UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

TESTAR para testing IoT

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

Autor: Mirella Oreto Martínez Murillo

Tutor: Tanja E. J. Vos

2015/2016

Resumen

El número de dispositivos conectados a internet ha aumentado en los últimos años. Con ello, la llamada Internet de las Cosas (IoT, por sus siglas en inglés) se está convirtiendo en una realidad. Se trata, por tanto, de un tema que dispone del potencial necesario para cambiar el modo en el que las personas vivimos e incluso trabajamos.

Para poder aprovechar las ventajas que la IoT puede aportarnos es necesario asegurar la calidad de los dispositivos masivamente interconectados. Aplicar testeo automático se presenta como un medio para satisfacer dichas necesidades. Sin embargo, conlleva varias dificultades como la falta de estándares y las limitaciones en recursos como batería y memoria.

En este trabajo se parte de una herramienta de testeo automático a nivel de interfaz de usuario que ha sido aplicada con éxito en diversos entornos industriales. A partir de ella, se desarrolla una nueva aplicación que, manteniendo su filosofía y aproximación, es aplicable al mundo de la IoT. La herramienta desarrollada es evaluada mediante el testeo de una vivienda inteligente y se presentan los resultados obtenidos.

Palabras clave: Internet de las Cosas, IoT, testeo automático, vivienda inteligente.

Abstract

As the number of devices connected to the Internet is increasing, the so-called Internet of Things (IoT) is becoming a reality. It even has the required potential to change both the way we live and the way we work.

In order to take advantage of the benefits that the IoT can bring us, ensuring the quality of massively interconnected devices becomes a pressing necessity. A means of satisfying this need would be automated testing of IoT devices. However, this presents many difficulties such as the lack of standards and limitations in battery and memory.

In this work we start from an automated testing tool at the user interface level that has already been successfully applied in several industrial cases. Maintaining its philosophy and approach, a new tool is developed which is applicable to the IoT environment. The tool is evaluated by testing a smart home and the results are presented.

Keywords: Internet of Things, IoT, automated testing, smart home.

Índice de ilustraciones

Índice de tablas

Índice de contenidos

1. Introducción

Actualmente la mayoría de los objetos que nos rodean (coches, frigoríficos, luces del hogar, …) disponen del potencial necesario para conectarse a Internet e interactuar con otras máquinas. Por ello, no es de extrañar que el número de dispositivos conectados a la red haya experimentado un crecimiento enorme en los últimos años y que se espere que siga creciendo.

La idea de que los objetos se comuniquen e interactúen los unos con los otros es lo que se conoce como la Internet de las Cosas (IoT, por sus siglas en inglés). Haller et. Al.[1] definen la IoT como un mundo en el que los objetos físicos están perfectamente integrados en la red de información y pueden convertirse en participantes activos en los procesos de negocio. Añaden, también, que se dispone de servicios para interactuar con los "objetos inteligentes" a través de Internet, consultándolos y cambiando tanto su estado como cualquier tipo de información asociada con ellos, eso sí, teniendo en cuenta problemas de seguridad y de privacidad.

Nos encontramos, por tanto, con un tema con gran popularidad debido al hecho de que dispone del potencial necesario para cambiar el modo en el que trabajamos y vivimos. Así mismo, puesto que la IoT nos hace capaces de monitorizar y controlar remotamente dispositivos, también se considera que podría suponer una revolución en la industria, siendo este uno de los sectores que más beneficios puede obtener. Muchas empresas están aprovechando las oportunidades que brinda para obtener información en tiempo real que, tras procesarse y analizarse, puede ayudarles a mejorar la toma de decisiones, reducir riesgos y aumentar su eficiencia.

No obstante, para poder aprovechar las ventajas que la IoT nos puede aportar, es necesario garantizar la calidad de los proyectos llevados a cabo. El testeo, que ya es importante en el ciclo de desarrollo de cualquier aplicación, cobra un papel vital para asegurar que todo funcione según lo esperado. Tal y como se indica en [2], testear la IoT conlleva una serie de retos entre los cuales encontramos las limitaciones en la memoria, el ancho de banda y la vida de la batería.

En este trabajo fin de máster realizaremos una introducción a la IoT en la cual revisaremos la literatura con la intención de identificar el estado del arte de dicho campo. Tras ello, se presentará la herramienta de testeo automático a nivel de interfaz de usuario llamada TESTAR, con la que he tenido la oportunidad de trabajar gracias a una beca de colaboración en la Universitat Politécnica de València (UPV).

En la realización de un *hacklab* (evento organizado por la Alianza Española de Innovación en Software Testing, cuyo objetivo es fomentar la importancia del testeo) presentamos la herramienta en directo para diversas compañías de Zaragoza. Dicha experiencia, junto con la oportunidad de llevar a cabo la implantación de la aplicación en diferentes compañías de la Comunidad Valenciana, nos ha permitido corroborar su utilidad para mejorar la calidad del software testeandolo a través de sus interfaces gráficas. Una muestra de ello se ha transmitido en el *paper* llamado *Another Eperience with Test* in industry: automated localisation testing* que ha sido aceptado en la *28th International Conference on Testing Software and Systems* (ICTSS – 2016).

La obtención de resultados positivos ha motivado la expansión de la filosofía de TESTAR hacía nuevos entornos. Por ello, servirá como base para desarrollar una nueva aplicación enfocada al testeo de la IoT.

Seguidamente, se introducirá la aplicación utilizada como SUT en el caso de estudio realizado. En dicho punto, se especificará tanto en qué consiste como los componentes que la conforman.

Continuaremos detallando la herramienta desarrollada durante la realización de este proyecto. Para ello, se explicará la solución que se ha decidido llevar a cabo, las funcionalidades de las que cuenta y la implementación realizada. Seguidamente, evaluaremos la herramienta ejecutando una serie de pruebas con el SUT explicado con anterioridad y comentaremos los resultados obtenidos en dicha evaluación.

Para finalizar, terminaremos este trabajo con una serie de conclusiones obtenidas durante su realización y establecernos el trabajo futuro a realizar con el objetivo de seguir mejorando la herramienta resultante de este proyecto.

1.1 Objetivos

El objetivo principal de este trabajo de fin de máster es fomentar la expansión de la IoT contribuyendo a los avances en el testeo automático de la misma. Para ello, se parte de TESTAR: una herramienta de testeo automático a nivel de interfaz de usuario cuyo funcionamiento exitoso ha sido validado en aplicaciones industriales[3, 4, 5, 6] y se pretende verificar si su filosofía es aplicable dentro del entorno de la IoT.

Aplicando dicha filosofía se espera conseguir que, sin necesidad de acceder al código fuente del sistema, se puedan realizar pruebas automáticas de manera desatendida. Así mismo, los encargados de testear los sistemas podrán desarrollar sus propios conjuntos de instrucciones que les permitan comprobar si los requisitos de los cuales disponen se cumplen (oráculos).

Por otra parte, durante la realización de este trabajo se espera alcanzar una serie de objetivos personales. Teniendo en cuenta el enorme potencial de la IoT, se pretende que este proyecto sea un punto de partida en dicho campo. Así mismo, se espera poder seguir mejorando la herramienta desarrollada y aplicando los conocimientos adquiridos en futuros proyectos. Además, se pretende aprovechar esta oportunidad para poner en práctica la información aprendida tanto durante los estudios universitarios cursados, como en el periodo de tiempo en el que he participado en el grupo de investigación en *Software Testing and Quality* (STaQ).

1.2 Metodología

Para alcanzar los objetivos planteados se han realizado los siguientes pasos:

1. Se ha investigado el estado del arte del testeo de la IoT para poder posicionar el trabajo realizado en este proyecto
2. Se ha realizado un ejercicio de comprensión del código de la herramienta ya implementada. Esto nos ha permitido entender su funcionamiento interno y adquirir el conocimiento necesario para extrapolarlo a una nueva herramienta cuyo SUT dista de los previamente testeados
3. Puesto que en la aplicación desarrollada no se testea la interfaz gráfica de usuario, se ha establecido como necesario realizar un ejercicio de extracción de posibilidades. Así pues, se han implementado diferentes modos de ejecución teniendo en cuenta las características de los componentes que se van a testear
4. Se ha implementado una nueva versión de TESTAR destinada al testeo de la IoT, aplicando la filosofía de la herramienta original. Para ello, se ha tratado de reutilizar la estructura de dicha aplicación en la medida de lo posible
5. Se ha evaluado el comportamiento de la nueva herramienta desarrollada, concretamente en el entorno de una vivienda inteligente mediante una maqueta virtual. Dicha maqueta ha sido proporcionada por el grupo de investigación en inteligencia ambiental y tecnologías web del Departamento de Sistemas Informáticos y de Computación (DSIC) de la Universitat Politécnica de València (UPV)

2. Internet of Things (IoT)

La IoT está formada por diferentes componentes y conceptos que la constituyen. En medio de la vorágine de términos que se utilizan frecuentemente de un modo no uniforme, en [7] se trata de proporcionar luz a este problema describiendo tanto los términos más importantes como las relaciones entre ellos.

A pesar de existir diversas definiciones para el término IoT, todas ellas tienen en común el hecho de que intentan integrar el mundo físico con el mundo virtual de Internet. Los objetos físicos con los cuales se desea interactuar son las llamadas "Cosas" de la IoT o como las definen en [7] las entidades de interés. Ahora bien, para poder interactuar con dichos objetos físicos estos necesitan dispositivos que pueden estar tanto adjuntados o incrustados en las entidades (constituyendo las llamadas cosas inteligentes) como instalados en el entorno que se desea monitorizar. Entre dichos dispositivos encontramos, por ejemplo, los lectores de RFID, sensores, actuadores, computadores embebidos e incluso dispositivos móviles.

Por otra parte, dichos dispositivos suelen alojar recursos que proporcionan un enlace a las entidades de interés y, por ejemplo, proporcionan información sobre las cosas e incluso pueden proveer capacidades de actuación. Finalmente, el acceso a estos recursos se realiza mediante servicios que pueden ser RESTful[8] o SOAP[9] entre otros.

Debido a la gran heterogeneidad de las cosas conectadas a internet y al aumento a un ritmo acelerado de la cantidad de ellas, asegurar su seguridad y confiabilidad es una necesidad. Más aún, teniendo en cuenta que conectando objetos cotidianos a Internet, los riesgos de seguridad de la información disponen de potencial para extenderse mucho más de lo que se ha experimentado con Internet antes de la aparición de la IoT [10].

Una medida para satisfacer las necesidades comentadas puede ser el testeo automático. No obstante, desarrollar y testear dispositivos IoT representa muchas dificultades [2,10,11] entre las cuales se manifiestan la falta de estándares, el rendimiento de la red de comunicaciones e incluso problemas relacionados con los bajos recursos, en términos de capacidad de computación y energía, de los que disponen.

En [12] se indica que, a pesar de que las tecnologías facilitadoras para la IoT han incrementado en los últimos años, siguen habiendo muchos problemas abiertos a los que hay que hacer frente. Entre ellos se describen, por ejemplo, los relacionados con la selección de una arquitectura adecuada para la IoT, los diferentes ataques que pueden recibir los sistemas y que pueden comprometer su privacidad y seguridad (p.ej. deshabilitar la disponibilidad de la red y enviarle datos erróneos) y los protocolos de comunicación puesto que muchos de ellos fallan a la hora de garantizar confiabilidad *end-to-end* [12].

Por otra parte, la organización mundial sin ánimo de lucro conocida como OWASP (Open Web Application Security Project)[13] que trata de mejorar la seguridad del software, indica en su guía para el testeo de la IoT[14] que las interfaces *cloud*

inseguras necesitan tenerse en consideración para mejorar la seguridad de cualquier producto IoT. Esto incluye la evaluación de las *Application Programing Interfaces* (APIs) que sirven para la comunicación entre componentes software [15] y de las interfaces web basadas en la nube.

Puesto que tal y como se ha comentado, la IoT está formada por diversos componentes y abarcarlos todos mediante una herramienta automática resulta inviable, para la realización de este trabajo se ha meditado cuál de ellos es el más adecuado para aplicarle la filosofía de TESTAR. Tras ello, se ha decidido realizar las pruebas sobre los servicios que nos permiten acceder a los recursos. Así pues, se testeara el API a través de la cual se puede acceder a los diferentes recursos disponibles en una solución. Testear una API requiere utilizar software para enviar llamadas y procesar la respuesta proporcionada por el sistema, lo cual parece idóneo para una herramienta como TESTAR.

En la literatura podemos encontrar trabajos relacionados con el testeo de servicios web de características muy diferentes entre sí. En [16] proponen un *framework* para proporcionar, a programas que dependen de servicios web, una serie de entradas entre las cuales se encuentran casos de prueba predeterminados y la generación de casos de prueba aleatorios mediante la perturbación de plantillas proporcionadas. Para ello, resulta necesario especificar tanto las llamadas a los servicios que están disponibles como sus respectivas entradas válidas. Por otro lado, en [17] han desarrollado un algoritmo para testear la conectividad en los servicios web RESTful, es decir, comprobar si todos los recursos disponibles son alcanzables desde el recurso base a través de peticiones sucesivas. Siguiendo con el testeo de servicios web RESTful, en [18] presentan una herramienta llamada Test-the-REST (TTR) para la cual desarrollaron un lenguaje de especificación de pruebas basado en el lenguaje XML. En [19] introducen una nueva técnica basada en la aplicación de análisis de mutaciones sobre los documentos que describen los accesos a los servicios web (escritos en WSDL – *Web Service Description Language*). Sin embargo, cabe tener en mente la creciente popularidad de los servicios RESTful para los cuales no tiene por qué resultar necesario generar dichos documentos, ya que pueden ser auto-descriptivos.

Estas opciones, a pesar de ser prometedoras, parecen haberse quedado en el mundo de la investigación. Si realizamos una búsqueda de las aplicaciones más utilizadas a nivel profesional encontramos diversas herramientas entre las cuales aparecen las siguientes:

- Postman[20]: Se ofrece gratuitamente en forma de *plugin* para el navegador Google Chrome. Es un cliente HTTP que permite crear peticiones rápidamente e incluso añadirlas a colecciones que pueden ser ejecutadas posteriormente.
- SoapUI[21]: Se trata de una herramienta de código abierto que permite hacer pruebas funcionales, de regresión, de cumplimiento y de carga. Existen diversos libros sobre cómo testear servicios web con esta herramienta [22, 23]
- Ready! API TestServer[24]: Permite a sus usuarios escribir pruebas sobre la API mediante código desde un entorno de desarrollo integrado (IDE)
- RunScope[25]: Se trata de una aplicación web que permite realizar peticiones a una API, crear aserciones y definir variables de manera simple. Para obtener

aserciones más complejas, también soporta la opción de desarrollar scripts mediante el lenguaje JavaScript.

Cabe destacar que se tratan de aplicaciones consolidadas en el mercado, que llevan años en desarrollo y cuentan con un amplio abanico de posibilidades. Por ello, el objetivo de la herramienta desarrollada en este trabajo no es sustituirlas y establecerse como la herramienta definitiva sino, más bien, aportar una nueva aproximación que pueda servir como complemento a otras pruebas realizadas mediante las herramientas ya comentadas.

Puesto que se pretende que la aplicación implementada en este trabajo se pueda ejecutar de forma desatendida, se podrá dejar realizando pruebas automáticamente mientras el personal dedicado a testear un SUT se dedica a realizar otras pruebas. De este modo, la cantidad de pruebas realizadas será mayor y por tanto habrá más posibilidades de encontrar errores existentes.

Así pues, cuando la herramienta detecte un error lo indicará apropiadamente en las salidas que producirá. Por tanto, se podrá observar los resultados de las pruebas ejecutadas durante horas de un modo rápido e intuitivo. Además, puesto que se pretende que con la aproximación seguida no se requiera llevar a cabo un mantenimiento de los casos de prueba generados, los recursos consumidos no serán significativos para una empresa o un grupo de desarrollo.

En [18] recalcan una lista de características deseables en un *framework* para poner a prueba servicios web RESTful, las cuales resulta interesante tener el mente para el desarrollo de nuestra herramienta. Las diferencian del siguiente modo:

- Pruebas funcionales: Compatibilidad de versiones, almacenamiento en caché, Códigos de estado HTTP, pruebas de conectividad, etc.
- Pruebas no funcionales: Pruebas de rendimiento, de estrés, de disponibilidad y de confiabilidad.

Así pues, en este trabajo se presenta una nueva aproximación al testeo de servicios web mediante la cual, tras proporcionar una especificación válida del sistema que se quiere testear, las llamadas realizadas para interactuar con él se derivarán automáticamente. De este modo, siempre y cuando se mantenga actualizada la especificación del sistema, no resultará necesario mantener los casos de prueba ya que estos se crearán en cada ejecución y se permitirá llevar a cabo el testeo desatendido.

3. TESTAR

Actualmente, las interfaces de usuario se encuentran en casi todas las aplicaciones modernas y representan el punto de unión entre los usuarios finales y la aplicación[26]. Por ello, testear a nivel de interfaz de usuario resulta de especial interés. Además, puesto que pueden llegar a representar el 45-60% del código[27] total de las aplicaciones, testearlas resulta complicado. Por ello, automatizar dicho testeo es especialmente necesario para lograr resultados efectivos.

En este contexto, surge la aplicación TESTAR (*TEST Automation at the useR interface level*) [28, 29] la cual genera secuencias de prueba a partir de un modelo derivado automáticamente de la información proporcionada por la API de Accesibilidad del sistema operativo. Puesto que el modelo es inferido en cada estado de la aplicación, el hecho de que la interfaz de usuario sea modificada no afectará a la ejecución de las pruebas.

3.1 Flujo de TESTAR

A continuación se detallan los pasos llevados a cabo por la herramienta cuando se inician las pruebas automáticas con un SUT determinado.

1. Se realiza la instrucción necesaria para empezar a ejecutar el SUT
2. Se produce un escaneo de la interfaz de usuario a partir del cual se obtiene el estado de la aplicación (conocido como *Widget Tree*)
3. A partir del *Widget Tree*, se deriva el conjunto de posibles acciones que un usuario final podría ejecutar (p. ej. clics derechos, clics izquierdos y escritura de texto)
4. Se selecciona una acción del conjunto previamente derivado
5. Se ejecuta la acción seleccionada
6. Se comprueba la validez del nuevo estado de la interfaz de usuario según unos conjuntos de instrucciones, definidos previamente, que permiten decidir si una acción a causado un error o un comportamiento indeseado (oráculos)
7. Si se ha encontrado un error o se ha alcanzado el número de acciones por secuencia especificado por el usuario (mediante la interfaz gráfica de TESTAR), se para la ejecución del SUT guardando la secuencia ejecutada para poder reproducirla si así se desea.

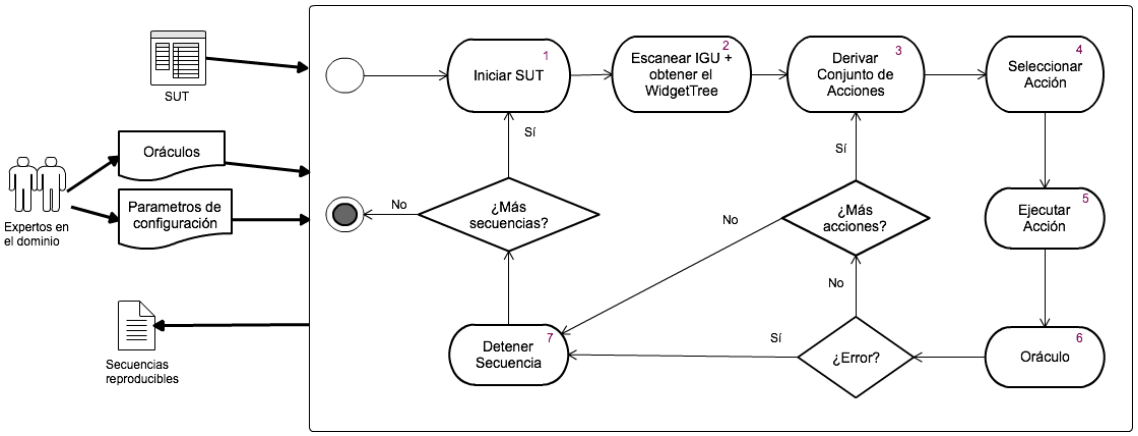


Ilustración 1 – Flujo de TESTAR

Tal y como se indica en la ilustración 1, mientras no se haya alcanzado el número de secuencias especificadas por el usuario, el ciclo descrito se repetirá. Además, se observa que tanto los diferentes parámetros de configuración (número de secuencias, número de acciones, etc.) como los oráculos son proporcionados por expertos del dominio de la aplicación. Esto se debe a que disponen de un mejor criterio para ayudar a que la herramienta decida qué debe considerarse un error y qué no. Así mismo, como salida del ciclo de testeo automático realizado, encontramos las secuencias ejecutadas de modo que podemos reproducirlas mediante la herramienta TESTAR.

4. Vivienda inteligente

Con el objetivo de comprobar el funcionamiento correcto de la aplicación desarrollada en este proyecto, disponemos de un simulador de una vivienda inteligente que cuenta con diversos sensores y actuadores. En este apartado detallaremos las características de dicho sistema, el cual será utilizado como SUT en los experimentos realizados.

4.1 Plataforma IoT

La vivienda inteligente de la que disponemos cuenta con una serie de dispositivos físicos (cosas) ofrecidos en una plataforma IoT [30] a través de un conjunto de servicios RESTful.

La plataforma IoT dispone de un microcontrolador (Arduino) al cual se conectan los diferentes dispositivos electrónicos que ofrecen capacidades tanto para percibir situaciones externas como para actuar sobre el mundo físico. Al mismo tiempo, dicho microcontrolador se conecta a un ordenador reducido de bajo coste (Raspberry Pi) que funciona como pasarela de conexión con los dispositivos físicos y, además, alberga la plataforma de accesibilidad IoT basada en servicios RESTful. Estos servicios podrán ser consumidos por diversos clientes entre los cuales se incluyen desde portátiles y teléfonos inteligentes hasta cualquier otra cosa que forme parte de la IoT y quiera comunicarse con los recursos disponibles en la vivienda inteligente.

La vivienda inteligente, puesto que se trata de un sistema con fines académicos, dispone de un mecanismo de seguridad básico mediante el cual sólo podrán modificar el estado de los recursos aquellos usuarios que dispongan de credenciales válidas. No obstante, debido a la implementación actual del sistema, cualquier usuario podrá consultar el estado de las diferentes cosas.

Tal y como se había indicado, en esta implementación se proporciona un servidor que contiene una capa de servicios REST siguiendo las directrices de la Web of Things[1]

[1] Refinamiento del IoT en el cual se utiliza la Web para exponer e interactuar con las diferentes cosas

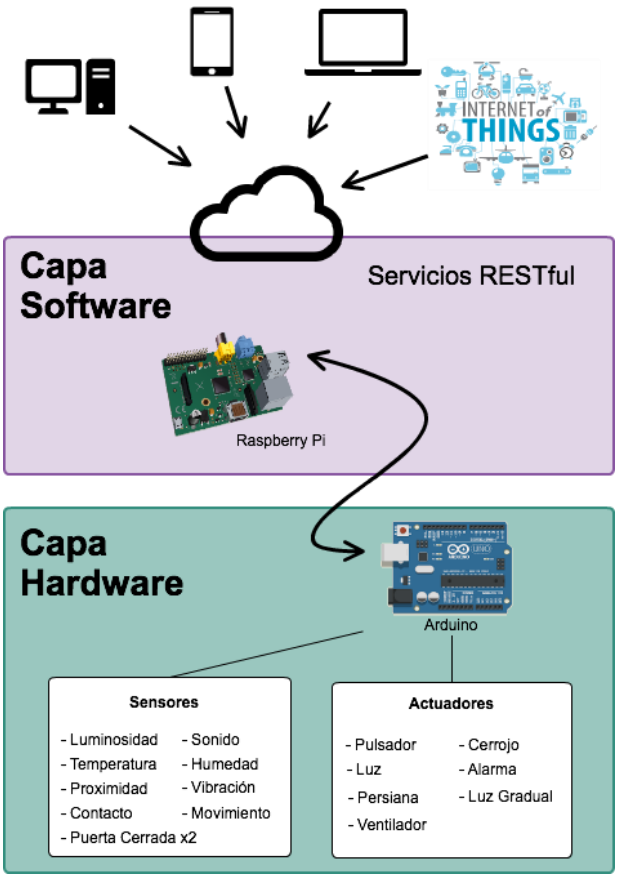


Ilustración 2 - Organización por capas de la plataforma IoT

4.2 Maqueta Virtual

Para facilitar el acceso al sistema durante la realización del proyecto, se nos ha facilitado una maqueta virtual que permite la interacción con los diferentes recursos existentes del mismo modo que su homóloga, sin que estos estén conectados a una plataforma física real. Puesto que dicha maqueta virtual no dispone de sensores ni actuadores reales, no puede interactuar con la realidad.

Sin embargo, gracias a la simulación del comportamiento real, esta limitación resulta inocua para nosotros. En el caso de los actuadores, el sistema simula la interacción con objetos reales permitiéndonos que, por ejemplo, podamos abrir una puerta y comprobar si efectivamente la acción se ha realizado de modo satisfactorio.

Para el caso de los sensores, encargados de medir magnitudes físicas como la temperatura, la humedad o la luminosidad, se ha habilitado la posibilidad de enviar mediciones bajo demanda indicando las características que se desean simular. De este modo, podremos simular, por ejemplo, que la temperatura en la vivienda es de 13ºC enviando una petición al sensor que así lo indique.

4.3 Recursos disponibles

Disponemos, por tanto, de una maqueta virtual con un total de 17 recursos con los cuales podremos interactuar a través de sus respectivos identificadores uniformes de recurso (URIs, del inglés *Uniform Resource Identifiers*)

Así mismo, cada recurso también dispone de un tipo de funcionalidad que marcará qué tipo de interacciones están disponibles para interactuar con él (p. ej. Una luz se puede encender y apagar mientras esas acciones no tienen sentido si el recurso objetivo es una puerta)

Recurso	**ID_THING**	**Funcionalidad**
Sensor de luminosidad	SensorLuminosidad	Numeric
Sensor de temperatura	SensorTemperatura	Numeric
Sensor de sonido	SensorSonido	Numeric
Sensor de humedad	SensorHumedad	Numeric
Sensor de proximidad	SensorProximidad	Numeric
Sensor de vibración	SensorVibracion	Bistate
Sensor de contacto	SensorContacto	Bistate
Sensor de movimiento	SensorMovimiento	Bistate
Pulsador	Pulsador	Bistate
Cerrojo	Cerrojo	Bistate
Sensor de puerta cerrada	SensorPuertaCerrada	Bistate
Ventilador	Ventilador	Bistate
Sensor de puerta cerrada	DoorClosedSensor	Bistate
Luz	Luz	Bistate
Alarma	Alarma	Bistate
Persiana	Persiana	Movement
Luz Gradual	LuzGradual	Dimmer

Tabla 1 - Recursos disponibles en la vivienda inteligente

Funcionalidad	**Acciones**	**Admite Valor**
Numeric	set	Si
	read	No
Bistate	on	No
	off	No
	toggle	No
	pulseOn	Si
	pulseOff	Si
	read	No
Move	open	No
	stop	No
	close	No
	stepOpen	No
	stepClose	No
	up	No
	down	No
	stop	No
	stepUp	No

	stepDown	No
	read	No
Dimmer	set%	Si
	setOx	Si
	setAng	Si
	on	No
	off	No
	pulseOn	Si
	pulseOff	Si
	read	No

Tabla 2 - Interacciones disponibles por funcionalidad

4.4 RESTful

Tal y como se ha indicado, la aplicación desarrollada durante la elaboración de este proyecto se centra en probar los servicios que nos permiten acceder a los recursos disponibles en una solución IoT.

Los servicios RESTful[31], los cuales nos permiten acceder a los recursos de la vivienda inteligente, se han alzado como los servicios web predominantes llegando, incluso, a desbancar otros tipos de servicios web como por ejemplo los SOAP y aquellos basados en lenguajes de descripción de servicios web (WSDL). De hecho, el estudio realizado en [32] indica que los desarrolladores los encuentran más fáciles de desarrollar e incluso más apropiados para la programación de cosas inteligentes.

Otras ventajas que los hacen apropiados para la IoT[33] se basan en el hecho de que tienen menor sobrecarga, una mayor integración con *Hypertext Transfer Protocol* (HTTP) y funcionan mejor que otras alternativas en nodos de sensores inalámbricos con recursos limitados.

Las implementaciones REST siguen cuatro principios de diseño principales[31]:

- Utilizan de modo explícito y consistente los métodos del HTTP. De hecho, se establece una asociación entre las operaciones de crear, leer, actualizar y borrar (CRUD) y los métodos http.
    - POST se utiliza para crear un recurso en el servidor
    - GET permite obtener un recurso disponible
    - PUT se utiliza para llevar a cabo actualizaciones de un recurso
    - DELETE permite eliminar un recurso disponible

- Realizan comunicaciones sin estado (*Stateless*). Una aplicación o cliente de un servicio web REST incluye entre la cabecera y el cuerpo de la petición http todos los datos necesarios para generar una respuesta, de modo que cada petición funciona de manera aislada. Esto permite mejorar el rendimiento del servicio y, debido a que no hay necesidad de sincronizar datos de sesión con aplicaciones externas, el diseño y la implementación de componentes del lado del servidor es más sencillo.

- Están orientados a recursos y cada uno dispone de un identificador uniforme de recurso (URI). Los diferentes URIs sirven tanto de nombre como de dirección de un recurso. Además, deben ser sencillas, predecibles y fáciles de entender de modo que se favorezca el hecho de que los desarrolladores puedan predecir a que se está apuntando e incluso derivar recursos relacionados.
- El formato de codificación de los datos que se intercambian entre una aplicación y el servicio ha de ser estándar como por ejemplo el lenguaje de marcas extensible (XML) o *JavaScript Object Notation* (JSON).

Así mismo, puesto que el protocolo de transferencia es http, se deben utilizar los códigos de error y excepciones de dicho protocolo los cuales se han extraído de [34] y se pueden observar en la tabla 3.

Código de estado	**Descripción**
1xx	Respuestas informativas
2xx	Peticiones correctas
3xx	Redirecciones
4xx	Errores del cliente
5xx	Errores del servidor

Tabla 3 - Códigos de estado de http

Por otra parte, las operaciones realizadas en REST se clasifican según si son seguras o inseguras e idempotentes o no indempotentes.

- Operaciones seguras: No modifican el estado de los recursos y, por tanto, se pueden realizar tantas veces como se desee. Suelen tratarse de operaciones para realizar consultas. P. ej. Obtener el estado de la luz gradual.
- Operaciones inseguras: Se trata de aquellas operaciones que modifican el estado de un recurso tras ser ejecutadas. P. ej. eliminar una luz ya existente.
- Operaciones idempotentes: Aquellas cuyo resultado no varía independientemente de que se realicen una o *n* veces. P. ej establecer el valor del sensor de la temperatura a 13ºC.
- Operaciones no idempotentes: Son aquellas cuyo resultado sí varía en función del número de veces que se hayan ejecutado. P. ej. Añadir una alarma nueva.

5 TESTAR para la IoT

En este apartado se detalla la aplicación desarrollada para testear un dispositivo IoT, concretamente la vivienda inteligente introducida en el apartado anterior.

5.1 Solución implementada

La aplicación TESTAR de la cual partimos para desarrollar nuestra solución obtiene la información que necesita sobre la interfaz gráfica de usuario a través de la API de accesibilidad del sistema operativo. A partir de dicha información conoce las interacciones que puede realizar con el SUT y lo somete a una serie de acciones que le permiten deducir si el comportamiento de la aplicación es el adecuado.

No obstante, en esta solución no pretendemos testear a nivel de interfaz de usuario. Si nos centramos en los servicios web utilizados para interactuar con los diferentes recursos disponibles en las soluciones IoT, se considera que realizando una especificación del sistema a testear se puede conseguir que nuestra aplicación sea capaz de derivar las acciones que pueden llevarse a cabo. De este modo, el SUT podrá someterse a una serie de peticiones y se realizarán comprobaciones para asegurarse de que su comportamiento es el esperado, tal y como ocurre con la herramienta de la que partimos.

A continuación se indica el flujo de la herramienta desarrollada cuando se inicia el testeo automático.

1. Se inicia una secuencia con tantas acciones como se haya especificado a través de la interfaz gráfica de usuario de la aplicación
2. Ateniendo a la especificación realizada en el *plugin* para el SUT, se deriva el conjunto de acciones posibles
3. Se selecciona una acción del conjunto previamente obtenido
4. Se ejecuta la acción seleccionada
5. Mediante los oráculos y la respuesta obtenida por parte del servidor, se comprueba si el comportamiento del SUT es el esperado
6. Si se detecta un error o se alcanza el número de acciones indicadas para cada secuencia (a través del campo habilitado para ello en la interfaz gráfica de usuario) se procede a detener la secuencia en ejecución y se comprueba si aún quedan por realizar. En dicho caso, se procede a iniciar otra secuencia y se completa el ciclo entero de nuevo. En cambio, si ya se han ejecutado todas las secuencias indicadas por el usuario, se procede a finalizar la ejecución de las pruebas automáticas y se cierra la aplicación TESTAR para la IoT.

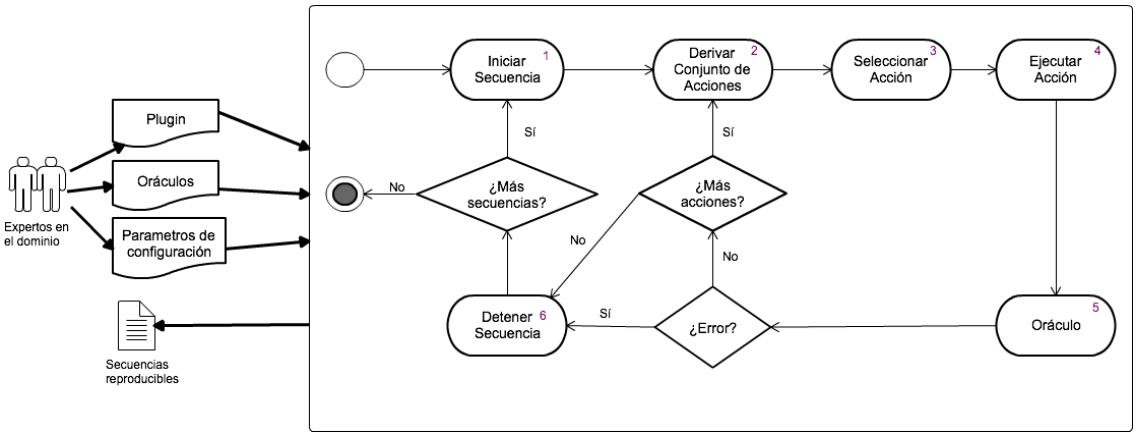


Ilustración 3 - Flujo de TESTAR para la IoT

Como se puede observar en las ilustraciones 1 y 3, el comportamiento de ambas herramientas es muy similar. Esto tiene sentido debido al hecho de que se considera que el enfoque de TESTAR resulta interesante para el testeo de los servicios de la IoT.

No obstante se pueden identificar varias diferencias:

1. Los expertos del dominio proporcionan como entrada necesaria para ejecutar la aplicación un *plugin* que contiene la información requerida para poder interactuar con la solución a testear. En el apartado 5.4.6 se indican los componentes que ha de tener todo *plugin* desarrollado para la herramienta y se muestran los detalles de implementación del específico para la vivienda inteligente
2. En este caso no disponemos de un ejecutable que hayamos de poner en marcha. Por ello, podemos iniciar directamente la secuencia cuyas acciones ya contienen los URIs de los recursos con los cuales se desea interactuar
3. Se utiliza un nuevo elemento llamado ActionBuilder (descrito en el apartado 5.4.6) para derivar las posibles acciones que se pueden ejecutar en el SUT teniendo en cuenta tanto sus características como el modo de ejecución seleccionado. Puesto que las acciones pueden contener valores (p. ej. Encender la luz gradual al 60%), se derivarán en cada ciclo previamente a la selección de una de ellas. De este modo, el conjunto de acciones será más dinámico puesto que en cada iteración se derivarán unos valores diferentes.

5.2 Componentes a alto nivel

Tal y como se ha comentado, TESTAR utiliza la API de accesibilidad del sistema operativo para adquirir la información que necesita. Puesto que dicha API depende del sistema operativo que se esté utilizando, si queremos testear aplicaciones, por ejemplo, en Windows necesitamos un *plugin* diferente al que se utilizará para testear en MacOSX.

No obstante, en este caso la API de accesibilidad no nos puede devolver la información que necesitamos. Por ello, se ha decidido que nuestra solución también funcione mediante *plugins* pero esta vez desarrollados para cada SUT que se desee poner a prueba. Dichos *plugins* deberán contener la especificación de los recursos con los cuales cuenta el sistema, de modo que TESTAR pueda derivar todas las acciones posibles a partir de dicha información. Así pues, siempre dispondrá como mínimo de dos clases: una que implementa la interfaz Recurso y otra que implementa la interfaz ActionBuilder y que permitirá derivar las acciones desde la aplicación teniendo el cuenta el modo de ejecución seleccionado.

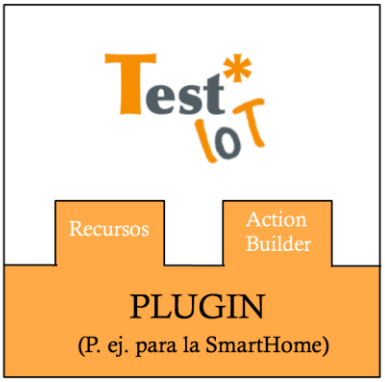


Ilustración 4 - Estructura mediante *plugin* de TESTAR

En la ilustración 4 se puede observar que TESTAR para la IoT se puede separar en dos partes, la primera se corresponde a la aplicación en sí y la otra se trata del *plugin* necesario para testear un sistema determinado.

Del mismo modo, dentro de la parte correspondiente a la aplicación en sí podemos diferenciar diferentes partes:

- Core. Consiste en la parte central de la aplicación. Entre sus componentes se encuentran las implementaciones de las acciones que se llevarán a cabo durante la ejecución de las pruebas. Se ha tratado de reutilizar el core de la aplicación TESTAR original adaptándolo a las necesidades actuales. Por ejemplo, las acciones que contiene difieren de las disponibles durante el testeo a nivel de interfaz de usuario (clics, escribir, desplazar, etc.). En la tabla 4 podemos encontrar los diferentes componentes del core indicando sí existen en la aplicación original y, en dicho caso, la adaptación necesaria para la nueva herramienta (si se ha requerido alguna).

Componente	Existe en TESTAR original (Sí/No)	Adaptaciones necesarias
Action (Interfaz)	Sí	Se mantiene el componente pero los métodos especificados son diferentes ya que son característicos de la versión de TESTAR para la IoT: - run - getResponse - isTimeout - isConnectionProblems - getUri - getPaylod
CompoundAction	Si	Puesto que implementa la interfaz Action, contiene la implementación de los nuevos métodos especificados en ella
Delete	No	
Get	No	
Post	No	
Put	No	
Resource (Interfaz)	No	
Tag	Sí	
Tags	Sí	Se eliminan *tags* relativos a elementos de las interfaces gráficas puesto que ya no resultan necesarios (p. ej. Slider, Modal, KeyboardFocus y Shape). Además, se añade el *tag* ActionsExecuted para referenciar al conjunto de acciones que se han ejecutado en una secuencia (nos facilita poder reproducirla posteriormente)
Taggable (Interfaz)	Sí	
TaggableBase	Si	
Verdict	Sí	Se elimina el método visualizer del componente original puesto que su función es devolver el objeto Visualizer encargado de pintar la zona en la que se encuentra un error en la interfaz gráfica del SUT y en este caso no disponemos de ella.
ActionBuildException	Sí	
ActionFailedException	Sí	
NoSuchTagException	Sí	
TestarException	Sí	
Assert	Sí	Se ha aplicado la refactorización sobre un método para aplicarle un nombre intuitivo. Puesto que comprueba si los argumentos pasados como parámetros son validos, pasa de llamarse isTrue a llamarse checkValidParameters
Pair	Sí	
UnProc	Sí	
Util	Sí	Se añaden los métodos getResponseBody, encodeBase64 y decodeBase64 (ver 5.4.5)

Tabla 4 - Componentes del core de la nueva herramienta

- Interfaz gráfica de usuario. Esta parte se ha tenido que implementar desde cero para dar soporte a las nuevas necesidades actuales. Puesto que los modos de ejecución de la herramienta original estaban muy enfocados al testeo de interfaces gráficas, no resultaban apropiados para el testeo de los servicios web. Así pues, se ha desarrollado una nueva interfaz gráfica que permita a los usuarios seleccionar nuevos parámetros de configuración pensados específicamente para la solución desarrollada (en las figuras 5, 6, 7 y 8 se muestran diversas capturas de pantalla mediante las cuales se puede observar la nueva interfaz gráfica de la aplicación).
- Protocolo. Puesto que, tal y como se ha comprobado en las ilustraciones 1 y 3, se pretende que el comportamiento de la aplicación desarrollada sea muy similar al de la original, la estructura del protocolo se mantiene.

5.3 Funcionalidades

Puesto que las pruebas realizadas en TESTAR para la IoT se centran en un nivel diferente al de la aplicación original (servicios web frente a interfaces gráficas de usuario) las necesidades a cubrir también difieren de las originales. Así pues, pese a tener una estructura y un flujo muy similar, la herramienta se ha tenido que rediseñar para dar soporte a dichas necesidades. De hecho, gran parte de la funcionalidad se ha extraído de la herramienta original, no obstante, para conseguir su correcto funcionamiento se han requerido cambios. En la tabla 5 se resumen las funcionalidades con las que cuenta la aplicación desarrollada, indicando para cada una de ellas si existe en la aplicación original y, si es el caso, las adaptaciones requeridas para la nueva versión de TESTAR. Tras ello, se explican dichas funcionalidades con más detalle, proporcionado de este modo una visión general de la herramienta.

Funcionalidad	**Existe en TESTAR original (Sí/No)**	**Adaptaciones necesarias**
Iniciar generación de pruebas	Sí	Se modifican los parámetros de configuración a tener en cuenta para generar los casos de prueba.
Reproducir secuencia	Sí	A pesar de que la funcionalidad es la misma, la implementación difiere. Puesto que en la aplicación original las acciones eran esencialmente distintas, el replay test se ha reescrito quedando tal y como se muestra en la figura 14
Cargar un fichero de configuración	Sí	Se ha modificado para atender a los nuevos parámetros de configuración y para rellenar correctamente los campos de la nueva interfaz gráfica de usuario según el contenido del fichero
Guardar un fichero de configuración	Sí	Se han reutilizado los pasos a realizar para guardar el fichero testar.settings. No obstante,

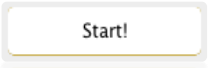
		puesto que el contenido de dicho fichero es diferente al original, se han realizado modificaciones para extraer su información y almacenarla para su futuro uso
Pruebas positivas	No	
Pruebas negativas	No	
Pruebas sin autorización	No	
Oráculos simples	Sí	Se mantiene la posibilidad de crear oráculos simples a través de la interfaz gráfica. No obstante, dichos oráculos son totalmente diferentes a los originales ya que están pensados para los nuevos modos de ejecución de la herramienta
Edición del protocolo	Sí	
Establecer número de secuencias	Sí	
Establecer número de acciones por secuencia	Sí	
Establecer tiempo de espera entre acciones	Sí	
Establecer tiempo máximo de respuesta (*time out*)	No	
Establecer directorio de salida	Sí	
Establecer directorio temporal	Sí	
Obtención de un fichero .log	Sí	Puesto que las acciones realizadas difieren de las originales, se ha modificado el contenido del fichero .log para mostrar la nueva información de un modo claro (ver figura 10)

Tabla 5 - Funcionalidades de TESTAR para la IoT

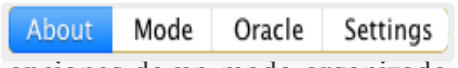
5.3.1 Opciones principales

Cuando iniciamos TESTAR para la IoT hay una serie de opciones que están disponibles desde cualquier lugar de su interfaz gráfica de usuario.

Ilustración 5 - Pestaña *About* en TESTAR para la IoT

- 

: Permite empezar la ejecución de pruebas automáticas atendiendo a la configuración establecida.
- 
: La herramienta guarda las secuencias realizadas y ofrece la posibilidad de reproducirlas. Mediante este botón podemos seleccionar la secuencia a reproducir desde nuestro sistema de archivos. Esto es especialmente útil tanto para comprobar si un error encontrado es reproducible como para ver si las modificaciones realizadas tras encontrar un error han sido exitosas y se ha solucionado el problema
- 
: Permite cargar un fichero de configuración previamente guardado. Este fichero tiene la extensión *.settings* y es utilizado por la herramienta para conocer los valores que el usuario desea para su ejecución.
- 
: Permite guardar un archivo con la configuración actual establecida a través de la interfaz de usuario. De este modo, podemos disponer de un conjunto de configuraciones y utilizar aquella que más nos interese sin tener que modificar los parámetros en cada ocasión

- 

: Con el objetivo de mostrar las diferentes opciones de un modo organizado, se han habilitado diferentes pestañas en las cuales se han agrupado los parámetros de configuración de la herramienta. Por defecto, la pestaña mostrada al iniciar la aplicación es la que contiene información sobre la herramienta como por ejemplo su versión y su logo.

5.3.2 Modos de ejecución

Con el objetivo de obtener una mejor experiencia de uso y ampliar el abanico de posibilidades que ofrece la herramienta, se han establecido 3 modos de ejecución en función de los cuales se derivarán una acciones u otras.

- Pruebas positivas: En este modo de ejecución se derivarán todas las acciones válidas teniendo en cuenta si se necesita autorización para poder ejecutarlas o no. Es decir, se realizarán únicamente acciones bien formuladas que tengan sentido teniendo en cuenta la especificación realizada para el *plugin* del SUT (p. ej. encender una luz en la vivienda inteligente)
- Pruebas negativas: Únicamente se derivarán acciones que traten de actualizar los recursos (acciones PUT) de un modo inválido (p. ej. tratar de abrir una luz en la vivienda inteligente). Esto nos servirá para comprobar si las acciones inválidas suponen algún tipo de riesgo para el SUT
- Pruebas sin autorización: Se derivarán únicamente aquellas acciones para las cuales se haya indicado que necesitan autorización. Sin embargo, se ejecutarán sin las credenciales (*token*) necesarias. Esto nos permitirá comprobar si en algún momento es posible interactuar con la API sin disponer de dicha información

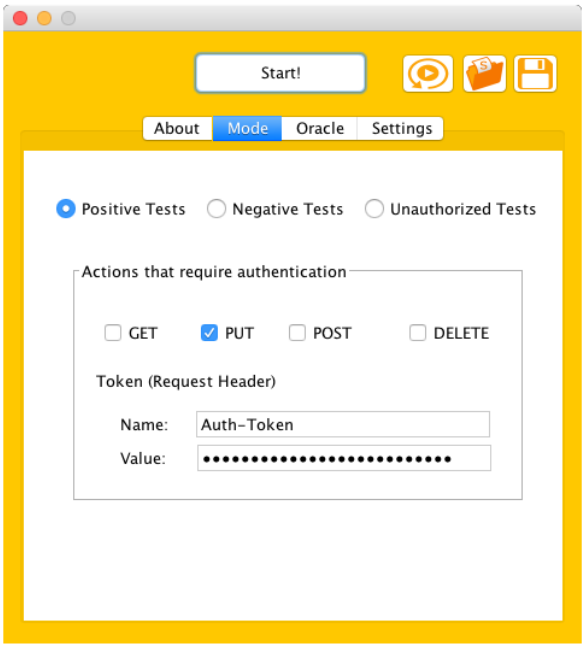


Ilustración 6 - Pestaña *Mode* en TESTAR para la IoT

Puesto que según el SUT que se esté tratando las operaciones para crear, actualizar, obtener y borrar un recurso pueden requerir o no autorización, TESTAR para la IoT permite que se indique en cada tipo de acción si se requiere utilizar un *token* en el momento de realizar la petición. Cabe indicar que si ninguna acción requiere el *token* de autorización como parámetro, el modo de pruebas no autorizadas se deshabilitará puesto que no derivaría ninguna acción a ejecutar. Del mismo modo, si ninguna acción requiere el *token* o si se ha seleccionado el modo de pruebas no autorizadas, los campos disponibles para indicar el valor de las credenciales también se deshabilitarán por no ser necesarios en dicho casos.

5.3.3 Oráculos simples

TESTAR para la IoT ofrece, por defecto, varios oráculos simples que se ejecutarán en función del modo de ejecución seleccionado. Todos ellos se especifican mediante expresiones regulares[35].

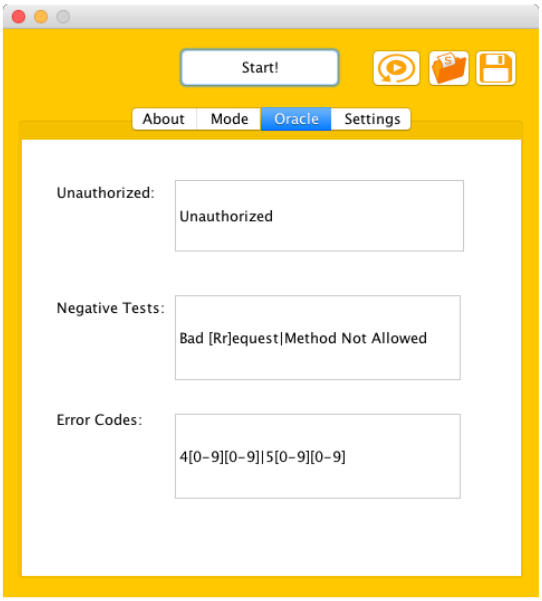


Ilustración 7 - Pestaña Oracle en TESTAR para la IoT

- No Autorizado: Si, en el modo de pruebas no autorizadas, el mensaje de respuesta proporcionado por el servidor tras realizar una acción no coincide con la expresión proporcionada, la herramienta considerará que se ha producido un error en la secuencia actual. Por ejemplo, sabemos que en la vivienda inteligente se requiere el *token* de autorización para actualizar el estado de un recurso y que si no se proporciona la respuesta recibida por parte del servidor deberá ser *401 Unauthorized*. Mediante este oráculo nos aseguramos de que la aplicación comprobará que efectivamente se recibe dicho mensaje y, en caso contrario, nos informará de lo ocurrido mediante un mensaje como el siguiente *Detected fault: severity 1.0 info: Error: [Mensaje Recibido] received while expecting [Mensaje Esperado].*

- Pruebas negativas: Si, en el modo de pruebas negativas, el mensaje recibido por parte del servidor no coincide con el indicado en el campo disponible, TESTAR para la IoT considerará que se ha producido un error y avisará mediante el mensaje *Detected fault: severity: 1.0 info: Error: [Mensaje Recibido] received while expecting [Mensaje Esperado]*. En el caso de la vivienda inteligente, sabemos que la respuesta esperada frente a este tipo de acciones puede ser o bien 400 Bad Request o 405 Method Not Allowed. Así pues, mediante este oráculo si recibimos una respuesta que difiera de uno de esos dos mensajes, la aplicación nos informará de que algo inesperado ha ocurrido.
- Pruebas positivas: Mediante este oráculo simple, los usuarios pueden indicar aquellos códigos que consideran indicadores de que se ha producido un error. Cuando se esté ejecutando TESTAR para la IoT en el modo de pruebas positivas, se comprobará el código recibido por parte del servidor tras cada acción ejecutada. Si dicho código coincide con uno de los que se han indicado como erróneos, la aplicación avisará mediante el mensaje *Detected fault: severity: 1.0 info: Error: [Código + Mensaje Recibido] received*. Por defecto se indican como códigos de error todos aquellos que empiecen por 4 (errores del cliente) y por 5 (errores del servidor).

5.3.4 Configuración

Mediante la pestaña *Settings* de la interfaz de usuario de la herramienta, se pueden indicar diferentes parámetros de configuración que se tendrán en cuenta durante la ejecución de las pruebas.

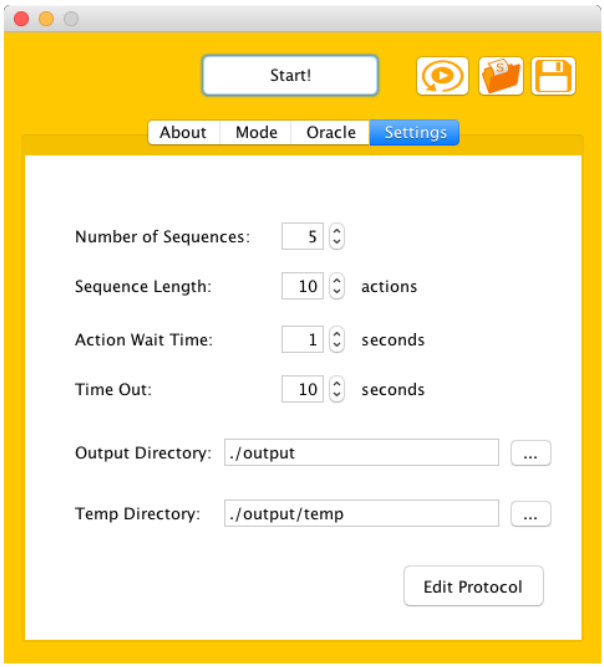


Ilustración 8 - Pestaña Settings en TESTAR para la IoT

1. Número de secuencias: Permite indicar la cantidad de secuencias que se desean llevar a cabo
2. Tamaño de las secuencias: Indica la cantidad de acciones que se producirán en cada una de las secuencias realizadas
3. Tiempo de espera entre acciones: Tras realizar una acción, la herramienta esperará el tiempo indicado en este campo para ejecutar la siguiente acción
4. Tiempo máximo de espera: Si una acción necesita más tiempo del indicado en este campo para recibir una respuesta por parte del servidor, se considerará que se ha producido un error en la secuencia y se avisará mediante el mensaje *Connection to [URI del Recurso] timed out*
5. Directorio de salida: Permite indicar el directorio en el que se guardarán las salidas producidas por TESTTAR para la IoT.
6. Directorio temporal: Permite indicar el directorio en el que se almacenarán archivos temporales durante la ejecución de las secuencias.
7. Editar el protocolo: La aplicación desarrollada permite que los usuarios editen el código fuente que determina su comportamiento mediante el editor del protocolo. Esto permite a los usuarios implementar acciones más complejas así como protocolos más sofisticados. Tras pulsar el botón disponible para ello, se muestra el código fuente del protocolo en lenguaje Java junto con un botón para guardar/compilar los cambios modificados y una consola que informa de cómo se ha realizado la compilación.

Método	Descripción
void initialize(Settings settings)	Configuración inicial. Se ejecuta antes de comenzar las secuencias de pruebas
void beginSequence()	Se ejecuta antes de empezar una nueva secuencia de pruebas.
Verdict getVerdict(String response)	Permite especificar oráculos más sofisticados
Set<Action> deriveActions()	Obtiene el conjunto de las acciones disponibles. Por tanto, se puede usar para modificar la derivación de dichas acciones.
Action selectAction(Set<Action> actions)	Sirve para seleccionar la próxima acción a ejecutar. Actualmente las acciones se seleccionan aleatoriamente pero se podrían utilizar, por ejemplo, metaheurísticas.
boolean executeAction(Action action)	Ejecuta la acción que recibe por parámetro. Así pues, permite modificar el comportamiento de TESTAR para la IoT cuando procede a ejecutar una acción.
boolean moreActions()	Determina el criterio de parada.
void finishSequence()	Se invoca cada vez que se va a finalizar una secuencia.
boolean moreSequences()	Determina si se van a continuar realizando secuencias de pruebas o si, por el contrario, ya se ha llegado al fin de la ejecución

Tabla 6 - Métodos modificables desde el protocolo

5.3.5 Salidas producidas

Durante la ejecución de las pruebas, TESTAR para la IoT almacena todas las secuencias en el directorio de salida que se le ha especificado. Dichas secuencias son reproducibles, lo cual nos permitirá volver a realizar un conjunto de acciones en el mismo orden con el cual se realizaron en su momento.

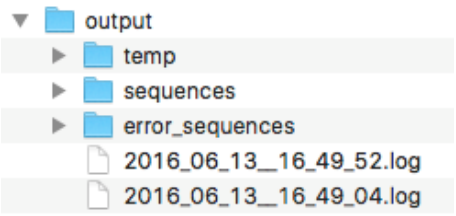


Ilustración 9 – Directorio de salida

- /output/sequences: TESTAR para la IoT guarda todas las secuencias en este directorio. Aquí podemos encontrar tanto las secuencias en las que se ha producido un error como aquellas que se han completado exitosamente
- /output/error_sequences: Este directorio únicamente contiene aquellas secuencias en las cuales se ha identificado un error. Por tanto, si este directorio no está vacío tras realizar ejecuciones con la herramienta, podemos deducir que el comportamiento no ha sido el esperado y proceder a investigar lo ocurrido.

- Ficheros .log: Por cada ejecución de la herramienta se almacena un fichero .log que se puede abrir con cualquier editor de texto. Estos ficheros disponen de un título con formato *año_mes_dia_hora_minutos_segundo.log* para facilitar su identificación y nos proporcionan información extensa sobre todo lo sucedido (véase el anexo C para contemplar un ejemplo real de un fichero .log producido por la herramienta).

1. Hora y fecha del incio de la ejecución
2. Parámetros de configuración utilizados
3. Para cada secuencia producida
   3.1. Fecha y hora del inicio de la secuencia
   3.2. Nombre con el que se guardará la secuencia en el directorio de salida
   3.3. Para cada acción producida
      3.3.1. Cantidad total de acciones derivadas
      3.3.2. Acción seleccionada y ejecutada
         P. ej. Get Resource http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/SensorPuertaCerrada'
      3.3.3. Respuesta del servidor
         P. ej. Code: 200 Message: OK Body:
{"lastAction":"unknown","currentState":"UNKNOWN","link":"/thing/SensorPuertaCerrada","actuator":true,"isEnabled":true,"id":"SensorPuertaCerrada","sensor":true,"isQuiescent":false,"name":"Sensor Puerta Cerrada", "device":{"id":"Puerta","name":"Puerta","isEnabled":true},"previousState":"UNKNOWN","isStarted":true,"devfunc-type":"bistate"}
      3.3.4. Si error == true
         3.3.4.1 Información del error
            P. ej. Detected fault: severity: 1.0 info: Error: 404 Not Found received
4. Fecha y hora del final de la ejecución

Ilustración 10 - Estructura del fichero .log de salida

5.4 Implementación

En este apartado se pretende explicar la implementación de aquellos aspectos que han resultado más relevantes durante el desarrollo de la herramienta.

5.4.1 BuildFile

Con el objetivo de conseguir una aplicación multiplataforma y evitar la dependencia con órdenes del intérprete de comandos de un sistema operativo concreto, se ha recurrido a la herramienta Apache ANT[36]. Dicha herramienta nos permite realizar tareas que resultan mecánicas y repetitivas, lo cual nos es especialmente útil durante la fase de compilación y de construcción de la aplicación.

Cuando no se especifican argumentos en la ejecución, la herramienta busca en el directorio actual un fichero con nombre build.xml y, si lo encuentra, ejecuta el objetivo por defecto. Por ello, en nuestro caso hemos creado un fichero build.xml en el cual disponemos del siguiente conjunto de objetivos:

- jar: crea el archivo Jar de la aplicación. Además, se encarga de elaborar la siguiente estructura de directorios
  - target: lugar donde se ubica el Jar elaborado
  - target/resources: destino de las imágenes utilizadas durante la compilación de la herramienta
  - target/output: directorio en el cual se ubicarán las salidas de las ejecuciones llevadas a cabo
  - target/output/temp: directorio para guardar archivos temporales
- compile: como su propio nombre indica, se encarga de compilar el código fuente guardándolo en el directorio *bin*
- clean: se encarga de eliminar los directorios *bin* y *target* para proceder a la creación del Jar sin archivos no deseados, fruto de una compilación anterior

Puesto que se pueden indicar dependencias que establezcan que antes de ejecutar un objetivo se tenga que ejecutar otro del cual dependa, se ha establecido como objetivo por defecto el encargado de construir el Jar que a su vez depende del encargado de compilar el código fuente.

De este modo, cuando queramos compilar el código, bastará con situarnos en el directorio en el cual disponemos del fichero build.xml y ejecutar el comando “ant”. Si funciona según lo esperado obtendremos el directorio target con su respectivo iot_testar.jar

```
BUILD SUCCESSFUL
Total time: 10 seconds
priv2g17-81:iot_testar_ mimarmu1$ ls target/
iot_testar.jar  output          resources       test.settings
```

Ilustración 11 – Contenido del directorio target tras compilar mediante Apache ANT

5.4.2 testar.settings

La herramienta desarrollada carga y almacena los diferentes parámetros de configuración a partir del fichero testar.settings del cual dispone.

```
1  TimeToWaitAfterAction = 1.0
2  SuspiciousCodes = 4[0-9][0-9]|5[0-9][0-9]
3  UnauthorizedText = Unauthorized
4  DeleteNeedsAuthentication = false
5  TimeOutTime = 10
6  TokenValue = M29lNDd2cWRmaWd0OTJ1MGF2bmRlM2plY2s=
7  ShowSettingsAfterTest = true
8  PathToReplaySequence = ./output
9  PutNeedsAuthentication = true
10 OutputDir = ./output
11 GetNeedsAuthentication = false
12 SequenceLength = 6
13 Mode = Positive
14 TempDir = ./output/temp
15 ShowVisualSettingsDialogOnStartup = true
16 NegativeTestsText = Bad [Rr]equest|Method Not Allowed
17 TokenName = Auth-Token
18 Replay = false
19 PostNeedsAuthentication = false
20 Sequences = 6
21
```

Ilustración 12 - Contenido del fichero testar.settings

Los valores que aparecen en la ilustración 12 se corresponden con los explicados previamente en el apartado 5.3 (modificables a través de la interfaz gráfica de la herramienta), no obstante cabe indicar que el valor del *token* (propiedad llamada TokenValue) aparece cifrado con la intención de evitar escribirlo en plano en un fichero de texto.

Nada más arrancar la aplicación, TESTAR para la IoT ejecuta el método loadSettings que, tal y como se puede observar en la ilustración 13, se encarga de cargar en un ArrayList de pares clave-valor todas las propiedades de configuración con un cierto valor por defecto. Tras ello, ejecuta el método estático fromFile de la clase Settings al cual le pasa la lista de valores por defecto y el fichero en el cual se encuentran los valores que queremos cargar (test.settings si no se indica lo contrario). Así pues, dicho método se encarga de obtener las propiedades del fichero y crear un objeto que se utilizará cada vez que se quiera acceder a los parámetros de configuración durante la ejecución de la herramienta.

```
public static Settings loadSettings(String[] argv, String file) throws ConfigException{
    try{
        List<Pair<?, ?>> defaults = new ArrayList<Pair<?, ?>>();

        defaults.add(Pair.from(OutputDir, "."));
        defaults.add(Pair.from(TempDir, "."));
        defaults.add(Pair.from(TimeToWaitAfterAction, 0.1));
        defaults.add(Pair.from(TimeOutTime, 5));
        defaults.add(Pair.from(SequenceLength, 10));
        defaults.add(Pair.from(SuspiciousCodes, "4[0-9][0-9]|5[0-9][0-9]"));
        defaults.add(Pair.from(UnauthorizedText, "Unauthorized"));
        defaults.add(Pair.from(NegativeTestsText, "NegativeTestText"));
        defaults.add(Pair.from(Sequences, 1));
        defaults.add(Pair.from(ShowSettingsAfterTest, true));
        defaults.add(Pair.from(Mode, "Positive"));
        defaults.add(Pair.from(PutNeedsAuthentication, true));
        defaults.add(Pair.from(GetNeedsAuthentication, false));
        defaults.add(Pair.from(DeleteNeedsAuthentication, false));
        defaults.add(Pair.from(PostNeedsAuthentication, false));
        defaults.add(Pair.from(TokenValue, ""));
        defaults.add(Pair.from(TokenName, ""));
        defaults.add(Pair.from(Replay, false));
        defaults.add(Pair.from(PathToReplaySequence, "./output"));

        return Settings.fromFile(defaults, file == null ? "./test.settings" : file);
    }catch(IOException ioe){
        throw new ConfigException("Unable to load configuration file!", ioe);

    }
}
        public static Settings fromFile(List<Pair<?, ?>> defaults, String path) throws IOException{
            Assert.notNull(path);
            Properties props = new Properties();
            props.load(new FileInputStream(path));
            return new Settings(defaults, props);
        }
```

Ilustración 13 - Carga de los parámetros de configuración

Con el objetivo de mostrar los datos en la interfaz gráfica de la herramienta y permitir que el usuario los modifique a través de ella, disponemos de los métodos populateInformation y extractInformation respectivamente. El primero extrae los valores de configuración e introduce cada uno de ellos en su campo correspondiente de la interfaz, mientras el segundo se ejecuta cada vez que se lanza una ejecución de pruebas y se encarga de obtener los valores de los campos de la interfaz para actualizarlos en el fichero testar.settings.

Además, tal y como se ha indicado en el apartado 5.3.1 disponemos de funcionalidades para guardar y cargar diversos archivos de configuración. Esto se consigue reutilizando los métodos ya comentados. Cuando deseamos cargar un fichero, se ejecuta el método loadSettings pasándole como parámetro el fichero que ha seleccionado el usuario y, tras ello, se muestran los datos en la interfaz gráfica (método populateInformation). Así mismo, si se desean guardar los parámetros en un fichero de configuración sin haber iniciado antes una ejecución de pruebas, la pulsación del botón guardar invocará el método extractInformation obteniéndose así los datos actualizados y guardándolos en el correspondiente fichero de configuración.

5.4.3 Protocolo

Tal y como se ha comentado, el protocolo dictamina el comportamiento de la aplicación y dispone de los métodos mostrados en la tabla 6. Así pues, cabe indicar que en la herramienta disponemos de 3 niveles de protocolo diferentes, todos ellos relacionados entre sí tal y como se indica en la ilustración 14.

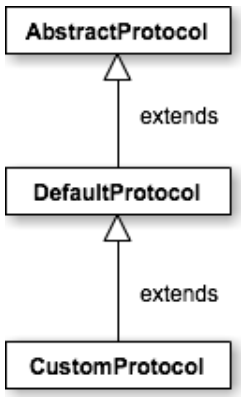


Ilustración 14 – Relación entre los diferentes protocolos

- AbstractProtocol: Se trata de una clase abstracta cuyos métodos son abstractos, es decir, no disponen de implementación. En dicha clase se describe el flujo a seguir tanto cuando se inicia una nueva ejecución de pruebas como cuando se procede a reproducir una secuencia previamente ejecutada. Así pues, dispone de los métodos runTest y replayTest los cuales serán ejecutados en función de la opción deseada. En la ilustración 15 se muestran dichos métodos en pseudocódigo para facilitar su comprensión.

runTest()

1. INICIO
2. *sequenceCount* = 1
3. Mientras *moreSequences()*
4. Generar fichero para la secuencia actual
5. *actionCount* = 1
6. Mientras *moreActions()*
7. *deriveActions()*
8. *selectAction()*
9. *executeAction()*
10. *actionCount++*
11. Guardar acción en la lista de acciónes ejecutadas
12. *getVerdict()*
13. Guardar acciones ejecutadas en el fichero de la secuencia actual
14. *finishSequence()*
15. Copiar secuencia a output/sequences
16. Si (*faultySequence*)
17. Copiar secuencia a output/error_sequences
18. *sequenceCount++*
19. FIN

replayTest()

1. INICIO
2. *success* = true
3. Obtener secuencia a reproducir (a través del tag *PathToReplay*)
4. Obener actionsExecuted de la secuencia a reproducir
5. Si (*actionsExecuted*.isEmpty || *actionsExecuted* == null)
6. *success* == false
7. Si no
8. Para *action* en *actionsExecuted*
9. *executeAction()*
10. Si (*actionSucceded*)
11. *getVerdict()*
12. si no
13. *success* = false
15. *finishSequence()*
16. Si (*success*)
17. Log(Sequence successfully replayed)
18. Si no
19. Log(Failed to replay sequence)
20. FIN

Ilustración 15 - Pasos para implementar la ejecución de pruebas y la reproducción de secuencias

- DefaultProtocol: Implementa los métodos abstractos de los cuales dispone el AbstractProtocol. Por tanto, proporciona el comportamiento por defecto del cual ha de disponer la herramienta. A continuación se indica la implementación para cada uno de ellos.

    - initialize(Settings settings): Instancia el objeto de la clase ActionBuilder que será utilizado en el protocolo de la aplicación para derivar las acciones correspondientes. Además, como parte de la inicialización también obtiene los parámetros de configuración establecidos por el usuario (tiempo límite de espera, nombre y valor del *token*, acciones que requieren parámetros de autorización y el modo de ejecución seleccionado)
    - beginSequence(): Durante la ejecución de una secuencia, la variable faultySequence se establece a verdadero si se detecta un error. Este método se aprovecha para reiniciar su valor igualándola a falso, lo cual nos permite asegurarnos de que cada secuencia se inicia sin indicar que un error ya ha sido encontrado
    - getVerdict(String response): La respuesta recibida está formada por tres componentes: código, mensaje y cuerpo. Tal y como se ha indicado en el apartado 5.3.3, existen una serie de oráculos simples que se proporcionan por defecto y se aplican en función del modo de ejecución. Teniendo esto en cuenta, en este método se proporciona un *switch* que permitirá aplicar un oráculo u otro en función de la variable *mode* instanciada en el método *initialize*.

```
switch (mode) {
case "Positive":
    String codesRegEx = settings().get(ConfigTags.SuspiciousCodes);
    if (responseCode.matches(codesRegEx)) {
        return new Verdict(1.0, "Error: " + responseCode + " " + responseMessage + " received");
    }
    break;
case "Negative":
    String notCoherentRegEx = settings().get(ConfigTags.NegativeTestsText);
    if (!responseMessage.matches(notCoherentRegEx)) {
        return new Verdict(1.0, "Error: " + responseMessage + " received while expecting " + notCoherentRegEx);
    }

    break;
case "Unauthorized":
    String unauthorizedRegEx = settings().get(ConfigTags.UnauthorizedText);
    if (!responseMessage.matches(unauthorizedRegEx)) {
        return new Verdict(1.0, "Error: " + responseMessage + " received while expecting " + unauthorizedRegEx);
    }
    break;

default:
    break;
}
```

Ilustración 16 - Código para aplicar los oráculos simples

- deriveActions(): Se derivan las acciones a partir del ActionBuilder (ver apartado 5.4.6), teniendo en cuenta el modo de ejecución seleccionado

```
protected Set<Action> deriveActions() throws ActionBuildException {
    Set<Action> actions = new HashSet<Action>();

    switch (mode) {
    case "Positive":
        actions = actionBuilder.derivePositiveActions(tokenName, tokenValue, getNeedsToken, putNeedsToken,
                postNeedsToken, deleteNeedsToken);
        break;
    case "Negative":
        actions = actionBuilder.deriveNegativeActions(tokenName, tokenValue, getNeedsToken, putNeedsToken,
                postNeedsToken, deleteNeedsToken);
        break;
    case "Unauthorized":
        actions = actionBuilder.deriveUnauthorizedActions(tokenName, tokenValue, getNeedsToken, putNeedsToken,
                postNeedsToken, deleteNeedsToken);
        break;

    default:
        actions = null;
        break;
    }

    return actions;
}
```

Ilustración 17 - Código para la derivación de acciones

- selectAction(Set<Action> actions): Se selecciona aleatoriamente una acción perteneciente al conjunto de acciones derivado previamente
- executeAction(Action action): Se invoca el método run de la acción pasada por parámetro y se pausa el tiempo de espera entre acciones indicado por el usuario mediante los parámetros de configuración.

- CustomProtocol: Se trata del protocolo modificable por los usuarios a través del editor proporcionado por la herramienta. En su implementación inicial (ver ilustración 18) proporciona el comportamiento por defecto de la herramienta, puesto que extiende del protocolo por defecto y no cambia ninguno de sus métodos. No obstante, realizando modificaciones sobre él, el usuario puede obtener el comportamiento que desee.

Así mismo, para proporcionar el editor del protocolo se ha recurrido a la librería JSyntaxPane[37] la cual nos ofrece, mediante un único archivo JAR, la posibilidad de crear de un modo sencillo editores de diversos lenguajes de programación (Java, Ruby, XML, Python, etc).

```
public class CustomProtocol extends DefaultProtocol{

    protected void initialize(Settings settings){
        super.initialize(settings);
    }

    protected void beginSequence(){
        super.beginSequence();
    }

    protected Verdict getVerdict(String response){
        Verdict verdict = super.getVerdict(response);

        //-------------------------------------------------------------------------
        // MORE SOPHISTICATED ORACLES CAN BE PROGRAMMED HERE (the sky is the limit ;-)
        //-------------------------------------------------------------------------

        // ... YOU MAY WANT TO CHECK YOUR CUSTOM ORACLES HERE ...

        return verdict;

    }

    protected Set<Action> deriveActions() throws ActionBuildException{
        return super.deriveActions();
    }

    protected Action selectAction(Set<Action> actions){
        return super.selectAction(actions);
    }

    protected boolean executeAction(Action action){
        return super.executeAction(action);
    }

    protected boolean moreActions() {
        return super.moreActions();
    }

    protected void finishSequence(){
        super.finishSequence();
    }

    protected boolean moreSequences() {
        return super.moreSequences();
    }
}
```

Ilustración 18 - Clase CustomProtocol.java

En la ilustración 19 se muestra lo sencillo que es utilizar esta librería. Nos basta con llamar el método initKit del kit sintáctico elegido e indicar el *Multipurpose Internet Mail Extensions* (MIME) deseado, en nuestro caso "text/java". Así mismo, al JEditor Pane (al cual hemos llamado codeEditor) le introducimos el texto de la clase java que queremos compilar.

```
public ProtocolEditor() {
    DefaultSyntaxKit.initKit();
    initComponents();
    codeEditor.setContentType("text/java");
    codeEditor.setText(Util.readFile(new File(dirSRC + "/CustomProtocol" + ".java")));
}
```

Ilustración 19 - Código para utilizar la librería JSyntaxPanel

Por otra parte, el paquete javax.tools[38] nos proporciona interfaces a herramientas que pueden ser invocadas desde los programas. Por ello, se ha podido utilizar para invocar un compilador de Java que nos permite añadir en el editor del protocolo la opción de compilar las modificaciones realizadas y que se tengan en cuenta durante las ejecuciones de TESTAR para la IoT.

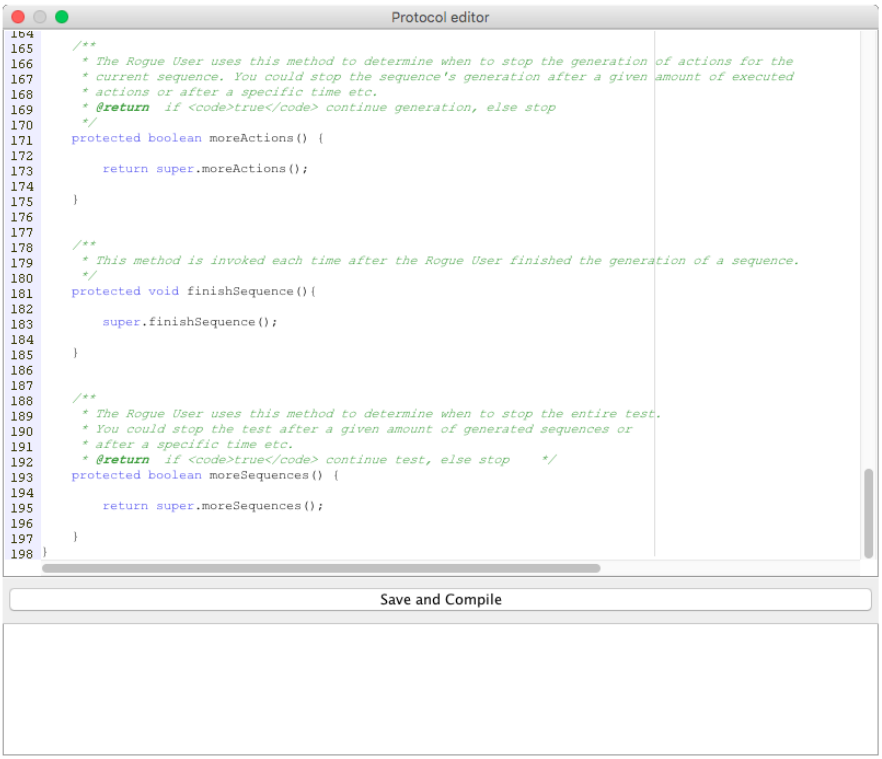


Ilustración 20 - Editor del protocolo en TESTAR para la IoT

5.4.4 Acciones

TESTAR para la IoT es capaz de ejecutar los cuatro métodos de HTTP, comentados en el apartado 4.4, que se asocian a las operaciones CRUD. Así pues, disponemos de una interfaz Action la cual es implementada por las clases Get, Put, Post y Delete.

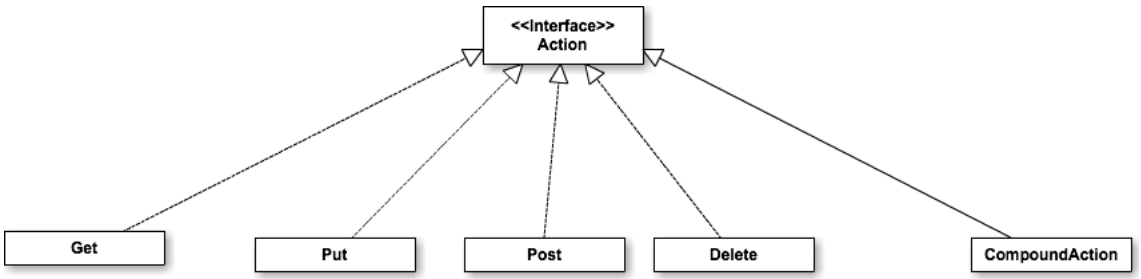


Ilustración 21 - Acciones disponibles en TESTAR para la IoT

Tal y como se puede observar en la ilustración 21, también disponemos de la clase CompoundAction la cual nos permite crear acciones compuestas. Dichas acciones están formadas por un conjunto de acciones simples (Get, Put, Post o Delete) las cuales se

ejecutarán una detrás de la otra. Esto nos puede ser útil cuando, por ejemplo, deseamos establecer una relación entre acciones de modo que una siempre vaya precedida por otra en concreto.

Puesto que el método run se encuentra declarado en la interfaz, nos aseguramos de que todas las acciones lo implementarán, no obstante la implementación en cada una de ellas variará. Además, disponer de una interfaz nos permite tratar el conjunto de acciones sin tener que diferenciar entre su tipo, es decir, podeemos mantener un Set<Action> o un ArrayList<Action> sin necesidad de diferenciar el tipo de acciones de las cuales se disponga. Esto, unido a lo anterior, nos permite seleccionar una acción y ejecutar su método run() sin tener que indicar si se trata de un Get, Put, Post o Delete.

Así mismo, si llegamos a un punto en el que queremos comprobar qué tipo de acción es la que se va a ejecutar para así poder tratarla de un modo determinado, disponemos del operador instaceOf para descubrirlo.

A continuación se explica, a modo de ejemplo, la implementación de la acción Put y, tras ello, se comentan los aspectos que diferencian las otras acciones de la explicada.

Disponemos de un constructor que recibe los siguientes parámetros:

- String uri: Identificador único del recurso sobre el cual vamos a ejecutar la acción
- Map<String, String> headers: Se trata de las cabeceras HTTP las cuales contienen información necesaria para la comunicación (p. ej. Content-Type: application/java para indicar el tipo MIME del cuerpo de la petición o Accept: application/json para establecer los Content-Types que son aceptados en la respuesta). Como se puede observar en los ejemplos propuestos, las cabeceras contienen una clave y un valor, por ello el tipo del atributo headers es un Map que nos permite almacenar pares clave-valor.
- String payload: Contenido del cuerpo de la petición
- int timeoutTime: Tiempo máximo en segundos (indicado por el usuario) que se esperará para obtener una respuesta a la petición por parte del servidor. Pasado este tiempo sin obtener respuesta, la herramienta considerará que se ha producido un error
- String tokenName: Clave de la cabecera utilizada para enviar el *token* de autorización
- String tokenValue: Valor del *token* de autorización

```
public Put(String uri, Map<String, String> headers, String payload, int timeOutTime, String tokenName,
        String tokenValue) {
    Assert.notNull(uri);
    this.uri = uri;
    this.timeOutTime = timeOutTime;
    this.headers = headers;
    this.payload = payload;
    this.tokenName = tokenName;
    this.tokenValue = tokenValue;
}
```

Ilustración 22 - Constructor de la clase Put

El método run de cada acción se encarga de realizar la petición HTTP correspondiente, mediante código Java, utilizando la clase HttpURLConnection del paquete java.net. En la ilustración 24 se muestra el cuerpo de dicho método para la acción Put.

En primer lugar creamos un localizador de recursos uniforme (URL del inglés Uniform Resource Locator) a partir de la uri de la que disponemos. Mediante dicha URL abrimos una conexión a la cual le indicamos el método de la petición HTTP (Put en este caso).

Utilizando el método setDoOutput indicamos que deseamos enviar un cuerpo de la petición (*request body*) lo cual es apropiado para las acciones Put y Post. Seguidamente recorremos el atributo headers mediante un iterador e introducimos en la conexión los pares clave-valor existentes.

Puesto que las peticiones pueden necesitar *token* o no (según lo indicado por el usuario a través de la interfaz gráfica) comprobamos si su valor es distinto de nulo y sólo en dicho caso añadimos los valores correspondientes a las propiedades de la petición.

Para indicar el tiempo máximo de espera al realizar peticiones, pasamos el valor indicado por el usuario (en segundos) a milisegundos y, tras ello, lo establecemos como el tiempo máximo de conexión y de lectura mediante los métodos setConnectTimeout y setReadTimeout respectivamente.

A continuación se escribe el cuerpo de la petición, si se dispone de uno. Para ello se instancia un OutputStreamWriter que nos permitirá escribir en la conexión, utilizando el *output stream* devuelto mediante el método getOutputStream. Una vez utilizado, se cierra para liberar los recursos asociados y nos encontramos en un punto en el que ya hemos enviado todos los datos de los que disponemos.

Por ello, estamos en las condiciones necesarias para obtener la respuesta a la acción realizada. Mediante los métodos getResponseCode, getResponseMessage y getResponseBody podemos obtener diferente información sobre la respuesta. En nuestro caso, almacenamos toda esa información con el formato Código|Mensaje|Cuerpo

200|OK{"lastAction":"unknown","currentState":"UNKNOWN","link":"/thing/SensorPuertaCerrada","actuator":true,"isEnabled":true,"id":"SensorPuertaCerrada","sensor":true,"isQuiescent":false,"name":"Sensor Puerta Cerrada","device":{"id":"Puerta","name":"Puerta","isEnabled":true},"previousState":"UNKNOWN","isStarted":true,"devfunc-type":"bistate"}

Ilustración 23 - Ejemplo de una posible respuesta almacenada

De este modo disponemos, en una única cadena, de toda la información requerida para obtener los veredictos a través de TESTAR para la IoT.

```
URL url = new URL(uri);
HttpURLConnection connection = (HttpURLConnection) url.openConnection();
connection.setRequestMethod("PUT");
connection.setDoOutput(true);

if (headers != null) {
    Iterator<String> it = headers.keySet().iterator();
    while (it.hasNext()) {
        String key = it.next();
        String value = headers.get(key);
        connection.setRequestProperty(key, value);
    }
}

if (tokenValue != null) {
    connection.setRequestProperty(tokenName, tokenValue);
}

int CONNECTION_TIMEOUT_MS = timeOutTime * 1000; // Timeout in
                                                // millis.
connection.setConnectTimeout(CONNECTION_TIMEOUT_MS);
connection.setReadTimeout(CONNECTION_TIMEOUT_MS);
if (payload != null) {
    OutputStreamWriter osw = new OutputStreamWriter(connection.getOutputStream());
    osw.write(payload);
    osw.flush();
    osw.close();
}

response = connection.getResponseCode() + "|" + connection.getResponseMessage() + "|"
        + Util.getResponseBody(connection);
```

Ilustración 24 – Contenido del método run

Por otra parte, el paquete java.net también nos proporciona una serie de excepciones que nos permiten tratar las situaciones excepcionales que pueden producirse. Así pues, las excepciones ConnectException y SocketException nos permite tratar aquellas situaciones en las que se existan problemas en la conexión. Del mismo modo, mediante SocketTimeoutException podemos tratar los casos en los que se ha sobrepasado el tiempo máximo de espera e informar al usuario de lo sucedido apropiadamente.

A pesar de que la utilización de la clase HttpURLConnection en el resto de acciones es muy similar, cada acción dispone de distintas peculiaridades:

- El método de la petición varía en función de si estamos haciendo un Get, Put, Post o Delete
- Las acciones Get y Delete no requieren cuerpo de la petición por lo que no se establece la propiedad DoOutput a cierto. En cambio, dicha propiedad si es necesaria en las acciones Put y Post
- Tal y como se indica en [39] si se introduce un cuerpo en la petición de una acción Get o Delete, algunas implementaciones pueden incluso rechazarlas. De hecho, existen clientes como por ejemplo Postman que directamente no aceptan la introducción de un cuerpo en una petición Get. Por ello, en este tipo de acciones no se ha implementado la posibilidad de añadir un cuerpo a la petición

Cabe indicar que, con el objetivo de mostrar información apropiada a través del log, todas las acciones disponen de un método toString que nos permite sacar, mediante texto, información sobre el tipo de acción que se está realizando y el recurso con el cual se está interactuando.

5.4.5 Utilidades

En el *core* de la aplicación disponemos de una clase Util.java en la que podemos encontrar diversos métodos estáticos que resultan de interés en diferentes partes del código. Recogiendo dichos métodos en una única clase y haciéndolos estáticos podemos reutilizarlos sin necesidad de duplicar código ni de crear un objeto del tipo Util. A continuación se listan los métodos disponibles en la clase Util.java junto con una breve descripción de cada uno de ellos:

- String dateString(String format): Devuelve una cadena con la fecha actual en el formato pasado por parámetro
- int hashCode(Object o): Se comprueba que el objeto pasado por parámetro no sea nulo y, en ese caso, se devuelve un int con su HashCode[40] calculado. El HashCode de un objeto consiste en un identificador de 32 bits y resulta especialmente interesante para agrupar objetos en base a dicho cálculo y, así, mejorar el rendimiento de las tablas hash y otras estructuras de datos
- File generateUniqueFile(String dir, String prefix): Genera un archivo en el directorio indicado cuyo nombre contiene el prefijo pasado por parámetro concatenado con un entero (empezando desde cero, el más pequeño que no exista con anterioridad). Esto nos sirve para generar los ficheros de secuencias con el formato sequence1, sequence2, sequence3, …, sequenceN
- String getLineSep(): Devuelve el separador de línea del sistema actual (Sytem.getProperty("line.separator")
- void saveToFile(String content, String pathToFile): Guarda el contenido pasado por parámetro en el archivo indicado (mediante la ruta pasada como cadena)
- void pause(double seconds): Realiza una pausa de la cantidad de segundos pasada por parámetro
- void delete(File fileOrDirectory): Elimina el archivo o directorio pasado como parámetro. En el caso de los directorios se eliminan, mediante recursión, todos los archivos que contiene
- void copyToDirectory(File fileOrDirectory, File DestDir, String targetName): Copia en el directorio indicado como parámetro el archivo o directorio proporcionado con el nombre indicado en targetName. Si el nombre es nulo, se copia con el nombre del archivo fileOrDirectory
- String readFile(File path): Devuelve en una cadena de texto el contenido del archivo pasado como parámetro
- List<File> getAllFiles(File dir, String extension): Devuelve una lista con los archivos que se encuentran dentro del directorio indicado cuyo nombre termine con la extensión pasada como parámetro. En el caso de existir otros directorios

dentro del directorio del cual se desean extraer los archivos, se utiliza la recursión para sacar aquellos que cumplan dicha condición dentro de ellos

- getResponseBody(HttpURLConnection connection): Permite extraer el cuerpo de la respuesta de una petición HTTP
- String encodeBase64(String text): Permite cifrar en base64 el texto pasado como parámetro
- String decodeBase64(String text): Permite descifrar en base64 el texto pasado como parámetro

5.4.6 *Plugin* de la Vivienda Inteligente

La realización de un *plugin* específico para un SUT requiere conocer las características de dicho sistema y, así, poder implementar un ActionBuilder que permita derivar las operaciones apropiadas en función del modo de ejecución seleccionado.

Así pues, la clase ActionBuilder.java junto con la clase Resource.java (cabe tener en cuenta que estamos testeando los servicios que nos permiten acceder a los recursos) estarán presentes en todo *plugin* implementado.

El primer paso, por tanto, es ver cómo podemos implementar los recursos en nuestro *plugin*. La vivienda inteligente (ver apartado 4) está formada por 17 recursos y cada uno de ellos tiene asociada una funcionalidad concreta. Así mismo, cada funcionalidad dispone de una serie de interacciones válidas (las cuales pueden requerir un valor o no). Teniendo esto en cuenta se han desarrollado 3 clases relacionadas entre ellas tal y como muestra en la ilustración 25.

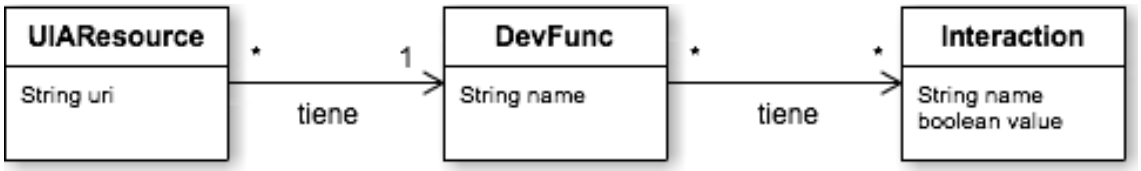


Ilustración 25 - Relación entre las clases necesarias para crear los recursos de la vivienda inteligente

El siguiente paso consiste en crear la clase en la cual se instancian los recursos disponibles en la vivienda inteligente. Para ello, se ha desarrollado la clase Smarthome.java en la cual, además, se proporciona el método estático getResources que devuelve todos los recursos disponibles y nos será útil para llevar a cabo la derivación de las acciones. En la ilustración 25 se puede observar la instanciación de varios recurso del tipo *bistate*.

```
// Common
private final static Interaction read = new Interaction("read");

// Interaction for bistates
private final static Interaction on = new Interaction("on");
private final static Interaction off = new Interaction("off");
private final static Interaction toggle = new Interaction("toggle");
private final static Interaction pulseOn = new Interaction("pulseOn");
private final static Interaction pulseOff = new Interaction("pulseOff");

private final static List<Interaction> listBistate = Arrays.asList(read, on, off, toggle, pulseOn, pulseOff);

private final static DevFunc bistate = new DevFunc("bistate", listBistate);

private final static UIAResource luz = new UIAResource(
        "http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/Luz", bistate);
private final static UIAResource alarma = new UIAResource(
        "http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/Alarma", bistate);
```

Ilustración 26 - Instanciación de recursos bistate

Una vez establecido lo anterior, es el momento de implementar la clase ActionBuilder del *plugin* en desarrollo. Dicha clase tiene que implementar la interfaz IActionBuilder la cual contiene la especificación de los métodos derivePositiveActions, deriveNegativeActions y deriveUnauthorizedActions que devolverán el conjunto de acciones derivadas en cada uno de los tres modos de ejecución, respectivamente.

Tal y como se ha indicado, en el modo de pruebas positivas se derivan todas las acciones válidas en función de si necesitan *token* o no, en el modo no autorizado se derivan aquellas acciones que necesitan un *token* sin proporcionarlo y en el modo de pruebas negativas únicamente se derivan las acciones Put pero de un modo inválido para comprobar si este tipo de acciones provocan algún tipo de riesgo para el SUT.

Con el objetivo de evitar la duplicación de código, se ha encapsulado en diversos métodos el código destinado a derivar cada tipo de acción. Así pues, para cada modo de ejecución realizamos las llamadas correspondientes tal y como se muestra en la tabla 7.

Modo de Ejecución	Métodos Invocados para Derivar las Acciones
derivePostiveActions	deriveGets(tokenName, tokenValue, getNeedsToken)
	derivePuts(tokenName, tokenValue, putNeedsToken)
	derivePosts(tokenName, tokenValue, postsNeedsToken)
	deriveDeletes(tokenName, tokenValue, deleteNeedsToken)
deriveUnauthorizedActions	if(getNeedsToken) deriveGets(tokenName, tokenValue, false)
	if(putNeedsToken) derivePuts(tokenName, tokenValue, false)
	if(postNeedsToken)

	derivePosts(tokenName, tokenValue, false)
	if(deleteNeedsToken) deriveDeletes(tokenName, tokenValue, false)
deriveNegativeTests	deriveNegativePuts(tokenName, tokenValue, putNeedsToken)

Tabla 7 - Derivación de acciones según el modo de ejecución

Disponemos por tanto de los métodos deriveGets, derivePuts, derivePosts, deriveDeletes y deriveNegativePuts, siendo este último necesario por ser realmente diferente al caso de la derivación de las acciones Put de forma válida.

- deriveGets: Derivamos una acción Get por recurso. Para ello, se añaden las cabeceras necesarias (Content-Type: application/json y Accept: application/json en el caso de la vivienda inteligente) y seguidamente se recorre la lista de los recursos disponibles creando una acción Get por cada uno de ellos. Para crear dicha acción se tiene en cuenta si se requiere o no autorización en función de lo establecido por el usuario a través de la interfaz gráfica. Cada acción se añade al conjunto de acciones derivadas que es devuelto al finalizar el método
- deriveDeletes: Cabe indicar que la vivienda virtual, por motivos internos, no acepta ni operaciones Post ni Delete. Sin embargo, nos conviene derivarlas para comprobar que el sistema informa de la situación sin que se produzca ningún error. Así pues, para el caso de las acciones Delete también crearemos una acción por cada recurso. Por ello la implementación de este método sigue los mismos pasos que el método anterior, pero comprobando si las acciones Delete necesitan parámetros de autorización y creando una acción de este tipo en función de ello
- derivePuts: En este caso añadiremos una acción PUT por cada interacción válida de cada recurso (p. ej para la luz que es del tipo *bistate* añadiremos acciones encargadas de realizar las interacciones *on*, *off*, *toggle pulseOn* y *pulseOff* respectivamente). Para ello, recorremos la lista de recursos y tras extraer las interacciones soportadas según su funcionalidad, llamaremos al método addPutPerInteraction el cual, pasándole el recurso objetivo, la interacción que se desea llevar a cabo y la información sobre si requieren o no parámetros de autorización, instanciará la acción apropiada y la añadirá a la lista de acciones derivadas.
- addPutPerInteraction: La derivación de las acciones Put para la vivienda inteligente es la más compleja. Esto se debe a la necesidad de construir un *payload* con un formato apropiado que varia en función de si se requiere un valor para la interacción o no (véase la tabla 2 para identificar qué interacciones requieren un valor)

{“action”: “on”} -> Casos en los que no se require un valor

{“action”: “set%”,”value”:”50”} -> Casos en los que se require un valor

Ilustración 27 - Ejemplos de payload en función de si se requiere valor

Para ello, tras instanciar las cabeceras apropiadas, se comprueba si la interacción sobre la cual se quiere crear la acción requiere un valor. Así pues, dependiendo de ello se construye el *payload* incluyéndolo o no tal y como se puede observar en la ilustración 28. Tras ello, se instancia la acción atendiendo a si requiere parámetros de autorización.

```
Action ac;
String payload = "";
String value = "";

//headers neeeded
Map<String, String> headers = new HashMap<String, String>();
headers.put("Content-Type", "application/json");
headers.put("Accept", "application/json");

if(interaction.isValue()){
    value = getRandomValue(interaction);
    //Eg.payload = {"action":"set","value":"21"}
    payload = "{\"action\":"+ "\"" + interaction.getName()+ "\",\"value\":\""+value+"\"}";
}else{
    //Eg payload: {"action":"on"}
    payload = "{\"action\":" + "\"" + interaction.getName()+ "\"}";
}
if(putNeedsToken){
    ac = new Put(resource.getUri(), headers, payload, timeOutTime, tokenName, tokenValue);
}else{
    ac = new Put(resource.getUri(), headers, payload, timeOutTime, null, null);
}
setActions.add(ac);
```

Ilustración 28 - Contenido del método addPutPerInteraction

- deriveNegativePuts: En este caso derivaremos un Put por interacción no válida según el tipo de funcionalidad de cada recurso. Así pues, recorremos el total de recursos disponibles y para cada tipo de funcionalidad que no coincida con la del recurso, extraeremos las interacciones válidas y creamos una acción Put para ella. No obstante, en la vivienda inteligente disponemos de casos particulares que nos fuerzan a añadir una comprobación extra. En la tabla 2 mostrada anteriormente, se puede observar que la interacción *read* es válida para todas las funcionalidades y que, además, los dispositivos *dimmer* aceptan las interacciones propias de los *bistate*. Si no controlamos estos casos, estaremos derivando acciones válidas para algunos recursos y esperando que el servidor nos responda que no lo son (p.ej. tratar de hacer un *on* a la luz gradual). Por ello, necesitamos comprobar que la interacción con la cual se va a derivar una acción no pertenezca a la lista de interacciones soportada por el recurso y, si eso se cumple, invocaremos el método addPutPerInteraction explicado previamente.

- derivePosts: Tal y como se ha comentado, la vivienda inteligente no acepta peticiones Post sobre los recursos disponibles. No obstante, se ha decidido añadir una acción por recurso para comprobar que la respuesta proporcionada por el servidor no difiere de la esperada. El procedimiento es el mismo seguido para las acciones Get y Delete: añadimos las cabeceras adecuadas, recorremos los recursos disponibles y para cada uno de ellos instanciamos una acción del tipo Post teniendo en cuenta si requiere parámetros de autorización o no.

Cabe recordar que todos estos métodos añaden las acciones instanciadas al conjunto de acciones derivadas. Dicho conjunto será devuelto por los métodos derivePositiveActions, deriveUnauthorizedActions y deriveNegativeActions invocados desde el protocolo de la aplicación en función del modo de ejecución seleccionado.

6 Evaluación de la herramienta

La herramienta desarrollada se ha evaluado mediante la ejecución de pruebas sobre el caso de estudio de la vivienda inteligente, para la cual disponemos del *plugin* implementado. Durante este trabajo se han diferenciado tres fases de pruebas que nos han permitido evaluar diversos aspectos sobre la viabilidad de la herramienta para el testeo del SUT.

La primera fase de evaluación ocurrió durante el desarrollo de TESTAR para la IoT. Con el objetivo de evitar esperar a tener la herramienta desarrollada completamente para empezar a comprobar su utilidad, se realizaron ejecuciones desde el momento en el que se disponía de un producto mínimo que fuera funcional. Así pues, en esta fase se ejecutaba la herramienta sin una metodología estratégica sobre la cantidad de pruebas a realizar, con el único objetivo de descubrir si el desarrollo de TESTAR iba por buen camino y verificar que el comportamiento de la herramienta fuera el esperado. Para la realización de dicha fase de pruebas, ciertos aspectos característicos de la vivienda virtual se tuvieron en cuenta.

Tal y como se ha indicado anteriormente, el SUT está implementado para que no acepte acciones Post ni Delete. Teniendo esto en cuenta, se ha adaptado el protocolo de la aplicación indicando que si la acción ejecutada es una de las mencionadas, se espera la respuesta *405 Method Not Allowed*. En caso contrario, TESTAR para la IoT considera que un error se ha producido. Para conseguirlo, basta con editar el método getVerdict tal y como muestra la ilustración 29.

```
protected Verdict getVerdict(String response) {
    // For the Smarthouse: It should not be possible to delete a resource.
    // We exepect an 405 Method Not Allowed.
    Verdict verdict;
    if (action instanceof Delete || action instanceof Post) {
        getPartsOfResponse(response);
        Main.logln("Server response (It should be \"405 Method Not Allowed\"):");
        Main.logln("Code: " + responseCode + " Message: " + responseMessage + " Body: " + responseBody);
        if (!responseCode.equals("405")) {
            verdict = new Verdict(1.0,
                    responseCode + " " + responseMessage + " received while expecting 405 Method Not Allowed");
        } else
            verdict = Verdict.OK;

    } else {
        verdict = super.getVerdict(response);
    }

    return verdict;

}
```

Ilustración 29 - Modificación del método getVerdict para la vivienda inteligente

Una vez desarrollada completamente la nueva versión de TESTAR para la IoT, pasamos a realizar un testeo más intensivo sobre la vivienda inteligente. De este modo, se pretendía dejar la herramienta ejecutándose automáticamente y de forma desatendida para, posteriormente investigar las salidas obtenidas en busca de resultados. Para cada modo de ejecución se establecieron diversas ejecuciones (ver tabla 8) para las cuales se realizaron los siguientes pasos:

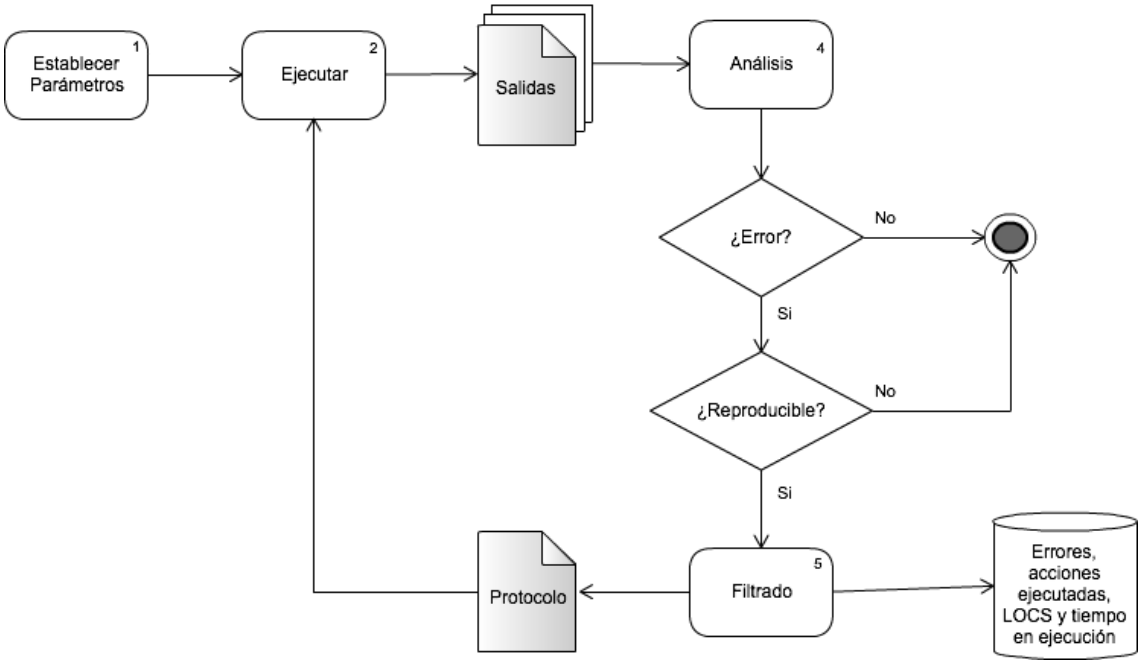


Ilustración 30 - Flujo seguido para la ejecución de pruebas en la segunda fase de pruebas sistemáticas

1. Se establecen los parámetros de configuración adecuados
2. Se ejecutan las pruebas automáticamente y de forma desatendida. De este punto obtenemos las salidas en el directorio /output
3. Analizamos las salidas obtenidas para averiguar qué ha ocurrido durante la ejecución de las pruebas y ver si se ha detectado algún error. Si no se ha producido ningún error, el total de acciones que deseamos ejecutar se habrán llevado a cabo exitosamente, por ello, terminaremos el ciclo y empezaremos las pruebas con los siguientes parámetros de configuración (tabla 8). En cambio, si se han detectado errores comprobaremos mediante la reproducción de las secuencias si estos son reproducibles
4. Si se han detectado errores reproducibles, se filtrarán mediante el protocolo para poder seguir ejecutando pruebas sin que se repita la misma detección una y otra vez

Modo	Número de Secuencias	Acciones por secuencia	Time Out	Tiempo de espera entre acciones
Pruebas Positivas	5	300	10	0
	2	500	10	0
	3	1000	10	0
Pruebas Negativas	5	300	10	0
	2	500	10	0
	3	1000	10	0
Pruebas No Autorizadas	5	300	10	0
	2	500	10	0
	3	1000	10	0

Tabla 8 - Parámetros de las ejecuciones realizadas en la segunda fase

Como se puede observar en la tabla 8, el tiempo de espera entre acciones es cero en todos los casos. Esto se debe a que la vivienda inteligente ha de ser capaz de aguantar una serie de acciones producidas sucesivamente sin pausa entre ellas. Del mismo modo, se establece en todos los casos un *Time Out* de 10 segundos por ser un valor mucho más alto de lo que debería tardar el servidor en proporcionarnos una respuesta.

La tercera fase consistió en la realización de pruebas más concretas sobre la vivienda inteligente. Para ello, contamos con la posibilidad de modificar el comportamiento por defecto de la herramienta a través del protocolo. Cabe tener en cuenta que algunos de los aspectos puestos a prueba en esta fase no están contemplados en la implementación actual de la vivienda. No obstante, aportan valor puesto que pueden ser interesantes en futuras versiones, sirven de ejemplo para llevar a cabo comprobaciones similares y permiten mostrar las posibilidades de la herramienta. Teniendo esto en cuenta, tras modificar el protocolo según el comportamiento deseado, se ha ejecutado la herramienta en el modo de pruebas positivas con el objetivo de comprobar que el funcionamiento obtenido corresponde con el esperado. Los aspectos puestos a prueba son los siguientes (en el anexo D se encuentran las modificaciones realizadas sobre el protocolo por defecto):

- Realizar peticiones repetidamente a un único recurso. Con ello comprobamos si el sistema es capaz de tolerar una serie elevada de peticiones sobre un mismo recurso, concretamente realizamos 3 secuencias de 1000 acciones cada una sobre la luz disponible en la vivienda inteligente
- Realizar peticiones a un único elemento desde diversas instancias de TESTAR para la IoT simultáneamente. Para ello, utilizamos el protocolo creado para las pruebas del punto anterior pero ejecutamos tres instancias de nuestra aplicación con 3 secuencias de 1000 acciones cada una. De este modo, el número de peticiones realizadas sobre un recurso determinado aumentó considerablemente
- En el caso de la vivienda inteligente si se permite que un usuario pueda, por ejemplo, encender y apagar repetidamente y sin intervalo de pausa una luz, dicho recurso podría llegar a fundirse. Por ello, de cara a la comercialización de la aplicación sería conveniente que existiera una limitación sobre ese tipo de comportamientos (p. ej. que no se pueda hacer la acción encender-apagar más de 20 veces seguidas). Mediante el protocolo de TESTAR para la IoT se puede comprobar si dicha limitación se aplica correctamente. No obstante, puesto que la implementación actual de la vivienda inteligente no contempla ese aspecto, la herramienta siempre va a detectar un error
- Comprobar que el sensor de la temperatura no devuelve valores más altos de 60ºC. Si esto ocurriese podría ser un indicativo de que el sensor no estuviera funcionando correctamente o de que el servidor no devolviera el valor adecuado.

Con relación al último punto, salta a la vista otra posible aplicación de la herramienta desarrollada: Monitorizar la vivienda inteligente de manera que si se observa un valor fuera de lo esperado, se avise a los responsables ya que una situación perjudicial puede estar sucediendo. Por ejemplo, si la temperatura es muy elevada puede haber un incendio o si el sensor de humedad da valores muy altos puede que se esté produciendo una situación que no sea buena para la salud de las personas.

6.1 Resultados

Con la realización de las diferentes pruebas ejecutadas y la revisión de las salidas producidas por la herramienta, se identificaron diversos errores que han sido comunicados al equipo de desarrollo de la vivienda inteligente. Cabe destacar que encontrar errores en la fase de pruebas durante el desarrollo nos permitió empezar a evaluar la herramienta desde una fase temprana de su implementación. Además, puesto que dichos errores fueron o bien solucionados o bien filtrados para evitar su repetición bloquando otros errores, conforme avanzamos en el desarrollo de nuestra herramienta resultaba menos probable localizar nuevos errores. A continuación se indican los resultados de las pruebas realizadas.

6.1.1 Pruebas durante el desarrollo

Probando el funcionamiento de las pruebas no autorizadas se descubrió que al realizar acciones sobre el recurso con funcionalidad *Dimmer* (luz gradual) éste no aceptaba sus interacciones correspondientes. Es decir, cada vez que se pretendía hacer una acción *set%*, *setOx* o *setAng* la respuesta por parte del servidor consistía en un *405 Method Not Allowed*.

Tras transmitir lo identificado al equipo encargado del desarrollo de la vivienda inteligente, nos comunicaron que se había producido un error al instanciar la luz gradual y que, por ello, no aceptaba las interacciones apropiadas. Así pues, una vez identificado el error, fue solucionado sin mayores problemas.

```
Selecting action...
Selected action 'Put Resource http://tambori.dsic.upv.
es:8100/smartcity/vsmarthome/thing/LuzGradual {"action":"set0x","value":"45"}'.
Executing (22): Put Resource http://tambori.dsic.upv.
es:8100/smartcity/vsmarthome/thing/LuzGradual {"action":"set0x","value":"45"}...
Executed [23]: ACTION_Put Resource http://tambori.dsic.upv.
es:8100/smartcity/vsmarthome/thing/LuzGradual {"action":"set%","value":"45"}
Server response:
Code: 405 Message: Method Not Allowed Body: null
Detected fault: severity: 1.0 info: Error: 405 Method Not Allowed
```

Ilustración 31 - Salida mostrando que la luz gradual no acepta las acciones de la funcionalidad dimmer

Por otra parte, durante un periodo de tiempo la vivienda inteligente no respondía adecuadamente a ninguna petición realizada. Es más, todas las respuestas obtenidas consistían en el error *404 Not Found*. Tras comentar la situación al equipo encargado de desarrollar el SUT, nos comentaron que por motivos internos habían realizado una limpieza del servicio *cloud* y que en dicha limpieza habían eliminado la dirección que

estábamos utilizando para las pruebas. Volvieron a registrar la dirección en el puerto correspondiente y tras modificar el fichero de configuración de la vivienda inteligente, pudimos volver a utilizarla del modo esperado.

```
Built action set! (141 actions)
Selecting action...
Selected action 'Get Resource http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/SensorLuminosidad'.
Executing (1): Get Resource http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/SensorLuminosidad...
Executed [2]: ACTION_Get Resource http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/SensorLuminosidad
Server response:
Code: 404 Message: Not Found Body: null
Detected fault: severity: 1.0 info: Error: 404 Not Found received
```

Ilustración 32 - Salida mostrando la respuesta obtenida cuando la dirección utilizada no estaba registrada adecuadamente

Finalmente, se identificó que en algunas ocasiones tras realizar peticiones a un recurso no disponible en ese momento, el sistema dejaba de responder por un instante y se recuperaba pasado un tiempo. En sistemas críticos dejar inactivo un servidor durante un tiempo, por breve que sea, podría traer consecuencias graves. En la ilustración 33, se puede observar un caso en el que tras realizar una petición a la luz gradual (acción 6 de la primera secuencia) cuando dicho recurso no estaba disponible, la siguiente acción ejecutada se resolvió con *time out*. Dicho comportamiento no es aislado de una única ejecución, ya que se ha identificado en varias secuencias de pruebas.

```
Selecting action...
Selected action 'Put Resource http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/LuzGradual
{"action":"read"}'.
Executing (6): Put Resource http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/LuzGradual
{"action":"read"}...
Executed [7]: ACTION_Put Resource http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/LuzGradual
{"action":"read"}
Server response:
Code: 404 Message: Not Found Body: null
Detected fault: severity: 1.0 info: Error: 404 Not Found

Sequence 1 finished.
Copying generated sequence ("sequence91") to output directory...
Copied generated sequence to output directory!
Copying erroneous sequence ("sequence91") to error_sequences directory...
Copied erroneous sequence to output directory!
Creating new sequence file...
Created new sequence file!

<==================>
20.May.2016 21:09:56 Starting...
Starting sequence 2 (output as: sequence92)

Building action set...
Built action set! (135 actions)
Selecting action...
Selected action 'Put Resource http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/SensorProximidad
{"action":"set","value":"35"}'.
Executing (1): Put Resource http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/SensorProximidad
{"action":"set","value":"35"}...
Connect to: http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/SensorProximidad timed out
Excecution of action failed!
Sequence contained problems!
```

Ilustración 33 - Salida mostrando que el servidor queda inactivo tras peticiones a recursos no disponibles

Error			Acción Causante
Cod.	**Mensaje**	**Descripción**	
405	Method Not Allowed	El recurso no acepta las interacciones que le pertenecen	Put (con interacción *set%*) sobre el recurso persiana
			Put (con interacción *setox%*) sobre el recurso persiana
			Put (con interacción *setAng*) sobre el recurso persiana
404	Not Found	Los recursos de la vivienda inteligente no están disponibles	Cualquier tipo de acción sobre cualquier recurso
_____	_____	Tiempo excedido	Acción tras haber obtenido una respuesta con 404 Not Found

Tabla 9 - Errores encontrados en la fase de pruebas el desarrollo

6.1.2 Pruebas sistemáticas

Tal y como se ha indicado, las pruebas sistemáticas se han llevado a cabo siguiendo un flujo determinado (ver ilustración 30 y tabla 8) para cada modo de ejecución.

- Modo de pruebas positivas

Tras realizar una ejecución de 5 secuencias con 300 acciones cada una de ellas e identificar los errores reproducibles obtenidos, se detectó que la persiana de la vivienda inteligente dejó de estar disponible (cada vez que se realizaba una petición sobre ella se obtenía un *404 Not Found*).

Además, se descubrió que al realizar un *read* sobre la luz gradual, la respuesta obtenida disponía del código 500 y el mensaje *Internal Server Error*. Para poder seguir ejecutando pruebas, filtramos en el protocolo todas aquellas acciones relacionadas con la persiana y la interacción *read* para la luz gradual (ver anexo D.A).

Así pues, tras reanudar las pruebas encontramos otro error reproducible relacionado con la luz gradual. Cada vez que se realizaba un *toggle* sobre dicho recurso (acción permitida según la especificación) la respuesta volvía a ser *500 Internal Server Error*. Tras filtrar dicha acción mediante el protocolo de nuestra herramienta (ver anexo D.A) se prosiguió con las pruebas y se completaron tanto las 5 secuencias de 300 acciones, como las 2 de 500 y las 3 de 1000 acciones, todas ellas sin detectar errores.

- Modo de pruebas negativas

Tras volver a establecer los parámetros de ejecución para realizar 5 secuencias de 300 acciones cada una e iniciar las pruebas, se observó que poco a poco los recursos iban dejando de estar disponibles y que cada vez se recibía la respuesta *404 Not Found* con más frecuencia.

Analizando los archivos de salida, se detectó que un recurso dado siempre dejaba de estar disponible después de que se hubiera realizado una interacción *stop* sobre él.

Cabe tener en cuenta que la interacción *stop* es realizable, según la especificación, sobre los recursos con funcionalidad *Move*, es decir, la persiana en nuestro caso. Por ello, dicha acción puede ser la responsable de que en las pruebas positivas comentadas anteriormente la persiana hubiera dejado de estar disponible.

Así mismo, en el caso de las pruebas negativas dicha acción llegó a realizarse para los demás recursos los cuales iban desapareciendo gradualmente. Tras comentarles lo identificado al equipo de desarrollo nos confirmaron que, efectivamente, la interacción *stop* da los dispositivos de baja. Además, después de dicha interacción los recursos sobre los que se ha interactuado se encuentran en una especie de limbo esperando a ser reactivados.

Sin embargo, el problema existente consiste en que la API REST disponible no soporta, actualmente, la posibilidad de interactuar con los recursos que se encuentran en dicha situación. Por tanto, simulando un usuario mediante nuestra herramienta pudimos dar de baja todos los recursos pero no disponíamos de la posibilidad de volver a reactivarlos.

No obstante, estas ejecuciones nos han servido para identificar un problema basándonos únicamente en los archivos de salida de TESTAR para la IoT. Tras llevar a cabo un reinicio de la vivienda inteligente por parte del equipo de desarrollo, los recursos vuelven a estar disponibles. Así pues, para evitar quedarnos sin recursos con los que interactuar y no poder hacer más pruebas, filtramos la operación *stop* para que no llegue a ejecutarse (ver anexo D.B) y proseguimos con la realización de las pruebas. Tras ejecutar 5 secuencias de 300 acciones, 2 de 500 y 3 de 1000 completamente, no se localizó otro comportamiento no esperado.

- Modo de pruebas no autorizadas

En este modo de ejecución se completaron las 5 secuencias de 300 acciones, las 2 de 500 y las 3 de 1000 sin encontrar ningún comportamiento que distara del esperado.

Modo de Ejecución	**Error**			**Acción Causante**
	Cod.	**Mensaje**	**Descripción**	
Pruebas Positivas	404	Not Found	Recurso no disponible	Get sobre el recurso persiana
				Put (con interacción *stop*) sobre el recurso persiana
				Put (con interacción *stepUp*) sobre el recurso persiana
	500	Internal Server Error	No se produce la interacción aceptada según la especificación	Put (con interacción *read*) sobre el recurso luz gradual
				Put (con interacción *toggle*) sobre el recurso luz gradual
Pruebas Negativas	404	Not Found	Recurso no disponible	Put (con interacción *stepDown*) sobre el recurso luz
				Put (con interacción *up*) sobre el

				recurso alarma
				Put (con interacción *set*) sobre el recurso sensor de puerta cerrada
				Put (con interacción *set%*) sobre el recurso sensor de vibración
				Put (con interacción *on*) sobre el recurso sensor de humedad

Tabla 10 - Errores encontrados en la fase de pruebas sistemáticas

6.1.3 Pruebas específicas

En primer lugar, se modificó el protocolo de TESTAR para la IoT (ver anexo D.C) para que sólo se realizarán acciones sobre un único recurso de la vivienda inteligente (en nuestro caso la luz). Tras ello, se realizaron 3 secuencias de 1000 acciones cada una con el objetivo de ver si la vivienda inteligente podía soportarlo. Todas las acciones se realizaron correctamente y la vivienda actúo según lo esperado.

Con el objetivo de aumentar la carga soportada por el servidor, se ejecutaron al mismo tiempo 3 instancias de nuestra aplicación realizando 3 secuencias de 1000 acciones cada una simultáneamente sobre la luz (se utilizó el mismo protocolo que en el caso anterior).

Comprobamos que mientras la vivienda era capaz de soportar sin ningún problema las acciones de una instancia de TESTAR para la IoT, cuando ejecutamos 3 de ellas al mismo tiempo, algunas peticiones no son tratadas con normalidad. De hecho, se produjeron diversos errores por *time out* de los cuales el SUT se recuperó pasado un tiempo, volviendo a funcionar con normalidad.

Por otra parte, tanto para el caso dedicado a controlar que un usuario no pudiera encender y apagar la luz más de 20 veces seguidas (sin pausa entre dichas acciones) como para el de controlar que la temperatura no superara los 60ºC, sabíamos de antemano que la vivienda inteligente no estaba implementada para controlar dichos aspectos.

Por ello, esperábamos que TESTAR para la IoT detectará un error cuando dichas situaciones se produjeran. No obstante, tal y como se había indicado, dichas pruebas nos sirven de ejemplo tanto para llevar a cabo comprobaciones parecidas como para observar posibles aplicaciones de nuestra herramienta. Modificando el comportamiento mediante el protocolo de TESTAR para la IoT (ver anexos D.D y D.E) hemos podido comprobar dichas pruebas y ver que efectivamente la herramienta devuelve un error cuando se producen los casos que se desean controlar.

Prueba	**Error**	**Acción ejecutada al detectarse el error**
Peticiones simultaneas a la luz desde 3 instancias de TESTAR para la IoT	Tiempo excedido	Put con interacción *toggle*
		Get
		Delete

		Post Put con interacción *off*
Encender-Apagar la luz más de 20 veces seguidas	Se permite apagar y encender la luz más de 20 veces seguidas	Acción compuesta sobre la luz: - Put con interacción *on* - Put con interacción *off*
Controlar que la temperatura no supere los 60ºC	El sensor de la temperatura recibe un valor mayor de 60	Get sobre el sensor de Temperatura

Tabla 11 - Errores encontrados en la fase de pruebas específicas

6.1.4 Información Adicional

Junto con la información referente a los errores encontrados, la cual ha sido expuesta anteriormente, disponemos de datos sobre la cantidad de acciones ejecutadas, las líneas de código (LOC) añadidas al protocolo y el tiempo que la herramienta ha estado en ejecución durante las pruebas sistemáticas y las específicas.

Cabe recordar que las pruebas llevadas a cabo durante el desarrollo se realizaron inicialmente como método para saber si lo implementado hasta el momento funcionaba. Sin embargo, puesto que se localizaron diversos errores, pareció indicado incluirla como fase de pruebas.

Fase de Pruebas Sistemáticas		
Modo de Ejecución	**Acciones Ejecutadas**	**Tiempo en Ejecución**
Pruebas Positivas	5660	1h. 46min. 48s.
Pruebas Negativas	5516	1h. 55min. 39s.
Pruebas No Autorizadas	5500	1h. 46min. 35s.
Total	16676	5h. 29min. 2s.
Fase de Pruebas Específicas		
Prueba Específica	**Acciones Ejecutadas**	**Tiempo en Ejecución**
Peticiones a un único recurso	3000	56min. 36s.
Peticiones a un único recurso desde 3 instancias de la herramienta	3376	38min. 42s.
Encender – Apagar la luz más de 20 veces seguidas	126	1min. 37s.
Control del valor de la temperatura	386	7min. 34s.
Total	6888	1h. 44min. 29s.

Tabla 12 - Acciones ejecutadas y tiempo de ejecución por fase de prueba

	Modo de ejecución/Prueba Específica	LOC añadidas al protocolo base
Pruebas Sistemáticas	Pruebas Positivas	21
	Pruebas Negativas	19
	Pruebas No Autorizadas	21
Pruebas Específicas	Peticiones a un único recurso (desde 1 o 3 instancias)	28
	Encender – Apagar la luz más de 20 veces seguidas	14
	Control del valor de la temperatura	41

Tabla 13 - LOC añadidas al protocolo base

Como se puede observar en la tabla 12, el modo en el que más acciones se han realizado dentro de la fase de pruebas sistemáticas es el de pruebas positivas. Esto se debe a que, siguiendo el flujo de la ilustración 30, una ejecución dada se repitió si se produjo en ella un error reproducible. Puesto que fue el primer modo ejecutado, en dichas pruebas se encontraron más errores y por ello se repitieron más ejecuciones que en el resto de modos.

Por otra parte, en la tabla 13 se puede observar que el número de líneas necesarias para adaptar el protocolo a nuestras necesidades es relativamente bajo. De hecho, una vez familiarizados con el protocolo de TESTAR para la IoT, conseguir que la herramienta se comporte de un modo determinado no resulta complicado.

7 Conclusiones

Al inicio de este proyecto se planteaban una serie de objetivos que, tras su realización, se han conseguido alcanzar. Por una parte, se ha comprobado que la filosofía de TESTAR (pese a ser una herramienta enfocada al testeo a través de interfaces gráficas de usuario) puede ser aplicada para el entorno de la IoT, permitiendo la realización de pruebas desatendías sin necesidad de conocer el código fuente del SUT.

Para ello, ha sido necesario investigar en qué parte de la IoT encajaba mejor la herramienta teniendo en cuenta sus características. Tras decidir centrarse en el testeo de los servicios que nos permiten acceder a los recursos de los sistemas, el proyecto ha girado en torno a la realización de pruebas en una vivienda inteligente. Esto nos ha permitido evaluar la nueva herramienta desarrollada en un caso práctico y tener un primer contacto con el mundo de la IoT.

Ya en una fase temprana del desarrollo de TESTAR para la IoT se detectaron errores que pudieron ser corregidos rápidamente. Con ello, se potenció el correcto funcionamiento de la vivienda inteligente, evitando errores que tendrían una repercusión negativa si no se hubieran detectado antes de salir al mercado.

Así mismo, se han implementado diversos modos de ejecución permitiendo realizar diferentes tipos de pruebas. Por una parte, TESTAR para la IoT permite realizar pruebas de lo inesperado pudiendo, de este modo, ver como reacciona el servidor frente a peticiones anómalas. Además, puesto que se pueden ejecutar diversas instancias de la aplicación simultáneamente, se puede utilizar para estresar un sistema y ver cómo reacciona a dichas situaciones.

Por otra parte, disponer de la opción de reproducir secuencias ejecutadas permite llevar a cabo pruebas de regresión. De este modo, los encargados de realizar el testeo pueden guardar secuencias que se han completado correctamente y ejecutarlas tras realizar cambios durante el desarrollo, comprobando si el comportamiento sigue siendo el esperado.

Relacionado con esto, cabe destacar que la herramienta desarrollada durante este proyecto resulta especialmente adecuada para realizar testeo en fases tempranas del ciclo de desarrollo del software. De este modo, una vez configurada de manera adecuada puede dejarse en ejecución automáticamente e ir comprobando si los cambios realizados en el SUT provocan algún comportamiento no deseado. Así pues, una vez realizado un *plugin* para un sistema concreto, dejar la aplicación desarrollada en ejecución e incluso modificar su comportamiento para adaptarlo a nuestras necesidades no requiere prácticamente recursos.

De hecho, tras realizar las pruebas específicas se ha llegado a la conclusión de que mediante la modificación de su comportamiento mediante el protocolo, TESTAR para la IoT podría incluso utilizarse para monitorizar los recursos de un sistema determinado. Por ejemplo, en el caso de la vivienda inteligente, comprobar que la temperatura no supere los 60ºC o que la humedad no rebase un umbral que resulte peligroso para la salud de los habitantes.

Respecto a los objetivos personales, este proyecto me ha permitido adquirir nuevos conocimientos sobre la IoT. Teniendo en cuenta que se trata de un tema que dispone de la capacidad necesaria para representar la próxima revolución tecnológica, el modo en el cual se desarrolle mi carrera profesional podría verse afectado por las ventajas y desventajas que la IoT puede aportar. De este modo, los conocimientos adquiridos durante la realización de este proyecto pueden resultar realmente útiles en un ámbito profesional, más aun teniendo en cuenta que cada vez son más las empresas que participan en dicha materia.

Por otra parte, tras haber trabajado durante más de un año en el desarrollo y mantenimiento de la herramienta TESTAR para el testeo a través de interfaces de usuario, este proyecto me ha permitido llevar a cabo una abstracción de sus características y aplicar los conocimientos de los que disponía para tratar de llevarla a un campo totalmente diferente. No obstante, también me ha permitido llegar a comprender mucho mejor tanto su funcionamiento como su código fuente.

Teniendo en cuenta que en la realización de muchos trabajos profesionales no se desarrolla software desde cero si no que más bien se adquieren sistemas legados que hay que mantener o se llega a proyectos ya iniciados, saber leer y entender código escrito por otros desarrolladores es de vital importancia para el desarrollo profesional. Así mismo, el ejercicio de comprensión del código fuente de TESTAR me ha ayudado a mejorar dichas habilidades permitiéndome experimentar en primera mano el hecho de enfrentarme a un software implementado por otro desarrollador.

Finalmente, este proyecto también me ha permitido poner en práctica los conocimientos adquiridos tanto durante el grado en ingeniería informática como en el máster universitario de ingeniera informática.

7.1 Trabajo Futuro

Tal y como se ha comentado, este proyecto ha representado un primer contacto con TESTAR aplicado a la IoT. Puesto que actualmente no es un campo estandarizado y son muchas las tecnologías utilizadas en su desarrollo, queda como trabajo pendiente expandir la herramienta consiguiendo que soporte otros protocolos de comunicación bidireccionales más allá de las API REST.

Así mismo, disponer de una vivienda inteligente nos ha permitido evaluar la herramienta con un dispositivo IoT. Sin embargo, será necesario probar la aplicación con sistemas de características diferentes. De este modo, podremos adquirir práctica en la creación de *plugins* y considerar si hay cambios requeridos para mejorar la aplicabilidad de la herramienta.

Por otra parte, tras desarrollar una primera versión de la aplicación, queda como trabajo futuro aumentar sus funcionalidades. Entre las mejoras pensadas se encuentran:

- Utilizar técnicas metaheurísticas [41] para mejorar el proceso de selección de acciones a ejecutar. De este modo, partiendo de una solución que normalmente no es óptima se obtienen soluciones parecidas de entre las cuales se elige la que satisface un criterio dado. A partir de dicha solución se vuelve a iniciar el proceso, el cual se detiene cuando se cumple una condición establecida previamente. Algunas de dichas técnicas como, por ejemplo, los algoritmos genéticos han empezado a utilizarse recientemente en la versión de TESTAR aplicada a las interfaces gráficas de usuario. Por tanto, queda como futuro trabajo estudiar su aplicabilidad en la versión para la IoT, buscando la selección de acciones que maximicen los errores encontrados o la cobertura de acciones ejecutadas.
- Relacionado con la cobertura, se añadirá información de ella en los archivos de salida de TESTAR para la IoT. En el caso práctico de este proyecto, puesto que el número de acciones disponibles no era excesivamente elevado no se consideró algo crítico y por motivos de limitación temporal se propuso como mejora futura. Así pues, se pretende disponer de métricas para saber la cobertura conseguida y proporcionar dicha información de manera fiable y comprensible.
- Añadir la posibilidad de que TESTAR para la IoT envíe correos electrónicos a una dirección proporcionada con un resumen de las ejecuciones producidas. De este modo, un usuario puede dejar la aplicación ejecutándose desatendidamente durante un periodo de tiempo. Cuando la aplicación haya terminado de ejecutar pruebas, podrá recibir información de interés en su cliente de correo. Puesto que muchos usuarios disponen de notificaciones para los correos electrónicos en el ordenador o en sus dispositivos móviles, pueden ser avisados de que la herramienta ya ha finalizado las pruebas y acceder en ese mismo momento al resumen de las ejecuciones (errores encontrados, secuencias ejecutadas, …). Tras investigar brevemente como se puede implementar dicha funcionalidad se ha encontrado la librería JavaMail[42] que permite enviar correos electrónicos mediante código java.

Estas mejoras funcionales forman parte del mantenimiento perfectivo que se realizará para la aplicación desarrollada. No obstante, puesto que en este proyecto se ha realizado una primera versión de la aplicación orientada a la IoT, también será necesario llevar a cabo un mantenimiento correctivo que nos permita diagnosticar errores que puedan surgir y, tras ello, solucionarlos.

Por otra parte, se dará difusión a la solución desarrollada mediante la redacción de un *paper* que se presentará en una conferencia o *workshop*. Además, la aplicación se subirá a github siendo de código abierto (bajo la licencia BSD-3). De este modo, los usuarios podrán utilizarla, crear sus propios *plugins* y adaptarla a sus necesidades tal y como lo deseen.

Bibliografía

[1]S. Haller, S. Karnouskos and C. Schroth, "The Internet of Things in an Enterprise Context", *Lecture Notes in Computer Science*, pp. 14-28.

[2]J. Hagar, (2014) "Testing Strategy for the IoT", *LogiGEAR Magazine*. [Online]. Available: http://www.logigear.com/magazine/issue/past-articles/testing-strategy-for-the-iot/.

[3]T.E.J. Vos, P.M. Kruse, N. Condori-Fernández, S. Bauersfeld and J. Wegener, "TESTAR: Tool Support for Test Automation at the User Interface Level", *International Journal of Information System Modeling and Design*, vol. 6, no. 3, pp. 46-83, 2015.

[4]S. Bauersfeld, A. de Rojas and T.E.J. Vos, "Evaluating rogue user testing in industry: An experience report", *2014 IEEE Eighth International Conference on Research Challenges in Information Science (RCIS)*, 2014.

[5]S. Bauersfeld, T. Vos, N. Condori-Fernandez, A. Bagnato and E. Brosse, "Evaluating the TESTAR tool in an industrial case study", *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '14*, 2014.

[6]U. Rueda, T.E.J. Vos, F. Almenar, M.O. Martínez and A.I. Esparcia-Alcazar, "TESTAR: from academic protype towards an industry-ready tool for automated testing at the user interface level", *Actas de las XX Jornadas de Ingeniería del Software y Bases de Datos* (JISBD 2015).

[7] S.Haller, "The Things in the Internet of Things", *Poster at the (IoT 2010), Tokyo, Japan, November 2010*. [Online]. Available: http://www.iot-a.eu/public/news/resources/TheThingsintheInternetofThings_SH.pdf

[8]L. Richardson and S. Ruby, *RESTful web services*. Farnham: O'Reilly, 2007.

[9]J. Snell, D. Tidwell and P. Kulchenko, *Programming Web services with SOAP*. Sebastopol, CA: O'Reilly & Associates, 2001.

[10]L. Atzori, A. Iera and G. Morabito, "The Internet of Things: A survey", *Computer Networks*, vol. 54, no. 15, pp. 2787-2805, 2010.

[11]R. van Kranenburg and A. Bassi, "IoT Challenges", *Communications in Mobile Computing*, vol. 1, no. 1, p. 9, 2012.

[12]R. Porkodi and V. Bhuvaneswari, "The Internet of Things (IoT) Applications and Communication Enabling Technology Standards: An Overview", *2014 International Conference on Intelligent Computing Applications*, 2014.

[13]"OWASP", *Owasp.org*, 2016. [Online]. Available: https://www.owasp.org/. [Accessed: 5- Mar- 2016].

[14]"IoT Testing Guides - OWASP", *Owasp.org*, 2016. [Online]. Available: https://www.owasp.org/index.php/IoT_Testing_Guides. [Accessed: 5- Mar- 2016].

[15]"Application programming interface", *Wikipedia*, 2016. [Online]. Available: https://en.wikipedia.org/wiki/Application_programming_interface. [Accessed: 7- Mar- 2016].

[16]H. Reza and D. Gilst, "A Framework for Testing RESTful Web Services", *2010 Seventh International Conference on Information Technology*, pp. 216-221, 2010.

[17]S. Chakrabarti and R. Rodriquez, "Connectedness testing of RESTful web-services", *Proceedings of the 3rd India software engineering conference on India software engineering conference - ISEC '10*, 2010.

[18]S. Chakrabarti and P. Kumar, "Test-the-REST: An Approach to Testing RESTful Web-Services", *2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*, 2009.

[19]R. Siblini and N. Mansour, "Testing web services", *The 3rd ACS/IEEE International Conference onComputer Systems and Applications, 2005.*.

[20]"Postman", *Postman*, 2016. [Online]. Available: https://www.getpostman.com/. [Accessed: 15- Mar- 2016].

[21]"SoapUI | Functional Testing for SOAP and REST APIs", *Soapui.org*, 2016. [Online]. Available: https://www.soapui.org/. [Accessed: 15- Mar- 2016].

[22]C. Kankanamge, *Web Services Testing with soapUI*. Birmingham: Packt Pub., 2012.

[23]R. Anderson, *SoapUI cookbook*. Packt Pub., 2015.

[24]"RESTful API and Web Service Test Framework | TestServer", *Smartbear.com*, 2016. [Online]. Available: https://smartbear.com/product/ready-api/testserver/overview/. [Accessed: 16- Mar- 2016].

[25]R. Inc., "Runscope", *Runscope.com*, 2016. [Online]. Available: https://www.runscope.com/. [Accessed: 16- Mar- 2016].

[26]T.E.J. Vos, U. Rueda, W. Prasetya, "Automated Testing at the User Interface level".

[27]A. Memon, "A comprehensive framework for testing graphical user interfaces", Doctoral Dissertation, University of Pittsburgh, 2001.

[28]"TESTAR - automated system testing of desktop, web and mobile applications at the GUI level.", *Testar.org*, 2016. [Online]. Available: http://www.testar.org/. [Accessed: 27- Feb- 2016].

[29]S. Bauersfeld and T.E.J. Vos, " User interface level testing with TESTAR; what about more sophisticated action specification and selection?", *2014* In Proceedings of the Seventh Seminar on Advanced Techniques and Tools for Software Evolution, SATToSE 2014, L'Aquila, Italy, 9-11 July 2014, pp. 60-78.

[30]J.J. Fons. "Acceso a plataforma IoT". Inteligencia Ambiental (INA). Universitat Politècnica de València, 2016

[31]"RESTful Web services: The basics", *Ibm.com*, 2015. [Online]. Available: https://www.ibm.com/developerworks/library/ws-restful/. [Accessed: 20- Mar- 2016].

[32]D. Guinard, I. Ion and S. Mayer, "In Search of an Internet of Things Service Architecture: REST or WS-*? A Developers' Perspective", *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pp. 326-337, 2012.

[33]M. Laine, "RESTful Web Services for the Internet of Things ".

[34]"Códigos de estado HTTP", *Es.wikipedia.org*, 2016. [Online]. Available: https://es.wikipedia.org/wiki/Anexo:C%C3%B3digos_de_estado_HTTP. [Accessed: 08- May- 2016].

[35]"Regular Expression Language - Quick Reference", *Msdn.microsoft.com*, 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/az24scfc%28v=vs.110%29.aspx. [Accessed: 17- Apr- 2016].

[36]"Manual básico de ANT - Wikilibros", *Es.wikibooks.org*, 2016. [Online]. Available: https://es.wikibooks.org/wiki/Manual_b%C3%A1sico_de_ANT. [Accessed: 20- Jun- 2016].

[37]"aymanhs/jsyntaxpane", *GitHub*, 2016. [Online]. Available: https://github.com/aymanhs/jsyntaxpane. [Accessed: 21- Jun- 2016].

[38]"javax.tools (Java Platform SE 7)", *Docs.oracle.com*, 2016. [Online]. Available: https://docs.oracle.com/javase/7/docs/api/javax/tools/package-summary.html. [Accessed: 21- Jun- 2016].

[39]"draft-ietf-httpbis-p2-semantics-14 - Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", *Tools.ietf.org*, 2016. [Online]. Available: http://tools.ietf.org/html/draft-ietf-httpbis-p2-semantics-14#section-7.7. [Accessed: 08- Jul- 2016].

[40]"HashCode() (Java)", *Es.wikipedia.org*, 2016. [Online]. Available: https://es.wikipedia.org/wiki/HashCode()_(Java). [Accessed: 08- Jul- 2016].

[41]A. García, "Técnicas metaheurísticas", [Online]. Available: http://www.iol.etsii.upm.es/arch/metaheuristicas.pdf. [Accessed: 5- Aug- 2016].

[42]"JavaMail API", *Oracle.com*, 2016. [Online]. Available: http://www.oracle.com/technetwork/java/javamail/index.html. [Accessed: 05- Aug- 2016].

Apéndice A: Requerimientos e instalación

TESTAR para la IoT se ha desarrollado siendo independiente de la plataforma, es decir, puede ejecutarse en cualquier sistema operativo. No obstante, para asegurarse de que la aplicación se ejecuta correctamente es necesario instalar el *Java Development Kit* (JDK) en su versión 1.7.

La herramienta, junto con el *plugin* para la vivienda inteligente, se encuentra en un archivo llamado TESTAR_IoT_SmartHome_v1_0.zip el cual contiene todos los archivos necesarios para ser ejecutada. Para ello, simplemente es necesario descomprimir el .zip en un directorio sobre el cual se disponga de permisos de escritura y ejecutar el archivo iot_testar.jar ubicado en el directorio /iot_testar/target.

Si en algún momento se desea volver a compilar el proyecto obteniendo el archivo .jar de nuevo, se puede utilizar la herramienta ANT de Apache. Para utilizarla se tienen que seguir los siguientes pasos:

- Descargarla desde el siguiente enlace http://ant.apache.org/bindownload.cgi
- Descomprimir el fichero descargado en el directorio deseado
- Añadir la variable de entorno ANT_HOME que apunte al directorio en el cual se ha descomprimido el archivo descargado (p. ej. C:\Program Files\apache-ant-1.9.7)
- Agregar al *path* la ruta (hasta \bin) en la cual se ha ubicado ANT (p. ej. C:\Program Files\apache-ant-1.9.7\bin)
- Verificar que la instalación se ha realizado correctamente. Para ello, abrimos la consola de comandos/terminal y ejecutamos el comando “ant”. Si la instalación es correcta se mostrará un mensaje indicando que el build ha fallado debido a que no encuentra un fichero build.xml

Apéndice B: Preparación del entorno para desarrollar

A continuación, se detallan los pasos a realizar para configurar adecuadamente el entorno de trabajo que permite utilizar TESTAR para la IoT como desarrollador. De este modo, los desarrolladores estarán en condiciones de implementar sus propios *plugins* específicos para los sistemas que deseen testear. Cabe indicar que en el ejemplo proporcionado se utiliza Eclipse como entorno de desarrollo integrado (IDE). Sin embargo, los usuarios pueden elegir el que deseen utilizar.

1. Descomprimir el fichero TESTAR_IoT_Smarthome_v1_0.zip en el directorio deseado
2. Abrir Eclipse
3. Seleccionar *File -> New -> Other… -> Java Project from Existing Ant Buildfile*
4. Seleccionar el fichero *iot_core_\build.xml* y marcar la opción *Link to the buildfile in the file system* seguido del botón *Finish*

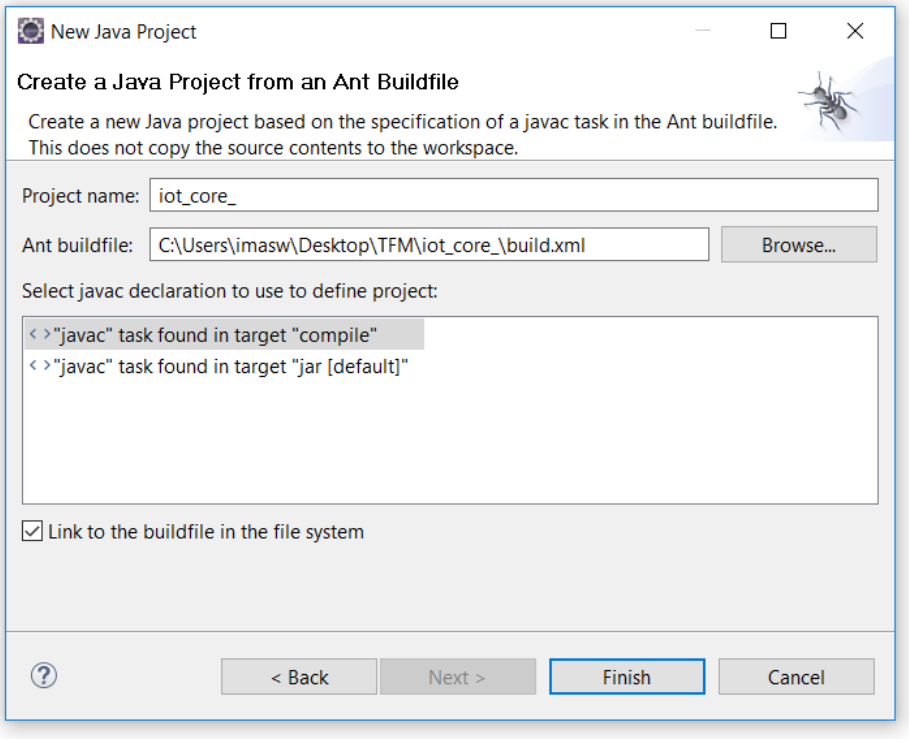


Ilustración 34 - de Create a Java Project from an ANT Buildfile en Eclipse

5. Repetir el paso 4 con los *buildfiles* localizados en *iot_smarthome_\build.xml* e *iot_testar_\build.xml*
6. En el explorador de paquetes de Eclipse, seleccionar el fichero *build.xml* del proyecto *iot_testar_* y pulsarlo con el botón derecho

7. Seleccionar *Run As* -> *Ant Build*. Si todo va bien se mostrará, en la consola de Eclipse, el mensaje *BUILD SUCCESSFUL* acompañado del tiempo que ha tardado en realizarse el *build*

```
Buildfile: C:\Users\imasw\Desktop\TFM\iot testar \build.xml
dependencies:
compile:
compile:
jar:
      [jar] Building jar: C:\Users\imasw\Desktop\TFM\iot_core_\target\intermediate.jar
      [zip] Building zip: C:\Users\imasw\Desktop\TFM\iot_core_\target\iot_core.jar
   [delete] Deleting: C:\Users\imasw\Desktop\TFM\iot_core_\target\intermediate.jar
dependencies:
   [depend] Deleted 3 out of date files in 0 seconds
compile:
compile:
jar:
      [jar] Building jar: C:\Users\imasw\Desktop\TFM\iot_core_\target\intermediate.jar
      [zip] Building zip: C:\Users\imasw\Desktop\TFM\iot_core_\target\iot_core.jar
   [delete] Deleting: C:\Users\imasw\Desktop\TFM\iot_core_\target\intermediate.jar
    [javac] Compiling 3 source files to C:\Users\imasw\Desktop\TFM\iot_smarthome_\bin
jar:
      [jar] Building jar: C:\Users\imasw\Desktop\TFM\iot_smarthome_\target\iot_smarthome.jar
    [mkdir] Created dir: C:\Users\imasw\Desktop\TFM\iot_testar_\bin\resources\icons
     [copy] Copying 10 files to C:\Users\imasw\Desktop\TFM\iot_testar_\bin\resources\icons
jar:
      [jar] Building jar: C:\Users\imasw\Desktop\TFM\iot_testar_\target\intermediate.jar
      [zip] Building zip: C:\Users\imasw\Desktop\TFM\iot_testar_\target\iot_testar.jar
   [delete] Deleting: C:\Users\imasw\Desktop\TFM\iot_testar_\target\intermediate.jar
    [javac] Compiling 1 source file to C:\Users\imasw\Desktop\TFM\iot_testar_\target
BUILD SUCCESSFUL
Total time: 2 seconds
```

Ilustración 35 - de Eclipse tras realizar un build correctamente

8. Seleccionar el proyecto *iot_testar_* en el explorador de archivos. Mediante el botón derecho accedemos a *Run As* -> *Run Configurations…* Una vez ahí, seleccionar la pestaña *Arguments* y en el apartado destinado a indicar el directorio de trabajo marcar *Other* y buscar en el sistema de archivos el directorio *iot_testar_\target*. Acto seguido pulsar *Apply* y *Run*

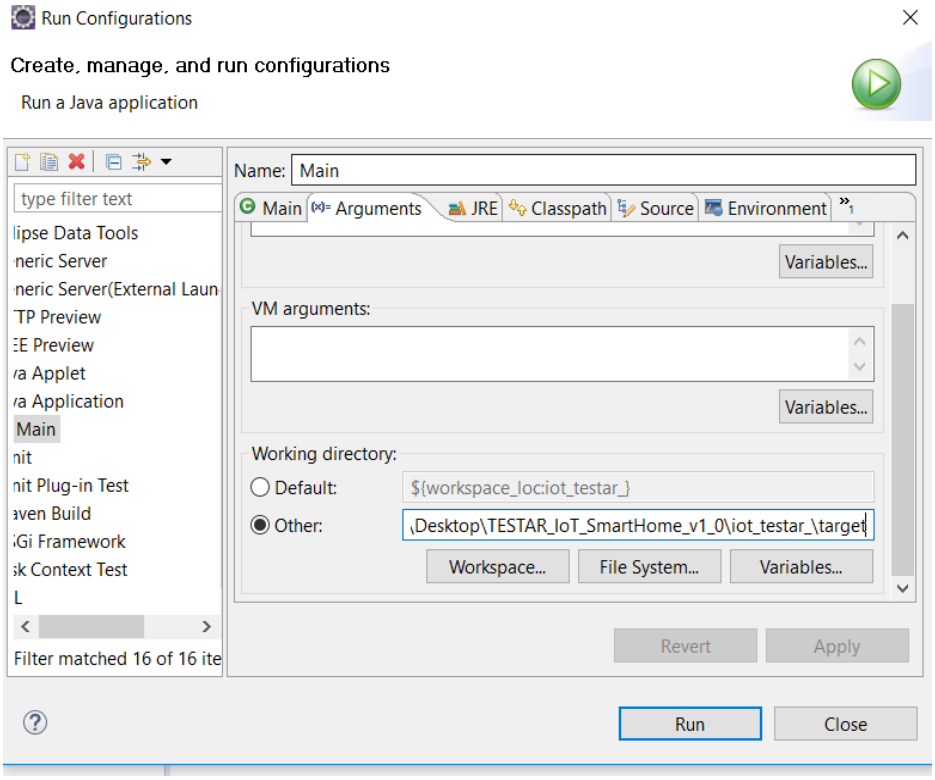


Ilustración 36 - Pantalla de Run Configurations en Eclipse

9. Tras ese punto, la aplicación ya debería poder ejecutarse. Sin embargo, en algunos casos cuando se desea compilar el protocolo desde TESTAR para la IoT aparece un mensaje indicando *JDK required (running inside of JRE)* y no se produce la compilación deseada.

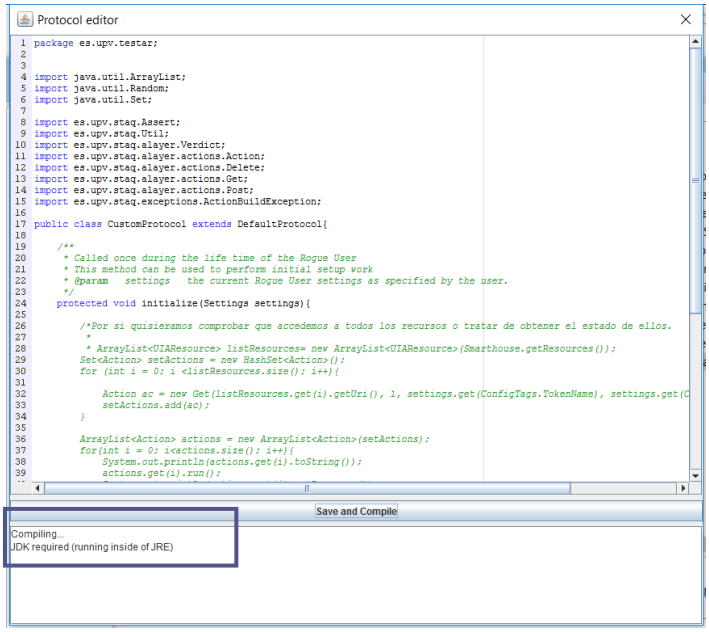


Ilustración 37 - Error en la compilación del protocolo

En dichos casos tenemos que indicarle a Eclipse que utilice el JDK instalado previamente en vez del *Java Runtime Environment* (JRE).
Para ello, seleccionamos el menú *Window -> Preferences -> Java -> Installed JRE's*.

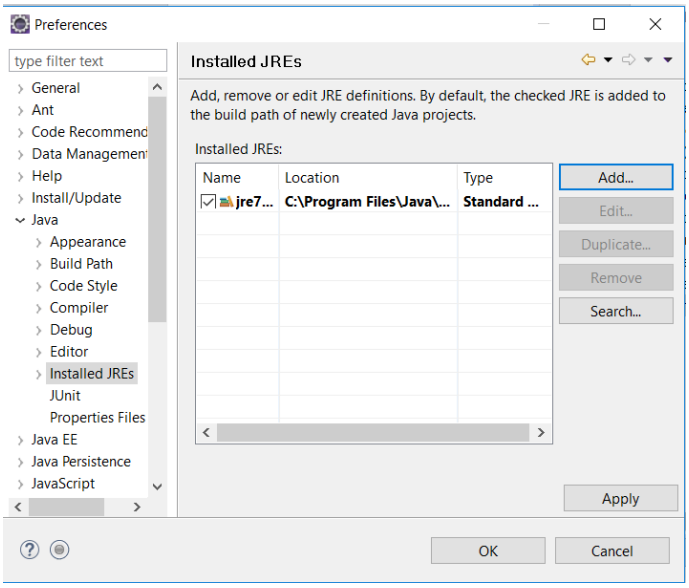


Ilustración 38 - Pantalla de Installed JREs en Eclipse

Tras ello, pulsamos *Add...* -> *Standard VM* -> *Next* y en el apartado *JRE home* buscamos el directorio en el cual tenemos instalado el JDK (p. ej. C:\Program Files\Java\jdk1.7.0_79)

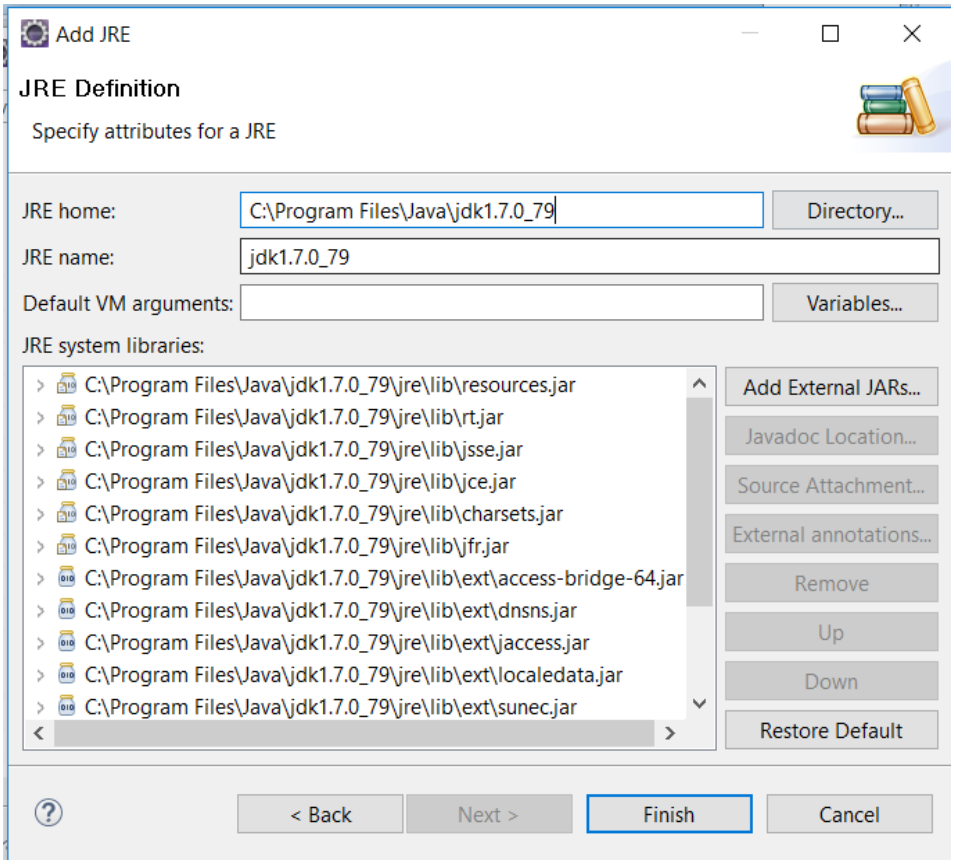


Ilustración 39 - de JRE Definition en Eclipse

Tras pulsar el botón *Finish*, se nos llevará de vuelta a la pantalla de *Installed JREs* en la cual desmarcaremos el JRE que aparecía anteriormente y marcaremos la nueva entrada para el JDK. Finalmente, pulsamos los botones *Apply* y *Ok*.

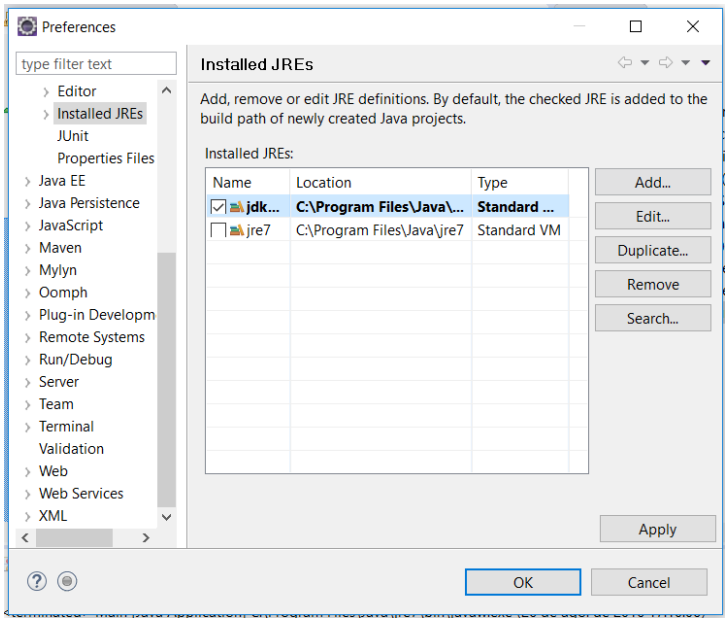


Ilustración 40 - Pantalla de Installed JREs en Eclipse con la nueva entrada para el JDK

Si todo ha salido según lo esperado, ya podremos ejecutar la herramienta correctamente. Para ello, seleccionamos el proyecto *iot_testar_* y tras pulsar el botón derecho elegimos la opción *Run As -> Java Application*.

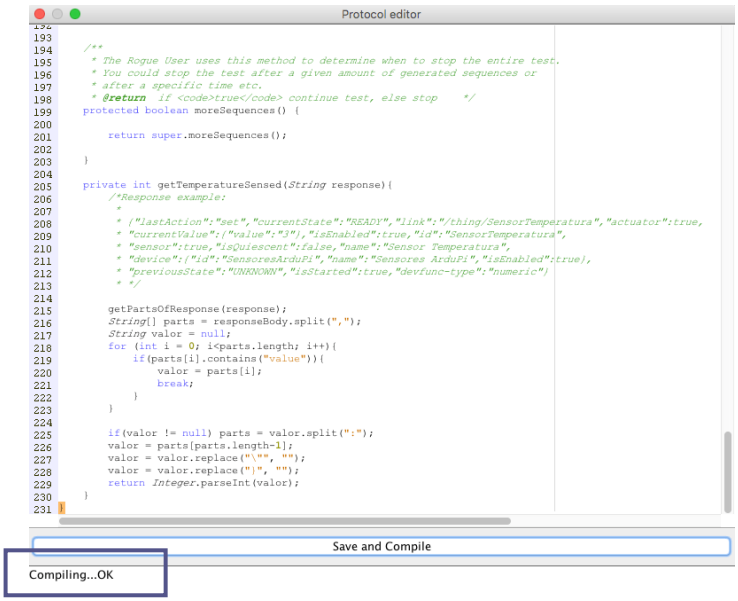


Ilustración 41 - Compilación correcta del protocolo

Una vez realizados estos pasos, disponemos del código fuente de la herramienta importado a Eclipse. De este modo, si se desea crear un nuevo *plugin* para un sistema determinado, se puede sustituir el proporcionado para la vivienda inteligente y desarrollar uno nuevo que se adapte a las necesidades de las cuales se disponga.

Apéndice C: Ejemplo de fichero .log

A continuación se adjunta un ejemplo real (aunque reducido por cuestiones de espacio) de un fichero .log en el que se proporciona información detalla sobre lo sucedido en una ejecución de TESTAR para la IoT.

Dicha ejecución consistía en una secuencia de 300 acciones pero fue detenida debido a la localización de un error al hacer un *toggle* sobre la luz gradual. Se puede observar que en la acción número 9 se detecta el error *500 Internal Server Error* y la aplicación procede a guardar la secuencia tanto en el directorio de salida como en el de las secuencias erróneas.

```
2016_06_27__13_51_52 TESTAR 0.0.1 is running

-- settings start ... --

SuspiciousCodes<String> : 4[0-9][0-9]|5[0-9][0-9]
Replay<Boolean> : false
OutputDir<String> : ./output
TempDir<String> : ./output/temp
ShowVisualSettingsDialogOnStartup<String> : true
GetNeedsAuthentication<Boolean> : false
TokenName<String> : Auth-Token
NegativeTestsText<String> : Bad [Rr]equest|Method Not Allowed
ShowSettingsAfterTest<Boolean> : true
TimeToWaitAfterAction<Double> : 1.0
PutNeedsAuthentication<Boolean> : true
TokenValue<String> : ZmtmbWRpMm1ibzNzMnB1NzgzbHJhMmM0OHU=
DeleteNeedsAuthentication<Boolean> : false
Mode<String> : Positive
Sequences<Integer> : 1
SequenceLength<Integer> : 300
TimeOutTime<Integer> : 10
PathToReplaySequence<String>:
/Users/mimarmu1/TESTAR_IOT/workspace/iot_testar_/target/output/
sequences
PostNeedsAuthentication<Boolean> : false
UnauthorizedText<String> : Unauthorized

-- ... settings end --

Trying to load TESTAR protocol
Starting TESTAR protocol ...
Creating new sequence file...
Created new sequence file!

<===================>
```

```
27.June.2016 13:51:53 Starting...
Starting sequence 1 (output as: sequence1)

Building action set...
Built action set! (126 actions)
Selecting action...
Selected action 'Put Resource
http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/SensorPuert
aCerrada {"action":"read"}'.
Executing (1): Put Resource
http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/SensorPuert
aCerrada {"action":"read"}...
Executed [2]: ACTION_Put Resource
http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/SensorPuert
aCerrada {"action":"read"}
Server response:
Code: 200 Message: OK Body:
{"lastAction":"on","currentState":"ON","link":"/thing/SensorPuertaCerr
ada","actuator":true,"isEnabled":true,"id":"SensorPuertaCerrada","sens
or":true,"isQuiescent":false,"name":"Sensor Puerta
Cerrada","device":{"id":"Puerta","name":"Puerta","isEnabled":true},"pr
eviousState":"OFF","isStarted":true,"devfunc-type":"bistate"}


Building action set...
Built action set! (126 actions)
Selecting action...
Selected action 'Post Resource
http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/SensorLumin
osidad '.
Executing (2): Post Resource
http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/SensorLumin
osidad ...
Executed [3]: ACTION_Post Resource
http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/SensorLumin
osidad
Server response (It should be "405 Method Not Allowed"):
Code: 405 Message: Method Not Allowed Body: null
      .

      .

      .

Building action set...
Built action set! (126 actions)
Selecting action...
Selected action 'Put Resource
http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/LuzGradual
{"action":"toggle"}'.
```

```
Executing (9): Put Resource
http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/LuzGradual
{"action":"toggle"}...
Executed [10]: ACTION_Put Resource
http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/LuzGradual
{"action":"toggle"}
Server response:
Code: 500 Message: Internal Server Error Body: null
Detected fault: severity: 1.0 info: Error: 500 Internal Server Error
received

Sequence 1 finished.
Copying generated sequence ("sequence1") to output directory...
Copied generated sequence to output directory!
Copying erroneous sequence ("sequence1") to error_sequences
directory...
Copied erroneous sequence to output directory!
TESTAR stopped execution at 27.June.2016 13:52:30
```

Apéndice D: Modificaciones en el protocolo

Durante la realización del proyecto, se han llevado a cabo cambios en el protocolo por defecto de la herramienta. Dichos cambios se han realizado con la intención de adaptarlo a las necesidades específicas del SUT e incluso para cambiar el comportamiento por defecto de TESTAR para la IoT.

A continuación se explican los cambios realizados para llevar a cabo las pruebas ejecutadas durante la evaluación de la herramienta con la vivienda inteligente como caso de estudio.

Apéndice D.A. Filtrar la Persiana y las operaciones *Read* y *Toggle* para la luz gradual

Para llevar a cabo dicho filtrado necesitamos recurrir al método selectAction. Así pues, comprobamos si la acción seleccionada cumple unas ciertas características que nos indiquen que se trata de una acción de las que queremos filtrar.

Para ello, indicaremos que si la acción contiene en su URI la palabra “Persiana” (se trataría de una acción sobre dicho recurso) o bien si contiene la palabra “LuzGradual” y al mismo tiempo se cumple que su *payload* sea distinto de nulo e incluya la cadena “read” o “toggle”, se deberá ignorar y llevar a cabo otra selección.

```
protected Action selectAction(Set<Action> actions){
    Assert.checkValidParameters(actions != null && !actions.isEmpty());
    Random rnd = new Random();
    Action actionSelected = new ArrayList<Action>(actions).get(rnd.nextInt(actions.size()));
    String payload = actionSelected.getPayload();
    if (actionSelected.getUri().contains("Persiana") ||
            (actionSelected.getUri().contains("LuzGradual") &&
                    (payload != null) &&
                    (payload.contains("read")|| payload.contains("toggle"))
            )) {
        actionSelected = selectAction(actions);
    }
    actions.clear();
    return actionSelected;
}
```

Ilustración 42 - Filtrado de la Persiana y las operaciones *read* y *toggle* para la Luz Gradual

Apéndice D.B. Filtrar la operación *Stop*

Para filtrar todas las acciones que contengan la operación *Stop* también recurrimos al método selectAction. En este caso, simplemente comprobamos si el *payload* de la acción seleccionada es distinto de nulo y contiene la palabra "stop". Si dichas condiciones se cumplen se puede afirmar que se trata de una acción que realizará un *stop* sobre un recurso determinado, por tanto, procedemos a ignorarla y seleccionar una acción alternativa.

```
protected Action selectAction(Set<Action> actions){
    Assert.checkValidParameters(actions != null && !actions.isEmpty());
    Random rnd = new Random();
    Action actionSelected = new ArrayList<Action>(actions).get(rnd.nextInt(actions.size()));
    String payload = actionSelected.getPayload();
    if ( payload != null && payload.contains("stop")) {
        actionSelected = selectAction(actions);
    }
    actions.clear();
    return actionSelected;
}
```

Ilustración 43 - Filtrado de la operación Stop

Apéndice D.C. Realizar peticiones únicamente sobre la luz

En esta ocasión modificaremos el método deriveActions. Así pues, en primer lugar se derivan las acciones del modo usual, obteniéndose un *Set* de acciones. Ahora bien, en lugar de devolver directamente dicho conjunto, instanciamos una lista con él.

Esa lista es recorrida con la intención de quedarnos únicamente con aquellas que tengan como recurso la luz (es decir, aquellas cuya URI se corresponda con la de la luz).

Cabe indicar que dichas acciones se vuelven a añadir a un Set<Action> ya que es el tipo devuelto por el método deriveActions. De este modo, forzaremos la herramienta a que sólo devuelva acciones que interactúen con el recurso deseado. Por tanto, tras ejecutar el método selectAction del modo usual siempre se seleccionará una acción que interactúe con dicho recurso.

```
protected Set<Action> deriveActions() throws ActionBuildException {
    Set<Action> actions = super.deriveActions();
    ArrayList<Action> actionsList = new ArrayList<Action>(actions);
    actions.clear();
    for (int i = 0; i < actionsList.size(); i++) {
        if (actionsList.get(i).getUri().equals("http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/Luz")) {
            actions.add(actionsList.get(i));
        }
    }
    return actions;
}
```

Ilustración 44 - Modificación para realizar peticiones únicamente a la Luz

Apéndice D.D. Comprobar que la luz no se pueda encender y apagar más de 20 veces seguidas

Para comprobar que la luz disponible en la vivienda inteligente no se pueda encender y apagar más de 20 veces seguidas sucesivamente, se han modificado varios métodos del protocolo de TESTAR para la IoT.

Por una parte, mediante el método selectAction forzaremos la herramienta a que siempre seleccione una acción compuesta que consista en encender y apagar la luz (dos acciones Put sobre la URI correspondiente a la luz, con las interacciones *on* y *off* respectivamente).

Para ello, instanciamos un Map<String, String> al cual le pondremos las cabeceras *Content-Type: application/json* y *Accept: application/json*. Del mismo modo, dispondremos de tres cadenas que contengan la URI del recurso Luz, el nombre y el valor del *token* respectivamente (estos últimos los obtendremos del fichero de configuración directamente).

Tras ello, estamos ya en condiciones de instanciar una acción Put con la interacción *on* y otra con *off*. Así pues, las pasaremos como parámetro a la acción compuesta la cual será devuelta por el método como acción seleccionada.

Cabe recordar que al tratarse de una acción compuesta, recibe por parámetro tantas acciones como se desee. Dichas acciones se ejecutarán secuencialmente cuando se llame al método run de la acción compuesta.

```
protected Action selectAction(Set<Action> actions) {
    Map<String, String> headers = new HashMap<String, String>();
    headers.put("Content-Type", "application/json");
    headers.put("Accept", "application/json");
    String uri = "http://tambori.dsic.upv.es:8100/smartcity/vsmarthome/thing/Luz";
    String tokenName = settings().get(ConfigTags.TokenName);
    String tokenValue = Util.decodeBase64(settings().get(ConfigTags.TokenValue));
    Action a1 = new Put(uri, headers, "{\"action\":\"on\"}",
            settings().get(ConfigTags.TimeOutTime), tokenName, tokenValue );
    Action a2 = new Put(uri, headers, "{\"action\":\"off\"}",
            settings().get(ConfigTags.TimeOutTime), tokenName, tokenValue);
    return new CompoundAction(a1, a2);
}
```

Ilustración 45 - Selección de acción para apagar y encender la Luz

Para controlar que el número de veces que se enciende y apaga la luz no sea mayor de 20, se ha añadido un contador de las veces que se ejecuta la acción compuesta seleccionada. Para ello, disponemos de la variable *cont* que se incrementa cada vez que se llama el método executeAction.

```
protected boolean executeAction(Action action) {
    cont++;
    return super.executeAction(action);
}
```

Ilustración 46 - Incremento del contador al ejecutar la acción

Por otra parte, en el método getVerdict se comprueba, previamente a las comprobaciones por defecto, si dicho contador supera el valor 20. Si eso ocurre, el contador se inicializa a cero y se avisa devolviendo un veredicto que indica que la luz de la vivienda inteligente se ha encendido y apagado más de 20 veces seguidas.

```
protected Verdict getVerdict(String response) {
    if(cont >20){
        cont = 0;
        return new Verdict(1.0, "The smart home allowed the user to"
                + "turn on and off the light more than 20 times in a row!");
    }
    return super.getVerdict(response);
}
```

Ilustración 47 - Comprobación de si la Luz se ha encendido y apagado más de 20 veces seguidas

Apéndice D.E Controlar que la temperatura no supere los 60ºC

Para las pruebas realizadas con el objetivo de comprobar que la temperatura no supere los 60ºC se decidió que la herramienta ejecutara acciones del modo usual. Así pues, cuando la acción seleccionada es un Get sobre el sensor de temperatura se comprueba si el valor obtenido supera o no los 60ºC.

Para conseguirlo, en el método getVerdict se ha añadido una comprobación para ver si la acción ejecutada es una instancia de un Get y si además su URI contiene la palabra “SensorTemperatura”. Si eso ocurre, obtenemos el valor de la temperatura a partir de la respuesta del servidor. Tras ello, observamos si es mayor que la temperatura umbral y, si se cumple, emitimos un veredicto indicándolo.

```
protected Verdict getVerdict(String response){
    if(action instanceof Get && action.getUri().contains("SensorTemperatura")){
        int temperature = getTemperatureSensed(response);
        if(temperature>60) return new Verdict(1.0, "The received temperature is above 60°C!");
    }

    Verdict verdict;
    if(action instanceof Delete
            || action instanceof Post){
        getPartsOfResponse(response);
        Main.logln("Server response (It should be \"405 Method Not Allowed\"):");
        Main.logln("Code: "+ responseCode + " Message: " + responseMessage +
                " Body: " + responseBody);
        if(!responseCode.equals("405")){
            verdict = new Verdict(1.0, responseCode + " " + responseMessage +
                    " received while expecting 405 Method Not Allowed");
        }else verdict = Verdict.OK;

    }else{
        verdict = super.getVerdict(response);
    }

    return verdict;

}
```

Ilustración 48 - Método getVerdict para comprobar que la temperatura no supere los 60ºC

Sin embargo, obtener el valor de la temperatura a partir de la respuesta del servidor no es trivial. Cabe tener en cuenta que recibimos un JSON con toda la información pertinente a la petición.


```
{"lastAction":"set","currentState":"READY","link":"/thing/SensorTemperatura",
          "actuator":true, "currentValue":{"value":"3"},"isEnabled":true,
          "id":"SensorTemperatura","sensor":true,"isQuiescent":false,
          "name":"Sensor Temperatura","device":{"id":"SensoresArduPi",
          "name":"Sensores ArduPi","isEnabled":true}, "previousState":"UNKNOWN",
          "isStarted":true,"devfunc-type":"numeric"}
```


Ilustración 49 - Ejemplo del cuerpo de una respuesta del servidor al hacer un GET

Para extraer, a partir de dicha respuesta, el valor que buscamos se ha implementado el método getTemperatureSensed en el cual realizan los siguientes pasos:

1. Obtenemos, por separado, las partes de la respuesta (código, mensaje y cuerpo)
2. Nos quedamos con el cuerpo (contiene la información que nos interesa) y lo separamos por el carácter "," mediante la función *split*.
3. De la separación realizada, nos quedamos únicamente con aquella que contenga la cadena "value" y la guardamos en la variable "valor". Por tanto, nos quedamos con una cadena que tiene el siguiente aspecto: *"currentValue":{"value":"3"}*
4. Realizamos una separación por el carácter ":" y nos quedamos con la parte que contiene el valor de la temperatura (En el caso del ejemplo nos quedaríamos con *"3"}*) .
5. Quitamos las comillas y el carácter "}"

6. Finalmente, ya disponemos del valor que deseamos obtener. Sin embargo, como se trata de una cadena, pasamos la variable de String a Integer para poder hacer las comprobaciones pertinentes

```
private int getTemperatureSensed(String response){
    getPartsOfResponse(response);
    String[] parts = responseBody.split(",");
    String valor = null;
    for (int i = 0; i<parts.length; i++){
        if(parts[i].contains("value")){
            valor = parts[i];
            break;
        }
    }

    if(valor != null) parts = valor.split(":");
    valor = parts[parts.length-1];
    valor = valor.replace("\"", "");
    valor = valor.replace("}", "");
    return Integer.parseInt(valor);
}
```

Ilustración 50 - Método para obtener el valor de la temperatura