Integración de sistemas embebidos basada en Web Services – Primeras experiencias

Nora Blet, Cristina Bender, Gustavo Muro, Rodolfo Recanzone, José L. Simón, José I. Sosa

Facultad de Ciencias Exactas, Ingeniería y Agrimensura, Universidad Nacional de Rosario
{nblet,bender,gmuro, mikerrr, jlsimon, jisosa}@fceia.unr.edu.ar

Resumen. En los últimos años se demostró que es posible implementar Arquitecturas Orientadas a Servicios (SOA) a nivel de dispositivos con recursos limitados utilizando la especificación estandarizada *Devices Profile for Web Services* (DPWS), que los habilita para integrarse uniformemente al mundo de las aplicaciones SOA. Este hecho de enorme relevancia para los sistemas embebidos y la automatización industrial puesto que DPWS apunta explícitamente a dispositivos con restricciones en sus recursos, tiene potencial para cambiar el panorama industrial caracterizado por una gran heterogeneidad de dispositivos computacionales, en especial a nivel de piso de planta manufacturera. El objetivo de este trabajo es investigar la factibilidad de implementar DPWS en dispositivos con recursos limitados, mediante la evaluación del soporte provisto por una plataforma embebida típica en este área de aplicaciones, identificando a través de esta primera experiencia limitaciones y dificultades para su integración con otros dispositivos en un entorno heterogéneo.

Palabras claves: Web Services· DPWS· Sistemas Embebidos· Microcontrolador ARM Cortex M3· Integración de Sistemas.

1 Introducción

El paradigma de las Arquitecturas Orientadas a Servicios (SOA, por *Service Oriented Architecture*) es adoptado cada vez más en el sector de las tecnologías de la información y comunicación, al proveer un *middleware* para la integración de sistemas originalmente independientes, implementado generalmente mediante las tecnologías *Web Services* (WS). A pesar que tal enfoque se limitó hasta ahora al dominio de la gestión empresarial, no es el único ámbito en que las SOA son capaces de proveer una integración uniforme entre aplicaciones software. En los últimos años se observa una creciente disponibilidad de dispositivos embebidos de bajo consumo y costo y, a la vez alta performance, interconectados a través de redes TCP/IP cableadas e inalámbricas. La necesidad que estos dispositivos no sólo se comuniquen sino que interactúen entre ellos a través de interfaces y protocolos de comunicación estándares, produce el advenimiento de SOA a nivel de dispositivos embebidos. Es en el área de la automatización industrial donde se implementa por primera vez, como software *open*

adfa, p. 1, 2011.

source, una pila de protocolos ligera portable directamente a dispositivos embebidos con recursos computacionales limitados, habilitándolos a participar proactivamente en un entorno SOA. Es de esperar, que el piso de planta de las industrias manufactureras del futuro sea orientado a servicios y más aún, para enfrentar los desafíos a los que están sometidos este tipo de empresas, se adopte una infraestructura de comunicación homogénea basada en este paradigma integrando todos los niveles de las mismas. Esto implicaría que las funciones del sistema de automatización industrial serían representadas como servicios. Sin embargo debido a que el software y el hardware utilizados a nivel de piso de planta son significativamente diferentes a aquellos usados en aplicaciones empresariales y, a que los ingenieros de automatización no están familiarizados con esta nueva filosofía, la adopción integral de una orientación a servicios en la industria manufacturera constituye todo un desafío [1, 2].

El objetivo de este trabajo es investigar la factibilidad del uso de WS para la integración de los heterogéneos dispositivos hallados en el piso de una planta manufacturera, generalmente con fuertes restricciones de recursos a fin de garantizar rentabilidad, mediante la evaluación del soporte provisto por una plataforma embebida típica en este tipo de escenario, identificando a través de esta primera experiencia limitaciones y dificultades para su integración con otros dispositivos.

2 SOA a nivel de dispositivos

2.1 SOA en pocas palabras

Existen muchas definiciones de SOA, variando de acuerdo a la formación del autor y dominio de aplicación. Puesto que, este trabajo se enfoca en los dispositivos embebidos típicos usados en piso de planta de una industria manufacturera, los autores adhieren a la definición presentada en la publicación pionera de Jammes y Smit del 2005 [3]: “SOA es un conjunto de principios para construir sistemas autónomos y a la vez interoperables”. Aunque simple y concisa esta definición sin duda incompleta, trata de demostrar que SOA promueve una fluida combinación entre autonomía e interoperabilidad, conceptos contradictorios por definición.

Un servicio se considera autónomo puesto que se crea y opera independientemente de su entorno y es autocontenido. A la vez es interoperable a través de una interfaz que expone su funcionalidad a dicho entorno, abstrayendo por tanto los detalles de su implementación. SOA por definición es agnóstica con respecto a plataformas, lenguajes e implementaciones; justamente el uso de estándares abiertos, en particular aquellos de la familia de los WS, permite implementar SOA de forma neutral con respecto a tecnologías [4]. Esta característica hace a SOA particularmente aplicable a entornos heterogéneos donde la interoperabilidad sea esencial.

2.2 Web Services

Web Services, la más popular de las tecnologías de implementación de las SOA, utiliza un conjunto de protocolos y estándares que permiten el intercambio de datos entre aplicaciones heterogéneas. Estos estándares, básicamente XML, HTTP, SOAP y

WSDL (por *Web Services Description Language*) actualmente son compatibles con la mayoría de las plataformas.

Un interesante poster disponible en `http://www.innoq.com/soa/ws-standards/poster/` provee una visión general del panorama de estándares y protocolos de los WS; en él pueden contarse más de 60 especificaciones y protocolos vinculantes entre dos o más de ellos. La utilización de WS en diferentes áreas promovió la aplicación de especificaciones más concretas para un dominio específico combinadas en los llamados perfiles (*profiles*). Mientras que, el trabajo en el consorcio W3C, se centra en nuevas versiones de las especificaciones WS, existe otra organización independiente que presta más atención a la interoperabilidad, la *Web Services Interoperability Organization*. Esta organización define un perfil como: "Un conjunto de definiciones/especificaciones comúnmente aceptadas por la industria y que apoyan estándares basados en XML, asociadas a un grupo de recomendaciones acerca de cómo deben usarse para desarrollar WS interoperables entre sí".

2.3 Perfil DPWS

Los estándares WS requieren demasiados recursos para poder implementarlos en dispositivos de pequeña escala. Como consecuencia un grupo liderado por Microsoft especifica el perfil DPWS (por *Devices Profile for Web Services*).

DPWS define un conjunto mínimo de funcionalidades que permiten ejecutar WS en forma nativa sobre dispositivos computacionales, formado básicamente por el intercambio de mensajes seguros para proporcionar descubrimiento dinámico, control/ejecución y descripción de servicios, como así también suscripción/notificación de eventos, permitiendo la implementación directa sobre sistemas embebidos en general sin comprometer el cumplimiento de los estándares WS [5].

Debido a que DPWS soporta descubrimiento dinámico (*plug-and-play*) y, es parte de la visión en permanente evolución de las aplicaciones distribuidas que impulsan los estándares WS, su uso en entornos de automatización industrial era previsible. Varios proyectos europeos tales como SIRENA (`www.sirena-itea.org`), SODA (`www.soda-itea.org`) y SOCRADES (`www.socrades.eu`), donde participan ABB, SAP, Schneider Electric y Siemens, están enfocados en proveer una plataforma para implementar la pila de protocolos DPWS sobre los dispositivos embebidos más comunes en el dominio de la automatización industrial.

2.4 Arquitectura DPWS

La pila de protocolos DPWS provee mecanismos de comunicación de alto nivel para permitir interoperabilidad entre dispositivos computacionales. Alineado con los estándares adoptados por los WS y aprobado en 2009 como un estándar OASIS (*Organization for the Advancement of Structured Information Standards*) constituye un enlace entre el mundo de las aplicaciones SOA y el escenario de los sistemas embebidos, ofreciendo a este tipo de sistemas el mismo nivel de interoperabilidad que aquél existente en aplicaciones empresariales.

La pila de protocolos presentada en Fig. 1 muestra cómo DPWS saca partido de protocolos Web tales como TCP, UDP (*unicast* y *multicast*) y HTTP. Para el intercambio de mensajes DPWS utiliza SOAP sobre UDP y SOAP sobre HTTP.

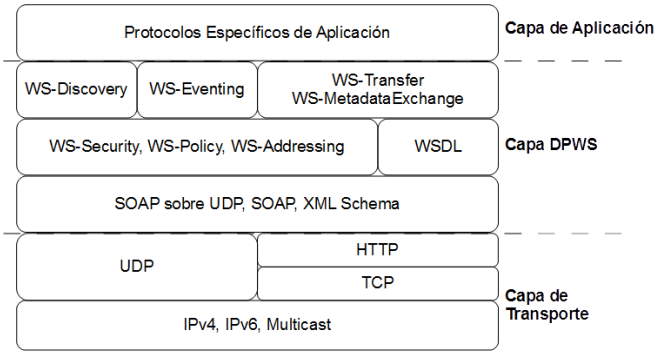

Fig. 1. Pila de protocolos DPWS

Los estándares WS que constituyen la pila de protocolos DPWS pueden considerarse organizados en tres capas principales (no oficial en las especificaciones):

- Capa de mensajes: destinada a encapsular y estructurar los mensajes merced a los protocolos SOAP sobre HTTP, SOAP sobre UDP y XML Schema. WSDL permite describir la interfaz pública de los WS.
- Capa de propiedades: dedicada a suministrar elementos de descripción de los servicios o añadir información complementaria para el enrutamiento de los mensajes. WS-Security provee mecanismos de seguridad a los mensajes SOAP, tales como encriptación y firmas digitales. WS-Policy permite informar sobre políticas inherentes a clientes y servidores. WS-Addresing proporciona un mecanismo por el cual pueden identificarse WS independientemente del protocolo de transporte utilizado.
- Capa de características DPWS: provee las principales características de alto nivel, concretamente: el descubrimiento de dispositivos y detección de servicios *plug-and-play* (WS-Discovery), el intercambio de metadatos para invocarlos (WS-MetadataExchange), la operación Get de WS-Transfer para recuperar todos los metadatos de un servicio o dispositivo y, la notificación de eventos producidos por algún servicio (WS-Eventing). Por cada fase de una conversación entre dispositivos DPWS depende de las dos capas anteriores.

En una arquitectura DPWS los dispositivos pueden adoptar diferentes roles: consumidores de servicios (o clientes), servicios o ambos [6]. DPWS distingue dos tipos de servicios:

- Anfitriones o proveedores de servicios (*Hosting services*): Servicios que alojan a otros servicios; en general representan los dispositivos. Juegan un papel importante en los protocolos de descubrimiento e intercambio de metadatos. Para limitar el

ancho de banda de la red sólo los anfitriones de servicios se anuncian según WS-Discovery.

- Alojados (*Hosted services*): Servicios alojados por un anfitrión; sus tiempos de vida están limitados por los de este último. Son visibles en la red (no encapsulados) y direccionados independientemente de sus anfitriones. Proveen el comportamiento funcional de los dispositivos donde se alojan y dependen de ellos para su descubrimiento [7].

Todos los mensajes del protocolo WS-Discovery son enviados usando UDP (*multicast/unicast*) para no sobrecargar el tráfico de red. En el peor de los escenarios, con el máximo número de mensajes intercambiados, el proceso de descubrimiento de un servicio puede tomar varios segundos; por tanto los clientes suelen acceder a los metadatos de un dispositivo por otros medios sin dejar de ajustarse a DPWS. Por ejemplo usan datos que puedan conocer de antemano o, aplican mecanismos tales como cacheo de metadatos. WS-Discovery contempla el uso de mensajes UDP *multicast* `Hello` cuando un dispositivo se conecta a la red y *multicast* `Bye` cuando la abandona; permitiendo a los clientes actualizar el caché con información sobre los mismos.

En caso de dispositivos con grandes restricciones (caso estático, tal como el contemplado en este trabajo) DPWS asume que el cliente conoce toda la información necesaria para invocar los servicios alojados en un dispositivo y, no es necesario realizar los procedimientos de descubrimiento y obtención de descripciones de dispositivos/servicios. Las últimas versiones de WS-Discovery introducen la noción de mediador entre clientes y dispositivos anfitriones (*Discovery proxy*) con dos objetivos: suprimir el proceso de descubrimiento *multicast* para reducir el tráfico de red utilizando mensajes SOAP *unicast* HTTP en lugar de SOAP sobre UDP y a la vez, permitir al protocolo WS-Discovery extender el alcance de la red más allá de la subred local. El hecho de trasladar a otro dispositivo computacional, la carga de procesamiento generada por la etapa de descubrimiento, permite a aquéllos más limitados economizar recursos.

3 Implementación de WS en un microcontrolador

3.1 Plataforma de desarrollo

Como plataforma de desarrollo para los WS se propone el microcontrolador LPC1769 del fabricante NXP con tecnología ARM Cortex-M3 y sistema operativo de tiempo real FreeRTOS [8]. Constituye una opción económica (aproximadamente US $8) dentro de la gama de procesadores de 32 bits, de gran poder de cálculo, bajo consumo de energía y programable en ANSI C.

Los ARM Cortex-M3 son ideales para las aplicaciones de tiempo real conocidas como "*deeply embedded*" [9] tales como: microcontroladores, aplicaciones automotrices, sistemas de control industrial y aquellas con comunicaciones inalámbricas donde se realicen tareas de monitoreo y control del entorno. Normalmente este tipo de aplicaciones constituyen los bloques básicos de la llamada "Internet de las cosas" [1].

En este trabajo se pretende medir el desempeño de una implementación basada en WS sobre un microcontrolador de relativamente escasos recursos, tal como los que suelen hallarse en el piso de planta de una industria manufacturera. Una performance aceptable en una plataforma con estas características asegura un correcto desempeño en otra con mayores prestaciones. Otro de los motivos por los cuales se escoge esta plataforma es la disponibilidad de un sistema operativo de tiempo real tal como el FreeRTOS que puede utilizarse en combinación con un *stack* TCP/IP ligero, garantizando además la planificación de las distintas tareas que se ejecutarán en orden correcto y en los instantes de tiempo especificados [9].

El LPC1769 se presenta con un procesador ARM Cortex-M3 de NXP que opera a una frecuencia de 120 MHz, memoria de datos SRAM de 64 kB y flash de 512 kB. Entre los periféricos integrados se destacan: interfaz Ethernet, 4xUART, 3xI2C serie, SPI serie, 2xSSP serie, 2xCAN bus interface, PWM modulación por ancho de pulsos, USB 2.0 Device/Host/OTG, RTC, diversos puertos ADC y GPIO, etc. La placa puede alimentarse mediante fuente externa de entre 3.15V y 3.3V o desde el conector USB del *debugger* JTAG.

El LPC1769 es parte de la plataforma de desarrollo del fabricante compuesta por un Entorno Integrado de Desarrollo (IDE, por *Integrated Development Environment*) gratuito basado en Eclipse que se ejecuta en PC con Windows o Linux y, una placa de bajo costo que incluye un *debugger* JTAG (LPCXpresso *target board*). El IDE, LPCXpresso, desarrollado por la empresa Code Red incluye la *toolchain* con un compilador cruzado que corre en PC y genera el archivo binario que ejecutará el LPC1769 [8]. LPCXpresso permite utilizar dos bibliotecas de código C: Redlib, librería propietaria de Code Red que soporta el estándar ISO C90 (con la inclusión de algunas funcionalidades extras) y Newlib, de código GNU preparada para utilizarse en sistemas embebidos; que soporta todo el estándar ISO C99 a expensas de generar código que consume mayor cantidad de recursos (en algunos casos mucho mayor) que Redlib [8]. En este trabajo se optó por Newlib debido a que la herramienta utilizada para implementar WS usa C puro.

Las restricciones de recursos en el microcontrolador hacen que sea prohibitivo el uso de protocolos TCP/IP intensivos en software. En este trabajo se usó la implementación lwIP (por *LighWeight Internet Protocol*) del stack TCP/IP bajo FreeRTOS [9, 10]. LwIP, librería de amplio uso en sistemas embebidos es *open source* y fue desarrollada por Adam Dunkels para controladores con arquitecturas de 8 bits. El diseño de este *stack*, sin dejar de cumplir con el estándar TCP/IP, se enfocó en reducir el consumo de memoria, tamaño de código y potencia computacional requerida, a fin de poder utilizarlo en sistemas embebidos con decenas de kB de RAM y unos 40kB de ROM, con o sin sistemas operativos, en los cuales sería imposible utilizar la API de *sockets* BSD UNIX (BSD, por *Berkeley Software Distribution*), debido al alto nivel de abstracción que provee.

LwIP añade algunos aspectos no tenidos en cuenta en el *stack* de protocolos TCP/IP, por ejemplo provee funciones para asegurar compatibilidad con la librería de *sockets* BSD; característica necesaria en este trabajo.

3.2 Toolkit de desarrollo gSOAP

El *toolkit* gSOAP se elige principalmente por dos motivos: es de código abierto y está escrito en lenguaje C puro lo que facilita migrar un desarrollo a sistemas embebidos [11]. Sólo hay dos herramientas que cumplen con estas premisas, gSOAP y Apache Axis2/C. Mientras que gSOAP soporta y ha sido transplantada a varias plataformas embebidas, Axis2/C se ejecuta principalmente bajo Windows o Linux. Más aún, según las estadísticas gSOAP tiene mejor performance e incorpora características que pueden incluirse o excluirse selectivamente, haciendo que esta herramienta *open source* pueda competir con implementaciones comerciales. Está disponible para su descarga desde SourceForge, incluyendo la guía del usuario (`http://sourceforge.net/projects/gsoap2/files/`) bajo la licencia de código abierto y software libre Mozilla Public License 1.1 (MPL1.1).

gSOAP es un entorno de desarrollo independiente de la plataforma para construir aplicaciones cliente/servidor completas basadas en WS y C/C++. Provee una eficiente técnica de *parsing* de XML además de un compilador RPC (por *Remote Procedure Call*) fácil de usar, que genera las rutinas *stub* (lado cliente) y *skeleton* (lado servidor) y permite integrar aplicaciones C/C++, posiblemente ya existentes, en aplicaciones basadas en WS. Un aspecto único de gSOAP es el mapeo automático de datos de una aplicación C/C++ a tipos XML semánticamente equivalentes y viceversa. Como consecuencia, con una API simple puede obtenerse un desarrollo totalmente compatible con SOAP, concentrándose en la lógica esencial de la aplicación en lugar de tener que lidiar con todos los detalles de SOAP.

Este *toolkit* portable a distintas plataformas, soporta APIs de red que usen sockets BSD, versiones 1.1/1.2 de SOAP y 1.1 de WSDL. Los protocolos que proveen las características de alto nivel de DPWS se implementan sobre gSOAP utilizando el concepto de *plugin*, mecanismo que le proporciona flexibilidad y extensibilidad.

En [11] se ilustran en detalle las etapas de desarrollo e implementación de WS, a partir de un archivo de cabecera C/C++ conteniendo los prototipos o interfaces de las funciones RPC que implementan los WS (Figura 1), ídem para aplicaciones clientes a partir de un archivo de descripción de WS en formato WSDL (Figura 4).

Vale destacar que no es necesario crear el cliente a través de gSOAP para invocar un WS creado por este *toolkit* como así también que puede aplicarse como herramienta de preprocesamiento autónoma o como complemento (*custom-build step*) en algún IDE; en particular en este trabajo se utilizó el IDE Microsoft Visual C++ 2010 Express Edition para implementar el cliente y crear el ambiente de ejecución de los WS.

3.3 Ajustes para la migración de gSOAP a la plataforma de desarrollo

En este trabajo se utiliza la capacidad disponible en gSOAP para crear servidores autónomos (*stand-alone*); los WS se ejecutan como procesos en *background* que atienden peticiones utilizando HTTP y cualquier puerto TCP/IP. Esta solución se conoce como aplicación bajo demanda o *Software as a Service*. En un futuro se plantea utilizar sistemas embebidos para extender este concepto a aquél llamado *Industrial*

Machine as a Service para exponer elementos de la industria manufacturera como servicios.

Para implementar WS en la plataforma LPC1769 utilizando gSOAP y LPCXpresso debieron realizarse ciertos ajustes. gSOAP soporta API de sockets TCP/IP BSD; sin embargo su arquitectura modular permite extender sus capacidades mediante el concepto de *plugin*. Este mecanismo funciona conjuntamente con un grupo de funciones *callback* (punteros a funciones en C/C++) provistas por gSOAP, a implementar por el usuario (detalles en sección 19.38 de la guía de usuario). Se aprovecha esta facilidad para utilizar una API de *sockets* lwIP en lugar de una BSD, para lo cual deben implementarse nueve funciones *callback*. En [13] puede encontrarse el código completo de una aplicación cliente/servidor autónomo y, en la guía de usuario de gSOAP o en ejemplos que trae el *toolkit*, el de un servidor básico modificado para usar *plugin*.

Un grupo de investigación de la Universidad de Rostock crea el *toolkit* WS4D-gSOAP basado en gSOAP [9], que usa su mecanismo de *plugin* para implementar los protocolos que proveen las características de más alto nivel de DPWS: WS-Discovery, WSMetadataExchange/WS-Transfer y WS-Eventing. Bajo este mismo concepto implementan algunas extensiones de apoyo a desarrollos sobre sistemas embebidos. Para el presente trabajo, en particular, se modificó el código de adaptación (disponible en [12]) a una API de *sockets* lwIP [15]. Adicionalmente debieron utilizarse algunas directivas de compilación particulares tanto para eliminar las características no esenciales en aplicaciones sobre dispositivos con poco espacio de memoria, como así también para eliminar la necesidad de enlazar el código de la aplicación con la librería de sockets BSD (secciones 19.32 y 19.33, respectivamente).

Excepto la tesis de maestría de G. B. Machado [13], no existe ninguna publicación donde se impartan pautas para la migración de gSOAP a plataformas que no estén basadas en Linux. Los autores proponen una arquitectura de integración de embebidos con otros sistemas a través de WS utilizando gSOAP y la plataforma SHIP. Relatan con amplio detalle las modificaciones realizadas al *firmware* del microcontrolador y código fuente del *runtime* de gSOAP para adaptarlos a las prestaciones de la plataforma utilizada. En realidad, las implementaciones WS sobre microcontroladores de la literatura están casi todas realizadas sobre plataformas de muchas mayores prestaciones; la mayoría usan procesadores ARM9 y, Linux Embebido para el cual la implementación es directa o, .NET Micro Framework que incluye el *stack* DPWS completo facilitando el desarrollo de aplicaciones [5].

Durante la implementación de los WS sobre la plataforma elegida surgieron algunas dificultades en cuanto a los ajustes necesarios para la migración de gSOAP al LPC1769, complicando aún más la de por si ardua tarea de desarrollo sobre sistemas embebidos. La curva de aprendizaje de DPWS fue más grande de lo esperado: debido a la naturaleza altamente flexible y extensible de las especificaciones resulta difícil determinar la funcionalidad básica requerida; cada especificación de protocolos, en constante evolución, referencia a las de otros. Falta documentación que pueda considerarse una guía con autoridad como así también información adicional sobre implementaciones en dispositivos con recursos limitados que no estén basadas en Windows/Linux. Lo mismo ocurre con las herramientas de implementación, en sus primeras etapas de desarrollo, sin suficiente solidez y con lógicas inestabilidades.

4 Futuro escenario de validación

En este trabajo, en una siguiente etapa se utilizará como escenario de validación de DPWS, réplicas a pequeña escala de procesos industriales con las cuales se cuenta, conformadas por celdas de trabajo controladas por PLCs con capacidad de comunicación mediante protocolo PROFIBUS DP, diseñadas con el objetivo de trabajar cooperativamente en forma integrada en una pirámide CIM [1]. Actualmente se cuenta con dos celdas: una destinada a la transferencia de piezas mediante cintas transportadoras y otra a la dosificación del llenado de tanques desde un depósito, las cuales pueden visualizarse en Fig. 2 y se describen con mayor detalle en [2]. A nivel de campo, están en etapa de desarrollo dos paneles táctiles que oficiarán de HMI, conectados a cada PLC de las celdas mediante una red PROFIBUS. Ídem a nivel de control que contará con un PLC de alta gama y a nivel de gestión con un SCADA ejecutándose en una PC conectada a una red Ethernet.

Fig. 2. Modelos de celdas de trabajo

5 Resultados preliminares

En esta primera experiencia se implementaron exitosamente WS en el LPC1769 usando gSOAP. Como cliente de los mismos se utilizó una PC con procesador AMD Sempron 145, 2.8 GHz con Windows 7 Professional; conectados mediante una red Ethernet de 100 MBps. Los resultados preliminares se obtuvieron a través de peticiones de servicios SOAP/XML-RPC de la herramienta JMeter 2.9 ejecutándose en la PC cliente [13]. Se utiliza como métrica de rendimiento el tiempo de respuesta a una petición de servicio por parte del cliente. El propósito del experimento es obtener indicadores de aplicabilidad de una técnica de interface (WS) sin especificación de tiempo real y creada para plataformas informáticas con importantes recursos de cómputo, memoria, etc., en aplicaciones con restricciones de tiempo real basadas en

sistemas embebidos y la habilidad de una plataforma particular (LPC1769) para su utilización.

Los servicios ensayados en este trabajo son los que ofrece una calculadora expuesta como WS extraída de los ejemplos que acompañan la distribución de gSOAP, en este caso versión 2.8.14 [13]. Cuando un cliente requiere una de las cinco operaciones matemáticas básicas de la calculadora: suma, resta, multiplicación, división y potencia se envía un mensaje SOAP al LPC1769 el cual tiene implementado dichos servicios; éste realiza la operación y replica con un mensaje SOAP conteniendo el resultado de la misma, luego de lo cual queda a la espera de otra solicitud de servicio. La aplicación cliente también se desarrolló en C usando gSOAP [13] y el IDE Microsoft Visual C++ 2010 Express Edition, ejecutando para estas pruebas la solicitud de la operación potencia, usando valores aleatorios entre 1 y 1000 tanto para la base como para el exponente.

Cuando un único cliente realiza requerimientos de servicios a través de una petición SOAP/XML-RPC cuyo tamaño es de 430 bytes, el tiempo de respuesta promedio es de 8 ms con muy pequeñas desviaciones entre muestras (0.96). Se simula un escenario frecuente en piso de planta donde, un sistema embebido es encuestado desde una PC sobre una red TCP/IP usando la especificación y protocolos WS.

También se simuló el caso de dos clientes, a fin de estimar la capacidad de procesamiento en situaciones de sobrecarga, para lo cual se configuró JMeter de forma tal que deje un intervalo de: 0.5 s, 1 s, 2.5 s y 5 s entre cada petición, repitiendo este proceso 20 veces para extraer datos estadísticos.

Tabla 1. Ensayos realizados. Caso 2 clientes solicitando servicios al LPC1769

Frecuencia de las peticiones (seg)	Tiempo de respuesta min. (ms)	Tiempo de respuesta max. (ms)	Media (ms) de 20 muestras	Desvío Estandar
0.5	7	15	9	2.13
1	7	11	8	1.04
2.5	6	26	9	4.15
5	6	14	7	1.81

En tabla 1 pueden observarse los resultados obtenidos. En el contexto de los proyectos SIRENA y SOCRADES se midió la performance alcanzada por el *stack* DPWS usando dispositivos basados en ARM9 en una red Ethernet 100 Mbps, con menos de 10 dispositivos conectados a un *switch* con topología estrella. Se comprobó que los tiempos de respuesta a una petición de servicio están en el orden de las decenas de milisegundos con una alta variabilidad en los tiempos de ejecución de un servicio [16]. Finalmente, en un último ensayo realizado para el caso de 2 clientes en condiciones más exigidas con peticiones instantáneas de ambos (cada 0 s.), pudo observarse dicha variabilidad: la mayoría de los requerimientos son procesados en 10ms como máximo y uno de ellos en particular (test Nº 11 de un total de 20) en más de

3000 ms, pudiendo en algunos de los repetidos ensayos considerar JMeter que hubo un fallo de atención de la petición.

Los resultados iniciales confirman que actualmente DPWS sólo es viable para aplicaciones sin restricciones temporales (*best-effort*) o con restricciones temporales suaves [14], [16]. Las actividades con restricciones temporales suaves deben completarse antes de cierto *deadline*, si ello no es así no ocurre nada catastrófico; la calidad de servicio brindada por dicha actividad depende de la frecuencia de incumplimiento de dichos tiempos límites. Ejemplos en automatización de este tipo de actividades: *streaming* de video para monitoreo de procesos industriales, HMI, diagnóstico y mantenimiento. La recolección de datos estadísticos para poblar una base de datos sobre la eficiencia de algún dispositivo de automatización, es un ejemplo de actividad que no tiene restricciones temporales (*best-effort*) y donde, tiempos de respuesta del orden de los segundos son usualmente aceptables. Algunas actividades con requerimientos temporales estrictos pueden acomodarse dentro de las capacidades actuales de los WS, por ejemplo el control directo está limitado a la capacidad de arrancar/detener el ciclo automático de un proceso o cambiar el modo de trabajo de automático a manual.

6 Consideraciones finales

El presente trabajo pretendió demostrar la factibilidad de implementar WS en dispositivos embebidos. Cabe remarcar que, actualmente y hasta donde llega el conocimiento de los autores, no se encuentra en la literatura ninguna publicación en la que se considere la implementación de DPWS sobre microcontroladores basados en ARM Cortex-M3 con FreeRTOS y stack TCP/IP lwIP.

Los resultados iniciales muestran que la aplicabilidad de SOA a nivel de dispositivos embebidos depende de la especificación de tiempo real del sistema, lo que implica la necesidad de evaluar la viabilidad de usar esta filosofía en una aplicación de tiempo real antes de desarrollarla.

Referencias

1. Blet, N. Simón, J.L.: SOA in industrial automation for SMEs. In IJIE Iberoamerican Journal of Industrial Engineering. Ed. Universidade Federal de Santa Catarina, Vol. 3, Nº. 2, pp 190-208 (2011）
2. Recanzone, R. R., Sosa, J. I., Bender, C., Blet, N., Simón, J. L.: Experiencias en la enseñanza de la Informática Industrial en una carrera de Ingeniería Electrónica. VIII Congreso de Tecnología en Educación y Educación en Tecnología, TE & ET 2013, Red de Universidades con Carreras en Informática (2013)
3. Jammes, F., Smit, H.: Service-oriented paradigms in industrial automation. IEEE Transactions on Industrial Informatics, vol. 1, pp. 62–70 (2005)
4. Cândido, G. M.: Service-oriented Architecture for Device Lifecycle Support in Industrial Automation. Phd diss., Fac. de Ciências e Tecnologia da Univ. de Nova de Lisboa (2013)
5. Dias, R. A., Mendonca I. T. M., Regis, A.: Integrated Manufacturing Management using Internet of Things. International Journal of Computer Applications, vol. 51 Nº 11, pp.20-25. Foundation of Computer Science, New York, USA (2012)

6. Araújo, G. M., Siqueira, F.: The device service bus: a solution for embedded device integration through web services. Proceedings of the 2009 ACM symposium on Applied Computing, pp. 185-189. Honolulu, Hawaii: ACM, (2009)
7. Bangemann, T., Diedrich, C., Riedl, M. Wuwer, D., Harrison, R., Monfared, R. P.: Integration of Automation Devices in Web Service supporting Systems, 30th IFAC Workshop on Real-Time Programming, pp. 161-166 (2009)
8. Espósito, J. E.: Diseño, implementación y validación de una biblioteca para algoritmos de control para sistemas embebidos. Fac. de Ingeniería, UBA. Tesis de grado (2013)
9. Moritz, G., Prüter, S., Timmermann, D., Golatowski, F., Web services on deeply embedded devices with real-time processing. Emerging Technologies and Factory Automation . IEEE International Conference on , Vol., No., pp.432-435, (2008)
10. Dunkels, A.: Full TCP/IP for 8-Bit Architectures. International Conference On Mobile Systems, Applications And Services , pp. 85-98, San Francisco, California (2003)
11. Van Engelen, R., Gupta, G., Pant, S.: Developing web services for C and C++. Internet Computing, vol. 7, no. 2, pp. 53–61 (2003)
12. Plugin para lwip. `https://trac.e-technik.uni-rostock.de/svn/ws4d-gsoap/trunk/src/alt-io/lwip/gsoap-lwip-io.c`
13. Machado, G. B.: Uma arquitetura baseada em web services com diferenciação de serviços para integração de sistemas embutidos a outros sistemas. Universidade Federal de Santa Catarina, Dissertação de Mestrado. (2006)
14. Jammes, F., Mensch, A., Smit, H.: Real-time performance Web Services using EXI. In proceeding of: IEEE IECON (2011)
15. Moritz, G., Zeeb, E., Golatowski, F., Timmermann, D., Stoll, R.. Web services to improve in-teroperability of home healthcare devices. In Proceedings of the 3rd International Conference on Pervasive Computing Technologies for Healthcare, pp. 1–4 (2009)
16. Checcozzo, R. Rusina, F., Mangeruca, L., Ballarino, A., Abadie, C., Brusaferri, A., Harrison, R., Monfared, R.: RI-MACS: an innovative approach for future automation systems. Int. Journal of Mechatronics and Manufacturing Systems , Vol. 2, Nº. 3 (2009)