control, (ii) ubicados en silos independientes entre ellos y (iii) gestionados por aplicaciones fragmentadas, sin integración entre ellas y solo intercomunicadas gracias a canales de comunicación específicos, generalmente propietarios.

El objetivo de las Smart Grids es el de proveer mejores servicios y características (también conocidas como funciones inteligentes), tanto para los consumidores como para los productores y prosumers. Además, la mayor utilización de generación distribuida y renovable de energía demanda cambios en el sistema de gestión de la electricidad. Se hace necesaria la mejora de los sistemas de automatización, inteligencia distribuida, minería de datos en tiempo real y gestión para mejorar las funciones de control de la red, reducir la configuración y reducir los tiempos de recuperación y auto cicatrización de los sistemas.

Las TIC de las Smart Grids se pueden considerar un caso particular del IoT. Uno de los objetivos y retos más relevantes de las Smart Grids es la integración de datos y comunicaciones entre una red de diferentes tipos de sensores y actuadores que las integran (sensores con y sin cables, medidores inteligentes o *smart meters*, generadores distribuidos, etc.) que, a su vez, deben ser capaces de cooperar y coordinarse entre ellos para que puedan ejecutar cualquier función deseada. Esto es, el objetivo de las Smart Grids es el de crear un único sistema de integración para que las diferentes aplicaciones puedan aprovecharse de las mismas ventajas [4]. En [5] se exponen esta y otras problemáticas inherentes al IoT y, por ende, a las Smart Grids.

Para dar solución a la integración de los dispositivos heterogéneos se ha propuesto el uso del concepto de la Web of Things [1] para acceder a los múltiples dispositivos usando una misma interfaz facilitada por las tecnologías web, implicando uniformidad en los protocolos de comunicación (HTTP y WebSockets), en el modelo de representación de los datos y en el descubrimiento de dispositivos (Web Thing Model [6] y Web Semántica [7]). Este concepto es, hasta el momento, de difícil aplicación en escenarios reales que involucren la gestión de la energía debido a la posibilidad de crear nuevas vulnerabilidades de seguridad, la falta de dispositivos que implementen estándares de la Web of Things y la oposición de la industria de la energía a incluir módulos o dispositivos externos en sus sistemas propietarios.

El objetivo del trabajo que se presenta en este artículo es el de crear una arquitectura basada en el paradigma del IoT para gestionar las necesidades de almacenamiento y de comunicación de las Smart Grids y a la vez enlazar las Smart Grids con el usuario final a través de las metodologías de la Web of Things. Para ello, se establece una interfaz bidireccional H2M (*human-to-machine*) inspirada por la WoT que permite un control ubicuo de los sistemas de energía, la Web of Energy [2].

De este modo, la WoE podría representar una oportunidad para que las eléctricas puedan disponer de representaciones virtuales de sus dispositivos más flexibles (basadas en software, actualizable, configurable y con posibilidad de desplegar nuevas aplicaciones encima de ellos), posibilitando una la distribución y gestión de bajo coste de la red eléctrica [8]. La WoE facilita la compartición de datos de diferentes dispositivos (puntos de recarga de vehículos eléctricos, *smart metering* o monitorización de subestaciones) con terceros. Además, el uso de tecnologías web permite la creación de herramientas de visualización multiplataforma y sin coste de instalación, pudiendo ofrecer interfaces gráficas simples y usables para una mayor adopción para los DSO (*Distribution Systems Operator*) o cualquier usuario interesado en la consumición o producción de energía (*prosumer*). Así pues, podemos identificar distintos campos relacionados con la gestión de la energía en los que la WoE sería de gran utilidad:

- Acceso remoto desde subestaciones a los servidores centrales
- Monitorización y control de DER (*Distributed Energy Resources*)
- Distribución de SCADA a subestaciones secundarias
- Control de EVSE (*Electrical Vehicle Supply Equipment*)

II. Web of Things

Aunque las predicciones iniciales de 1 billón de dispositivos conectados a Internet en 2015 [9] fueron rápidamente rebajadas a 26.000 millones en 2020 [10] y 20.800 millones en 2020 [11], es evidente que el número de dispositivos conectados a Internet incrementa día a día. Cómo acceder a todos estos sensores y actuadores a través de una interfaz uniforme es indudablemente uno de los mayores retos del IoT. Actualmente, el IoT está dividido en silos, esto es, existen soluciones propietarias que ayudan a la integración de un conjunto determinado de dispositivos, pero se alejan de la integración global prevista para el IoT. Es por este motivo se propone el uso de tecnologías web ya existentes para la integración global de los dispositivos. Los prerrequisitos básicos para la habilitación de los dispositivos del IoT en la web son dos: (1) capacidad mínima de procesamiento de datos y (2) conectividad a la red (no es necesario que se conecten directamente a Internet).

A. De la heterogeneidad a la homogeneidad

El modelo que propone la Web of Things [1] pretende dar solución al reto de la heterogeneidad de los dispositivos que componen el IoT. Esto lo consigue, en gran parte, generando traductores o *mappings* entre el lenguaje hablado por cada dispositivo (protocolo de comunicación y formato de los datos) y el lenguaje que podemos considerar "universal" por su uso extendido: las tecnologías web. En concreto disponemos del protocolo HTTP y las APIs RESTful [12]. Estas traducciones habilitan a los dispositivos a hablar un mismo lenguaje, lo que permite que puedan ser accesibles de forma homogénea por actores humanos o

computerizados, como otros dispositivos. La acción a realizar sobre estos dispositivos puede ser tanto para actuar como para captar información.

Si nos centramos en el flujo de datos, este se compone de dos estados que se pueden entender como un ciclo: de heterogeneidad a homogeneidad y de homogeneidad a heterogeneidad.

- De heterogeneidad a homogeneidad. Un dispositivo envía datos que ha captado con formato y protocolo específicos a través de un traductor WoT, que traduce los datos a un formato común y el protocolo a HTTP.
- De homogeneidad a heterogeneidad. El actor recibe estos datos y decide actuar sobre el dispositivo, por ejemplo, cambiando su configuración. Entonces envía una instrucción mediante el protocolo HTTP y un formato común al traductor WoT que, a su vez, traduce el protocolo y el formato de los datos al protocolo y formato específicos.

III. La arquitectura de la Web of Things

El concepto de Web of Things se desarrolla principalmente en los trabajos de Dominique Guinard [1] y Vlad Trifa [13], donde se presentan la WoT y los métodos de habilitación de dispositivos del IoT a la WoT respectivamente. En [1] se propone organizar la WoT en cuatro capas, cada una con una función específica. Sin embargo, estas capas no siguen un modelo de aislamiento y encapsulación entre capas no contiguas como ocurre con el modelo OSI o TCP/IP, sino que, por el contrario, las aplicaciones se pueden construir encima de cada una de ellas (Fig. 1), ya que todas ellas forman parte del nivel de aplicación de los dos modelos de red mencionados anteriormente. A continuación, se presentan las diferentes capas de la WoT propuestas en [1].

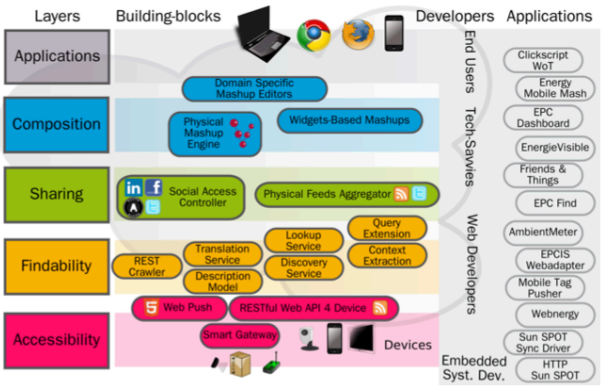

Fig. 1 Modelo de capas propuesto en [1]. Imagen extraída de [1]

Las funciones de cada una de estas capas son:

(i) Accesibilidad: Habilita un acceso consistente a todo tipo de dispositivos del IoT exponiendo sus funcionalidades mediante una API RESTful HTTP.

(ii) Localización: Facilita el descubrimiento de las representaciones de los diferentes dispositivos, modelando de manera uniforme el método de acceso a estas (a través ya del protocolo HTTP) y estableciendo relaciones entre ellas en el momento de su representación.

(iii) Participación: Se encarga de preservar la privacidad entre las representaciones de los dispositivos y de gestionar la autenticación y autorización del acceso a las representaciones por parte de otros actores distintos al propietario del dispositivo.

(iv) Composición: Su función es la de habilitar la integración entre las distintas representaciones de los dispositivos que, en última instancia, permite la integración entre las diferentes funcionalidades de los dispositivos físicos.

A. Capa de Accesibilidad

La capa de accesibilidad actúa de interfaz entre el IoT y las tecnologías web. Por lo tanto, es la más próxima a la heterogeneidad de protocolos del IoT, tales como MQTT [14] o CoAP [15], entre otros. En esta capa se encuentran soluciones en forma de proxy o puente entre los protocolos del IoT y HTTP.

Como metodologías básicas se pueden considerar la incrustación de servidores web a los dispositivos o la creación de *gateways* [16] con las funciones de agregación y/o proxy entre protocolos del IoT y protocolos web. Se encuentran en esta capa también soluciones cloud, generalmente transversales a más capas de la WoT, como ThingWorx [17], Watson IoT Platform de IBM [18], Octoblu [19] o EVRYTHNG [20], entre otros. Estas soluciones *cloud* ofrecen *gateways* de traducción IoT – WoT.

B. Capa de Localización

Esta capa la componen aquellas tecnologías que permiten explorar y encontrar los distintos dispositivos expuestos a la WoT. Agrupa aquellas tecnologías que permiten la publicación de datos estructurados e interconectados. En la publicación de las representaciones de los dispositivos se aplica el concepto de REST [12] y Linked-Data [21] para interconectar distintas representaciones (generalmente en forma de URLs – *Uniform Resource Locators*) .Las tecnologías de la Web Semántica [7] tales como RDF [22], RDFa [23] o OWL [24] intervienen para dotar de significado semántico tanto a las representaciones como a las conexiones con otras representaciones, permitiendo, por ejemplo, su exposición a los motores de búsqueda como Google.

C. Capa de Participación

Esta capa agrupa todos aquellos métodos que permiten autenticar y autorizar a diferentes actores a realizar una acción sobre la representación virtual del dispositivo y, en última instancia, sobre el dispositivo físico. Agrupa métodos de autenticación y autorización

simples desde la autenticación con usuario y contraseña hasta el uso de protocolos más avanzados como claves API u OAuth.

	HTTP	WebSocket	CoAP	MQTT	XMPP	AMQP	DDS
Topología	*Req/Resp*	*Two-way, realtime*	*Req/Resp*	*Pub/Sub*	*Pub/Sub yReq/Resp*	*Pub/Sub*	*Pub/Sub*
Arquitectura	*P2P*	*P2P*	*P2P*	*Broker*	*P2P*	*P2P o Broker*	*Global Data Space*
Nivel de Transporte	*TCP*	*TCP*	*UDP*	*TCP*	*TCP*	*TCP*	*TCP/UDP*
Encriptación	*TLS*	*TLS*	*DTLS*	*TLS*	*TLS*	*TLS*	*TLS/DTLS*
Autenticación	*TLS*	*TLS*	*DTLS*	*User/Pass*	*TLS*	*SASL*	*TLS/DTLS*
QoS	-	-	*Confirmable*	*3*	-	*3*	*23*

Tabla 1. Protocolos del WoT/IoT

En [25] se propone un middleware, el *Social Access Controller* que, combinando las APIs OAuth de las diferentes redes sociales y el acceso por credenciales simples a los dispositivos (usuario y contraseña p.ej.) permite (1) preservar la privacidad de los dispositivos físicos, (2) aprovechar la estructura y la función de las redes sociales para autenticar a los diferentes actores potencialmente interesados en participar en las acciones que se puedan realizar sobre la representación del dispositivo, (3) integrar las representaciones de los dispositivos en las redes sociales, creando un *Social Web of Things* y (4) posibilitar la publicación de agregaciones de datos mediante protocolos de sindicación de datos como ATOM [26].

D. Capa de Composición

Esta capa permite la composición de las diferentes funcionalidades expuestas por las representaciones de los dispositivos. Teniendo en cuenta que cada representación es accesible mediante el protocolo HTTP, es sencillo elaborar un *script* para que los diferentes dispositivos actúen de forma coordinada. La elaboración de esta idea consiste en proporcionar al usuario web (humano) una interfaz visual para componer mediante relaciones diferentes dispositivos. Soluciones como ThingWorx Composer [27], NODE-Red [28] de IBM o Octoblu [19] ejemplifican la composición de dispositivos físicos (*physical mashups).*

IV. Protocolos de la WoT y la WoE

El elemento principal de la propuesta del Web of Things es el uso de tecnologías web para la transmisión de información. En lo que se refiere a protocolos web, se encuentran el protocolo HTTP(S) y el protocolo WebSocket (WS) o WebSocket Secure (WSS). HTTP es un protocolo de petición/respuesta mientras que WebSocket permite una conexión bidireccional entre cliente y servidor.

No obstante, muchos dispositivos no disponen de las características necesarias para conectarse directamente a la web mediante HTTP o WebSocket. Para poder conectarse a la web, necesitan hacer uso de *gateways* que se encarguen de traducir la heterogeneidad de protocolos del IoT a la homogeneidad de la WoT. Para ello se necesitan establecer puentes de traducción o mapeo de protocolos IoT a WoT y viceversa.

Por ejemplo, tanto CoAP como MQTT son dos protocolos en auge[1] para el IoT, entre otros. CoAP, estándar abierto, fue diseñado específicamente para el IoT y para ser directamente compatible con HTTP [29]. Es un protocolo basado en petición/respuesta mediante paquetes UDP y sigue los mismos esquemas que HTTP, permitiendo crear recursos REST. El protocolo MQTT, diseñado por IBM, es ahora también de estándar abierto, pero sigue un paradigma publicador/subscriptor sobre TCP, por lo que se hace más complicado establecer un puente de traducción entre HTTP-REST y MQTT [30].

En la Tabla 1 se pueden apreciar las diferencias y similitudes entre los protocolos CoAP, HTTP, MQTT y WebSocket. Como se ha comentado, la traducción entre CoAP y HTTP es directa pues la topología de ambos es petición/respuesta (Req/Resp). En cambio, la topología entre HTTP y MQTT es distinta por lo que la traducción entre estos dos protocolos es más difícil de conseguir. Otro ejemplo de traducción (casi) directa se da entre WebSocket y MQTT, ya que la comunicación bidireccional de WebSocket se puede considerar una particularidad del protocolo publicador/subscriptor (Pub/Sub) de MQTT. Así pues, HTTP y WebSocket son las dos tecnologías web con topologías distintas que permiten la traducción con protocolos del IoT con topologías petición/respuesta y bidireccionales respectivamente. En la Tabla 1 se presentan también otros protocolos que pueden formar parte del IoT, los cuales presentan más similitudes con HTTP o WebSocket según su topología del mismo modo que ocurre con CoAP y MQTT.

Además de los dos protocolos ya presentados en la Tabla 1 y de muchos otros tantos propietarios como no propietarios para el IoT, a medida que nos adentramos en organizaciones cuya infraestructura es más antigua, surgen otros protocolos mucho más adaptados a las necesidades específicas de estas y, a su vez, más difíciles de adaptar para la WoT debido a que (1) su especificidad

[1]https://trends.google.es/trends/explore?q=MQTT,CoAP,XMPP,AMQP. A fecha de abril 2017, las búsquedas de MQTT superan las búsquedas de CoAP, con 92 y 15 puntos sobre 100 respectivamente. Se puede observar como a partir de la segunda mitad de octubre de 2015, las búsquedas relacionadas con MQTT superan a las búsquedas relacionadas con XMPP.

reduce el interés de terceras personas a contribuir a la exposición de los dispositivos (o conjuntos de ellos) a la WoT y, por lo tanto, las empresas deben invertir más capital en la traducción y (2) en el caso de que sea de interés para la empresa exponer algunos de sus dispositivos, la WoT añade un *stack* de nuevas tecnologías, abriendo nuevos agujeros de seguridad a su sistema. Estos retos se agravan en la Web of Energy, pues los sistemas implicados abarcan una gran cantidad de dispositivos que usan protocolos muy específicos y adaptados a las necesidades de estos.

V. Complementando la Arquitectura WoT

El objetivo de esta sección es el de presentar los parámetros clave que creemos que debe cumplir una arquitectura WoT. Estos son:

(i) Aunque ciertos dispositivos del IoT pueden soportar *stacks* HTTP, existen muchos de ellos que por el hecho de disponer de recursos limitados solo pueden soportar protocolos más ligeros. Aunque idealmente un estrecho abanico de protocolos del IoT (p. ej. CoAP y MQTT) facilitaría la integración de los dispositivos a Internet y a la Web, un requisito básico e indispensable para que los dispositivos puedan formar parte de la Web es que se puedan conectar a Internet.

(ii) La arquitectura debe proveer abstracciones para que los desarrolladores puedan interactuar con los dispositivos independientemente del protocolo de comunicación, ya sea en dirección hacia la heterogeneidad de protocolos del IoT como en la dirección de la homogeneidad de protocolos de la WoT.

(iii) La arquitectura debe ser capaz de escalar horizontalmente y proporcionar auto cicatrización para sus sistemas. Debe permitir el despliegue bajo demanda de recursos.

(iv) Para la arquitectura, los dispositivos se convertirán en objetos virtuales o *Things*, aunque también se podrán crear objetos virtuales a partir de agregaciones de otros objetos virtuales.

(v) Cada objeto virtual será accesible desde una interfaz REST HTTP y, en caso de que las características del objeto virtual lo permitan, desde un enlace WebSocket.

(vi) Debe permitir ejecutar protocolos de autenticación y autorización hacia los distintos dispositivos implicados.

En nuestra propuesta, identificamos 5 capas para la creación de una arquitectura de la Web of Energy (Fig. 2) que, aunque tienen una profunda relación con las capas propuestas en [1], también se pueden identificar algunas diferencias, principalmente debido a que las capas que se presentan a continuación están más enfocadas al desarrollo de la arquitectura:

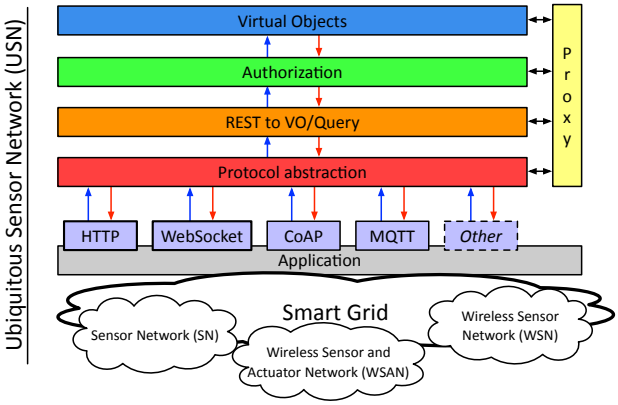

Fig. 2 Arquitectura para el Web of Energy

(i) Capa de Abstracción de Protocolos: El objetivo de esta capa es el de proveer una capa de abstracción para los desarrolladores para que puedan interactuar con los dispositivos físicos, tanto en el sentido de desarrolladores de funcionalidades de la arquitectura a nivel de capas más internas como de los desarrolladores de aplicaciones para la Smart Grid que, como ya se ha especificado, agrupa protocolos distintos. La idea no es solo exponer los dispositivos del IoT como recursos HTTP REST, sino proveer a los desarrolladores de mecanismos de abstracción tanto del protocolo HTTP como de protocolos del IoT. En este sentido, se mantiene la premisa expuesta en [1] de la habilitación de acceso a los dispositivos físicos a través de una interfaz HTTP.

(ii) REST a VO/Query: El propósito de esta capa es el de traducir las diferentes URIs generadas a partir de la información de los dispositivos físicos a acciones sobre los objetos virtuales o *Things*. De este modo, no se actúa directamente sobre los dispositivos físicos sino a través de los objetos virtuales. Esta capa también incluye una interfaz para realizar consultas más complejas sobre un *Thing* o las relaciones de diferentes *Things*.

(iii) Autorización: Esta capa es responsable de pedir y conceder acceso entre objetos virtuales o *Things*.

(iv) Espacio de Objetos Virtuales: Cuando los dispositivos físicos deban realizar acciones sobre otros dispositivos físicos enviarán la instrucción contra una representación virtual (objeto virtual o *Thing*). El objetivo de esta representación virtual es el de aumentar los recursos y funcionalidades de los dispositivos físicos. No podemos aportar una lista detallada de las funcionalidades añadidas, pues son específicas de cada solución. Aun así, nuestro objetivo es proporcionar un método (inyección de código) para que estas funcionalidades se puedan añadir de forma dinámica. Algunas funcionalidades

comunes son la de razonamiento (inteligencia artificial) o la de caché.

(v) Capa de Proxy: La arquitectura de la Web of Things presentada en [1] propone que todas las representaciones de los dispositivos utilicen tecnologías web (HTTP, p.ej.) para comunicar y exponer sus características y conseguir el acceso homogéneo a los dispositivos, tanto entre dispositivos como entre dispositivos y humanos. Sin embargo, usar tecnologías web para todo tipo de comunicación entre servidores específicos de la Web of Things no siempre será lo más óptimo. La motivación de esta capa es pues explicitar que la comunicación eficiente entre servidores se puede llevar a cabo mediante otro protocolo que no esté agrupado dentro de las tecnologías web. No obstante, es importante mencionar que incluso los servicios a los que se hace referencia podrían disponer de una interfaz HTTP para la integración en la WoT.

Finalmente, la capa de aplicación conceptualiza todas aquellas aplicaciones que son susceptibles de interactuar con la arquitectura. Dentro de todo el rango de aplicaciones soportadas por la Web of Things y, por lo tanto, del Internet of Things, se agrupan aquellas aplicaciones propias de las Smart Cities y las Smart Grids. En concreto la generación de una Red de Sensores Ubicua [4] es una aplicación que comprende la aglomeración de Redes de Sensores y Actuadores por cable y sin cable.

VI. Implementación

A. Primera propuesta, implementación en PHP

El objetivo que nos propusimos en [3] era el de crear una arquitectura de la Web of Things de alto rendimiento, a la vez que experimentábamos con el lenguaje de programación web más usado hasta ese momento, PHP, para comprobar si se podía facilitar a los muchos programadores web en PHP [31, 31] las interficies de programación para la Web of Things consiguiendo así una mayor adopción en menor tiempo. Los resultados no fueron positivos por las siguientes razones:

- Los servidores web de PHP siguen un modelo de ejecución muy estricto y dificultan el uso de tecnologías web como WebSocket. Las conexiones WebSocket son permanentes y los servidores PHP acotan la conexión con el cliente hasta un cierto límite de tiempo o hasta que se ha finalizado de ejecutar el script ya que el modelo de ejecución principal es *"load, execute, die"*. Existen alternativas a este modelo de ejecución como el que se implementa en React PHP [33] (patrón reactor), pero esta librería no está disponible para entornos de producción a gran escala como la WoT.
- PHP, un lenguaje interpretado, es más lento que los lenguajes compilados. Además, con el modelo de ejecución comentado anteriormente, los desarrolladores principales nunca se han preocupado de optimizar las instrucciones generadas ni de eliminar los múltiples *"memory leaks"*. Ahora, con la llegada de PHP 7 sí que se ha conseguido una mejora del rendimiento [34], aunque las otras razones que se comentan en esta lista siguen siendo válidas.
- Existen muy pocas librerías enfocadas a funcionalidades que involucren cálculos matemáticos intrínsecos al *Machine Learning* o modelos de computación distribuida. Esto es debido, una vez más, a la idiosincrasia de este tipo de lenguajes, los cuales se emplean principalmente para servir páginas web dinámicas.

B. Segunda propuesta, aproximación a través del Actor Model

En [3] se concluye que PHP no es suficientemente eficiente ni dispone de las herramientas necesarias para una implementación a gran escala de una infraestructura para la Web of Things. Una vez descartado el uso de PHP para desarrollar la infraestructura de la Web of Things, buscamos un lenguaje de programación que tuviera el rendimiento, el ecosistema y las herramientas necesarias para desarrollar tal infraestructura. La JVM (Java Virtual Machine) provee un entorno de ejecución estable (ya que está apoyada por Oracle) y muchas herramientas y librerías (protocolos, frameworks web, librerías de *Machine Learning*) están disponibles para esta plataforma.

Por otro lado, nos interesaba encontrar un modelo de programación que facilitase la creación de sistemas distribuidos, flexibles y auto cicatrizantes, características obligatorias para la WoE. Es por este motivo que el *Actor Model* [35] parece el modelo ideal. Este modelo o paradigma de programación facilita la creación de sistemas con estos requerimientos y, además, gracias a que permite concurrencia, es capaz de aprovechar los diferentes núcleos de computación de una máquina [36].

Los principios básicos de funcionamiento de este modelo son que cuando un Actor recibe un mensaje, éste puede:

- Enviar mensajes a otros Actores
- Crear nuevos Actores
- Designar como gestionar el siguiente mensaje que reciba

Aunque el *Actor Model* puede ser usado para crear agentes y, por lo tanto, Sistemas Multi Agente (MAS), este artículo se centra en la exposición y utilización del modelo *per se* en el IoT o la WoT.

En [37] los autores se benefician de este modelo para habilitar características como el *multi-cloud* y la *multi-tenencia* para dispositivos del IoT. Se aprovechan de los pocos recursos que necesita un actor para operar, compartimentando los dispositivos físicos en diferentes módulos de *software* (actores) aislados y conectados con diferentes infraestructuras *cloud* y diferentes propietarios.

En [38] se propone el uso de CAF (C++ Actor Framework) para proveer a los desarrolladores de un sistema de abstracción del Sistema Operativo de alto nivel para desarrollar aplicaciones del IoT. Destacan también las propiedades de abstracción, distribución y flexibilidad de este modelo de computación.

Gracias a los principios sobre el que está construido el *Actor Model* podemos conseguir propiedades interesantes para la infraestructura WoE.

- Distribución y flexibilidad: Uno de los principios básicos del modelo es la creación de nuevos actores. Esto facilita la utilización de los recursos de computación no solo en un mismo nodo sino en diversos nodos de forma distribuida ya que la comunicación es asíncrona y transparente entre actores de nodos locales y remotos. Además de poder crear nuevos actores, estos también se pueden destruir y, por lo tanto, permite la gestión de la utilización de recursos.
- Auto cicatrización: Esta propiedad no se basa en ningún principio del *Actor Model* teórico, pero todas las librerías que implementan este modelo la codifican desde la creación de *Erlang* debido a su gran uso práctico. Por definición, el estado de un actor está aislado en ese actor y solo se puede comunicar con el exterior a través de mensajes. La auto cicatrización se aprovecha de este principio de aislamiento para gestionar el fallo de un actor determinado. Por ejemplo, Akka [39] implementa la supervisión parental. La creación de actores por parte de otro actor resulta en la creación de una jerarquía con un actor "padre". En el caso de que un actor "hijo" falle, el actor "padre" puede decidir cómo gestionar este fallo: reiniciarlo o no hacerlo, por ejemplo. Gracias al aislamiento entre actores y a la supervisión entre actores se pueden aislar y gestionar los errores de ejecución de forma controlada.

Gracias a los principios básicos de un actor y a las propiedades que se derivan de estos, proponemos además que cada actor (o grupo de ellos) represente la virtualización de un dispositivo físico, aumentando los recursos y las funcionalidades de tal dispositivo y aislando así los fallos de ejecución de los demás actores, es decir, del sistema. Realmente, esta propuesta no es novedosa pues existen diferentes referencias [38, 39] que proponen este mismo uso para el IoT o incluso implementaciones específicas. Nuestra intención es proveer de un marco de trabajo y de una implementación para la WoE que cumpla con sus expectativas. También es nuestro objetivo promover el uso del *Actor Model* pues provee de importantes abstracciones para desarrollar sistemas distribuidos, flexibles, auto cicatrizantes y diseñados para alcanzar un uso máximo y óptimo de recursos gracias a la computación paralela.

C. Prueba de concepto

Se han implementado prototipos de algunas de las capas descritas anteriormente. En concreto se ha implementado el Espacio de Objetos Virtuales, la capa de Abstracción de Protocolos y, para comunicar dos servidores remotos, se ha considerado usar un protocolo publicador/suscriptor para la capa de proxy. A continuación se detallan cada uno de ellos:

a) Espacio de Objetos Virtuales: Cada dispositivo físico es representado por un objeto virtual y éste a su vez por uno o más actores. Esta capa define la lógica y las abstracciones necesarias para que cada dispositivo pueda ser representado por uno o más actores, de tal forma que el flujo de datos, tomando como punto de partida la recepción de datos de un protocolo de los que consideramos heterogéneos o del IoT (MQTT, p.ej.), es transformado a un protocolo homogéneo (capa de proxy) para que sea entendible por las demás partes de la arquitectura.

b) Capa de Proxy: Según las necesidades de esta prueba de concepto, se ha implementado la capa de proxy mediante un protocolo publicador/subscritor. De esta forma, los objetos virtuales que representan a los dispositivos pueden publicar aquellos datos captados y pueden suscribirse a mensajes también recogidos por otros dispositivos o a mensajes de actuación enviados por otros dispositivos.

c) Capa de Abstracción de Protocolos: Contiene las implementaciones que hacen de interfaz entre diferentes protocolos como MQTT o HTTP.

En la Fig. 3 se puede observar la interacción entre las distintas capas, encontramos, desde arriba abajo, (i) la capa de aplicaciones con las funciones inteligentes de las Smart Grids, (ii) el middleware que se propone en este documento, (iii) la capa de accesibilidad al USN con los agregadores de múltiples sensores y, finalmente, (iv) las redes de sensores. Intencionadamente, este esquemático tiene un gran parecido a la Fig. 2 que se muestra en [4]; en este caso nos hemos centrado en el desarrollo del Middleware para una USN (Ubiquituous Sensor Network).

Existen tres secciones de la implementación bien diferenciadas:

(i) Sección MQTT: La conforman aquellos dispositivos que se comunican mediante el protocolo MQTT y los servidores o servicios encargados de traducir el protocolo MQTT hacia un protocolo "entendible" por las capas internas de la arquitectura.

(ii) Espacio de objetos virtuales y capa de proxy: Esta sección la conforman aquellos módulos encargados de representar cada dispositivo físico mediante un objeto virtual y el protocolo publicador/subscriptor.

(iii) HTTP/WebSocket: Análoga a la sección MQTT, la conforman aquellos dispositivos que se comunican con la arquitectura mediante protocolos web y los servicios encargados de traducir estos protocolos hacia un protocolo "entendible" por las capas internas de la arquitectura.

Como servidor MQTT se ha usado Mosquitto [41]. Como implementación del Actor Model se ha usado

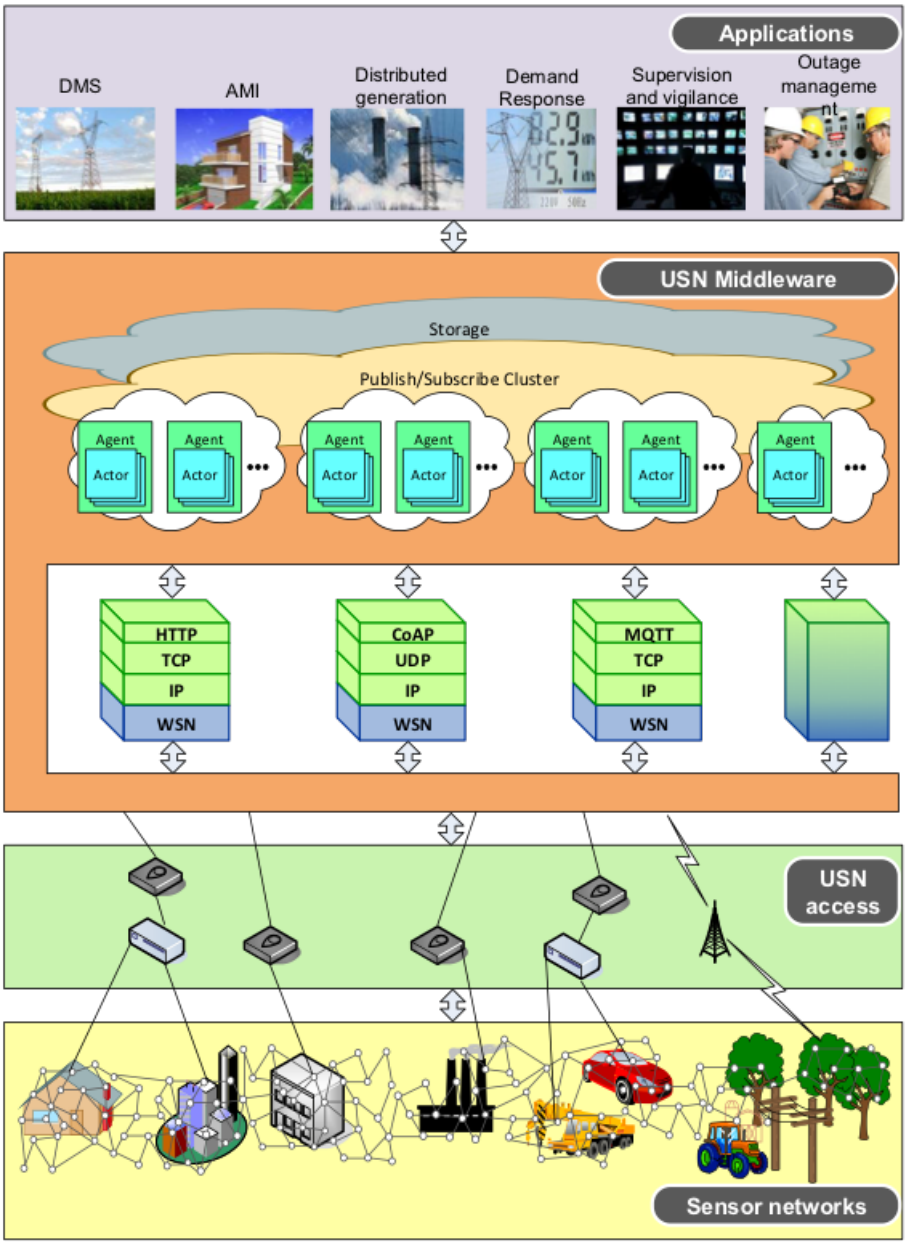

Fig. 3 Esquemático de las capas para un USN aplicado a las Smart Grids

Akka [39] a través de la API de Scala. Para la capa de proxy con la función de publicación/subscripción se ha usado una implementación con Akka-Cluster. Como servidor HTTP y WebSocket se ha usado Play Framework [42].

VII. CONCLUSIONES

En este artículo se usa el término Web of Energy para referirse a las características críticas que debe cumplir una arquitectura de la Web of Things para ser aplicada

al dominio de las Smart Grids. Se destacan sobre todo las necesidades de poder llevar a cabo funciones inteligentes, como son la distribución, resiliencia y auto cicatrización del sistema y la dificultad de renovar los sistemas tradicionales desplegados para la generación y gestión de energía, tanto a nivel de los dispositivos como a nivel de software, generalmente interrelacionados. Dados estos retos, se propone el uso del *Actor Model* para afrontarlos. Este paradigma está diseñado específicamente para realizar el modelado de sistemas concurrentes y distribuidos, aplicando así una mejora a las inherentes características de los sistemas distribuidos. En este sentido, se ha presentado en este artículo una propuesta de arquitectura *middleware* usando este paradigma para la creación de una red de sensores ubicua (USN). Esta propuesta se ha implementado mediante una prueba de concepto capaz de representar, de forma individualizada y virtual (y, por ende, con capacidad ilimitada de recursos) hasta 1000 dispositivos físicos simulados, con una representación visual web y a tiempo real.

AGRADECIMIENTOS

Parte de esta investigación ha estado financiada por la SUR de la DEC de la Generalitat de Cataluña y por Fondos Sociales Europeos 2017 FI_B 00583.

REFERENCIAS

[1] D. Guinard, “A Web of things application architecture: Integrating the real-world into the Web”, Tesis doct., ETH Zurich, 2011.

[2] J. Navarro, A. Sancho-Asensio, A. Zaballos, V. Jiménez-Ruano, D. Vernet y J. E. Armendáriz-Iñigo, “The Management System of INTEGRIS”, en Proceedings of the 4th International Conference on Cloud Computing and Services Science, SCITEPRESS-Science and Technology Publications, Lda, 2014, págs. 329-336.

[3] D. Vernet, A. Zaballos, R. Martin de Pozuelo y V. Caballero, “High performance web of things architecture for the smart grid domain”, International Journal of Distributed Sensor Networks, vol. 2015, 2015.

[4] A. Zaballos, A. Vallejo y J. M. Selga, “Heterogeneous communication architecture for the smart grid”, IEEE Network, vol. 25, n.º 5, 2011.

[5] D. Miorandi, S. Sicari, F. De Pellegrini e I. Chlamtac, “Internet of things: Vision, applications and research challenges”, Ad Hoc Networks, vol. 10, n.º 7, págs. 1497-1516, 2012.

[6] V. Trifa, D. Guinard y D. Carrera, “Web Thing Model”, W3C Member Submission, 2015.

[7] N. Shadbolt, T. Berners-Lee y W. Hall, “The semantic web revisited”, IEEE intelligent systems, vol. 21, n.o 3, págs. 96-101, 2006.

[8] R. Martín de Pozuelo, A. Zaballos, J. Navarro y G. Corral, “Prototyping a Software Defined Utility,” Energies, vol. 10, no. 6, p. 818, 2017.

[9] J. Iwata. (2012). “Making Markets: Smarter Planet”, [Online]. Disponible: https://www.ibm.com/investor/events/investor0512/presentation/05_Smarter_Planet.pdf (visitado el 05/04/2017).

[10] “Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020”, Gartner.com, 2017. [Online]. Disponible: http://www.gartner.com/newsroom/id/2636073. (visitado el: 05/04/2017).

[11] “Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015”, Gartner.com, 2017. [Online]. Disponible: http://www.gartner.com/newsroom/id/3165317. (visitado el: 05/04/2017).

[12] R. T. Fielding, “Architectural styles and the design of network-based software architectures”, Tesis doct., University of California, Irvine, 2000.

[13] M. V. Trifa, “Building blocks for a participatory web of things: devices, infrastructures, and programming frameworks”, Tesis doct., ETH Zurich, 2011.

[14] Banks y R. Gupta, “MQTT Version 3.1.1”, OASIS standard, 2014.

[15] Z. Shelby, K. Hartke y C. Bormann, “The constrained application protocol (CoAP)”, 2014.

[16] V. Trifa, S. Wieland, D. Guinard y T. M. Bohnert, “Design and implementation of a gateway for web-based interaction and management of embedded devices”, Submitted to DCOSS, págs. 1-14, 2009.

[17] “Enterprise IoT Solutions and Platform Technology”, ThingWorx, 2017. [Online]. Disponible: https://www.thingworx.com/. (visitado el 05/04/2017). [17]

[18] "IBM Watson IoT - IoT Platform", Ibm.com, 2017. [Online]. Disponible: https://www.ibm.com/internet-of-things/platform/watson-iot-platform/. (visitado el 05/04/2017).

[19] “Octoblu | Integration of Everything”, Octoblu.com, 2017. [Online]. Disponible: https://www.octoblu.com/. (visitado el: 05/04/2017).

[20] “EVRYTHNG IoT Smart Products Platform”, EVRYTHNG IoT Smart Products Platform, 2017. [Online]. Disponible: https://evrythng.com/. (visitado el 05/04/2017).

[21] C. Bizer, T. Heath, K. Idehen y T. Berners-Lee, “Linked data on the web (LDOW2008)”, en Proceedings of the 17th international conference on World Wide Web, ACM, 2008, págs. 1265-1266.

[22] W. W. W. Consortium y col., “RDF 1.1 concepts and abstract syntax”, 2014.

[23] Adida, M. Birbeck, S. McCarron y S. Pemberton, “RDFa in XHTML: Syntax and processing”, Recommendation, W3C, vol. 7, 2008.

[24] S. Bechhofer, “OWL: Web ontology language”, en Encyclopedia of Database Systems, Springer, 2009, págs. 2008-2009.

[25] D. Guinard, M. Fischer y V. Trifa, “Sharing using social networks in a composable web of things”, en Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on, IEEE, 2010, págs. 702-707.

[26] J. Gregorio y col., “The atom publishing protocol”, inf. téc., 2007.

[27] "ThingWorxComposer™ - 1Worx", 1Worx, 2017. [Online]. Disponible: http://www.1worx.co/the-thingworx-platform/thingworx-composer/. (visitado el 18/04/2017).

[28] "Node-RED", Nodered.org, 2017. [Online]. Disponible: https://nodered.org/. (visitado el 05/04/2017).

[29] E. Dijk, A. Rahman, T. Fossati, S. Loreto y A. Castellani, “Guidelines for HTTP-to-CoAP Mapping Implementations”, 2016.

[30] M. Collina, G. E. Corazza y A. Vanelli-Coralli, “Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST”, en Personal indoor and mobi- le radio communications (pimrc), 2012 ieee 23rd international symposium on, IEEE, 2012, págs. 36-41.

[31] "The 2016 Top Programming Languages", IEEE Spectrum: Technology, Engineering, and Science News, 2017. [Online]. Disponible: http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages. (visitado el 06/04/2017).

[32] "Usage Statistics and Market Share of Server-side Programming Languages for Websites, April 2017", W3techs.com, 2017. [Online]. Disponible: https://w3techs.com/technologies/overview/programming_language/all. (visitado el 21/04/2017)

[33] “reactphp/react", GitHub, 2017. [Online]. Disponible: https://github.com/reactphp/react. (visitado el 05/04/2017).

[34] "Get performance insight into the upcoming release of PHP 7", Zend.com, 2017. [Online]. Disponible: http://www.zend.com/en/resources/php7_infographic. (visitado el 05/04/2017).

[35] C. Hewitt, P. Bishop y R. Steiger, “Session 8 Formalisms for Artificial Intelligence A Universal Modular ACTOR Formalism for Artificial Intelligence”, en Advance Papers of the Conference, Stanford Research Institute, vol. 3, 1973, pág. 235.

[36] H. Sutter, “The free lunch is over: A fundamental turn toward concurrency in software”, Dr. Dobb’s journal, vol. 30, n.o 3, págs. 202-210, 2005.

[37] D. D. Sanchez, R. S. Sherratt, P. Arias, F. Almenarez y A. Marin, “Enabling actor model for crowd sensing and IoT”, en Consumer Electronics (ISCE), 2015 IEEE International Symposium on, IEEE, 2015.

[38] R. Hiesgen, D. Charousset, T. C. Schmidt y M. Wa¨hlisch, “Programming Actors for the Internet of Things”, ERCIM NEWS, pág. 25, 2015.

[39] "Akka", Akka.io, 2017. [Online]. Disponible: http://akka.io/. (visitado el 05/04/2017).

[40] M. Asay, "How One Developer Set Out To Make The Internet Of Things Manageable - ReadWrite", ReadWrite, 2017. [Online]. Disponible: http://readwrite.com/2014/07/10/akka-jonas-boner-concurrency-distributed-computing-internet-of-things/. (visitado el 05/04/2017).

[41] "An Open Source MQTT v3.1 Broker", Mosquitto.org, 2017. [Online]. Disponible: https://mosquitto.org/. (visitado el 18/04/2017).

[42] "Play Framework - Build Modern & Scalable Web Apps with Java and Scala", Playframework.com, 2017. [Online]. Disponible: https://www.playframework.com/. (visitado el 18/04/2017).