Propuesta de una Arquitectura de Dispositivos como Servicios con Procesamiento de Eventos

Juan Boubeta-Puig[1], Javier Cubo[2], Adrián Nieto[2],
Guadalupe Ortiz[1], y Ernesto Pimentel[2]

[1] Universidad de Cádiz, Dpto. Ingeniería Informática, España
{juan.boubeta,guadalupe.ortiz}@uca.es
[2] Universidad de Málaga, Dpto. Lenguajes y Ciencias de la Computación, España
{cubo,adrian,ernesto}@lcc.uma.es

Resumen. Internet de las Cosas representa un paradigma en el que los objetos que nos rodean están interconectados. En esta visión, existen problemas como la detección de dispositivos heterogéneos, la inexistencia de estándares para interoperar con los dispositivos, o la eficiencia en la obtención de la información. Para la detección e interacción, existen estándares e iniciativas que exponen los dispositivos y las cosas como servicios, siguiendo un enfoque de arquitecturas orientadas a servicios. Para procesar la información generada, además del análisis de datos, es fundamental reaccionar ante cambios en los dispositivos, estableciendo pautas de comportamiento. Las técnicas de procesamiento de eventos complejos junto con las arquitecturas dirigidas por eventos permiten diseñar sistemas reactivos y desacoplados, analizando los cambios en el entorno y adaptando su comportamiento en base a patrones de eventos. Actualmente, la mayoría de las soluciones que permiten interactuar con dispositivos heterogéneos son complejas y requieren conocimiento avanzado. En este trabajo, proponemos una arquitectura de dispositivos orientada a servicios y dirigida por eventos, exponiendo los dispositivos como servicios para unificar su manejo, que interactúan con el entorno mediante eventos que son procesados, y dotando al ecosistema de la capacidad de comportarse de forma autónoma y reactiva.

Palabras clave: IoT, DaaS, CEP, SOA 2.0, SODA, DPWS

1. Introducción

La implantación de la Internet de las Cosas (*Internet of Things*, IoT) en la sociedad representa un cambio en la forma de entenderla, donde los dispositivos que nos rodean —electrodomésticos, vehículos, dispositivos *wearables* instalados en las propias ciudades— están conectados entre sí. Para lograr esta interconexión de dispositivos, se ha demostrado que una visión de arquitectura orientada a servicios (*Service-Oriented Architecture*, SOA) de los dispositivos es fundamental, dando lugar a dispositivos como servicios (*Devices as a Service*, DaaS) o cosas como servicios (*Things as a Service*, TaaS). Sin embargo, SOA no garantiza resolver la heterogeneidad de los dispositivos, problema que tratan de resolver diversos estándares como *Device Profile for Web Services* (DPWS) [1] o *Universal Plug and Play* (UPnP) [4] empleando *Web Services* (WS), o enfoques

Representational State Transfer (REST) como HyperCat [3]. Así, se requiere una arquitectura común para abordar los problemas de integración y comunicación entre dispositivos heterogéneos, que podrían ser direccionados considerando una arquitectura de dispositivos orientada a servicios (*Service-Oriented Device Architecture*, SODA) [13].

Entre los numerosos retos que plantea IoT encontramos el de gestionar gran cantidad de información generada por los dispositivos que componen un sistema. Para poder procesar y correlacionar toda esta información proveniente de dispositivos y sensores, así como detectar situaciones críticas y/o relevantes y tomar decisiones en tiempo real, existe una tecnología emergente denominada procesamiento de eventos complejos (*Complex Event Processing*, CEP) [16], que en conjunción con las arquitecturas dirigidas por eventos (*Event-Driven Architecture*, EDA), permite diseñar sistemas reactivos y desacoplados. Éstos analizan los cambios en el entorno y adaptan su comportamiento en base a unos patrones de eventos definidos mediante lenguajes de procesamiento de eventos (*Event Processing Language*, EPL) [6]. Uno de los principales problemas a los que deben enfrentarse los expertos de dominio, es precisamente la implementación de dichos patrones usando algún EPL. Aunque algunas soluciones software actuales ofrecen herramientas gráficas con la finalidad de solventar este problema, éstas no son lo suficientemente amigables, puesto que requieren que el usuario escriba manualmente, al menos, una parte del código necesario para la definición de los patrones. Para paliar esta necesidad, recientemente hemos propuesto MEdit4CEP [5], una solución dirigida por modelos para la toma de decisiones en tiempo real en una arquitectura orientada a servicios y dirigida por eventos (*Event-Driven Service-Oriented Architecture*, ED-SOA o SOA 2.0).

En este artículo proponemos una arquitectura de dispositivos orientada a servicios y dirigida por eventos, SODA 2.0 (*Event-Driven Service-Oriented Device Architecture*), como extensión a la definición actual de SODA, mediante la combinación de SOA, DaaS y EDA. Se trata de una arquitectura novedosa en la que los dispositivos implementan SOA para unificar el manejo de los mismos y, a su vez, interactúan con el entorno mediante eventos que son procesados con la tecnología CEP, dotando al ecosistema de la capacidad de comportarse de forma reactiva. Esta arquitectura se ha diseñado para ser extensible; se pretende analizar su integración con la Nube (*Cloud*), como ya se ha realizado en trabajos previos haciendo uso de Google AppEngine [10]. Los autores están actualmente estudiando alternativas abiertas, como la plataforma FIWARE [2] —una iniciativa europea para diseñar un ecosistema *Cloud* abierto y accesible—, para extender el concepto de SODA 2.0 a un escenario global y ubicuo aprovechando el potencial de la Nube para almacenamiento y procesado de datos.

El resto del artículo está organizado como sigue. La Sección 2 describe las tecnologías y conceptos implicados en esta propuesta, y la Sección 3 presenta los antecedentes y trabajos relacionados. La Sección 4 describe nuestra propuesta SODA 2.0. La Sección 5 presenta dos escenarios que ilustran la aplicación de nuestra propuesta. Finalmente, se presentan las conclusiones y el trabajo futuro.

2. Fundamentos

La propuesta que se presenta en este trabajo se sustenta en los paradigmas y tecnologías de DaaS o TaaS, SODA, ED-SOA o SOA 2.0, y CEP.

Dispositivos como servicios: DaaS basado en DPWS. DPWS es un estándar OASIS que emplea un modelo SOA construido sobre el estándar W3C de arquitectura de servicios Web (SOAP + WSDL + XML schema), creando un perfil sobre la pila de protocolos WS-*. Está diseñado como un conjunto de guías basadas en especificaciones WS-* que proporciona interoperabilidad entre diferentes dispositivos y servicios compatibles con DPWS dentro de un entorno de red. DPWS está integrado en Windows, es ligero, soporta descubrimiento dinámico y puede ser usado por estándares de orquestación y coreografía, como WS-BPEL o WS-CDL.

Arquitectura de dispositivos orientada a servicios: SODA. SODA es una adaptación de SOA que permite integrar un gran conjunto de dispositivos en sistemas empresariales distribuidos, modelándolos como servicios. Los requisitos del paradigma SODA son los siguientes [13]: 1) uso de un modelo adaptador de servicios para encapsular las interfaces de programación específicas de dispositivos; 2) uso de mensajería mínimamente acoplada, que permita la utilización de un ESB (*Enterprise Service Bus*); 3) uso de estándares abiertos a nivel de interfaz de dispositivos y servicios; 4) mecanismos que proporcionen interfaces de servicios abiertas a dispositivos que tengan protocolos propietarios; 5) soporte para la implementación de una variedad de adaptadores de dispositivos (simples y complejos); y 6) mecanismos de seguridad según el dominio en cuestión.

Arquitectura orientada a servicios y dirigida por eventos: SOA 2.0. SOA 2.0 es una evolución de una SOA tradicional en la que las comunicaciones entre usuarios y servicios se llevan a cabo a través de eventos, en lugar de a través de llamadas a procedimientos remotos [16]. EDA es un patrón de arquitectura software en el que los componentes reaccionan al reconocer eventos y están mínimamente acoplados [8]. EDA y SOA no son alternativas excluyentes entre sí, sino compatibles y complementarias. Para lograr esta integración, se requiere una capa de abstracción de software altamente distribuida e integradora [19]: un ESB que posibilita la interoperabilidad entre protocolos de comunicación diferentes, pudiéndose usar como una plataforma de integración donde las aplicaciones son expuestas como servicios [12].

Procesamiento de eventos complejos: CEP. CEP es una tecnología emergente que permite capturar, analizar y correlacionar una gran cantidad de eventos heterogéneos con el fin de detectar situaciones críticas o relevantes en tiempo real. Un *evento* es algo que ocurre o que se espera que ocurra [15]. Por otra parte, una *situación* es una ocurrencia de un evento o una sucesión de eventos que requiere alguna reacción inmediata [14]. Esta tecnología se basa en el filtrado de eventos irrelevantes y en el reconocimiento de los eventos que sí son relevantes para un dominio en particular. Para ello, se utilizan *patrones de eventos*, esto es, unas plantillas en las que se especifican cuáles son las condiciones que deben cumplirse para detectar dichas situaciones de interés, así como las acciones que deberán

ser ejecutadas tras su detección. A estas situaciones de una mayor complejidad semántica se les denominan *eventos complejos*, por tanto, un evento complejo es una abstracción de otros eventos simples o complejos que contiene la información necesaria para describir la situación recién acontecida [16]. La característica principal de estos eventos complejos es que pueden ser identificados y notificados en tiempo real, reduciendo la latencia en la toma de decisiones, a diferencia del software tradicional de análisis de eventos que no funciona en tiempo real. Para llevar a la práctica este tipo de procesamiento de eventos en los sistemas de información, se requiere un software específico conocido como *motor CEP*.

3. Antecedentes y trabajos relacionados

En esta sección, presentamos antecedentes de trabajos previos y relacionados.

En el desarrollo de soluciones para IoT, las dos corrientes más comunes son el uso de estándares complejos basados en WS-* o filosofías de diseño más simples mediante un enfoque RESTful. Ambas alternativas presentan una serie de ventajas e inconvenientes, cuyo análisis (fuera del ámbito del presente trabajo) se puede encontrar en [17]. En trabajos anteriores, Cubo et al. [9] han propuesto el uso de DPWS para abordar la composición de servicios, considerando la necesidad de especificar el comportamiento de los dispositivos durante la orquestación de dichos servicios. También se ha analizado el uso del paradigma DaaS (basado en DPWS), con la finalidad de desarrollar la plataforma DEEP (*DPWS-Enabled Devices Platform*) en Cubo et al. [10] para integrar dispositivos heterogéneos mediante la generación de un *gateway*. En dicho trabajo, los autores han presentado una integración de IoT y la Nube para alojar los datos recogidos por los dispositivos a través del *gateway*, y poder accederlos de manera remota y monitorizarlos en tiempo de ejecución. Además de la integración de dispositivos y monitorización de datos, DEEP pretende llevar a cabo el análisis de datos, que aún no se ha desarrollado, por lo que hay una necesidad de abordar esta cuestión.

Por otra parte, Boubeta-Puig et al. [7] han propuesto una arquitectura SOA 2.0 integrada con CEP y plataformas IoT, a través de un ESB. Esta arquitectura hace posible la detección de situaciones críticas en distintos escenarios de aplicación. Gracias a esta arquitectura, cuando dichas situaciones ocurran a partir de la detección de ciertos patrones de eventos predefinidos en el motor CEP, las alertas (eventos complejos) serán enviadas en tiempo real a los usuarios y sistemas que estén interesados en recibirlas. Asimismo, Boubeta-Puig et al. [5] han creado MEdit4CEP, una solución dirigida por modelos para la toma de decisiones en tiempo real en una SOA 2.0. En particular, se ha propuesto un enfoque dirigido por modelos para CEP en SOA 2.0, un editor gráfico para la definición de dominios CEP y un editor gráfico reconfigurable para la definición y la generación de código de patrones de eventos —una versión inicial de este editor se encuentra publicada en [6]—. Al poder reconfigurarse el editor de patrones mediante un modelo de dominio CEP, el usuario dispondrá en todo momento de una interfaz gráfica común adaptada al contexto específico para el que desee definir los patrones de eventos, modificándose dinámicamente la paleta de herramientas dependiendo del dominio en cuestión. Por tanto, el editor pone al alcance de cualquier usuario no tecnólogo la definición de las situaciones que necesite detectar en tiempo real.

En cuanto a trabajos relacionados, en los últimos años, IoT se está integrando con la tecnología CEP y SOA 2.0, combinando las ventajas de EDA con la toma de decisiones automática, lo que se traduce en acciones reales sobre dispositivos físicos. En varios trabajos se ha propuesto dicha integración. En [11] se propone una SOA que soporta servicios personalizados centrados en el usuario altamente escalables para IoT, mediante un ESB y el motor de reglas Drools. En [20] proponen un enfoque para comunicación máquina a máquina y procesamiento de datos en aplicaciones del Internet del futuro. Por otra parte, en [18], los autores estudian las ventajas de combinar *Big Data*, CEP y IoT en el ámbito de la salud, a partir de datos de dispositivos *wearables*, *smarthomes* y *smart cities*.

Aunque estos trabajos son interesantes, es complicado encontrar una solución sencilla que permita interactuar con los eventos de los dispositivos siguiendo algún estándar y sin requerir conocimiento avanzado. Sin embargo, nuestro enfoque se centra en presentar una arquitectura novedosa donde los dispositivos se exponen como servicios para unificar su manejo, e interactúan con el entorno, dotando al ecosistema de la capacidad de comportarse de forma autónoma y reactiva.

4. Nuestra propuesta: SODA 2.0

En esta sección proponemos una arquitectura de dispositivos orientada a servicios y dirigida por eventos, SODA 2.0, que aúna las ventajas de los siguientes enfoques integrados: SOA proporciona una solución eficiente para la implantación de sistemas en los que la modularidad y las comunicaciones entre terceros son un factor clave, SODA es una adaptación de SOA que permite integrar dispositivos físicos (sensores y actuadores) como servicios, y EDA se caracteriza porque las comunicaciones entre servicios y aplicaciones se llevan a cabo por medio de eventos de una forma totalmente desacoplada.

Con el objeto de analizar continuamente toda la información que fluye por esta SODA 2.0, para detectar cuanto antes y de forma automática las situaciones que son críticas para los procesos de negocio, se requiere su integración con CEP. La Figura 1 ilustra nuestra propuesta SODA 2.0: una extensión de la SOA 2.0, propuesta en Boubeta-Puig et al. [5], integrada con la plataforma DEEP, desarrollada en Cubo et al. [10], y que es descrita brevemente a continuación.

El elemento principal de la arquitectura propuesta es el **ESB**, que actuará como capa de integración entre los productores de eventos, los consumidores de eventos, el motor CEP y MEdit4CEP [5], reduciendo así el acoplamiento. MEdit4CEP proporciona un editor de modelado que permite a usuarios no tecnólogos diseñar de forma gráfica e intuitiva los patrones de eventos o situaciones que se desean detectar en el sistema y transformarlos al código que los implementa.

Los **productores de eventos** son los componentes de la arquitectura desde los que se obtiene la información que debe ser procesada y correlacionada con la intención de detectar posibles situaciones críticas y/o relevantes en el sistema. Existen distintos tipos de productores de eventos; en esta arquitectura se han considerado los más relevantes en cuanto a la integración con dispositivos físicos, contemplados en la plataforma DEEP:

- *Dispositivos simples*: dispositivos que monitorizan el entorno donde se encuentran ubicados con el propósito de captar información (temperatura,

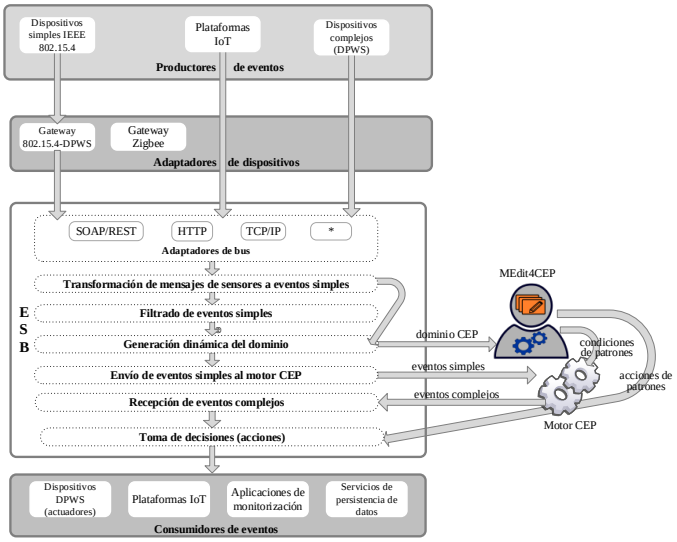

Fig. 1. SODA 2.0: Arquitectura general de la propuesta

luminosidad, etc.), tales como dispositivos TinyOS (sensores TelosB y MicaZ), dispositivos JavaME (sensores SunSpot) y dispositivos basados en el estándar IEEE 802.14.5 (como Waspmote o Arduino + Xbee).

- *Dispositivos complejos*: dispositivos con un comportamiento más sofisticado que los dispositivos simples (en nuestro caso, basados en DPWS con composición de varios dispositivos o sus servicios, como móviles o cámaras).
- *Plataformas IoT*: plataformas que permiten gestionar la información proveniente de sensores localizados en diversos entornos.

Los componentes homólogos a los productores de eventos son los denominados **consumidores de eventos**. Estos componentes reaccionan ante los eventos recibidos desde el ESB. Al igual que ocurre con los productores de eventos, existe una gran variedad de consumidores de eventos, entre los que caben destacar:

- *Dispositivos complejos*: dispositivos como actuadores (basados en DPWS) que realizan una acción en particular tras detectar algún evento complejo.
- *Plataformas IoT*: plataformas donde podrán hacerse públicas las situaciones críticas y/o relevantes detectadas.
- *Aplicaciones de monitorización*: aplicaciones diseñadas específicamente para avisar a los usuarios de las situaciones.
- *Bases de datos*: colecciones de datos donde se almacenarán los eventos complejos detectados.

Para dotar a los sensores de conectividad TCP/IP se propone el uso de **adaptadores de dispositivos**, en concreto, de un *gateway* que se encarga de recuperar la información de los sensores mediante el protocolo IEEE 802.14.5 y exponerlos a la red local mediante DPWS. El *gateway* realiza la función de agregar la capa de WS necesaria a los dispositivos que por sí mismos no sean capaces (microcontroladores, sensores simples, o cualquier dispositivo carente de

TCP/IP). Así pues, los dispositivos más avanzados (como un teléfono móvil o un electrodoméstico) que sean capaces de exponer DPWS autónomamente, ignorarán la presencia del *gateway*, permitiendo así la interacción con los dispositivos de forma homogénea. Una vez abstraídos los detalles hardware de los dispositivos, la interacción con los dispositivos mediante DPWS facilita su uso de manera homogénea mediante cualquier cliente SOAP, operando con los dispositivos de la misma manera que se haría con un cliente para servicios Web tradicionales.

Las funcionalidades que proporciona la arquitectura SODA 2.0 propuesta son:

- *Recepción de eventos*: se reciben los datos generados por los productores de eventos. Para cada tipo de estos productores se utilizarán adaptadores que se encargarán de mediar entre protocolos de transportes.
- *Transformación de eventos*: los datos provenientes de los productores de eventos serán transformados a un formato común, facilitando el procesamiento de estos al motor CEP.
- *Filtrado de eventos*: en el caso de que sea necesario, podrán filtrarse los eventos a enviar al motor CEP.
- *Generación dinámica del dominio*: previo al envío de cada evento al motor CEP, se comprobará si su tipo ya es reconocido por el sistema o si se trata de un nuevo tipo de eventos. En este último caso, se inferirá automáticamente el tipo de evento junto con los tipos de sus propiedades, registrándose en el motor CEP. A partir de estos tipos de eventos se creará automáticamente un modelo gráfico de dominio CEP que será incluido en el editor MEdit4CEP.
- *Envío de eventos al motor CEP*: los eventos serán enviados e introducidos en los flujos de eventos del motor CEP en tiempo de ejecución. Además, el código EPL de los patrones de eventos generado automáticamente por MEdit4CEP será añadido al motor en tiempo de ejecución.
- *Recepción de eventos complejos*: en el caso de que se cumplan las condiciones especificadas en uno de los patrones de eventos registrados en el motor CEP, se creará un evento complejo y se enviará al ESB.
- *Toma de decisiones (acciones)*: el editor MEdit4CEP transformará automáticamente las acciones de los patrones modelados por los usuarios finales a código y lo desplegará en el ESB. Entonces, el ESB ejecutará estas acciones para enviar los eventos complejos a los consumidores de eventos suscritos.

5. Escenarios de aplicación de nuestra propuesta

Para ilustrar nuestra propuesta, aquí presentamos dos escenarios de aplicación.

Escenario *SmartCity*. Uno de los mayores retos en el desarrollo de una *SmartCity* es la gestión autónoma de la ciudad en tiempo real. Los sistemas reactivos son los candidatos ideales para gestionar este tipo de escenarios, en los que se pretende automatizar la respuesta a cualquier eventualidad. Dentro de estas eventualidades, el primer ejemplo motivador se centra en la prevención y actuación temprana en el caso de un accidente de tráfico. En un automóvil moderno existen numerosos sensores para medir, por ejemplo, magnitudes tales como la velocidad de giro de las ruedas, la existencia de lluvia, la temperatura, si hay pasajeros en un asiento, o si un *airbag* ha sido disparado. En este contexto planteamos

un escenario donde sucede un accidente involucrando a un vehículo que dispone de los sensores anteriormente mencionados, los cuales se asumen que pueden ser actualizados para soportar DPWS y que se encuentran conectados de manera inalámbrica (Wi-Fi o red telefonía móvil). El uso de DPWS facilita la interacción autónoma y segura de los sensores, gracias a la descripción estandarizada de los sensores, al descubrimiento automático y al cifrado de las comunicaciones.

Durante la circulación normal del vehículo los sensores envían sus mediciones al ESB, el cual transforma estos datos a eventos que son enviados al motor CEP. En este motor se habrán registrado previamente los patrones de eventos que se requieran detectar para este escenario. Supongamos que se ha definido un patrón denominado *AccidenteTráfico*, donde se han indicado las condiciones que deben cumplirse para detectar automáticamente y en tiempo real un accidente de tráfico: 1) una o más ruedas del vehículo pierde presión, y 2) la velocidad del vehículo disminuye drásticamente a 0 Km/h y 3) se acciona el *airbag*. Conforme llegan eventos al motor CEP, éste los va analizando y correlacionando en tiempo real. En el instante en que se detecte dicho patrón, se creará un evento complejo *AccidenteTráfico* que será enviado al ESB. Éste se encargará de notificar la situación acontecida (el evento complejo) a los servicios operativos para actuar de forma rápida y eficaz. Este escenario se ilustra en la Figura 2.

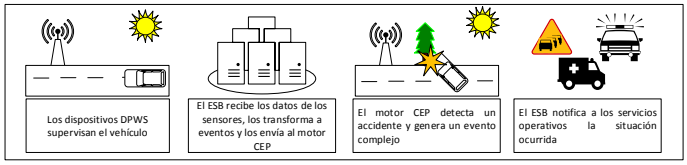

Fig. 2. Detección automática de un accidente en carretera

Escenario *eHealth* y *Ambient Assisted Living* (AAL). La implantación de IoT en escenarios relacionados con el ámbito de la salud supondrá una mejora de la calidad de vida de las personas convalecientes. Gracias al empleo de DPWS, los dispositivos conectados a los pacientes son descubiertos dinámicamente, notificando al sanitario que realiza la conexión y la asociación al encender estos dispositivos. En este conjunto de dispositivos se pueden tener medidores de presión arterial, temperatura, de ritmo cardíaco, saturación de oxígeno o actuadores como bombas de insulina o reanimadores automáticos.

Cuando los sensores son vinculados a un paciente comienzan a generar información sobre su estado que es enviada al ESB. Si en un determinado momento el paciente sufre una bajada de sus constantes vitales, el motor CEP inmediatamente lo detectará, generando un evento complejo; tal y como se describió en el escenario anterior. Entonces, dicho evento será notificado al doctor, indicándole que este paciente requiere atención urgente. Asimismo, este evento también podría notificarse a actuadores —dispositivos conectados al paciente—, los cuales podrían llevar a cabo una actuación concreta como inyectar una dosis de insulina al paciente ante una subida de azúcar. Este escenario se presenta en la Figura 3.

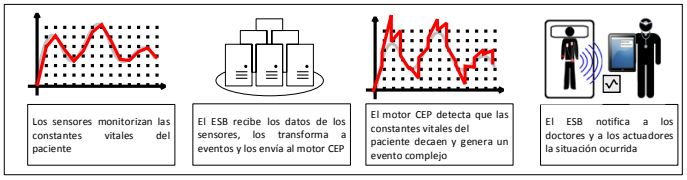

Fig. 3. Detección automática de la bajada de las constantes vitales de un paciente

6. Conclusiones y trabajo futuro

En este trabajo hemos propuesto la integración de una arquitectura de dispositivos orientada a servicios con una arquitectura dirigida por eventos: SODA 2.0. Se trata de una arquitectura novedosa en la que los dispositivos son modelados como servicios y donde las comunicaciones se llevan a cabo a través de eventos de forma desacoplada, haciendo uso de un ESB. Gracias a la integración de esta arquitectura con un motor CEP, es posible detectar en tiempo real situaciones de interés para un dominio en particular, a partir de la información proporcionada tanto por plataformas IoT como por dispositivos simples —TinyOS (sensores TelosB y MicaZ), JavaME (sensores SunSpot) y basados en el estándar IEEE 802.14.5 (Waspmote o Arduino + Xbee)— y dispositivos DPWS (móviles, cámaras, impresoras, etc.). Además, esta arquitectura se ha integrado con un editor de modelado de patrones de eventos (MEdit4CEP), que facilitará a cualquier usuario, experto en un dominio pero no en CEP, la definición de las condiciones que deben cumplirse para detectar situaciones críticas y/o relevantes para el dominio en cuestión. También permitirá la definición de las acciones que se requieran ejecutar una vez acontezcan dichas situaciones. Este editor se encargará de transformar estos patrones a código y de desplegarlo en tiempo de ejecución en el motor CEP y en el ESB. A medida que ocurran nuevas situaciones de interés en tiempo real, el ESB las notificará a las partes interesadas, entre las que destacan los dispositivos DPWS (actuadores), plataformas IoT y consolas de monitorización.

Asimismo, hemos ilustrado la viabilidad de la arquitectura propuesta aplicándola a dos casos de estudio: uno sobre *smart cities* y el otro sobre *ambient assisted living*. Así pues, podemos concluir que es una arquitectura flexible y extensible a distintos dominios de aplicación. Actualmente, se está analizando el desarrollo de la arquitectura aquí propuesta, esperando proporcionar una prueba de conceptos a ser totalmente validada con escenarios reales, como los presentados en este trabajo. Como principal trabajo futuro, se plantea la extensión de nuestra arquitectura con plataformas *Cloud*, principalmente explotando la integración con la plataforma abierta FIWARE, considerando las ventajas de la misma.

Agradecimientos. Trabajo parcialmente financiado por la Red de Excelencia TIN2014-53986-REDT del Ministerio de Economía y Competitividad y por los Proyectos: TIN2012-35669, Ministerio de Economía y Competitividad y FEDER; P11-TIC-7659, Junta de Andalucía; FP7-610531 SeaClouds, Unión Europea.

Referencias

1. DPWS. `http://docs.oasis-open.org/ws-dd/dpws/wsdd-dpws-1.1-spec.html`, [último acceso: 30-06-2015]

2. FIWARE. http://www.fiware.org/, [último acceso: 30-06-2015]
3. HyperCat. http://www.hypercat.io/, [último acceso: 30-06-2015]
4. UPnP. http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v2.0.pdf, [último acceso: 30-06-2015]
5. Boubeta-Puig, J., Ortiz, G., Medina-Bulo, I.: MEdit4CEP: A Model-driven Solution for Real-time Decision Making in SOA 2.0. Knowledge-Based Systems Pendiente de publicación. doi:10.1016/j.knosys.2015.06.021
6. Boubeta-Puig, J., Ortiz, G., Medina-Bulo, I.: A Model-driven Approach for Facilitating User-friendly Design of Complex Event Patterns. Expert Systems with Applications 41(2) (2014)
7. Boubeta-Puig, J., Ortiz, G., Medina-Bulo, I.: Approaching the Internet of Things through Integrating SOA and Complex Event Processing. In: Sun, Z., Yearwood, J. (eds.) Handbook of Research on Demand-Driven Web Services: Theory, Technologies, and Applications, pp. 304–323. IGI Global book series Advances in Web Technologies and Engineering (AWTE), IGI Global (2014)
8. Chandy, K.M., Schulte, W.R.: Event Processing: Designing IT Systems for Agile Companies. McGraw-Hill, Estados Unidos (2010)
9. Cubo, J., Brogi, A., Pimentel, E.: Behaviour-Aware Compositions of Things. In: IEEE International Conference on Green Computing and Communications (GreenCom). pp. 1–8. IEEE (2012)
10. Cubo, J., Nieto, A., Pimentel, E.: A Cloud-Based Internet of Things Platform for Ambient Assisted Living. Sensors 14(8) (2014)
11. Da, Z., Bo, C., Yang, Z., Junliang, C.: Future Service Provision: Towards a Flexible Hybrid Service Supporting Platform. In: IEEE Asia-Pacific Services Computing Conference (APSCC). pp. 226 –233 (Dec 2010)
12. Davis, J.: Open Source SOA. Manning Publications (2009)
13. de Deugd, S., Carroll, R., Kelly, K., Millett, B., Ricker, J.: SODA: Service Oriented Device Architecture. IEEE Pervasive Computing 5(3), 94–96 (Jul 2006)
14. Etzion, O., Niblett, P.: Event Processing in Action. Manning, Stamford, Estados Unidos (2010)
15. Event Processing Technical Society: Event Processing Glossary - Version 2.0. http://www.complexevents.com/wp-content/uploads/2011/08/EPTS_Event_Processing_Glossary_v2.pdf (2010), [último acceso: 30-06-2015]
16. Luckham, D.: Event Processing for Business: Organizing the Real-Time Enterprise. Wiley, Nueva Jersey, Estados Unidos (2012)
17. Moritz, G., Zeeb, E., Prüter, S., Golatowski, F., Timmermann, D., Stoll, R.: Devices Profile for Web Services and the REST. In: 8th IEEE International Conference on Industrial Informatics (INDIN). pp. 584–591 (2010)
18. Naqishbandi, T., Qazi, S.: Big Data, CEP and IoT: Redefining Holistic Healthcare Information Systems and Analytics 4(1) (2015)
19. Papazoglou, M.P., Heuvel, W.J.v.d.: Service Oriented Architectures: Approaches, Technologies and Research Issues. The VLDB Journal 16(3), 389–415 (Jul 2007)
20. Walczak, D., Wrzos, M., Radziuk, A., Lewandowski, B., Mazurek, C.: Machine-to-Machine Communication and Data Processing Approach in Future Internet applications. In: The 8th International Symposium on Communication Systems, Networks Digital Signal Processing. pp. 1–5 (Jul 2012)