CoAP y los proxys.

BaseCoAPServer

Código de la clase base para el servidor.

```
class BaseCoAPServer extends EndPoint {
    new() throws SocketException {
          super()
    }
    def protected void createByPut(PutRequest request) {
          var identifier = getResourceIdentifier(request)
          var pos = identifier.lastIndexOf('/')
          if (pos != -1 && pos < identifier.length-1) {
                var parentIdentifier = identifier.substring(0, pos)
                var newIdentifier = identifier.substring(pos+1)
                var parent = getResource(parentIdentifier)
                if (parent != null) {
                      parent.createNew(request, newIdentifier)
                } else {
                      request.respond(Code.RESP_NOT_FOUND, "Unable
                                  to create '" + newIdentifier
                                  + "' in '" + parentIdentifier
                                  + "': Parent does not exist.")
                }
          } else {
                request.respond(Code.RESP_FORBIDDEN)
          }
    }

    def protected static String getResourceIdentifier(
                                          Request request) {
          var uriPaths = request.getOptions(Option.URI_PATH)
          if (uriPaths == null) {
                return ""
          }
          var builder = new StringBuilder
          for (var i=0; i<uriPaths.length; i++) {
                builder.append('/')
                builder.append(uriPaths.get(i).getStringValue)
          }
          var builderLength = builder.length
          if(String.valueOf(builder.charAt(
                            builderLength - 1)).equals("/")) {
                builder.deleteCharAt(builderLength - 1)
          }
          return builder.toString
    }
```

```
override void handleRequest(Request request) {
	System.out.println("Incoming request:")
	request.log
	super.handleRequest(request)
}

override void execute(Request request) {
	if (request != null) {
		var resourceIdentifier =
					getResourceIdentifier(request)
		var resource = getResource(resourceIdentifier)
		if (resource != null) {
			request.dispat(resource)
			if (request instanceof GetRequest &&
				request.hasOption(Option.OBSERVE)) {
				var obsVal = 0
				try {
					obsVal = HexUtils.bytesToInt(
						request.getFirstOption(
						Option.OBSERVE).rawValue)
				} catch (NumberFormatException e) {
					System.out.println("[" +
						getClass.getName +
				"] Bad OBSERVE option value: " +
							obsVal)

					request.respond(
						Code.RESP_BAD_OPTION)
				}

				if (obsVal == 0) {
					resource.addObserveRequest(
						request as GetRequest)
				} else if (obsVal == 1) {
					resource.removeObserveRequest(
							request.endpointID)

				request.respond(Code.RESP_CONTENT,

				   "Observation with resource " +
				  resource.resourceName + " ended.")
				} else {
					System.out.println("[" +
						getClass.getName +
				    "] Bad OBSERVE option value: " +
						obsVal)

				request.respond(Code.RESP_BAD_OPTION)
				}
			} else if (resource.isObserved(
				request.endpointID)) {
				resource.removeObserveRequest(
						request.endpointID)
				request.respond(Code.RESP_CONTENT,
					"Observation with resource "
					+ resource.resourceName +
					" ended.")
```

```
                        }
                    } else if (request instanceof PutRequest) {
                        createByPut(request as PutRequest)
                    } else {
                        System.out.println("[" + getClass.getName +
                            "] Resource not found: "
                            + resourceIdentifier)
                        request.respond(Code.RESP_NOT_FOUND)
                    }
                }
            }
        }
```

BaseCoAP2HTTPProxy

```
class BaseCoAP2HTTPProxy extends EndPoint {
    new(int port, boolean daemon) throws SocketException {
            super(port, daemon)
    }
    override void execute(Request request) {
            if (request != null) {
                var reply = request.newReply(true)
                request.log
                var host = request.getFirstOption(
                        Option.URI_HOST).displayValue
                var port = request.getFirstOption(
                        Option.URI_PORT).displayValue
                var builder = new URIBuilder
                builder.setScheme("http").setHost(
                        host).setPort(Integer.valueOf(port))
                var path = ""
                var paths = request.getOptions(Option.URI_PATH)
                for (var i = 0; i < paths.length; i++) {
                        path += "/" + paths.get(i).displayValue
                }
                builder.setPath(path)
                var httpClient = HttpClients.createDefault
                var dataValue = request.payloadString
                var code = 0
                var HttpResponse response = null
                if (request instanceof GetRequest) {
                        var uri = builder.build
                        var req = new HttpGet(uri)
                        response = httpClient.execute(req)
                        code = response.getStatusLine.statusCode
                } else if (request instanceof PostRequest) {
                        var uri = builder.build
                        var req = new HttpPost(uri)
                        var postParameters = new ArrayList<
                                                NameValuePair>
                        for (String param : dataValue.trim.split(
                                                        "&")) {
                              var p = param.split("=")
                              postParameters.add(
                                    new BasicNameValuePair(
                                          p.get(0), p.get(1)))
```

```
                        }
                        req.setEntity(
                                new UrlEncodedFormEntity(
                                        postParameters))
                        response = httpClient.execute(req)
                } else if (request instanceof PutRequest) {
                        for (String param : dataValue.trim.split(
                                                        "&")) {
                                var p = param.split("=")
                                builder.setParameter(p.get(0),
                                                        p.get(1))
                        }
                        var uri = builder.build
                        var req = new HttpPut(uri)
                        response = httpClient.execute(req)
                } else if (request instanceof DeleteRequest) {
                        var uri = builder.build
                        var req = new HttpDelete(uri)
                        response = httpClient.execute(req)
                        code = response.getStatusLine.statusCode
                }

                if (response != null) {
                        code = response.getStatusLine.statusCode
                } else  {
                        code = 502
                }
                var coapCode = CoAPCode(code, request)
                if (response.getEntity != null) {
                        reply.setPayload(EntityUtils.toString(
                                response.getEntity, "UTF-8").trim)
                }
                reply.setCode(coapCode)

                communicator.sendMessage(reply)
                httpClient.close
        }
}


def private String CoAPCode(int HttpCode, Request req) {
        switch (HttpCode) {
                case 200:
                        return Code.RESP_CONTENT
                case 201:
                        return Code.RESP_CREATED
                case 204:
                        if (req instanceof DeleteRequest) {
                                return Code.RESP_DELETED
                        } else if (req instanceof PostRequest ||
                                        req instanceof PutRequest){
                                return Code.RESP_CHANGED
                        }
                case 304:
                        return Code.RESP_VALID
                case 400:
                        return Code.RESP_BAD_REQUEST
```

```
                case 401:
                        return Code.RESP_UNAUTHORIZED
                case 403:
                        return Code.RESP_FORBIDDEN
                case 404:
                        return Code.RESP_NOT_FOUND
                case 405:
                        return Code.RESP_METHOD_NOT_ALLOWED
                case 406:
                        return Code.RESP_NOT_ACCEPTABLE
                case 412:
                        return Code.RESP_PRECONDITION_FAILED
                case 413:
                        return Code.RESP_REQUEST_ENTITY_TOO_LARGE
                case 415:
                        return Code.RESP_UNSUPPORTED_CONTENT_TYPE
                case 500:
                        return Code.RESP_INTERNAL_SERVER_ERROR
                case 501:
                        return Code.RESP_NOT_IMPLEMENTED
                case 502:
                        return Code.RESP_BAD_GATEWAY
                case 503:
                        return Code.RESP_SERVICE_UNAVAILABLE
                case 504:
                        return Code.RESP_GATEWAY_TIMEOUT
                default:
                        return ""
        }
    }
}
```

BaseHTTP2CoAPProxy

```
class BaseHTTP2CoAPProxy extends MessageSender implements
                                                    HttpHandler {

    new(int port, boolean daemon) throws SocketException {
        super(port, daemon)
    }

    override void handle(HttpExchange ex) throws IOException {
        var String response = null
        var code = 0

        var method = ex.getRequestMethod
        var uri = ex.getRequestURI

        var reqBody = ex.requestBody
        var payload = ""

        var out = new ByteArrayOutputStream
        var buf = newByteArrayOfSize(4096)

        var n = reqBody.read(buf)
        while (n > 0) {
```

```
        out.write(buf, 0, n)
        n = reqBody.read(buf)
}
var query = new String(out.toByteArray)
if (query != null && !query.trim.equals("")) {
      payload = new String(out.toByteArray).split(
                                    "=").get(1)
}
reqBody.close


var request = Request.newRequest(method)

if (request == null) {
      response = "Bad Gateway"
      code = 502
} else {
      if (uri == null) {
            response = "Bad Request"
            code = 400
      } else {
            try {
                  request.setURI(uri)
                  request.setOption(new Option(
                              request.getURI.host,
                              Option.URI_HOST))
                  var port = request.getURI.port
                  if (port == -1) {
                        port =
                        MessageSender.DEFAULT_PORT
                  }
                  request.setOption(new Option(port,
                                          Option.URI_PORT))
            } catch (URISyntaxException e) {
                  response = "Bad Request"
                  code = 400
            }

            request.setID(nextMessageID("P"))
            generateTokenForRequest(request)
            request.setPayload(payload)
            request.enableResponseQueue(true)
            try {
                  request.execute
            } catch (IOException e) {
                  response = "Internal server error"
                  code = 500
            }

            var Response resp = null
            try {
                  resp = request.receiveResponse
            } catch (InterruptedException e) {
                  response = "Internal server error"
                  code = 500
            }
```

```
                        if (resp != null) {
                                response = resp.getPayloadString


                                code = HTTPCode(resp.getCode)
                                resp.log
                                System.out.println(
                                    "Round Trip Time (ms): " +
                                            resp.getRTT)
                                if (resp.getType ==
                                            MessageType.CONFIRMABLE) {
                                       var reply = resp.newReply(true)
                                       communicator.sendMessage(reply)
                                }
                        } else {
                                response = "Gateway timeout"
                                code = 504
                        }
                }
        }

        var respLength = -1
        if (response != null) {
                respLength = response.length
        }
    ex.sendResponseHeaders(code, respLength)
    var os = ex.getResponseBody
    os.write(response.getBytes)
    os.close
}

def private int HTTPCode(String CoAPCode) {
        switch (CoAPCode) {
                case "2.01":
                        return 201
                case "2.02":
                        return 204
                case "2.03":
                        return 304
                case "2.04":
                        return 204
                case "2.05":
                        return 200
                case "4.00":
                        return 400
                case "4.01":
                        return 401
                case "4.02":
                        return 402
                case "4.03":
                        return 403
                case "4.04":
                        return 404
                case "4.05":
                        return 405
                case "4.06":
                        return 406
```

```
                case "4.12":
                    return 412
                case "4.13":
                    return 413
                case "4.15":
                    return 415
                case "5.00":
                    return 500
                case "5.01":
                    return 501
                case "5.02":
                    return 502
                case "5.03":
                    return 503
                case "5.04":
                    return 504
                default:
                    return 500
            }
        }
    }
```

Anexo 4 Ejecución de los ejemplos de uso

En este anexo se muestra el resultado de las ejecuciones de los ejemplos realizados. Se encuentran en el directorio examples del proyecto.

Ejemplo Cliente/Servidor Método DISCOVER y petición Simple

El resultado de la ejecución es el siguiente:

- La petición llega al servidor.

```
Incoming request:
================================================================
|                         COAP MESSAGE                         |
|--------------------------------------------------------------|
| URI            : coap://192.168.1.12:60103                   |
| Message ID     : 4097                                        |
| Message Type   : CON                                         |
| CoAP Code      : GET Request                                 |
| Token          : D9 25 05 9E 93 29 83                        |
| Options: 4                                                   |
|  * Uri-Host: 192.168.1.12 (12 Bytes)                         |
|  * Uri-Port: 5683 (4 Bytes)                                  |
|  * Uri-Path: .well-known (11 Bytes)                          |
|  * Uri-Path: core (4 Bytes)                                  |
| Payload length: 0 Bytes                                      |
================================================================
```

El cliente realiza la petición DISCOVER

- El servidor responde y el cliente descubre los servicios ofrecidos.

```
================================================================
|                         COAP MESSAGE                         |
|--------------------------------------------------------------|
| URI            : coap://192.168.1.12:5683                    |
| Message ID     : 4097                                        |
| Message Type   : ACK                                         |
| CoAP Code      : 2.05 Content                                |
| Token          : D9 25 05 9E 93 29 83                        |
| Options: 1                                                   |
|  * Content-Format: application/link-format (4 Bytes)         |
| Payload length: 59 Bytes                                     |
================================================================

Payload: </.well-known/core>;ct=40,</store>;n="Resource for storage"

Round Trip Time (ms): 46


Discovered resources:
+[] root
  +[.well-known]
    +[core]
  +[store] Resource for storage
```

El servidor responde con el formato link-format

- El cliente realiza el resto de peticiones y el servidor contesta.

```
Incoming request:
================================================================
|                        COAP MESSAGE                          |
|--------------------------------------------------------------|
| URI            : coap://192.168.1.12:60103                   |
| Message ID     : 4099                                        |
| Message Type   : CON                                         |
| CoAP Code      : POST Request                                |
| Token          : F6 B8 96 FF DF 86 BE                        |
| Options: 4                                                   |
|  * Uri-Host: 192.168.1.12 (12 Bytes)                         |
|  * Uri-Port: 5683 (4 Bytes)                                  |
|  * Uri-Path: store (5 Bytes)                                 |
|  * Content-Format: text/plain charset=utf-8 (4 Bytes)        |
| Payload length: 1 Bytes                                      |
================================================================

Payload: 4
```

El cliente envía datos con el método POST

```
================================================================
|                        COAP MESSAGE                          |
|--------------------------------------------------------------|
| URI            : coap://192.168.1.12:5683                    |
| Message ID     : 4099                                        |
| Message Type   : ACK                                         |
| CoAP Code      : 2.01 Created                                |
| Token          : F6 B8 96 FF DF 86 BE                        |
| Options: 0                                                   |
| Payload length: 0 Bytes                                      |
================================================================

Round Trip Time (ms): 0
```

El servidor informa de que se ha modificado el valor

```
Incoming request:
================================================================
|                        COAP MESSAGE                          |
|--------------------------------------------------------------|
| URI            : coap://192.168.1.12:60103                   |
| Message ID     : 4098                                        |
| Message Type   : CON                                         |
| CoAP Code      : GET Request                                 |
| Token          : 28 13 89 C9 C7                              |
| Options: 3                                                   |
|  * Uri-Host: 192.168.1.12 (12 Bytes)                         |
|  * Uri-Port: 5683 (4 Bytes)                                  |
|  * Uri-Path: store (5 Bytes)                                 |
| Payload length: 0 Bytes                                      |
================================================================
```

El cliente realiza una petición con el método GET

```
============================================================
|                        COAP MESSAGE                      |
|----------------------------------------------------------|
| URI            : coap://192.168.1.12:5683                |
| Message ID     : 4100                                    |
| Message Type   : ACK                                     |
| CoAP Code      : 2.05 Content                            |
| Token          : E0 0D F8 B5 8B 83 61                    |
| Options: 1                                               |
|  * Content-Format: text/plain charset=utf-8 (4 Bytes)    |
| Payload length: 1 Bytes                                  |
============================================================

Payload: 4
```

El servidor devuelve el valor del recurso almacenado

```
============================================================
|                        COAP MESSAGE                      |
|----------------------------------------------------------|
| URI            : coap://192.168.1.12:5683                |
| Message ID     : 4104                                    |
| Message Type   : ACK                                     |
| CoAP Code      : 4.04 Not Found                          |
| Token          : 16 99 5E 0F B8 5C                       |
| Options: 0                                               |
| Payload length: 0 Bytes                                  |
============================================================

Round Trip Time (ms): 0
```

El recurso no se ha encontrado

Ejemplo Petición OBSERVE

```
Establishing observation relationship with coap://192.168.1.12/store...
============================================================
|                        COAP MESSAGE                      |
|----------------------------------------------------------|
| URI            : coap://192.168.1.12:5683                |
| Message ID     : 4097                                    |
| Message Type   : ACK                                     |
| CoAP Code      : 2.05 Content                            |
| Token          : 41 F0 BA C0 04                          |
| Options: 1                                               |
|  * Observe: 0 (4 Bytes)                                  |
| Payload length: 0 Bytes                                  |
============================================================

Round Trip Time (ms): 48
============================================================
|                        COAP MESSAGE                      |
|----------------------------------------------------------|
| URI            : coap://192.168.1.12:5683                |
| Message ID     : 65535                                   |
| Message Type   : NON                                     |
| CoAP Code      : 2.05 Content                            |
| Token          : 41 F0 BA C0 04                          |
| Options: 2                                               |
|  * Observe: 10 (4 Bytes)                                 |
|  * Content-Format: text/plain charset=utf-8 (4 Bytes)    |
| Payload length: 1 Bytes                                  |
============================================================

Payload: 4
```

```
Round Trip Time (ms): 10095
============================================================
|                        COAP MESSAGE                      |
|----------------------------------------------------------|
| URI            : coap://192.168.1.12:5683                |
| Message ID     : 65535                                   |
| Message Type   : NON                                     |
| CoAP Code      : 2.05 Content                            |
| Token          : 41 F0 BA C0 04                          |
| Options: 2                                               |
|  * Observe: 10 (4 Bytes)                                 |
|  * Content-Format: text/plain charset=utf-8 (4 Bytes)    |
| Payload length: 1 Bytes                                  |
============================================================

Payload: 5

Round Trip Time (ms): 10111
Ending observation relationship with coap://192.168.1.12/store...
```

Mensajes recibidos al modificarse el recurso observado

Ejemplo Proxy HTTP-CoAP

En este ejemplo, un cliente HTTP se conecta al servidor CoAP mediante un proxy HTTP-CoAP. Este es el tráfico de mensajes:

Primero el cliente realiza la petición HTTP al proxy y este la convierte a petición CoAP y la envía al servidor.

```
Executing request DELETE /store HTTP/1.1 to coap://localhost:5683 via http://localhost:8000
---------------------------------------
```

Petición DELETE realizada por el cliente

```
Incoming request:
============================================================
|                        COAP MESSAGE                      |
|----------------------------------------------------------|
| URI            : coap://127.0.0.1:51462                  |
| Message ID     : 12289                                   |
| Message Type   : CON                                     |
| CoAP Code      : DELETE Request                          |
| Token          : F9 77 5E 30 8F 4A                       |
| Options: 4                                               |
|  * Uri-Host: localhost (9 Bytes)                         |
|  * Uri-Port: 5683 (4 Bytes)                              |
|  * Uri-Path: store (5 Bytes)                             |
|  * Content-Format: text/plain charset=utf-8 (4 Bytes)    |
| Payload length: 0 Bytes                                  |
============================================================
```

Petición DELETE recibida en el servidor

```
Sample HTTP to CoAP proxy listening at port 8000.
================================================================
|                        COAP MESSAGE                          |
|--------------------------------------------------------------|
| URI            : coap://127.0.0.1:5683                       |
| Message ID     : 12289                                       |
| Message Type   : ACK                                         |
| CoAP Code      : 2.02 Deleted                                |
| Token          : F9 77 5E 30 8F 4A                           |
| Options: 0                                                   |
| Payload length: 0 Bytes                                      |
================================================================

Round Trip Time (ms): 62
```

Respuesta CoAP recibida por el proxy

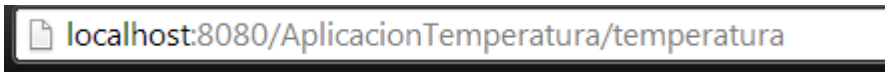
```
----------------------------------------
HTTP/1.1 204 No Content
```

Respuesta HTTP recibida por el cliente

Ejemplo Proxy CoAP-HTTP

En este último ejemplo, se realizará una petición POSTa un recurso HTTP por parte de un cliente CoAP a travñes de un proxy. Para ello se utiliza una pequeña aplicación web.

Primero se inicia el servidor web y se prueba que funciona la aplicación.




```
3.0
```

Tras la prueba, el cliente realiza una petición GET a través del proxy.

```
============================================================
|                        COAP MESSAGE                      |
|----------------------------------------------------------|
| URI            : coap://127.0.0.1:65313                  |
| Message ID     : 4097                                    |
| Message Type   : CON                                     |
| CoAP Code      : GET Request                             |
| Token          : 23 DF 7F 5B 34 FF B2                    |
| Options: 6                                               |
|  * Uri-Host: localhost (9 Bytes)                         |
|  * Uri-Port: 8080 (4 Bytes)                              |
|  * Uri-Path: AplicacionTemperatura (21 Bytes)            |
|  * Uri-Path: temperatura (11 Bytes)                      |
|  * Content-Format: text/plain charset=utf-8 (4 Bytes)    |
|  * Proxy-Uri: coap://localhost (16 Bytes)                |
| Payload length: 0 Bytes                                  |
============================================================
```

Petición recibida por el proxy

Finalmente el cliente CoAP recibe el valor de temperatura almacenado en el servidor HTTP.

```
============================================================
|                        COAP MESSAGE                      |
|----------------------------------------------------------|
| URI            : coap://127.0.0.1:5683                   |
| Message ID     : 4097                                    |
| Message Type   : ACK                                     |
| CoAP Code      : 2.05 Content                            |
| Token          : 23 DF 7F 5B 34 FF B2                    |
| Options: 1                                               |
|  * Content-Format: text/plain charset=utf-8 (4 Bytes)    |
| Payload length: 3 Bytes                                  |
============================================================

Payload: 3.0

Round Trip Time (ms): 46
```

Respuesta recibida por el cliente CoAP

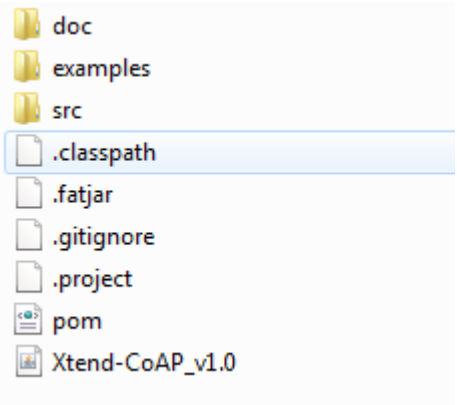
Anexo 5 Manual de descarga y creación de proyecto

En este anexo se explicará cómo utilizar el protocolo desarrollado en un proyecto de Eclipse.

Lo primero que hay que hacer es descargar el protocolo del repositorio:

https://github.com/ceesteg/xtend-coap

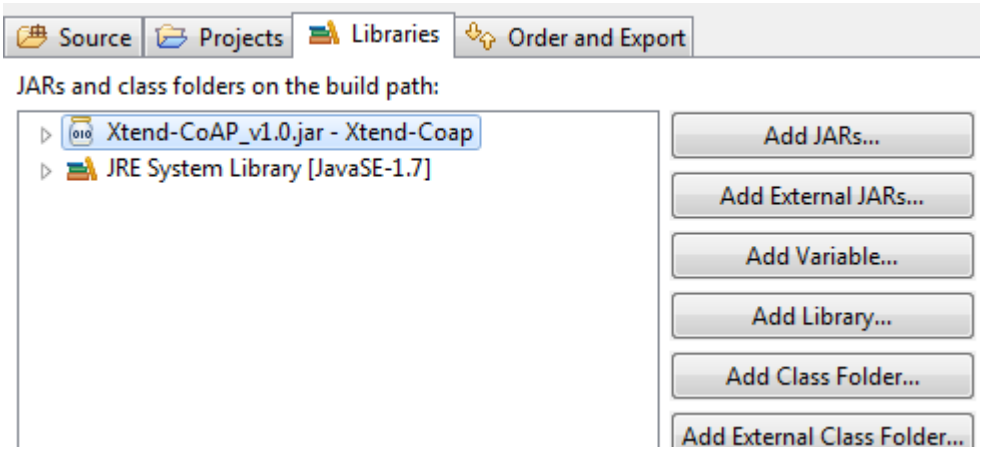
Una vez descargado el código, dentro de la carpeta Xtend-CoAP se tienen los siguientes directorios:



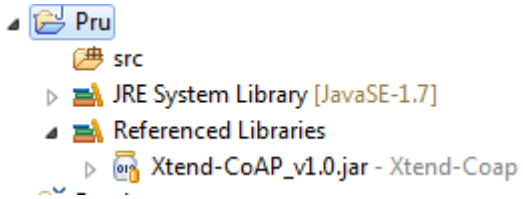
El directorio doc es el que contiene la documentación del protocolo. En la carpeta src se encuentra el código tanto desarrollado como generado. La carpeta examples contiene los ejemplos desarrollados como prueba y que se utilizarán más adelante.

Para crear un nuevo proyecto en eclipse se accede al menú File/New/Java Project.

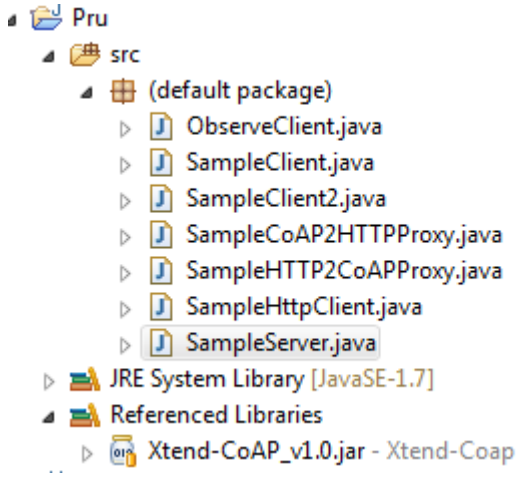
Tras seleccionar el nombre del proyecto, hacer click en next y añadir el fichero Xtend-CoAP_v1.0.jar al PATH del proyecto.



Tras dar en finalizar, se tendrá una estructura de proyecto como esta:



Ahora se copian los ejemplos contenidos en la carpeta examples del proyecto Xtend-CoAP en la carpeta src del nuevo proyecto.



Ya se pueden probar los ejemplos y modificarlos o tomarlos como referencia para realizar nuevas implementaciones.