Un Middleware Centrado en Datos para el Control en Tiempo Real de Redes de Energía Inteligentes

Jaime Chen[1], Eduardo Cañete[1], Manuel Díaz[1], Daniel Garrido[1], y Krzysztof Piotrowski[2]

[1]Universidad de Málaga, Málaga, España
{hfc, ecc, mdr, dgarrido}lcc.uma.es
[2]IHP, Frankfurt (Oder), Alemania
piotrowski@ihp-microelectronics.com

Resumen. Este artículo presenta un middleware centrado en datos que permite en tiempo real la comunicación y almacenamiento de datos en redes eléctricas inteligentes. El middleware ha sido diseñado teniendo en cuenta la heterogeneidad de dispositivos y plataformas software utilizadas. Para su utilización, se dispone de una interfaz de datos que permite la comunicación entre las diferentes partes del sistema a través de servicios web REST (Representational State Transfer), así como una interfaz funcional que permite realizar operaciones de más alto nivel. El middleware está siendo utilizado para la implemenación de los demostradores del proyecto europeo e-balance, y ha tenido en cuenta para su utilización, la presencia de requisitos blandos de tiempo real, aspectos de seguridad y la escasa capacidad de algunos de los dispositivos en los que tendrá que ejecutarse.

Palabras clave: Middleware, Sistema Empotrado, Smart Grid, Control

1 Introducción

Las redes eléctricas inteligentes, también conocidas como *Smart Grids*, tienen una importancia cada vez mayor en el concepto más amplio conocido como ciudades inteligentes o *Smart Cities*. Cada vez es más habitual disponer por parte de las compañías eléctricas o intermediarios, de mecanismos para conocer en tiempo real el consumo por parte de sus clientes. A su vez, los clientes tienen nuevas posibilidades en forma de aplicaciones para móviles o tabletas donde pueden conocer, también en tiempo real, el estado de su facturación.

Por otra parte, asistimos también al auge de los llamados electrodomésticos inteligentes: lavadoras, lavaplatos o incluso frigoríficos. Todos estos dispositivos pueden estar conectados, bien a través de redes propias, bien a través de Internet, de manera que es posible monitorizarlos y controlarlos de manera remota.

Todas estas nuevas posibilidades, llevan de manera implícita, el manejo de una gran cantidad de datos a distintos niveles: dispositivos, aplicaciones de usuario, estaciones eléctricas, suministradores, etc. Hay que tener en cuenta, además, no sólo la

comunicación de los datos, sino también su almacenamiento, tanto para poder ser transmitidos a su vez a otros dispositivos o elementos de la red eléctrica inteligente, como para la gestión de la red mediante algoritmos inteligentes. Todo ello en tiempo real y con dispositivos de capacidades muy diferentes: desde sensores, pasando por pequeños mini-PC (p.ej. BeagleBones) o servidores con mayores capacidades de cómputo. Existe, asimismo, un amplio conjunto de aplicaciones para los diferentes roles que podemos encontrar: clientes, suministradores, operadores de la red, etc.

Este artículo se enmarca en el proyecto europeo del 7º programa marco, e-balance [1]. El principal objetivo del proyecto es diseñar e implementar una infraestructura basada en algoritmos eficientes capaz de gestionar la red de una manera inteligente.

Desde el punto de vista del usuario final, e-balance será capaz de analizar y controlar la producción y consumo de energía de los productores/consumidores, tomando decisiones inteligentes que afecten al comportamiento de los usuarios y logren un uso mejor de la energía. Desde el punto de vista de la infraestructura, e-balance pretende desarrollar un sistema capaz de intercambiar información de manera eficiente (tiempo real blando) entre todos los dispositivos que forman parte del mismo. Para lograr esta comunicación, se ha desarrollado un middleware encargado tanto de las comunicaciones como del almacenamiento distribuido de los datos.

El proyecto, con una duración de 42 meses, se encuentra ya en su tercer año, en los que se realizará el despliegue del sistema en dos demostradores con usuarios reales, uno de ellos situado en la localidad de Batalha (Portugal) y otro en un parque de ocio situado en Bronsbergen (Países Bajos).

Este artículo se centra en la presentación del middleware desarrollado para lograr llevar a cabo los objetivos de control y de gestión de la energía del proyecto.El sistema desarrollado es altamente distribuido y heterogéneo, con una gran cantidad de dispositivos, muchos de ellos empotrados, ejecutándose en paralelo y teniendo que cooperar entre ellos para lograr llevar a cabo una ejecución eficiente y de tiempo real del sistema. Los requisitos de tiempo real del mismo, no son altamente exigentes en cuanto a periodos o plazos (tiempos en el orden de segundos), pero sí es muy importante mantenerlos, dado que está en juego un funcionamiento adecuado de la red eléctrica, vital en la vida de hoy día.

La estructura del artículo es como sigue: la sección 2 presenta la arquitectura del proyecto e-balance. En la sección 3 se detalla el diseño del middleware y los componentes que lo forman. La sección 4 presenta la interfaz de datos que puede utilizarse para conectar a las diferentes unidades. La sección 5 presenta algunos detalles de implementación. Finalmente, se presentan algunas conclusiones y trabajos futuros.

2 Arquitectura de e-balance

El proyecto e-balance presenta algunas características típicas de los sistemas fractales. En esta visión fractal, las soluciones que se aplican en un nivel (p.ej. una casa), pueden luego volver a aplicarse en un nivel superior (p.ej. un vecindario) y así sucesivamente (p.ej. una ciudad). Esta visión fractal simplifica en cierta manera el desarrollo de e-balance y tiene una gran influencia sobre su arquitectura.

En el estado actual del proyecto, la arquitectura de e-balance se compone de 3 niveles: HAN, LV-FAN y MV-FAN correspondientes, respectivamente, a los niveles del cliente, estación de bajo voltaje (o secundaria) y de medio voltaje (o primaria).

En la Fig. 1 los componentes de e-balance están representados por las formas en color azul oscuro. Las cajas en azul claro representan el nivel de generación y transmisión de la red.

El sistema formado por e-balance incluye las denominadas unidades de gestión (*Management Units* o MUs). La figura muestra la relación jerárquica entre los diferentes niveles de e-balance con una MU para cada nivel.

En esta arquitectura, el middleware se ejecuta en las MU, recibe los datos desde las capas inferiores de comunicaciones, se encarga de gestionar los aspectos de privacidad y seguridad, y procede a almacenar los datos localmente hasta que estos sean solicitados por otras partes del sistema. Es necesario tener en cuenta también la interacción con sensores y actuadores.

Las formas en amarillo representan la red eléctrica y los dispositivos que hay en ella. Finalmente, las cajas en rojo representan marcos virtuales como el mercado, acceso a datos globales y operaciones. Las diferentes líneas en la figura representan varias clases de interacciones posibles entre los módulos que conectan.

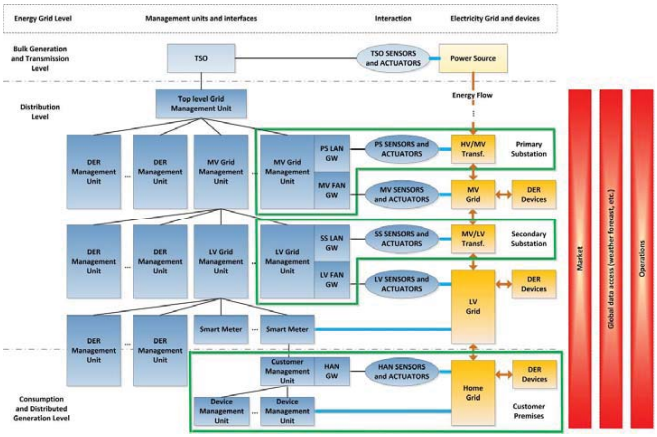

Fig. 1. Arquitectura de e-balance

Todas las MU tienen una arquitectura similar [2]. Sin embargo, dependiendo del nivel, pueden tener diferentes roles y obligaciones. Los procesos ejecutados en ellas pueden operar y procesar datos de diferentes partes, pero dada la arquitectura fractal de e-balance, los algoritmos de gestión aplicados en los diferentes niveles comparten la misma base conceptual, lo que mejora la escalabilidad de la propuesta.

El nivel de los dispositivos es el más bajo representado en la arquitectura. Un dispositivo puede ser, típicamente, un electrodoméstico que sólo consume energía, pero también puede ser una unidad de generación o almacenamiento. La denominada *Device Management Unit* (DMU) es la unidad central del dispositivo, que está al tanto de su estado y permite, además, su control. Para poder controlar las DMU, disponemos de las denominadas *Customer Management Unit* (CMU), que han sido diseñadas teniendo en cuenta la enorme heterogeneidad de dispositivos. Para ello, las CMUs utilizan plugins que pueden incorporarse dinámicamente. Por ejemplo, actualmente el consorcio está en contactos con una conocida empresa de electrodomésticos. La comunicación con estos dispositivos puede realizarse a través de Internet con una API proporcionada por esta empresa. Las CMUs están equipadas también para comunicarse con los sensores y actuadores de la denominada *Home Area Network* (HAN), red local de una casa, que interactúan con la red inteligente proporcionando monitorización, control, y, sobre todo, automatización.

El nivel por encima de las CMUs es el de la red eléctrica de bajo voltaje, donde encontramos las LVGMU (*Low Voltage Grid Management Unit*). Estas MUs están localizadas en las subestaciones secundarias, y cada una de ellas, controla los sensores, actuadores, CMUs y recursos de energía distribuidos (DER) de su nivel. Aquí, se repite la arquitectura, una LVGMU está equipada para comunicarse, en la denominada LV-FAN (*Low Voltage – Field Area Network*) con los niveles inferiores y también con el nivel superior, así como con los sensores y actuadores situados en el transformador de tensión media a baja y con los alimentadores de la estación secundaria.

Una MVGMU (*MediumVoltage Grid Management Unit*) es similar a su equivalente de bajo voltaje. Están situadas en las subestaciones primarias y equipadas para comunicarse con sensores, actuadores y LVGMU por debajo de la MU, en la denominada MV-FAN (*Medium Voltage – Field Area Network*).

La denominada TLGMU (*Top Level Grid Management Unit*) controla todas las MVGMU y DER conectados directamente a la red de medio voltaje. La TLGMU puede ser considerada como un centro de control que proporciona interfaces para las herramientas de gestión (sistemas SCADA), gestión de mercado, cortes, y otras.

Los párrafos previos dan una idea de la complejidad del sistema e-balance desde el punto de vista de las comunicaciones: está compuesto por diferentes dispositivos hardware (MUs, sensores, medidores inteligentes) con diferentes recursos, sistemas operativos y hechos por diferentes fabricantes. Además, todos estos dispositivos, tienen que comunicarse de manera segura entre sí, y con requisitos de tiempo real, generosos en cuanto a plazos (en el orden de segundos), pero existentes, al fin y al cabo.

3 Diseño del Middleware

El middleware presentado en este artículo se ha desarrollado con el objetivo de facilitar el desarrollo de aplicaciones en e-balance proporcionando un marco de abstracción sobre los detalles de las comunicaciones. Como se ha comentado, en nuestro sistema el

middleware se ejecutará en las diferentes unidades de gestión: TLGMU, MVGMU, LVGMU, DERMU y CMU. Los tres objetivos principales del middleware son:

1. Proporcionará medios para el intercambio de información: el middleware manejará automáticamente los niveles de red y proporcionará una manera simple de comunicación entre los dispositivos.
2. Proporcionará medios para almacenar/recuperar información: a través de una API. Los dispositivos pueden obtener y almacenar información persistente desde/de dispositivos remotos o desde/de el dispositivo local.
3. Elevar el nivel de abstracción sobre el que estos dispositivos están programados: el middleware proporciona una API simple de acceso a datos basada en un esquema de comunicación *publish/subscribe*.

3.1 Arquitectura del Middleware

La **Fig. 2** muestra la arquitectura de la plataforma de comunicaciones y su relación con la infraestructura de energía subyacente. La plataforma de comunicaciones se encuentra sobre las pilas de protocolos de red y por debajo de las aplicaciones e-balance. La plataforma de comunicaciones hace uso de un nivel de adaptación llamado *Communication Manager Module* que adapta las diferentes soluciones de red a un mismo "lenguaje" común para el middleware. El módulo de persistencia de datos (*Data Persistence Module*) almacena información y proporciona medios para almacenarla y recuperarla. El módulo de gestión de grupos y mantenimiento (*Maintenance and Group Management Module*) gestiona la red y la relación entre dispositivos. El módulo de procesamiento de solicitudes (*Request Processor Module*) se encarga de manejar todas las peticiones recibidas por la plataforma de comunicaciones. Este módulo es también responsable del procesamiento adecuado de las solicitudes de acuerdo con las políticas de seguridad y privacidad de los propietarios de los datos. Esta funcionalidad está implementada en el submódulo de control de acceso a datos (*Data Access Control Submodule*).

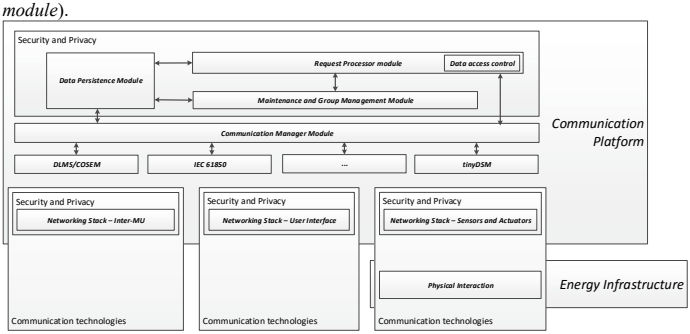

Fig. 2. – Arquitectura del middleware de comunicaciones de e-balance

3.2 Diseño software

La **Fig. 3** muestra una visión general del middleware desde el punto de vista del software. El diagrama muestra los diferentes módulos que conforman el middleware (persistencia de datos, procesamiento de peticiones, gestión de grupos y mantenimiento, control de acceso a datos y gestión de usuarios) y los servicios proporcionados por cada uno de ellos. Existe también una librería adicional denominada *EBCommon*, que se usa de manera interna para facilitar el desarrollo del middleware con funciones de apoyo y otra funcionalidad tales como manejo de tiempo, interfaz funcional para otros módulos, etc.

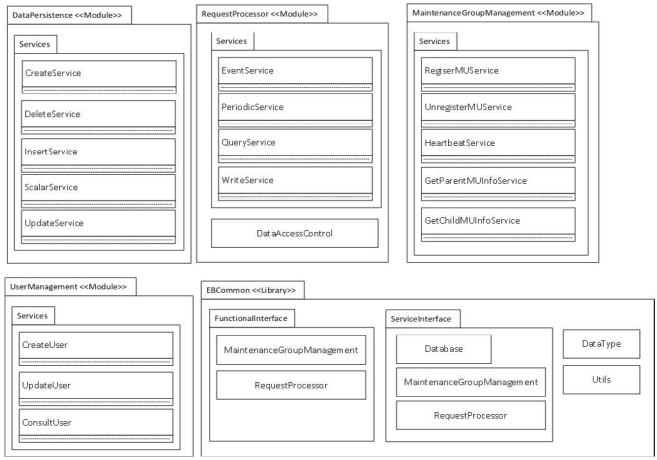

Fig. 3. – Diagrama de paquetes

El middleware está formado por un conjunto de módulos independientes que colaboran juntos a través de interfaces (servicios proporcionados por cada módulo). Esta alternativa de diseño permite a los diferentes módulos ser reemplazados sin afectar a los otros módulos. Por ejemplo, si se tiene que utilizar una nueva clase de base de datos, entonces, sólo se tiene que reemplazar el módulo de persistencia de datos. Es tarea del desarrollador adaptar la nueva base de datos a la API proporcionada por el módulo de persistencia de datos para hacer que sea compatible con el resto de módulos.

4 Interfaz de Datos de las Unidades de Gestión

La interfaz de datos (*Data Interface*) define las funciones a través de las cuales otros módulos software situados en la MU local o en otra remota pueden interactuar con una MU específica. Un ejemplo de este tipo de módulos es la Plataforma de Gestión de Energía de e-balance, que es la responsable de la ejecución de los algoritmos de balanceo de energía. La interfaz de datos permite interactuar con una MU específica leyendo y/o escribiendo datos desde/a ella. El principal objetivo de esta interfaz es hacer más fácil el desarrollo de aplicaciones en e-balance.

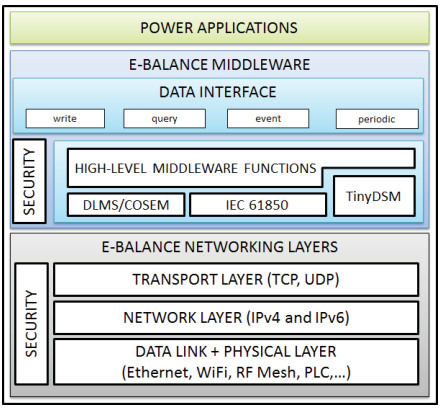

Fig. 4. – Contexto de la interfaz de datos (Data Interface)

El principal objetivo de diseño de la interfaz de datos (**Fig. 4**) fue la simplicidad Por esta razón, se compone únicamente de cuatro operaciones básicas:

- **Query:** esta operación permite leer valores de una variable específica de e-balance en un momento dado.
- **Write:** operación para actualizar/modificar el valor de una variable.
- **Event:** esta operación permite monitorizar una variable e-balance y recibir notificaciones en caso de que se cumplan determinadas condiciones sobre el valor de la misma (p.ej. superar un límite umbral).
- **Periodic:** esta operación permite subscribirse para recibir periódicamente los valores de una variable de e-balance. Un ejemplo puede ser la lectura periódica de una variable de consumo para refrescar los valores de la misma en la GUI de un cliente. La diferencia con respecto a la subscripción a eventos es que aquí no hay condiciones basadas en el valor de la variable, en su lugar, se define un periodo entre notificaciones.

Una operación puede únicamente ser realizada sobre variables existentes en el sistema si quien realiza la invocación tiene los permisos requeridos. La lista de variables existentes está predefinida dentro de e-balance.

Toda la comunicación realizada en el sistema se realiza únicamente a través de esta API. El middleware automáticamente convierte las llamadas a la API a los mensajes correspondientes en la plataforma de red específica que se esté usando. Los eventos y las operaciones periódicas siguen el paradigma de comunicación *publish/subscribe*. La aplicación se subscribe a una variable y los eventos son enviados cuando se actualice el valor o se cumpla alguna condición. Además de esta forma de comunicación, la aplicación también puede realizar peticiones bajo demanda con las operaciones de lectura y escritura.

5 Detalles de Implementación

En esta sección se describen los aspectos de implementación del middleware tal y como han sido aplicados en el proyecto e-balance.

5.1 Invocaciones remotas con ServiceStack

El middleware ha sido implementado utilizando ServiceStack [3], que es un marco de trabajo de código abierto diseñado para crear servicios web bajo el entorno .NET. ServiceStack permite modularizar la funcionalidad del middleware y estructurarlo en 2 niveles: *plugins* y servicios.

La funcionalidad básica del middleware se ha realizado con los servicios, y los *plugins* actúan como módulos capaces de contener tantos servicios como se requiera.

El concepto de *plugin* ha sido utilizado para implementar todos los módulos descritos en la sección 3.2.

Cada servicio web ServiceStack es modelado a través de un *Request DTO* (*Data Transfer Object*) y un *Response DTO*. Esto es, la entrada al servicio es un *Request DTO* y la salida es un *Response DTO*. Ambos *DTOs* definen las interfaces del servicio. Viendo el servicio como una función, el *Request DTO* podría representar a los argumentos de entrada y el *Response DTO* podría ser el conjunto de argumentos de salida.

5.2 Variables e-balance

Todos los datos de e-balance son modelados como tablas de una base de datos. Una variable e-balance es una tabla de datos donde el nombre de la tabla se refiere a la variable, y los campos de la tabla hacen referencia a las propiedades de la variable. Una variable e-balance puede tener tantas propiedades como haga falta, pero por defecto, hay tres propiedades obligatorias: *Id*, *Timestamp* y *Value*.

También podría ser posible tener todas las variables de e-balance en una tabla, pero la efectividad de la aproximación depende de la arquitectura subyacente del módulo de base de datos. En este caso, tener todas las variables en una tabla permitiría añadir nue-

vas variables más fácilmente, pero esto también puede hacerse en la aproximación elegida, si fuera necesario (las variables son predefinidas). Por otra parte, la principal ventaja de la aproximación utilizada es que tener una tabla por variable permite separar los datos con diferentes significados/funciones.

Supongamos que necesitamos una variable para almacenar el consumo de energía en el sistema. Para lograr, esto existe una tabla como la mostrada en la Tabla 1 que puede ser creada y usada para tal fin.

ENERGY_CONSUMPTION		
Id	**Timestamp**	**Value**
1	12/02/2015 14:10:00	30.4
2	12/02/2015 14:20:00	25.8

Tabla 1. Ejemplo: Tabla de consumo de energía

La tabla representa la variable e-balance llamada *energy_consumption*. Esta variable tiene las siguientes propiedades.

- **Id**: identificador de una tupla específica – la estructura de datos representando un valor simple de la variable.
- **Timestamp**: instante de tiempo del valor *energy_consumption* almacenado, el identificador temporal del valor.
- **Value**: el valor de la variable *energy_consumption* para cada punto en el tiempo.

Para trabajar con esta variable, la implementación proporciona una estructura de datos denominada `EBVariable` con los siguientes campos.

- **Name**: se refiere al nombre de la variable, p.ej. `energy_consumption`.
- **Properties**: propiedades de una variable e-balance específica, p.ej. `[Time, Value]`
- **Values**: contiene los valores que van a ser almacenados/leídos desde una variable e-balance.
- **Condition** – permite definir condiciones tipo SQL para leer conjuntos de tuplas que satisfagan una condición, p.ej., `"Id >2 and Id<5"` o `"Timestamp = 12/02/2015 14:20:00"`. Se basa en las propiedades definidas.

5.3 Interfaz de servicios web RESTful

Como se describió anteriormente, la interfaz de datos del middleware se compone de cuatro operaciones básicas. Cada una de estas operaciones (`write, query, event` y `periodic`) está implementada utilizando un servicio web RESTful a través del cual una aplicación puede interactuar fácilmente con una MU a través de Internet.

Representational State Transfer (REST) define un conjunto de principios arquitecturales a través de los cuales se pueden diseñar servicios web que sigan estos principios. Típicamente, un servicio web que utilice REST (un servicio RESTful) utiliza HTTP como su protocolo subyacente.

La utilización de los servicios web hace posible, en general, comunicarse con las MU desde prácticamente cualquier clase de hardware, sistema operativo y lenguaje de programación.

Cada servicio web tiene una forma de solicitud definida, así como una respuesta. Con ServiceStack esto se hace a través de los *DTO*. Por ejemplo, para la operación de escritura usaríamos el siguiente *DTO* para realizar solicitudes (ejemplo en Fig.5):

- **RequestSource:** identificador de quien realiza la llamada; dispositivos y servicio/participante. La URL del servicio web está compuesta de los siguientes campos:
  - **Protocol**: puede ser *http* o *https*.
  - **IP**: Dirección IP de la MU donde el servicio web está localizado.
  - **Port**: el Puerto a través del cual el servicio web puede ser accedido.
  - **ServicePath**: ruta dentro de la MU donde el servicio web está localizado.
- **Variable**: variable que va a ser almacenada.
  - **Name:** nombre de la variable e-balance.
  - **Properties**: indica las propiedades exactas de la variable e-balance que va a ser almacenada o modificada. Si su valor es nulo, el parámetro `Values`, debe contener un valor para cada propiedad.
  - **Condition**: este parámetro debe establecerse a nulo cuando vaya a insertarse un nuevo valor. Sin embargo, será útil para modificar tuplas existentes, ya que, permitirá identificarlas.
- **Values:** define los valores de la variable que va a ser almacenada o modificada.

La respuesta sería también a través de otro *DTO*:

- **OperationResults**:
  - **OpCode**: indica un código para describir el resultado de la operación de una forma concisa.
  - **Info**: proporciona un mensaje descriptivo sobre la operación.
  - **Success**: indica si el servicio se ejecutó de manera satisfactoria.

Data Owner			
Protocol	**IP**	**Port**	**ServicePath**
http	192.168.43.98	2554	/

e-balance Variable		
Name	**Fields**	**Condition**
ENERGY_CONSUMPTION	Time, Value	Null

Values
12/04/2015 12:43:00, 23.2

Fig. 5. Ejemplo de DTO para operación de escritura

5.4 Interfaz de Datos Funcional

Todas las funciones proporcionadas por la interfaz de datos están basadas en servicios web RESTful, de manera que dos MUs pueden interactuar entre ellas o intercambiar datos a través de servicios web ejecutados sobre IP. Esta clase de interfaz es bastante restrictiva en el sentido de que sólo permite hacer a los usuarios direccionamiento *unicast*. Se se ha desarrollado también una interfaz funcional que permite, no solamente hacer *broadcast* (p.ej. una LVGMU podría solicitar información de varias CMUs), sino que mejora también la utilidad de la interfaz de datos al poder ser utilizada desde otros módulos internos de e-balance (aplicaciones e-balance). La **Tabla 2** muestra ejemplos de operaciones que pueden utilizarse con esta API funcional.

Operación	Cabecera método
Escritura de valores unicast en una MU específica	`WriteResponse Write(EBUrl destinationMU, EBVariable ebVariable)`
Escritura de valores en varias MUs especificadas por el parámetro destinationMU	`WriteResponse[] Write(EBUrl[] destinationMUs, EBVariable ebVariable)`
Escritura de varias variables en todas las MUs hijas de la que hace la llamada	`WriteResponse[] WriteInChildMUs(EBVariable ebVariable)`

Tabla 2. Ejemplos de operaciones en API funcional

El uso del broadcast no supone un gran impacto para los dispositivos de menor capacidad, ya que, por la arquitectura de e-balance el mayor impacto de las comunicaciones recae en los dispositivos de mayor capacidad como PCs o servidores y, por otra parte, la arquitectura fractal aisla en diferentes niveles el impacto.

5.5 Despliegue del sistema

Tal y como se ha comentado, e-balance utiliza varias MUs para la gestión de los diferentes niveles de la red. Son estas MUs las que intercambian información utilizando el middleware descrito en las secciones anteriores. Las aplicaciones, utilizan a la plataforma de comunicaciones con el objetivo de lograr un uso más eficiente de la red eléctrica. En esta sección se describe parte de dicho despliegue en tres niveles: CMU, LVGMU y MVGMU.

- CMU: En cada casa se dispone de una CMU, que controla los electrodomésticos inteligentes y monitoriza el consumo/generación de energía de la casa. Cada CMU

requiere conexión a Internet, utilizando la conexión de la casa donde se instala. Las CMUs se ejecutan sobre dispositivos BeagleBone Black [4]. Estos dispositivos son de bajo coste y escaso tamaño. Están equipados con 512 MB de DDR3L DRAM, 4GB 8-bit eMMC de flash memory, procesador AM3358 a 1GHz basado en el procesador ARM Cortex-A8, aceleradora 3D, y un acelerador NEON para punto flotante. La BeagleBone está equipada con USB, Ethernet y Micro HDMI. La conexión Wi-Fi puede realizarse a través de un *dongle* utilizando el Puerto USB.
El software se ejecuta en la BeagleBone utilizando Mono [5], una implementación de código abierto de Microsoft .NET. Adicionalmente, la CMU ejecuta aplicaciones Java, que utilizan la plataforma de comunicaciones para acceder a otras MUs.

- LVGMU y MVGMU: Este tipo de MUs ejecutan también el mismo software que las BeagleBones. La implementación actual incluye el uso de máquinas virtuales y restricciones adicionales en las comunicaciones, que por motivos de simplicidad no son incluidas en este artículo.

6 Conclusiones

El desarrollo de un sistema de gestión de redes electricas inteligentes comprende una gran variedad de dispositivos hardware y elementos software que hacen muy difícil la reutilización de código. Con el fin de facilitar el desarrollo de aplicaciones en este tipo de plataformas, en este artículo se ha presentado un middleware centrado en datos que puede ser utilizado en los diferentes niveles de la arquitectura para comunicación y almacenamiento de datos en tiempo real. El middleware dispone de una interfaz de datos simple y una funcional, lo que facilita su reutilización y adaptación en diferentes sistemas. Además, la interfaz de datos está basada en servicios web RESTful, lo que permite su utilización desde prácticamente cualquier tipo de dispositivo y sistema operativo. Por último, el middleware tiene que encargarse además de los aspectos relativos a tiempo real y seguridad. Actualmente se está utilizando en el marco del proyecto europeo e-balance para el despliegue de sus demostradores.

7 Referencias

1. Página web proyecto e-balance. http://www.e-balance-project.eu/
2. K. Piotrowski, et al., “Deliverable D3.2 – Detailed System Architecture Specification”, Public deliverable of e-balance project, FP7-Smartcities-2013, Project number 609132, 2015.
3. ServiceStack framework. https://servicestack.net/
4. Beaglebone Black. http://www.beagleboard.org
5. Mono. Cross platform, open source .NET framework. http://www.mono-project.com

Agradecimientos

Este trabajo ha sido parcialmente soportado por el proyecto europeo e-balance (609132), nacional POLYCIMS (TIN2014-52034-R) y regional MIsTIca (TIC-1572).