Universidad Internacional de La Rioja (UNIR)

Escuela de Ingeniería

Máster universitario en Dirección e Ingeniería de Sitios Web

RAMA INVESTIGACIÓN

Metodología para describir servicios RESTful consumidos automáticamente por clientes máquina

Trabajo Fin de Máster

Presentado por: Rodríguez López, Martha Lucía

Director: Pérez Gálvez, Ignacio Javier

Ciudad: La Ceja, Antioquia, Colombia.

Fecha: Septiembre 05 de 2016.

CONTENIDO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

RESUMEN

Los servicios RESTful surgieron en el presente siglo para ser consumidos por clientes máquina. Sin embargo, el desarrollador humano debe indicar a estos clientes cómo funciona el servicio, cuáles parámetros enviar en la solicitud y cuáles parámetros recibirá como respuesta. Por ello, los clientes-máquina tienen dificultades para descubrir, consumir y combinar aquellos servicios RESTful que no fueron identificados en la etapa de desarrollo. Este TFM presenta una metodología para crear la descripción del servicio en un formato entendible por las máquinas. La novedad de esta metodología se basa en ofrecer la descripción como un recurso en sí misma, esto implica que la descripción se puede modificar en tiempo de ejecución para promover la composición de nuevos servicios.

Palabras Clave: RESTful, servicios web, metodología para describir servicios RESTful.

ABSTRACT

RESTful services emerged in this century to be consumed by machine-clients. However, the human developer should indicate these clients how the service works, what parameters to send in the request and which parameters receive in the response. Therefore, machine client have difficulty to discover, consume and combine those RESTful services that were not identified in the development stage. This TFM presents a methodology for creating the service description in a format understandable by machines. The novelty of this approach is based on providing the description as a resource itself. This implies that the description can be modified at runtime to promote the composition of new services.

Keywords: RESTful, web services, methodology for description of services.

1. INTRODUCCIÓN

El término Representational State Transfer (REST) fue acuñado por Roy Fielding (2000) para identificar un estilo de arquitectura orientada al diseño de software basado en la red. Luego el término se extendió para describir un estilo de construcción de servicios web, basado en los principios establecidos por Fielding, éste se ha denominado RESTful.

Bellido et al (2011) definen un servicio RESTful como "una telaraña de recursos interconectados identificados con URIs, que pueden ser manipulados a través de interfaces uniformes (métodos HTTP), cuyo estado es servido a través de representaciones (p. e. página HTML) que contienen enlaces y controles (p. e. un formulario indicando una operación POST), los cuales definen el modelo hipermedia que determina no solo la relación entre recursos, sino también la posible red de transiciones de estado del recurso".

Los servicios RESTful surgieron para ser consumidos por clientes máquina. Sin embargo, la mayoría de los RESTful se describe en páginas web, a través de documentos de texto, dónde se dan ejemplos de uso, se definen parámetros de entrada y de salida, y todos los detalles necesarios para que una persona sea capaz de programar una aplicación que consume estos servicios. El desarrollador humano debe indicar a los clientes cómo funciona el servicio. Esto limita a los clientes-máquina para descubrir, consumir y combinar, en tiempo de ejecución, servicios RESTful que no fueron identificados durante la etapa de desarrollo.

Con la aparición de nuevas aplicaciones donde dispositivos electrónicos se conectan a Internet, conseguir que un cliente máquina aprenda cómo utilizar un servicio web nuevo sin intervención humana, abre un mundo de posibilidades a la comunicación máquina a máquina.

Diversos autores proponen crear un documento semántico que describa el funcionamiento del servicio RESTful. Se busca que el cliente-máquina interprete este documento sin ayuda humana y pueda comprender cómo funciona, qué parámetros debe enviar en la solicitud y qué clase de parámetros va a recibir en la respuesta. Pero hasta la fecha no existe consenso acerca de cómo construir esta descripción.

Como solución al problema descrito se propone una metodología para describir servicios RESTful a través de un documento XML que contiene la semántica del servicio. La descripción del servicio constituye un recurso en sí mismo, al cual se

accede a través de su URI con el método GET. Un cliente-máquina con las credenciales adecuadas, puede modificar la descripción del RESTful para calificarlo positiva o negativamente (ranking) e incluir información de recursos de terceros para promover la composición de servicios.

En el capítulo dos se plantean los objetivos, el problema identificado y la solución a implementar. También explican los pasos que conforman la metodología de trabajo en el proyecto de investigación.

El siguiente capítulo expone los conceptos asociados al proyecto y las propuestas de varios autores para describir servicios RESTful. Estas propuestas van desde no describir los servicios y usar solo los parámetros entregados por los métodos HTTP, hasta completas metodologías para describir la ontología y la semántica de los servicios web.

El capítulo cinco explica en detalle la metodología para describir servicios RESTful consumidos automáticamente por clientes-máquina. “Automáticamente” significa que el cliente a través de la descripción identifica la función, los parámetros de solicitud y de respuesta para un servicio que era desconocido durante la etapa de desarrollo. También se describe el experimento que pondrá a prueba la metodología. En este caso se trata de un dispositivo que mide variables agronómicas y consume servicios RESTful para almacenar y recuperar datos, al tiempo que genera alarmas.

Junto con la metodología se presenta el framework desarrollado en HTML+CSS+JavaScript para automatizar la descripción del servicio. Se puede acceder a esta aplicación ingresando al sitio web http://diysmartdevices.com/framework/index.html.

La interpretación de los datos arrojados por el experimento, las conclusiones y las recomendaciones se exponen en el capítulo final. Los resultados expuestos en esta sección muestran que es posible para los clientes máquina consumir servicios no identificados durante la etapa de desarrollo. Y más aún, las máquinas pueden crear redes de colaboración a través de su intervención en la estructura de los recursos que describen servicios RESTful.

2. PLANTEAMIENTO DEL PROYECTO DE INVESTIGACIÓN

2.1. OBJETIVO GENERAL

Diseñar una metodología para describir servicios RESTful que permita a los clientes máquina descubrir, invocar, componer y recomendar servicios web sin intervención humana.

2.2. OBJETIVOS ESPECÍFICOS

- Analizar el estado del arte del tema propuesto.
- Identificar requisitos, los roles y las tecnologías asociadas a la metodología.
- Establecer los pasos de la metodología.
- Diseñar el framework de apoyo para la metodología.
- Evaluar la metodología para un caso de estudio.
- Realizar ajustes a la metodología.
- Elaborar el informe del proyecto de investigación.

2.3. METODOLOGIA DE TRABAJO

En el proyecto de fin de master “metodología para describir servicios RESTful consumidos automáticamente por clientes máquina” se va a realizar investigación cuantitativa. La cual recoge, procesa y analiza datos e información de variables numéricas relacionadas entre sí. Este proceso investigativo usa métodos deductivos que permitirán formalizar conclusiones a partir de los resultados obtenidos en los experimentos.

La metodología de trabajo se compone de los pasos indicados en la tabla 1, donde se muestra el cronograma de trabajo del proyecto de investigación.

Tabla 1. Cronograma de trabajo.

CRONOGRAMA DE TRABAJO						
ACTIVIDAD	**Marzo**	**Abril**	**Mayo**	**Junio**	**Julio**	**Agosto**
Planteamiento del problema.						
Revisión del estado del arte para identificar cómo describen servicios REST otros autores.						
Formulación de la hipótesis.						

Plantear la solución como una metodología que describe servicios web.						
Desarrollar Framework en HTML+CSS+JAVASCRIPT para automatiza la metodología.						
Realizar el experimento con aplicaciones Arduino que consumen servicios RESTful.						
Análisis de datos e interpretación de resultados.						
Elaboración de conclusiones e informe escrito.						

Planteamiento del problema de investigación

Consiste en describir la situación asociada con los servicios RESTful que representa un problema para los clientes máquina que los consumen. Luego se formula el problema, especificando claramente qué se va a estudiar y cuál es la pregunta relevante del asunto.

Revisión del estado del arte

Consiste en revisar fuentes de información relacionadas con servicios RESTful, para conocer cómo autores expertos en la temática han enfrentado la problemática de describir servicios web.

Formulación de la hipótesis de investigación

La hipótesis busca responder la pregunta relevante que plantea la definición del problema. Esta afirmación determina el plan de trabajo de la investigación, porque condiciona las técnicas a usar, las variables de estudio, los experimentos a realizar y las técnicas para la interpretación de los datos.

Planteamiento de la solución del problema como una metodología

En este proyecto de investigación en particular, se ha definido que la solución del problema será obtenida a través del diseño de una metodología. El conjunto de pasos a ser ejecutados secuencialmente para alcanzar la solución del problema se enumeran a continuación y se explican en detalle en el apartado correspondiente.

- Identificar las características de los posibles clientes máquina.
- Caracterizar los recursos requeridos por los clientes máquina.

- Describir la interface del recurso.
- Describir el servicio.
- Convertir la descripción en un recurso.
- Usar un cliente máquina de prueba para consumir la descripción.
- Publicar la descripción del servicio.

Desarrollar Framework que automatiza la aplicación de la metodología

Siendo el proyecto de investigación parte del Master en Dirección e Ingeniería de Sitios Web, se ofrece el desarrollo de una aplicación en HTML+CSS+JAVASCRIPT capaz de automatizar la implementación de la metodología propuesta. El framework se ha publicado en http://diysmartdevices.com/framework/index.html junto con una serie de video que explica su utilización.

Aplicación del Experimento

El experimento que se va a diseñar busca determinar la validez de la metodología propuesta como solución del problema. Primero se desarrolla el hardware usando sistemas embebidos Arduino y se programa el software necesario. Luego se realiza el experimento y se recolectan los datos, documentando cada paso para garantizar que otros investigadores reproduzcan el experimento obteniendo resultados similares.

Análisis de datos e interpretación de resultados

El hardware construido para consumir la descripción de los servicios RESTful se encuentra conectado a una interface serial que registra los datos intercambiados entre el cliente y el servidor. Se aplican métodos de análisis a los datos recolectados durante el experimento, se interpretan tales resultados y se concluye la validez del experimento y de la metodología planteada como solución del problema identificado.

Elaboración del informe

Se elabora un informe escrito con los resultados del proyecto, exponiendo claramente cada etapa del proceso de investigación llevado a cabo, para facilitar a otros investigadores la reproducción de los experimentos y la obtención de los hallazgos.

3. PLANTEAMIENTO DEL PROBLEMA DE INVESTIGACIÓN

Aunque los servicios web se programan para ser consumidos por las máquinas. En la actualidad, los servicios RESTful son descritos en lenguaje natural para los humanos. Esta descripción debe ser obtenida por el desarrollador antes de programar el cliente que consume el servicio. Lo anterior limita a los clientes-máquina* para:

- Intercambiar automáticamente credenciales de autenticación y autorización para acceder a servicios web.
- Deducir automáticamente cómo trabaja el servicio web, cómo es usado y qué datos debe proveer el cliente.
- Descubrir servicios web automáticamente.
- Invocar servicios web automáticamente.
- Realizar composición automática de servicios Web.
- Modificar la descripción del servicio en tiempo de ejecución.

(*) Cliente-Máquina. Para el alcance de este TFM, se define “cliente-máquina” como un dispositivo conectado a Internet para consumir servicios web sin intervención humana. Específicamente, dispositivos basado en microprocesadores de baja capacidad de procesamiento (bus de instrucciones de 8bits, velocidades hasta 20MHz) y baja capacidad de almacenamiento (inferior a 1MB para EEPROM y RAM), usados en aplicaciones tipo “Internet de las Cosas”.

Pregunta principal: “¿La descripción de un RESTful permite a los clientes máquina para promover, descubrir, consumir y componer servicios web sin intervención humana?

Afirmación o hipótesis: La descripción de un RESTful permite a los clientes máquina para promover, descubrir, consumir y componer servicios web sin intervención humana.

4. ESTADO DEL ARTE

4.1. CONTEXTUALIZACIÓN

Un servicio web es "*una tecnología que permite que las aplicaciones se comuniquen en una forma que no depende de la plataforma ni del lenguaje de programación. Un servicio web es una interfaz de software que describe un conjunto de operaciones a las cuales se puede acceder por la red*" (IBM).

Los primeros servicios web se crearon para hacer llamadas a procedimientos remotos (RPC) a través de la web. Con el tiempo surgieron una colección de estándares; comenzando por SOAP y WSDL, seguidos por toda la serie WS-*. Algunos consideran que esta prematura estandarización generó confusión en los desarrolladores, facilitando el surgimiento de los servicios RESTful, mucho más fáciles de desarrollar y de consumir (Paul Adamczyk, 2011).

Hoy en día Internet ofrece una amplia variedad de servicios con diversas funciones. Estos servicios se clasifican como basados en: WSDL, REST, XML-RPC, Atom o JSON-RPC. Siendo las dos primeras opciones las más populares para ofrecer plataformas, donde la reutilización y composición de servicios reducen costos y dan valor agregado (Bellido, 2011). Este trabajo se centra específicamente en los servicios basados en el estilo de arquitectura REST.

El término Representational State Transfer (REST) fue acuñado por Roy Fielding para identificar un estilo de arquitectura orientada al diseño de software basado en la red. Luego el término se extendió para describir un estilo de construcción de servicios web, basado en los principios establecidos por Fielding, éste se ha denominado RESTful (Paul Adamczyk, 2011).

Bellido et al (2011) definen un servicio RESTful como "*una telaraña de **recursos** interconectados identificados con **URIs**, que pueden ser manipuladas a través de **interfaces uniformes** (p. e. verbos HTTP), cuyo **estado** es servido a través de **representaciones** (p. e. una página HTML) que contienen **enlaces** y **controles** (p. e. un formulario indicando una operación POST), los cuales definen el **modelo hipermedia** que determina no solo la relación entre recursos, sino también la posible red de transiciones de estado del recurso*".

4.1.1 ARQUITECTURA REST

El estilo de arquitectura REST describe seis principios, las cuales aparecieron originalmente en la tesis doctoral de Roy Fielding y definen las bases del estilo REST (Fielding, 2000):

- Cliente- Servidor (Client-Server).
- Sin estado (Stateless).
- Almacenamiento en caché (Cacheable).
- Interface uniforme (Uniform Interface).
- Sistema por capas (Layered System).
- Código bajo Demanda (Code on Demand).

Cliente Servidor (Client-Server). Las funciones del cliente y el servidor están bien diferenciadas, ambos pueden evolucionar separadamente. Por ejemplo, el almacenamiento de datos es tarea del Servidor, mientras que la interface de usuario es tarea del cliente.

Sin estado (Stateless). "Sin estado" es un concepto clave en los servicios REST. Significa que toda la información necesaria (estado de la aplicación) para manejar una solicitud está contenida dentro de la misma como parte de la URI (que identifica el recurso) o en el cuerpo del mensaje (que contiene el estado o cambio de estado). Este principio mejora la visibilidad, fiabilidad y escalabilidad del servicio porque:

- Una solicitud contienen toda la información que determina su naturaleza, así un sistema de monitoreo no tiene que mirar más allá (visibilidad).
- Facilita la recuperación de fallos parciales (fiabilidad).
- El servidor no tiene que administrar el uso de recursos a través de las solicitudes (escalabilidad).

Almacenamiento en caché (Cacheable). En la Web los clientes pueden almacenar respuestas del servidor (responses) en la memoria caché. Estas respuestas se definen a sí mismas como almacenables o no (cacheable), para evitar que el cliente use inapropiadamente los datos o el estado. Lo anterior es muy importante para mejorar la escalabilidad o rendimiento de la aplicación.

Interface Uniforme (Uniform Interface). Define la interface entre clientes y servidores, permitiendo que ambas partes evolucionen independientemente, sin que tengan que compartir la misma arquitectura. Los principios que guían la construcción de interfaces uniformes son (Fredrich, 2012):

- **Basado en recursos**. La URI (Uniform Resource Interface) identifica un recurso individual durante la solicitud (request) realizada por el cliente. Estos recursos son independientes de las representaciones retornadas en la respuesta (response) del servidor.
- **Manipulación de recursos a través de representaciones**. Cuando un cliente tiene una representación de un recurso; se dice que él puede modificar o eliminar dicho recurso, siempre que tenga los permisos necesarios.
- **Mensajes auto descriptivos**. Cada mensaje incluye la información necesaria para procesarlo, ya sea de lado del servidor o del lado del cliente.
- **Hipermedia como máquina de estado de la aplicación (HATEOAS**). La hipermedia se define como hiperenlaces con hipertexto. Los clientes envían "el estado de la aplicación" a través de parámetros de consulta en el cuerpo, en el encabezado de la solicitud o en el nombre del recurso (URI). Mientras que los servicios envían el estado de un recurso a través del contenido del cuerpo o del encabezado del mensaje y en los códigos de respuesta.

Sistema por capas (Layered system). El sistema de capas se refiere al uso de servidores intermediarios, los cuales mejoran la escalabilidad del sistema, al tiempo que permiten el balanceo de cargas y ofrecen almacenamiento compartido. El sistema por capas puede reforzar las políticas de seguridad.

Código bajo Demanda (Code on demand). Los servidores pueden extender las capacidades de un cliente transfiriéndole código que puede ser ejecutado por éste. Como en el caso de los scripts de JavaScript. Esta es una característica opcional para el estilo de arquitectura REST. Sin embargo, las demás características son obligatorias para referenciar un servicio como RESTful.

En la práctica se espera que los servicios web RESTful sigan el modelo CRUD. Concepto tomado de las bases de datos, dónde se definen métodos para crear, leer, actualizar y borrar un recurso en el servidor. Esto se consigue aplicando los métodos HTTP (POST, GET, PUT, DELETE). HTTP/1.1 define ocho métodos (Paul Adamczyk, 2011).

4.1.2. MÉTODOS HTTP

Los verbos HTTP definen las acciones a realizar sobre los recursos. Los más usados son POST, GET, PUT y DELETE que corresponden a las acciones Create, Read, Update y Delete (CRUD) (Fredrich, 2012). Antes de explicar estos verbos, conviene explicar dos conceptos:

- **Idempotencia**. Una operación que produce el mismo resultado si se ejecuta una o muchas veces, aunque las respuestas del servidor no sean idénticas. Los verbos PUT y DELETE se consideran idempotentes, pero no seguros. Mientras que GET, HEAD, OPTIONS y TRACE son definidos como idempotentes y seguros.
- **Seguridad**. Un verbo HTTP se considera seguro si no cambia el estado del recurso en el servidor, son operaciones de solo lectura. Por definición una operación segura es también idempotente.

GET. El método GET es usado para leer la representación de un recurso, sin modificarlo o dañarlo. Una respuesta exitosa generalmente incluye una representación en XML o JSON y el código 200 para OK. En caso de error retorna el código 404 (not found) o 400 (bad request). GET es una operación segura e idempotente.

PUT. PUT se usa para actualizar recursos. En el cuerpo del mensaje se envían los datos que actualizarán el recurso original. Algunos servicios usan PUT para crear un nuevo recurso cuando la ID enviada por el cliente no corresponde a un recurso existente, aunque esto no es recomendable porque genera confusión con el verbo POST. El cuerpo en la respuesta del PUT es opcional. PUT es idempotente, pero no es seguro porque modifica el estado del recurso.

POST. Es el verbo usado para crear recursos subordinados. Es decir, que al nuevo recurso se le asigna una ID que lo asocia con el padre. La respuesta retorna el código 201 para indicar una creación exitosa. POST no es idempotente ni seguro, porque envíos sucesivos de la misma solicitud pueden generar la creación de sendos recursos.

DELETE. Usado para eliminar un recurso identificado por su URI. Un borrado exitoso retorna el código 200 con el cuerpo de la respuesta, o 204 si no se incluye contenido en el cuerpo de la respuesta. DELETE no siempre puede considerarse idempotente. Por ejemplo, si se trata de una base de datos, se podría recibir el error 404 (not found) en lugar de marcar el recurso como borrado.

4.2. DESCRIPCIÓN DE SERVICIOS RESTful

Si las máquinas no pueden deducir por sí mismas el funcionamiento de un servicio no podrán elegir automáticamente las rutas de navegación para alcanzar recursos que no han sido identificados durante la fase de desarrollo (Bellido, 2011).

A continuación se presentan las propuestas de diferentes autores para crear documentos que expliquen a las máquinas cómo utilizar un servicio sin intervención humana.

DESCRIPCIÓN TEXTUAL DE SERVICIOS RESTful

La mayoría de RESTful se describe en páginas web, a través de documentos de texto, dónde se dan ejemplos de uso, se definen parámetros de entrada y de salida, y todos los detalles necesarios para que una persona sea capaz de programar una aplicación que consume estos servicios (Kopecky, 2008).

WSDL

WSDL(Web Service Description Language) es un estándar W3C que usa un formato XML para describir un servicio como un conjunto de operaciones contenidas dentro de mensajes abstractos, que luego se asocian a un protocolo de red concreto (W3C, 2001). Aunque, WSDL es útil con servicios SOAP, no ha mostrado igual versatilidad con servicios que siguen la arquitectura REST (Kopecky, 2008).

WADL

Para Bellido et al (2011) un acercamiento entre los servicios REST y los clientes-máquina, lo constituye WADL (Web Application Description Language, lenguaje de descripción de aplicación web). WADL permite escribir descripciones para servicios, que pueden ser leídas por máquinas. Además permite incluir parámetros que contienen enlaces a otros recursos. Sin embargo, WADL no soporta el descubrimiento de enlaces ni la generación de enlaces para nuevos recursos, porque se trata de un modelo centrado en la operación.

ReLL

Otro acercamiento entre REST y clientes-máquina lo constituye ReLL (Resource Linking Language, lenguaje de enlace de recurso), un modelo centrado en la hipermedia para crear descripciones de servicios REST. ReLL integra los recursos, las representaciones, los enlaces y los mecanismos para identificar cambios en los REST descritos. Permitiendo a los clientes-máquina acceder en tiempo de ejecución a la información embebida en las representaciones, para que éstos sean capaces de descubrir los recursos enlazados dentro de un servicio REST. ReLL usa tipos para describir recursos y enlaces/controles, constituyendo la base de un modelo semántico (Bellido, 2011).

SAWSDL

Es un intento para agregar semántica a WSDL. Sin embargo, SAWSDL solo se centra en la semántica de los parámetros de entrada y salida del servicio, sin detallar la conexión funcional entre estos (Verborgh, 2011).

OWL-S

Ontology Web Language for Web Services, lenguaje de marcas para agregar ontología a servicios web. Ontología es una definición de tipos, propiedades y relaciones entre entidades que existen en un dominio de conocimiento específico. Usar ontología en servicios web busca responder a las preguntas (W3C, 2004):

- ¿Qué ofrece el servicio para los clientes potenciales? Esta pregunta se responde dentro del perfil (profile) usado para publicitar el servicio.
- ¿Cómo se usa? La respuesta se halla en el modelo de proceso usado (Process Model).
- ¿Cómo se puede interactuar con él? Los detalles acerca de los protocolos de transporte usados responden a esta pregunta.

OWL-S se enfoca en los parámetros de entrada y salida, así como en la descripción funcional de relaciones a través de expresiones del lenguaje. Sin embargo, estas expresiones no están integradas y forman una capa separada (Verborgh, 2011).

JSON DESCRIBIENDO HIPERMEDIA

Un recurso puede ofrecer hiperenlaces para permitir al cliente decidir cómo navegar a través de la API REST o cuando ejecutar una operación CRUD (Create, Read, Update, Delete). JSON es un formato para intercambio de datos, usado con frecuencia entre clientes y servicios RESTful. Algunos autores consideran que JSON podría usarse para describir la hipermedia dentro de un servicio web. Sin embargo, no ofrece soporte para la notación semántica propia de un servicio web (Salvadori, 2014).

Salvadori et al (2014) describen algunas alternativas para convertir JSON de un formato de datos a un formato de hipermedia:

- **JSON-LD**. Formato basado en JSON con soporte a enlaces de datos y controles hipermedia. Básicamente, permite que toda la respuesta enviada por el servicio, enlaces y datos, se exprese en formato JSON. Se crean atributos para describir información propia del contexto. Pero no ofrece mecanismos para especificar operaciones que alteren el estado del recurso.

- **HYDRA**. Amplía el alcance de JSON-LD, agregando vocabulario para representar recursos con sus atributos y operaciones. Al tiempo que ofrece una lista de enlaces a otros recursos. Pero este enfoque hace que no esté clara la separación entre la capa de datos y la capa de negocio.
- **JAX-RS**. Es una API Java para servicios REST. Integra las dos opciones anteriores en un framework que separa la aplicación en capas de representación, de negocios y de datos. Además se crean clases para representar nuevas operaciones, ya que los autores consideran que los verbos HTTP no son capaces de describir la semántica de solicitudes de negocios complejos (Salvadori, 2014). Pero crear nuevas operaciones aleja el servicio de su naturaleza REST.

RESTdesc

Uno de los principios de la arquitectura REST es el uso de hiperenlaces para guiar al cliente en el descubrimiento de nuevos servicios. Se espera que el servidor envíe dentro de la respuesta (response) enlaces a otros recursos de interés. Pero esto no suele suceder y el usuario se ve limitado a los servicios que conoce de antemano, en el momento de programar al cliente (Verborgh, 2011).

RESTdesc es un método para describir servicios RESTful. RESTdesc se basa en notación RDF y busca ofrecer una descripción simple y elegante, centrada en la funcionalidad del servicio, usando tecnologías de web semántica. Las principales características de este método son (Verborgh, 2011):

- Expresa las funciones como reglas semánticas.
- No introduce terminología nueva, ni especifica tipos. Porque asume que los clientes conocen la ontología con anticipación.
- La descripción es auto-contenida y auto-explicada, sin requerir información adicional.
- Describe el verbo HTTP requerido para ejecutar la operación.

RESTdesc cubre las necesidades de descripción semántica del servicio. Para la composición de servicios usa un proceso de razonamiento que lleva hasta la salida requerida siguiendo reglas de descripción. Pero no ofrece la posibilidad de que diferentes agentes intercambien enlaces y descripciones (Verborgh, 2011).

RDF

Debido a las características propias de los dispositivos conectados a una red IoT (bajas capacidades de procesamiento, almacenamiento y bajo consumo de energía) se requiere que los consumidores y proveedores de servicios web estén débilmente

acoplados, para que se adapten con facilidad a ambientes cambiantes, dónde los dispositivos o los servicios pueden dejar de estar disponibles con facilidad (Kovatsch, 2015).

Kovatsh et al (2015) describe el uso de la herramienta semántica RDF (Resource Description Framework), para definir un modelo de metadatos que describe datos semánticos sin considerar el dominio de la aplicación. RDF usa tres tipos de datos:

- Recursos (Resources) descritos con URIs.
- Propiedades (Properties) que describen atributos del recurso.
- Sentencias (Statements) como la combinación de tres entidades: Recurso (sujeto), propiedad (predicado) y un valor de la propiedad (objeto) que puede ser otro recurso.

Aplicando RDF en RESTdesc, Kovatsh (2015) consigue derivar nuevas sentencias (statements) a partir de las establecidas explícitamente, permitiendo la correcta aplicación de los principios de la arquitectura REST en servicios leíbles por máquinas. Los autores han incluido vocabulario ontológico para manejar las transiciones de cambio de estado físico de los sensores y actuadores propios del dominio de la Internet de las Cosas. Y se pasa al cliente-máquina un “plan” (lista de enlaces codificada en JSON), para que este lo ejecute.

hRESTS Y LOS MICROFORMATOS SEMÁNTICOS

Kopecky et al (2008) proponen una descripción para servicios RESTful basada en HTML para clientes-máquina, ellos la denominan hRESTS. A través de microformatos se describen las operaciones, las entradas y las salidas.

Los microformatos son una “adaptación de la semántica XHTML que hace más fácil publicar, indexar y extraer información semi-estructurada”. Con microformatos se puede ofrecer descripciones textuales, y descripciones operacionales acerca de cómo usar enlaces y formularios (Kopecky, 2008).

hRESTS implementa hojas de transformación XSLT para extraer datos RDF desde páginas web XHTML. Y la descripción de hRESTS es un documento XHTML que referencia esta hoja XSLT. Para realizar estas transformaciones usar Parsers y validadores de estructura (Kopecky, 2008).

CoAP

Constrained Application Protocol (CoAP) es un protocolo liviano de transferencia, similar a HTTP, para ser usado con dispositivos de baja capacidad de procesamiento y

de almacenamiento que operan en redes informáticas de bajo desempeño. Propuesto por Internet Engineering Task Force (IETF) para ser usado en entornos que implementan la arquitectura REST. Uno de los objetivos de sus creadores, es ofrecer servicios RESTful para clientes máquina con características limitadas (Kovatsch, 2015).

NO DESCRIBIR LOS SERVICIOS RESTful

Algunos autores (Wilde & Pautasso, 2011) consideran que los requisitos propios de la arquitectura REST evitan la necesidad de describir los servicios RESTful, sus argumentos se basan en:

- Interface uniforme (uniform interface) evita la necesidad de una descripción específica de la interface del servicio, porque estas interfaces están definidas por el protocolo HTTP.
- Hipermedia como controlador del estado de la aplicación evita la necesidad de especificar como descubrir los servicios, porque estos se descubren siguiendo enlaces.
- Un documento independiente que describa un RESTful puede quedar desactualizado ante nuevas versiones del servicio.
- Una descripción en lenguaje apropiado para clientes máquina podría hacer que el servicio deje de ser débilmente acoplado, evitando que sea escalable.
- Describir servicios a través de semántica web podría generar un conjunto de características (constraints) diferentes a las establecidas por REST.
- Si el servicio REST por naturaleza es auto-descrito, podría aplicarse semántica web dentro de éste para facilitar su consumo por parte de clientes máquina.

4.3. METODOLOGÍAS PARA DESCRIBIR SERVICIOS RESTful

Algunos autores van más allá, y proponen una metodología detallada para elaborar la descripción de un servicio RESTful.

METODOLOGÍA BASADA EN WSMO

Kerrigan et al (2009) proponen una metodología para describir servicios web usando ontología a través de un framework basado en el entorno de desarrollo Eclipse de Java. Esta propuesta se basa en la premisa de que los servicios web semánticos deben ser descubiertos, compuestos, ranqueados, seleccionados e invocados dinámicamente en tiempo de ejecución.

Para conseguir lo anterior, los autores usan WSMO (Web Service Modeling Ontology, Ontología para modelar servicios web). Cuyos cuatro pilares: ontología, servicio web, metas y mediadores; definen los pasos de esta metodología:

1. Construir las ontologías del dominio. Reutilizando las existentes o creando ontologías nuevas.
2. Describir el servicio web. Definir las interfaces de entrada y salida, así como las funcionalidades y condiciones de uso del servicio. También se incluyen palabras claves para facilitar su descubrimiento y composición.
3. Describir las capacidades meta. Básicamente consiste en describir un conjunto de muestras de lo que puede hacer el servicio, para que sirvan como guía de referencia.
4. Construir mapas de mediación. Los mediadores son usados por WSMO para resolver asuntos de heterogeneidad entre diferentes descripciones.
5. Probar las descripciones. Usar un framework para comprobar que las descripciones se comportan de la manera esperada.
6. Implementación de la descripción. Corresponde a la publicación de la descripción en un repositorio de servicios o en cualquier otro lugar de la web.

Esta metodología se basa en WSMO, un estándar de W3C para describir servicios web tipo SOAP, lo que no la hace directamente aplicable a la descripción de servicios RESTful, donde no es necesario el uso de mediadores.

RESTML

Sánchez et al (2014) proponen RESTML, un lenguaje específico para modelar servicios que aplican la arquitectura REST de manera estricta. Este lenguaje se basa en perfiles UML (Lenguaje unificado de modelado). Los pasos que propone esta metodología son:

1. Elaborar modelos de dominio para representar los requerimientos de la aplicación a través de un vocabulario común. Para ello usa casos de uso, diagramas de casos de uso y diagramas de actividades.
2. Plantear modelos independientes de la plataforma aplicando transformaciones a los modelos de dominio. RESTML propone dos opciones en este paso: los diagramas de clases y los diagramas de secuencias.
3. Implementar el modelo en una plataforma de desarrollo específica. RESTML propone usar Java Enterprise Edition Platform.
4. Codificar los modelos para obtener el código fuente.
5. Empaquetar e implementar el código fuente como una aplicación web.

6. Documentar. En cada paso del desarrollo del modelo se genera automáticamente la documentación que describe el servicio desarrollado.

Esta metodología implementa modelos de dominio que bien podrían representar la ontología necesaria para una descripción semántica del servicio. Sin embargo, la descripción semántica del servicio RESTful no es un paso explícito dentro de la metodología, ni la verificación de que el servicio desarrollado cumple con la arquitectura REST.

METODOLOGÍA BASADA EN CRUD

Lars Hagge (2011) propone una metodología para desarrollar servicios REST usados por sistemas de información. En el primer paso identifica la ontología del negocio específico. Luego programa los servicios REST con operaciones CRUD que representan los recursos de cada servicio. Estos servicios se apoyan de documentos como manuales de operación o especificaciones técnicas. El autor representa el flujo de trabajo como una secuencia de operaciones CRUD. Sin embargo, no hace referencia a una descripción semántica de los servicios.

METODOLOGÍA BASADA EN ATOM

Robinson (2011) apoyado en HTTP como un protocolo que ofrece un conjunto de reglas para acceder y manipular estados de recursos de forma uniforme, propone usar Atom Publication Protocol como convención para publicar contenido web, y The Sun Cloud API para indicar el manejo de recursos en la nube.

En HTTP cada cliente recuerda el estado de la aplicación porque sería muy costoso que el servidor recuerde el estado de todos los clientes. El servicio RESTful puede enviar en OPTIONS una serie de opciones para interactuar con nuevos recursos. Con esto en mente, el autor propone una serie de tres pasos para la capa de negocio en una aplicación web RESTful:

- Modelar la aplicación como una máquina de estado para el protocolo de aplicación. Para modelar la transición entre los estados de la aplicación. El objetivo de este paso es un mejor entendimiento del sistema y no será documentada para el consumidor.
- Implementar la máquina de estado basado en el ciclo de vida de los recursos.
- Documentar y ejecutar el paso dos usando tipos de hipermedia, relaciones como enlaces y verbos HTTP.

Sin embargo, puede objetarse que HTTP es un protocolo suficiente para ofrecer servicios RESTful, sin necesitar Atom Publication Protocol ni The Sun Cloud API.

DESCRIPCIONES AUTOMÁTICAS DE SERVICIOS EXISTENTES

Czyszczoń et al (2014) proponen un método automático para extraer la información que los clientes máquina requieren, directamente de la descripción HTML creada para los clientes humano, en servicios RESTful dispersos por la web. Los pasos propuestos por los autores son:

- Detectar si una página corresponde a la documentación de servicios RESTful. Para ello identifica palabras claves y la frecuencia de aparición en el documento.
- Identificar servicios RESTful a través de un patrón acorde con una URI.
- Extraer información del servicio RESTful. Cuando la URI se presenta fragmentada se deben identificar todos los trozos de la misma.
- Aplicar algoritmos para extraer información. Componer una descripción del RESTful para clientes máquina a partir de la información extraída en los pasos anteriores.

Aunque esta metodología ofrece una alternativa para que las máquinas comprendan por sí mismas cómo funciona un servicio, no ayuda implícitamente a la composición de servicios en tiempo de ejecución.

5. PLANTEAMIENTO DE LA SOLUCIÓN

5.1. SOLUCIÓN QUE SE PROPONE

Como solución del problema planteado se propone el diseño de una metodología para describir servicios RESTful a través de un documento XML que contiene la semántica del servicio. Este documento constituye, en sí mismo, un recurso del servicio y puede ser interpretado directamente por cualquier cliente. Esta descripción permite a los clientes-máquina descubrir e invocar el servicio, así como componer nuevos servicios sin ninguna intervención humana, respetando los principios de la arquitectura REST: cliente-servidor, sin estado, almacenamiento en caché, interface uniforme, sistema por capas, código bajo demanda.

5.2. NOVEDAD DE LA SOLUCIÓN

La particularidad de esta solución, en comparación con otras descripciones de servicios, radica en la combinación de los siguientes aspectos:

- La solución propuesta es apropiada para clientes-máquina basados en microprocesadores, con baja capacidad de procesamiento y almacenamiento.
- La descripción del servicio constituye un recurso en sí mismo. Al cual se accede a través de su URI con el método GET.
- La descripción puede ser leída por máquinas y por humanos, sin requerir documentos adicionales (como hojas de transformación).
- La descripción semántica que se propone, permite a los clientes-máquina descubrir, invocar y combinar servicios de manera automática.
- Un cliente-máquina con las credenciales adecuadas, puede modificar la descripción del servicio RESTful con el método POST para:
    - Incluir información de recursos de terceros, promoviendo la composición de servicios.
    - Calificar positiva o negativamente el servicio, y así promover o no el uso por parte de otros clientes.

5.3. RESTRICCIONES DE LA SOLUCIÓN PROPUESTA

Las restricciones asociadas a la solución propuesta son:

- Todos los servicios deben compartir la misma semántica (estructura de la descripción) y los clientes-máquina deben conocerla con anticipación.
- Un administrador humano debe obtener las credenciales para acceder a servicios que así lo requieran.

- Inicialmente, los clientes-máquina conocen la URI de la descripción de uno de los recursos, esto constituye el punto de partida del proceso.

5.4. PLANTEAMIENTO DE LA SOLUCIÓN COMO UNA METODOLOGÍA

Antes de explicar la metodología es necesario aclarar algunos términos usados en ella. Para el presente texto una API REST, se asume como un conjunto de servicios web que implementan la arquitectura REST. Un SERVICIO WEB es un conjunto de recursos a los que se accede a través de interfaces uniformes. Y un RECURSO es un objeto informático cuyas propiedades se pueden conocer o modificar remotamente a través de métodos HTTP. La metodología que se propone busca que los clientes máquina sean capaces de:

- Intercambiar automáticamente credenciales para autenticación y autorización para acceder a servicios web.
- Deducir automáticamente cómo trabaja el servicio web, cómo es usado y qué datos debe proveer el cliente.
- Descubrir servicios web automáticamente.
- Invocar servicios web automáticamente.
- Realizar composición automática de servicios Web.
- Modificar en tiempo de ejecución, la descripción del servicio, por parte del cliente.

Todo lo anterior gracias a una descripción del servicio RESTful que informa al cliente máquina la URI del recurso, las credenciales requeridas, los parámetros que debe enviar al servidor y los parámetros que recibirá con cada método invocado. Esta descripción constituye en sí misma un recurso, al cual el cliente accede con el método GET y la modifica con el método POST. Proponiendo al servidor que cambie el ranking que promueve el uso del recurso o informando de otros recursos similares el cliente ayuda al descubrimiento, invocación y composición de servicios por parte de otros usuarios.

Esta metodología se puede aplicar a servicios nuevos o a servicios disponibles en la web, incluso servicios de terceros. La figura 1 muestra los pasos propuestos para la metodología.

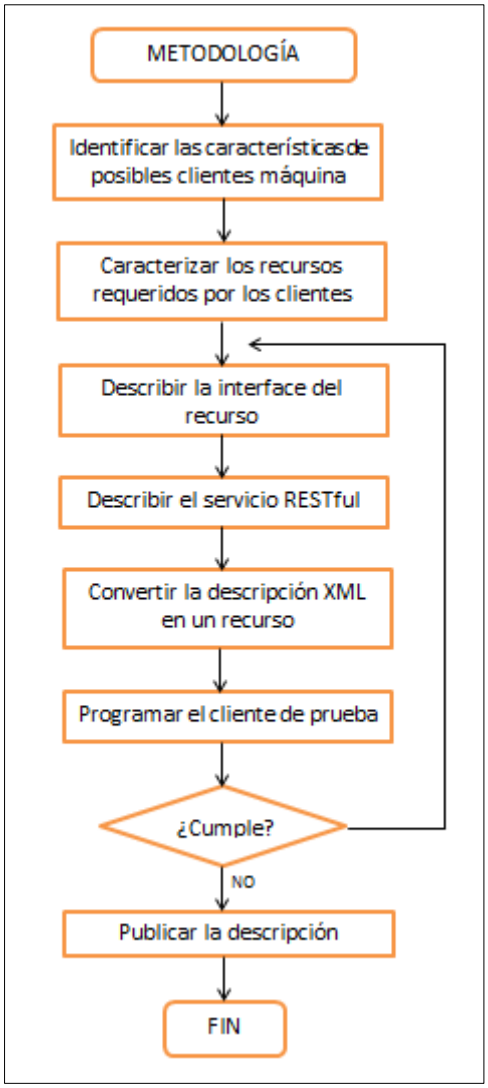

Figura 1. Diagrama de la metodología.

PASO 1: IDENTIFICAR LAS CARACTERÍSTICAS DE LOS POSIBLES CLIENTES MÁQUINA

Las características de los clientes máquina determinan la inteligencia con la que éstos podrán consumir los servicios descritos. Básicamente busca identificar:

- Capacidades de procesamiento. El cual suele limitarse a procesadores de 8 bits en sistemas Internet de las Cosas.
- Capacidades de almacenamiento, el cual puede ir de pocos kilobits hasta Gigabytes si se utilizan unidades externas.
- Ancho de banda de la conexión a internet.
- Posibles componentes, como tipo de procesador, periféricos, memorias externas, entre muchos otros.
- Dominio del objeto. Las variables físicas que mide o el tipo de dato consumido.

- Cosas que podría hacer el cliente máquina. como el tipo de procesamiento que realiza con los datos intercambiados con el servicio RESTful, o si se trata de un sensor, actuador o procesador.

PASO 2: CARACTERIZAR LOS RECURSOS REQUERIDOS POR LOS CLIENTES MÁQUINA

Después de construir una imagen de los clientes máquina que podrían consumir el servicio, el paso a seguir es caracterizar los recursos que ya estén publicados en la web o que se vayan a desarrollar. Básicamente se pide determinar tres cosas:

- La naturaleza del recurso, es decir para qué podría servir. Por ejemplo un recurso DATA que permite obtener, almacenar o borrar información, o un servicio ALARM para enviar mensajes a operarios humanos.
- Los métodos HTTP que implementa el recurso. Podría o no implementar los verbos GET, POST, PUT, DELETE, OPTIONS, entre otros.
- Los niveles de autenticación o autorización requeridos para que el cliente máquina pueda ejecutar los métodos implementados por el recurso.

PASO 3: DESCRIBIR LA INTERFACE DEL RECURSO

En este paso se describe la interface del servicio. Esta interface está determinada por el protocolo HTTP, el cual define la sintaxis de la comunicación y oculta la implementación del servicio ofreciendo una Interface Uniforme al cliente, la cual es independiente de la naturaleza del recurso. Cuando se describe la interface se deben definir ciertos parámetros:

- La versión del protocolo HTTP. La más reciente es “HTTP/1.1”.
- La estructura de la URI (Uniform Resource Identifiers, identificador uniforme del recurso), que junto con la URL del host que almacena el servicio, sirve para identificar unívocamente la ubicación del recurso.
- Caracter Sets o tabla de conversión de caracteres, por ejemplo UTF-8.
- Content Coding. La codificación del contenido permite realizar transformaciones sobre los datos intercambiados entre cliente y servicio sin cambiar el formato del documento.
- HTTP usa Internet Media-Type en el Content-Type (desde el cliente) y en Accept (desde el servidor) para negociar los tipos de datos intercambiados. Por ejemplo, texto plano, HTML, XML o JSON.

Se recomienda llegar en este paso de la metodología hasta la estructura del REQUEST (mensaje enviado por el cliente máquina) y la estructura del RESPONSE (mensaje enviado por el servicio que procesa el recurso).

PASO 4: DESCRIBIR EL SERVICIO RESTful

Los pasos anteriores han ayudado a identificar claramente la interface del recurso que es visible para el cliente, y teniendo claro la estructura de los mensajes REQUEST y RESPONSE. Es el momento de describir el servicio RESTful. Se recomienda elaborar un documento para describir sólo recursos de naturaleza similar ofrecidos por un servicio en particular. Recursos de diferente naturaleza se deberían describir en documentos independientes, para facilitar la composición de servicios.

Esta descripción se realiza usando un formato XML para que sea fácil de leer también por clientes humanos. La estructura de esta descripción se delimita por la etiqueta <description APIRest= “name”></description> que además identifica la API a la que pertenece el servicio. Los elementos contenidos en esta descripción son:

- Entre las etiquetas <service serviceID= “id”> </service> se ofrece información general del servicio, como el desarrollador, niveles de autenticación y autorización, pago o gratuito, fecha de desarrollo, entre otros datos.
- Entre las etiquetas <resource resourceID= “id”> </resource> se especifica el host y la URI del recurso, los métodos (GET, POST, DELETE, PUT, OPTIONS) implementados. Si el método está implementado se especifican los media-Type disponibles, los parámetros que debe enviar el cliente y los parámetros de recibe del servicio. Así como nivel de autorización para invocar el método.
- Otro elemento muy importante de la descripción del recurso se delimita por las etiquetas <ranking> </ranking>. Y es un número que los clientes máquina pueden recomendar incrementar o decrementar en función de su satisfacción con el uso del recurso.
- Entre etiquetas <otherServices> </otherServices> se recomienda otros recursos propios o de terceros que podrían ser de utilidad para el cliente. La información de estos recursos contiene la función que desempeña, el host y la URI. Este campo en la descripción es fundamental para que los clientes máquina compongan (combinen) varios servicios web sin intervención humana.

PASO 5: CONVERTIR LA DESCRIPCIÓN EN UN RECURSO

La novedad de esta metodología se basa en ofrecer la descripción del servicio como un recurso más. La descripción del servicio será obtenida por el cliente usando el método GET seguido de una URI con la forma URL+/nombre del recurso/description.

El cliente podrá modificar la estructura de la descripción con el método POST de dos maneras:

- En la carga del método POST (payload) puede enviar el parámetro RANKING seguido de uno de tres posibles valores: +1, para recomendar el recurso, -1 para desestimar el uso del recurso en caso de no haber obtenido el resultado esperado, o 0 para expresar una satisfacción neutra. En caso de recibir un número diferente, el desarrollador de la descripción podría interpretar números positivos como un +1 y número negativos como un -1.
- El cliente puede usar el método POST para enviar los parámetros FUNCTION, HOST y URI correspondientes a otros servicios con funciones similares para que el servidor modifique la descripción del servicio y los agregue entre las etiquetas <otherServices></otherServices> para promover la dinámica del consumo de servicios RESTful. Otra información que se podría incluir en la descripción de otros recursos es la necesidad o no de autorización, autenticación y pago.

PASO 6. PROGRAMAR UN CLIENTE DE PRUEBA QUE CONSUMA EL RECURSO DEL SERVICIO

Antes de publicar la descripción del servicio conviene verificar que un cliente máquina podrá descubrir y consumir el recurso, así como componer y promover la combinación de varios servicios. Para ello, se recomienda la programación de un cliente web que a partir de la descripción sea capaz de consumir el servicio. En su defecto, a falta de un cliente de prueba se pueden usar aplicaciones en línea para probar la eficiencia de la descripción.

PASO 7. PUBLICAR LA DESCRIPCIÓN DEL SERVICIO

El último paso es publicar la descripción en el servidor que contiene el servicio o en un servidor que almacene otras descripciones. Siempre que sea posible se debería implementar herramientas que ayuden a los robots de los buscadores.

6. DESARROLLO DEL FRAMEWORK PARA LA METODOLOGÍA PROPUESTA

El framework para automatizar la descripción de servicios RESTful se ha programado en HTML5, CSS, JavaScript y PHP. Esta aplicación se ha publicado en el sitio http://diysmartdevices.com/framework/index.html, donde el lector puede usarla para generar la descripción de un servicio propio.

En HTML se ha programado el formulario que reúne la información contenida en la descripción del servicio web. La figura 2 muestra parte de la interface de usuario.

Figura 2. Interface de usuario del Framework

Con JavaScript se recuperan los datos del formulario para generar el documento XML con la descripción del servicio web. Esta descripción se convierte en un recurso web para que los clientes puedan acceder a ella a través de los métodos HTTP. Lo anterior implica que el archivo XML debe ir acompañado de un archivo PHP que atienda las solicitudes GET y POST realizadas desde el cliente hacia el recurso de descripción.

El framework genera los siguientes archivos:

- **Descripción XML** del servicio.
- **Archivo HTACCESS** que redirecciona las solicitudes del recurso hacia el archivo PHP que atiende los métodos HTTP.
- **REST.inc** con las funciones auxiliares que permiten tratar la descripción como un recurso web.

- **API.php** con las funciones para manipular la descripción XML entrega el recurso como respuesta a la solicitud GET, o modificando el Ranking y los servicios referenciados cuando el cliente invoca el método POST.

La figura 3 muestra el contenido de la carpeta generada por JavaScript con el nombre asignado al recurso por parte del usuario. En ésta se almacena el documento XML y los archivos PHP, los cuales son iguales para cualquier “descripción”. Cuando el usuario da clic en el botón “Generar Descripción” se comprime la carpeta usando la librería JSZIP y se descarga en el disco local.

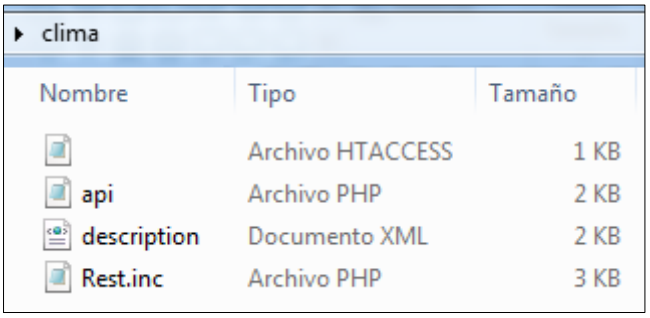

Figura 3. Contenido de la carpeta generada por el framework

7. EXPERIMENTO PARA VERIFICAR LA SOLUCIÓN

7.1. DESCRIPCIÓN DEL EXPERIMENTO

El experimento para verificar la validez de la solución propuesta se basa en un sistema de Internet de las Cosas, donde una estación meteorológica accede a la descripción de algunos servicios para encontrar el recurso que les permita almacenar datos, recuperar información y enviar alarmas a operarios humanos.

Se va a programar en PHP dos servicios RESTful, uno de manipulación de datos y otro de alarmas, ambos con recursos que podrán ser consumidos por dispositivos embebidos que realizan control de variables agronómicas en invernadero. También se van a describir dos servicios de terceros, específicamente de http://openweathermap.org

Se usa un sistema embebido Arduino MEGA ADK con sensores de humedad contenida en el suelo, humedad relativa del ambiente, temperatura e iluminación. Con acceso a Internet a través de tarjeta Wi-fi. El dispositivo conoce desde el principio la semántica de la descripción de los servicios RESTful, el token para acceder a los recursos de terceros y la URI de la descripción de un servicio.

Siguiendo la metodología propuesta se va a construir la descripción de cada servicio usado en el experimento. Esta descripción incluirá:

- La interface uniforme del servicio, incluida su URI.
- Funcionalidad del recurso, métodos que implementa.
- Parámetros de entrada.
- Parámetros de salida.
- Enlaces a otros servicios.
- Ranking de consumo de este servicio.
- Mecanismo para que clientes máquina modifiquen los enlaces a otros servicios y el ranking de consumo.
- Datos no funcionales: garantía de calidad del servicio, políticas de seguridad y de privacidad.

El cliente invoca la descripción del servicio y verifica si éste contiene un recurso para guardar datos. Si no lo encuentra visita la descripción de los servicios recomendados, y dentro de ellos busca nuevos recomendados hasta hallar el tipo de servicio que busca. Después de usar el servicio el cliente-máquina recupera el dato guardado para comprobar que se ha almacenado correctamente, luego afecta la descripción del

mismo para ofrecer una calificación (ranking de consumo) y recomendar otros servicios.

Posteriormente, el cliente-máquina busca un servicio para enviar mensajes de alarma a un operario humano cuando los datos medidos superan un intervalo establecido. En el caso de estudio, se ofrece un servicio que envían correos electrónicos. El cliente-máquina debe conocer con anticipación la cuenta de Twitter o de correo destino.

A medida que realiza las tareas, el cliente-máquina es monitoreado desde un computador a través de una interfaz serial, donde se crea un log para recopilar la información necesaria para determinar si la metodología ha tenido éxito.

7.2. CRITERIOS PARA DETERMINAR EL ÉXITO DEL EXPERIMENTO

Se considera que el experimento ha tenido éxito en plantear una metodología para describir servicios RESTful, si el cliente máquina consigue:

- Almacenar datos de variables agronómicas en la nube.
- Enviar mensajes de alarma a través de Twitter, correo electrónico o SMS.
- Recomienda o desestima el uso de un servicio web afectando su ranking desde la descripción.
- A través del método POST propone modificar la descripción para agregar la referencia a otro servicio.
- El cliente-maquina consigue las tres condiciones anteriores en menos de 24 horas.

7.3. APLICACIÓN DE LA METODOLOGÍA EN EL EXPERIMENTO

Se aplicará la metodología paso a paso en el experimento para que los resultados sirvan para juzgar la validez de la solución propuesta.

PASO 1: IDENTIFICAR LAS CARACTERÍSTICAS DEL CLIENTE MÁQUINA

El cliente máquina usado en el experimento se caracteriza por:

- Tarjeta de desarrollo: Arduino MEGA ADK.
- Sensores: humedad y temperatura (DHT11), iluminación (LDR), humedad contenida en el suelo.
- Capacidad de procesamiento: bus de 8 bits.
- Capacidad de almacenamiento: 1Mbits.
- Comunicación: Wi-Fi.
- Generación de datos: las variables agronómicas se actualizan cada hora.

PASO 2: CARACTERIZAR LOS RECURSOS REQUERIDOS POR LOS CLIENTES MÁQUINA

El clientes máquina usado en el experimento requiere almacenar variables agronómicas, recuperar variables agronómicas y enviar mensajes de alarma. La tabla 2 describe los recursos que se usarán en el experimento.

Tabla 2. Descripción de los servicios usados en el experimento.

RECURSO	MÉTODOS PERMITIDOS	DESCRIPCIÓN
estacion	GET, POST	Permite almacenar o recuperar las variables: 'station', 'latitude', 'longitude', 'altitude', 'date_time', 'wind_dir', 'wind_speed', 'temperature', 'humidity', 'pressure', 'water_content', 'rain_1h', 'rain_24h', 'rain_today', 'snow', 'lumens', 'dewpoint', 'uv'. No requiere credenciales para consumir el servicio. Permite recuperar datos con el método GET y guardar registros con el método POST.
alarma	POST	No requiere credenciales para usar el servicio. Con POST se envían los datos del destino del mensaje de alarma.
data	GET	Servicio de terceros (api.openweathermap.org). Requiere credenciales de acceso que se obtienen de la página del propietario. Permite leer datos de estaciones meteorológicas ubicadas en diferentes ciudades del mundo. Las variables que envía son: 'station', 'latitude', 'longitude', 'altitude', 'date_time', 'wind_dir', 'wind_speed', 'temperature', 'humidity', 'pressure', 'water_content', 'rain_1h', 'rain_24h', 'rain_today', 'snow', 'lumens', 'dewpoint', 'uv'.
station	POST	Permite enviar datos desde una estación meteorológica, siempre que se tengan las credenciales de acceso. Los datos que recibe son. 'station', 'latitude', 'longitude', 'altitude', 'date_time', 'wind_dir', 'wind_speed', 'temperature', 'humidity', 'pressure', 'water_content', 'rain_1h', 'rain_24h', 'rain_today', 'snow', 'lumens', 'dewpoint', 'uv'.

PASO 3: DESCRIBIR LA INTERFACE DEL RECURSO

Los parámetros de la solicitud (request) y respuesta (response) de los recursos del experimento se describen en la tabla 3.

Tabla 3. Descripción de la interface de los recursos

RECURSO	POST	GET
estacion	**URL:** http://diysmartdevices.com **URI:** /rest/estacion **Datos de la Solicitud:** 'station', 'latitude', 'longitude', 'altitude', 'date_time', 'wind_dir', 'wind_speed', 'temperature', 'humidity', 'pressure', 'water_content', 'rain_1h', 'rain_24h', 'rain_today', 'snow', 'lumens', 'dewpoint', 'uv'. **Datos de la respuesta:** 'id', 'codigo'. **Formatos:** application/x-www-form-urlencoded Json **Credenciales:** No requiere credenciales	**URL:** http://diysmartdevices.com **URI:** /rest/estacion?station={station name} **Datos de la respuesta:** 'station', 'latitude', 'longitude', 'altitude', 'date_time', 'wind_dir', 'wind_speed', 'temperature', 'humidity', 'pressure', 'water_content', 'rain_1h', 'rain_24h', 'rain_today', 'snow', 'lumens', 'dewpoint', 'uv'. **Formatos:** Json.
alarma	**URL:** http://diysmartdevices.com **URI:** /rest/alarma **Datos de la Solicitud:** 'receiver', 'message', **Datos de la respuesta:** 'codigo'. **Formatos:** application/x-www-form-urlencoded Json **Credenciales:** No requiere credenciales	
data		**URL:** http://api.openweathermap.org **URI:** /data/2.5/weather?APIid={APIid}&q={city name} **Datos de la respuesta:**

		'station', 'latitude', 'longitude', 'altitude', 'date_time', 'wind_dir', 'wind_speed', 'temperature', 'humidity', 'pressure', 'water_content', 'rain_1h', 'rain_24h', 'rain_today', 'snow', 'lumens', 'dewpoint', 'uv'. **Formatos:** Json, xml, html **Credenciales:** API ID obtenida cuando se crea una cuenta en http://openweathermap.org
station	**URL:** http://api.openweathermap.org **URI:** /data/post **Datos de la Solicitud:** 'station', 'latitude', 'longitude', 'altitude', 'date_time', 'wind_dir', 'wind_speed', 'temperature', 'humidity', 'pressure', 'water_content', 'rain_1h', 'rain_24h', 'rain_today', 'snow', 'lumens', 'dewpoint', 'uv'. **Datos de la respuesta:** 'id', 'codigo'. **Formatos:** application/x-www-form-urlencoded Json, xml, html. **Credenciales:** User:password en base64	

PASO 4: DESCRIBIR EL SERVICIO RESTful

A partir de los datos obtenidos en los pasos anteriores de la metodología se construye un archivo xml con la descripción de cada uno de los servicios usados en el experimento. La tabla 4 muestra la descripción del servicio "estación".

Tabla 4. Descripción del recurso "estacion"

DESCRIPCIÓN XML	EXPLICACIÓN
<?xml version="1.0" encoding="UTF-8"?> <description APIRest="diysmartdevices"> <service serviceID="weather"> <resource resourceID="estacion"> <function>weather data</function>	Información general de la API, del servicio y del recurso.

<host>http://diysmartdevices.com</host> <uri>/rest/estacion</uri>	
<GET> <authentication>none</authentication> <request> <bystation>?station={station name}</bystation> <bydate>?station={station name}&from={mm-dd-yy} &to={mm-dd-yy}</bydate> <byresults>?station={station name} &limit={number of results}</byresults> </request> <response> <mediatype>text/json,text/xml</mediatype> <parameters> <station>station</station> <longitude>lon</longitude> <latitude>lat</latitude> <temperature>temp</temperature> <pressure>pressure</pressure> <humidity>humidity</humidity> <wind_speed>wind_speed</wind_speed> <wind_direction>wind_dir</wind_direction> <water_content>water_content</water_content> <rain_1hour>rain_1h</rain_1hour> <rain_24hour>rain_24h</rain_24hour> <rain_today>rain_today</rain_today> <snow>snow</snow> <lumens>lumens</lumens> <dewpoint>dewpoint</dewpoint> <uv>uv</uv> </parameters> </response> </GET>	Descripción del método HTTP GET usado para recuperar registros almacenados. Se describe la estructura de la solicitud (request) y la respuesta (response)
<POST> <request> <header> <authentication>none</authentication> <content-type>application/x-www-form-urlencoded, application/json</content-type> </header> <body> <parameters> <name_station type="required">station</name_station>	Descripción del método HTTP POST para almacenar datos entregados por la estación meteorológica.

<latitude type="optional">lat</latitude> <longitude type="optional">long</longitude> <wind_direction type="optional">wind_dir</wind_direction> <wind_speed type="optional">wind_speed</wind_speed> <temperature type="optional">temp</temperature> <humidity type="optional">humidity</humidity> <pressure type="optional">wind_speed</pressure> <rain_1hour type="optional">rain_1h</rain_1hour> <rain_24hour type="optional">rain_24h</rain_24hour> <snow type="optional">snow</snow> <lumens type="optional">lum</lumens> <altitude type="optional">alt</altitude> <radiation type="optional">radiation</radiation> <dewpoint type="optional">dewpoint</dewpoint> <uv type="optional">uv</uv> </parameters> </body> </request> <response> <mediatype>text/json</mediatype> <parameters> <message>msg</message> <codeHTTP>cod</codeHTTP> <id_register>id</id_register> </parameters> </response> </POST>	
</resource> <ranking>-8</ranking> <otherservices> <service serviceID="data"> <function>weather data</function> <methods>GET</methods> <host_description>http://diysmartdevices.com</host_description> <uri_description>/description/data</uri_description> </service> </otherservices> </service> </description>	En esta sección de la descripción del servicio se usa la etiqueta <ranking>-8</ranking> para indicar la aprobación del recurso por parte de otros usuarios. La etiqueta <service> </service> hace referencia a recursos de otros servicios que pueden ser de utilidad para el cliente máquina.

PASO 5: CONVERTIR LA DESCRIPCIÓN EN UN RECURSO

En PHP se programa un servicio RESTful que manipula los archivos XML de las descripciones como recursos. Estas descripciones pueden ser obtenidas por el cliente máquina usando el método GET y se pueden modificar usando el método POST para cambiar el valor del ranking o para agregar referencias a nuevos servicios web. La tabla 5 muestra el código básico (sin control de errores) para convertir la descripción del recurso "estación" en un servicio RESTful.

Tabla 5. Programa en PHP de la descripción como un recurso RESTful

CODIGO PHP	EXPLICACIÓN

private function estacion(){
// Devolver la descripción de un servicio
if($this->get_request_method() == "GET"){
$path="description/estacion.xml";
readfile($path);
return;
}
	Si la solicitud enviada usa el método GET el servidor devuelve el archivo XML sin realizar ninguna modificación sobre él.

// Actualizar ranking o referenciar otros servicios
elseif($this->get_request_method() == "POST"){
$ranking = (int)$this->_request['ranking'];
$serviceID_ = $this->_request['serviceID'];
$function_ = $this->_request['function'];
$methods = $this->_request['methods'];
$host_description = $this->_request['host_description'];
$uri_description = $this->_request['uri_description'];
// validar si se ha ingresado un ranking:
if(!empty($ranking)){
$description = new SimpleXMLElement
(file_get_contents('description/estacion.xml'));
// cambiar el valor del ranking:
if($ranking>0)
$description->service[0]->ranking =
(int)$description->service[0]->ranking+1;
else
$description->service[0]->ranking =
(int)$description->service[0]->ranking-1;
$description->asXML('description/estacion.xml');
$result = array('status' => "OK",
"msg" => "Se ha modificado el ranking");
$this->response($this->json($result), 200);
}
	Esta parte del código se usa para modificar el ranking del servicio.
	Si el cliente envía un valor positivo el ranking se incrementa, en caso contrario el ranking disminuye en una unidad.

if(!empty($host_description)||!empty($uri_description)){	

$description = new SimpleXMLElement
(file_get_contents('description/estacion.xml'));
$service=$description->service[0]->
otherservices->addChild('service');
$service->addAttribute('serviceID',$serviceID_);
$service->addChild('function',$function_);
$service->addChild('methods',$methods);
$service->addChild('host_description',$host_description);
$service->addChild('uri_description',$uri_description);
$description->asXML('description/estacion.xml');
$result =array('status' => "OK",
"msg" => "se ha modificado descripción del servicio");
$this->response($this->json($result), 200);
}
	También se programa la modificación del archivo XML para agregar la referencia a un nuevo servicio en la descripción.

PASO 6. PROGRAMAR UN CLIENTE DE PRUEBA QUE CONSUMA EL RECURSO DEL SERVICIO

Antes de programar el cliente máquina del experimento, el cual es una estación meteorológica programada en Arduino, se va a usar la herramienta "Advanced REST Client" de Google para verificar las descripciones como recursos de servicios web. La figura 4 muestra cómo se usa el método GET para obtener la descripción del servicio web.

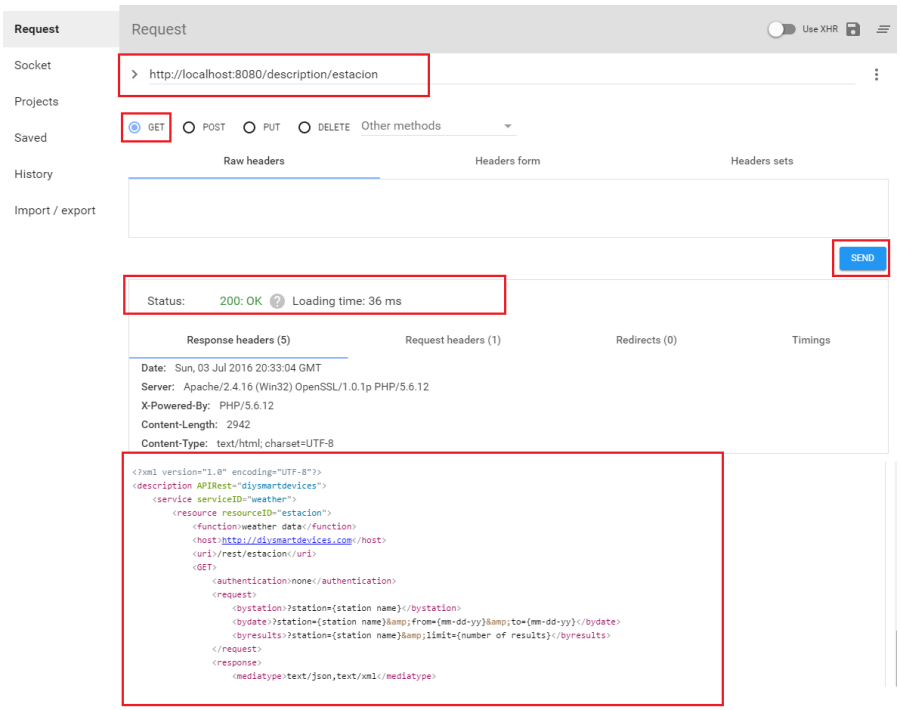

Figura 4. Descripción como recurso. Método GET.

La figura 5 muestra el cuerpo de la solicitud con el método POST para modificar el archivo XML de la descripción incrementando el ranking y recomendando otro servicio Web.

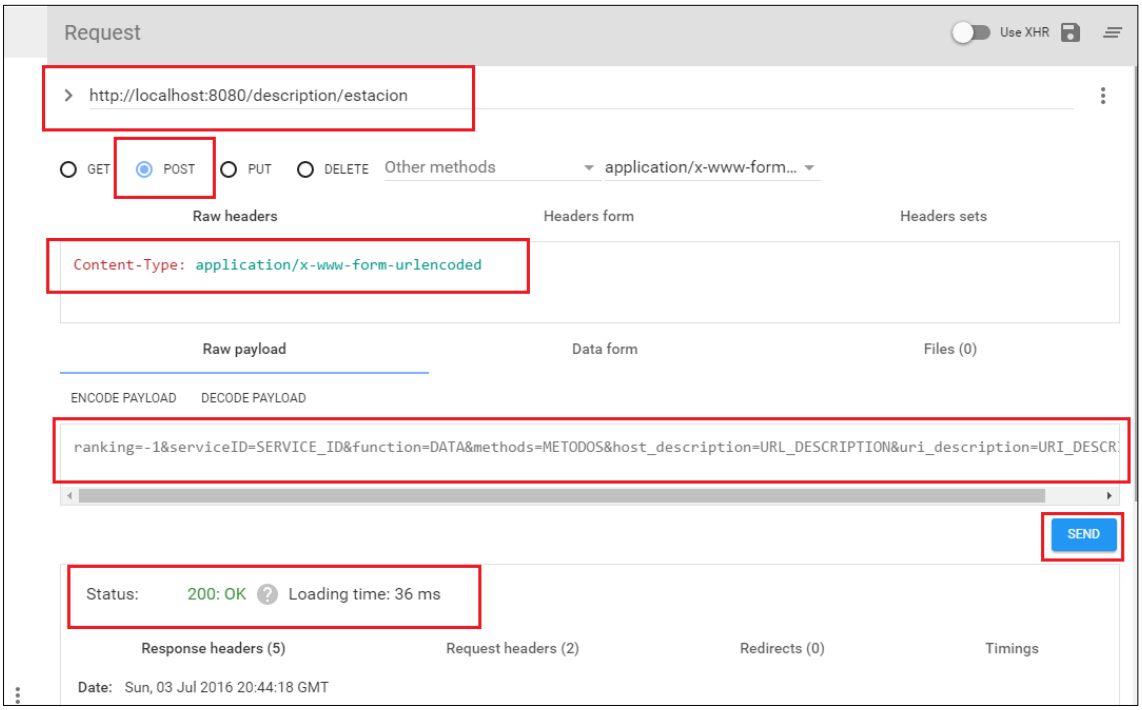

Figura 5. Modificación de la descripción del servicio

PASO 7. PUBLICAR LA DESCRIPCIÓN DEL SERVICIO

El siguiente paso es publicar todos los recursos de descripción en un servidor que pueda ser alcanzado por el cliente máquina.

8. ANÁLISIS DE DATOS E INTERPRETACIÓN DE RESULTADOS

Los resultados del experimento muestran que es posible para un cliente acceder a la descripción de un servicio para descubrir por sí mismo cómo debe usarlo. Para el cliente también es posible modificar la descripción del servicio para promover su uso por parte de otros clientes o para referenciar nuevos servicios web.

En el experimento se ha programado un dispositivo Arduino para consumir los servicios RESTful descritos. A través del puerto serial se ha monitoreado el desempeño del sistema. La figura 6 muestra como el cliente solicita la descripción de un servicio usando el método HTTP GET, el servidor responde con el documento XML que describe un servicio.

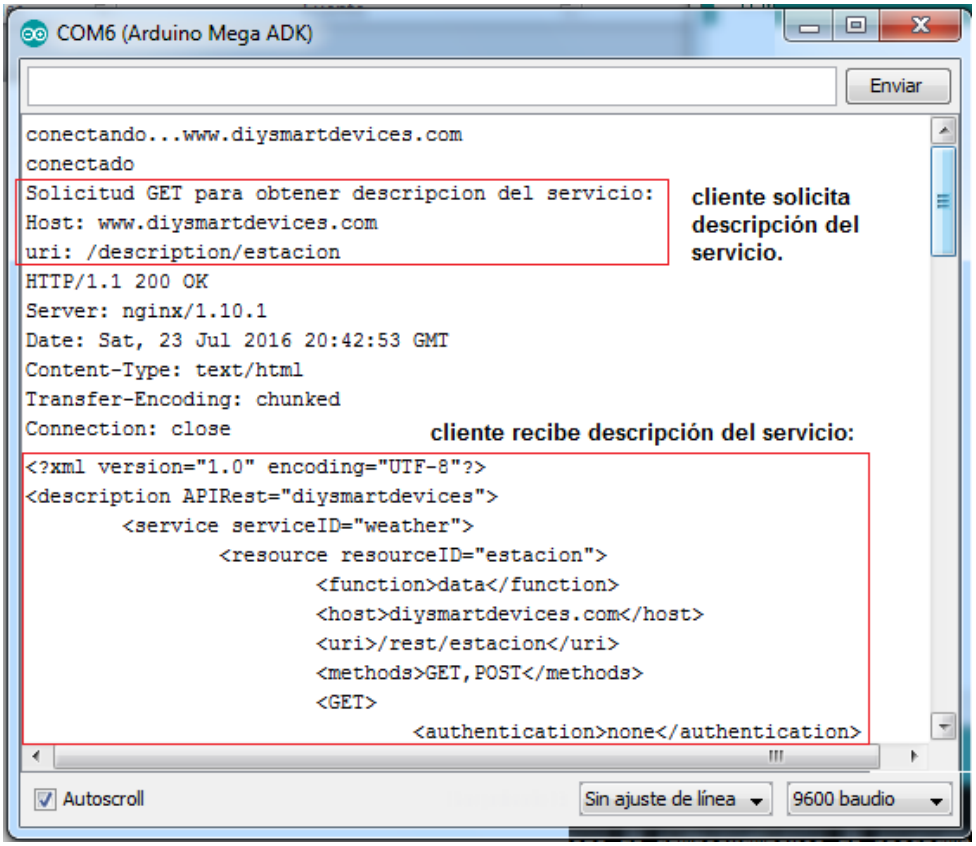

Figura 6. Cliente solicita descripción de servicio.

El cliente extrae la información del documento XML para identificar el tipo de servicio, métodos permitidos, y el formato para envío de parámetros. La figura 7 muestra la interacción entre el cliente Arduino y el servicio web.

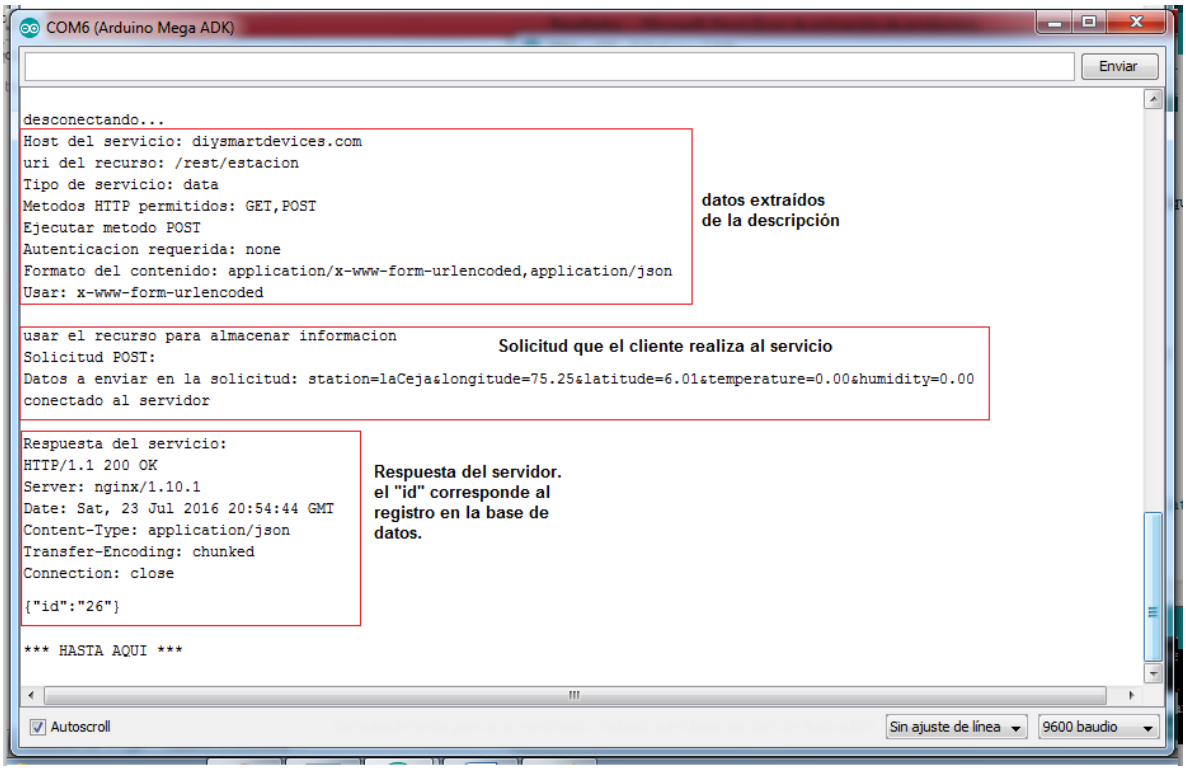

Figura 7. Cliente aprende cómo usar el servicio.

Una de las principales características de la metodología propuesta es que ofrece la descripción del servicio como un recurso web, el cual puede ser leído con el método GET y modificado con el método POST. El cliente puede incrementar o decrementar el ranking para promover su uso. También puede agregar referencias a otros servicios para facilitar la composición de soluciones más complejas. La figura 8 muestra el ranking y la referencia a un servicio antes de que el cliente modifique la descripción.

```
                        <mediatype>text/json</mediatype>
                        <parameters>
                                <message>msg</message>
                                <codeHTTP>cod</codeHTTP>
                                <id_register>id</id_register>
                        </parameters>
                        </response>
                </POST>
        </resource>
    <ranking>9</ranking>
    <otherservices>
        <service serviceID="data">
                <function>data</function>
                <methods>GET</methods>
                <host_description>diysmartdevices.com</host_description>
                <uri_description>/description/data</uri_description>
        </service>
    </otherservices>
</description>
```

Ranking en 9

un servicio recomendado

Figura 8. Ranking y referencia a otro servicio antes de ser modificado por el cliente

Mientras la figura 9 muestra la descripción del servicio después de ser modificada por el cliente, se puede observar que se ha incrementado el ranking y se hace incluye información de otro servicio.

```
        </resource>
        <ranking>10</ranking>                    Ranking actualizado: 10
        <otherservices>
                <service serviceID="data">
                        <function>data</function>
                        <methods>GET</methods>
                        <host_description>diysmartdevices.com</host_description>
                        <uri_description>/description/data</uri_description>
                </service>
        <service serviceID="WeatherData">
                        <function>DATA</function>   **Nueva referencia a servicio**
                        <methods>POST</methods>
                        <host_description>diysmartdevices.com</host_description>
                        <uri_description>description/station</uri_description>
                </service>
        </service>
</description>
```

Figura 9. Ranking y referencia después de ser modificado por el cliente

Inicialmente cada descripción incluye la referencia a otro servicio. La figura 10 muestra cuál servicio es incluido en cada descripción. Por ejemplo, la descripción del servicio "Estación" incluye información de la ubicación de la descripción del servicio "Data". Lo anterior implica que para hallar el servicio "Alarma" el cliente debe revisar la descripción de los demás servicios.

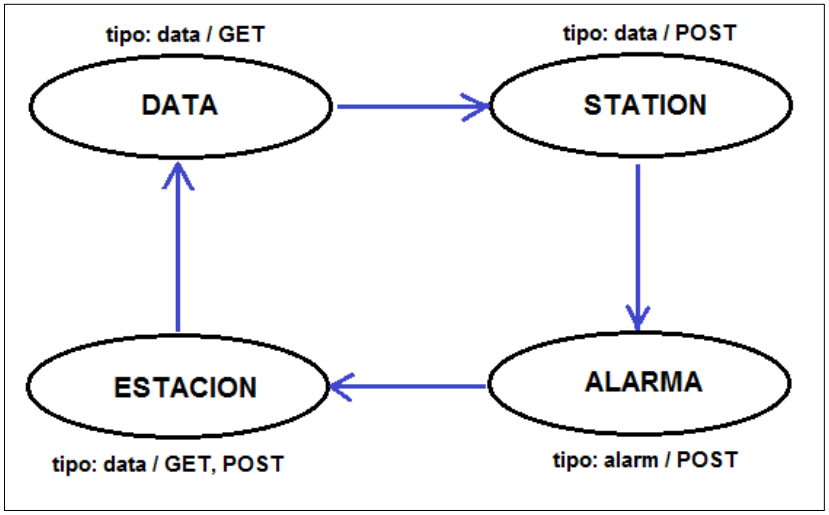

Figura 10. Referencias a otros servicios antes de actuar el cliente máquina

Después de realizar el experimento, las referencias a otros servicios se han ampliado. La figura 11 muestra que la descripción de "Estación" se ha modificado para incluir referencias a los servicios "Station" y "Alarma".

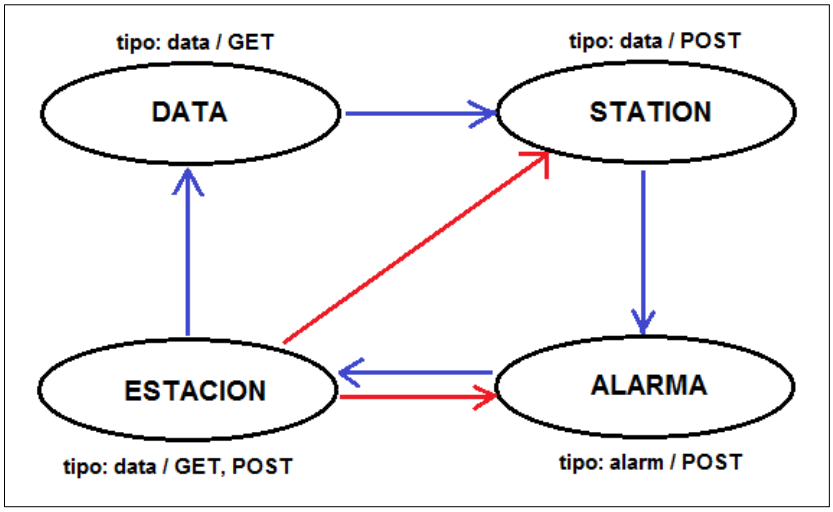

Figura 11. Referencias a otros servicios después de actuar el cliente máquina

Para enviar alarmas el cliente máquina visita la descripción de diferentes servicios hasta hallar uno tipo "Alarma". La figura 12 muestra la solicitud realizada por el dispositivo Arduino para enviar un mensaje de alarma.

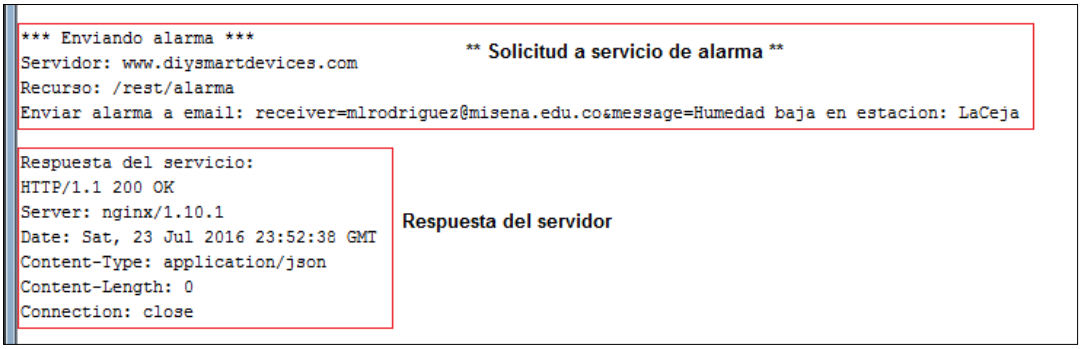

Figura 12. Interacción entre el cliente y el servicio de Alarma

La figura 13 muestra el correo (alarma) recibido por el operario del sistema.

Figura 13. Alarma recibida por el operario del sistema

9. CONCLUSIONES Y TRABAJO FUTURO

La tecnología actual permite a las máquinas interactuar de manera inteligente a través de la red, y los servicios RESTful ofrecen una manera liviana para acceder a los recursos disponibles. Siendo necesaria una descripción semántica que permita a los clientes máquina comprender el funcionamiento de los servicios de forma automática.

Los resultados de este proyecto muestran que es posible para los clientes máquina descubrir, invocar y combinar servicios RESTful que eran desconocidos para el desarrollador en el momento de programar el cliente.

Sin embargo, la limitada capacidad de procesamiento de los dispositivos embebidos usados en sistemas de Internet de las Cosas, señalan como un desafío el proveer la inteligencia suficiente para que éstos interactúen con un gran conjunto de descripciones de servicios RESTful. Incluso puede ser recomendable acceder a partes de la descripción sin invocar el documento XML completo, para facilitar el procesamiento del mismo.

Los resultados de este proyecto de investigación abren la posibilidad a trabajos futuros para desarrollar redes sociales para clientes máquina, donde interactúen para promover, descubrir, invocar y componer nuevos servicios web.

Un trabajo posterior acerca de la descripción semántica de servicios RESTful consumidos por clientes máquina, podría enfocarse en ofrecer a los motores de búsqueda una breve descripción semántica del servicio, para que no sea necesario que el cliente máquina conozca la dirección URL del directorio de servicios. Así el sistema embebido podría realizar una consulta en un buscador web, revisar los resultados de la búsqueda e identificar cuáles de ellos son servicios, y cuáles de estos servicios se ajustan a su ontología.

10. BIBLIOGRAFÍA

Bellido, J. A. (2011). Web Linking-based protocols for guiding RESTful M2M interaction. *In Current Trends in Web Engineering. Springer Berlin Heidelberg*, 74-85.

Czyszczoń, A. &. (2014). Automatic RESTful web service identification and information extraction. *Computer Networks*, 318-327.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Retrieved Abril 15, 2016, from https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Fredrich, T. (2012). *RESTful Service Best Practices. Recommendations for creating web services.* www.RestApiTutorial.com: Pearson eCollege.

IBM. (n.d.). *IBM developerWorks*. Retrieved Mayo 01, 2016, from Introducción a SOA y servicios web: http://www.ibm.com/developerworks/ssa/webservices/newto/service.html

Kerrigan, M. M. (2009). Modeling Semantic Web Services with the Web Service. *Journal of Network and Systems Management 17(3)*, 326-342.

Kopecky, J. G. (2008). hrests: An html microformat for describing restful web services. In Web Intelligence and Intelligent Agent Technology. *WI-IAT'08. IEEE/WIC/ACM International Conference* (pp. Vol.1. 619-625). IEEE Computer Society.

Kovatsch, M. H. (2015). Practical Semantics for the Internet of Things. *2015 5th International Conference on the Internet of Things (IoT)* (pp. 54-61). IEEE.

Lars Hagge, D. S. (2011). A Framework for Rapid Development. In E. Wilde, & C. P. Editors, *REST: from Research to Practice* (pp. 279-300). Springer.

Paul Adamczyk, P. H. (2011). REST andWeb Services: In Theory. In E. &. Wilde, *REST: from research to practice* (pp. 46-68). New York, USA: Springer Science & Business Media.

Robinson, I. (2011). RESTful domain application protocols. In *REST: from research to practice* (pp. 61-91). Nueva York, EU: Springer.

Salvadori, I. &. (2014). A Framework for Semantic Description of RESTful Web APIs. *2014 IEEE International Conference on Web Services*, 630-637.

Sanchez, R. V. (2014). RestML: Modeling RESTful Web Services. In E. &. Wilde, *REST: Advanced Research Topics and Practical Applications* (pp. 125-143). New York: Springer.

Verborgh, R. S. (2011). Efficient runtime service discovery and consumption with hyperlinked RESTdesc. ,. *In Next Generation Web Services Practices (NWeSP). 7th International Conference* (pp. 373-379). IEEE.

W3C. (2001, marzo 15). *Web Services Description Language (WSDL) 1.1*. Retrieved Mayo 05, 2016, from https://www.w3.org/TR/wsdl

W3C. (2004, Nov. 22). *OWL-S: Semantic Markup for Web Services*. Retrieved Mayo 02, 2016, from https://www.w3.org/Submission/OWL-S/

Wilde, E., & Pautasso, C. (2011). Introducción. In E. Wilde, & C. E. Pautasso, *REST: From Research to Practice* (pp. 14-21). Spriger.