Implementación del Estándar ISO/IEEE11073 en Dispositivos de Salud Personal Inalámbricos: Lecciones Aprendidas

M. Martínez-Espronceda Cámara[1], I. Martínez Ruiz[2], S. Led Ramos[1], L. Serrano Arriezu[1], J. D. Trigo Vilaseca[2], A. Marzo Pérez[1], J. Escayola Calvo[2], H. G. Barrón[3], J. García Moros[2]

[1] Dep. Ingeniería Eléctrica y Electrónica - Univ. Pública Navarra (UPNA), Campus de Arrosadía s/n. 31006 – Pamplona, {miguel.martinezdeespronceda,lserrano,sled,amarzo}@unavarra.es

[2] Instituto de Investigación en Ing. Aragón (I3A) - Univ. Zaragoza (UZ), c/ María de Luna, 1. 50018 – Zaragoza, {imr,jtrigo,javier.escayola,jogarmo}@unizar.es

[3] Dep. de Electrónica y Comunicaciones, Univ. Autónoma de Nuevo León, Pedro de Alba s/n, Ciudad Universitaria-San Nicolás De Los Garza, – N.L. México, {hector.barrongn}@uanl.edu.mx

Resumen

En este artículo se presentan diferentes formas de implementar el estándar ISO/IEEE11073 en dispositivos de salud personal con requerimientos de bajo consumo de potencia. Para ello se aplica la metodología de patrones planteada por los autores previamente proponiendo cuatro posibilidades de implementación: una implementación basada en un solo hilo empleando funciones no bloqueantes, una implementación basada en un Sistema Operativo en Tiempo Real (RTOS), una implementación empleando un desarrollo ad-hoc multihilo y una solución basada en código interpretado.

1. Introducción

En los últimos años han aparecido una serie de nuevas soluciones y servicios basados en la aplicación de las Tecnologías de la Información y las Comunicaciones (TICs) a los servicios clásicos de cuidados asistenciales. El así llamado empoderamiento del paciente [1] permite a éste controlar su salud y, además, mejora su calidad de vida. Desde el punto de vista empresarial, estos nuevos servicios suponen un nuevo mercado que puede conducir a un nuevo motor de crecimiento económico [2]. No obstante, estos servicios centrados en el paciente deben ser implantados de forma consensuada, y ordenada [3]. Como respuesta, algunos estándares están siendo desarrollados. En el caso de las normas de información en salud personal, un campo importante de desarrollo en los últimos años está siendo el estándar de interoperabilidad para comunicaciones de dispositivos médicos ISO/IEEE11073 (X73) [4], [5]. Esta norma, orientada inicialmente a escenarios del Punto de Cuidado (*Point of Care*, PoC) y Unidades de Cuidados Intensivos (UCIs) (X73PoC) [6], ha evolucionado recientemente a entornos de Salud Personal (X73PHD) [7] impulsada por la industria. Debido al uso de tecnologías de baja tensión y potencia (*Low Voltage Low Power*, LV-LP), al uso de sistemas inalámbricos como Bluetooth [8] y ZigBee [9] y al uso de baterías en dispositivos médicos y de salud personal por sus requerimientos de ergonomía, reducir el uso de los recursos de procesador y necesidades de memoria resulta imprescindible.

Por ello es común encontrar limitaciones tanto en recursos hardware como las posibilidades del software de estos dispositivos. Por poner ejemplo más concreto, en gran parte de aplicaciones de ZigBee empleando sistemas *System-on-Chip* (SoC), el espacio de memoria de acceso aleatorio (RAM) disponible para la aplicación queda reducido a unos pocos kilobytes al estar compartida con con la pila de ZigBee. Teniendo en cuenta que además, en función de la especialización X73 el tamaño de los paquetes (*Application Protocol Data Units*, APDU) empleados puede llegar hasta los 63KB y 8 KB, respectivamente, se hace evidente la complejidad a la hora de implementar X73 [10], [11] en estos dispositivos.

Con el fin de optimizar el uso de los recursos de dispositivos o sensores médicos (a.k.a agentes), y teniendo en cuenta la reducción el consumo de energía, nuestro grupo propuso una metodología de implementación de agentes conformes a X73 basada en patrones [12]. Gracias a ella, el procesamiento de X73 resulta ser muy eficiente en términos de memoria y procesador. A modo de ejemplo, la generación de un APDU X73 saliente utilizando algunos patrones y algunas variables en la memoria RAM se muestra en la Figura 1. Una prueba de concepto de la metodología se hizo en el proyecto INTENSA [13]. Hay diferentes enfoques para aplicar la norma según está metodología basada en patrones.

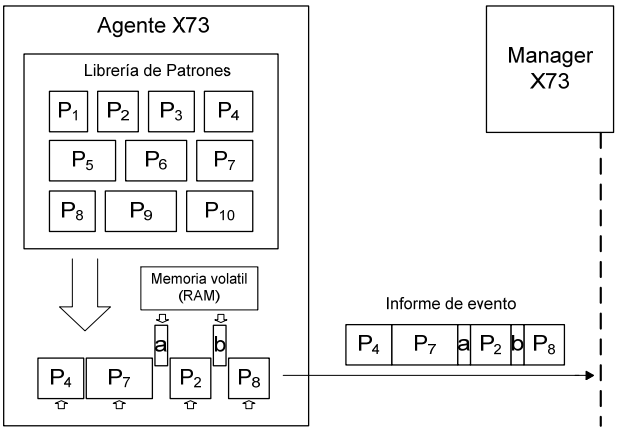

Figura 1.*Síntesis de una APDU usando una técnica de patrones*

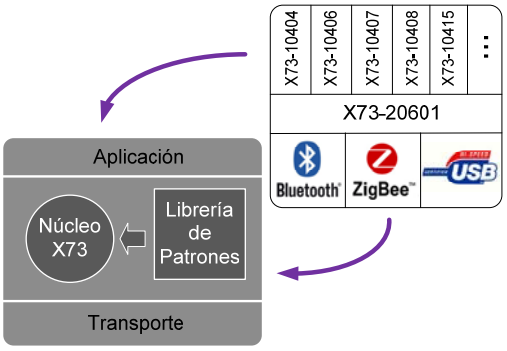

Figura 2. *División en capas de X73PHD y división en bloques de la arquitectura empleada en las propuestas*

El trabajo se organiza de la siguiente manera. En la Sección 2, se presenta primeramente la arquitectura empleada, para entonces, proponer cuatro alternativas de implementación que incluyen hilo único con llamadas no bloqueantes, un sistema basado en un sistema operativo en tiempo real (*Real Time Operating System*, RTOS), una implementación multihilo específica, y una solución basada en código interpretado. En la Sección 3 se presenta una discusión y comparación cualitativa de los enfoques. Finalmente se presentan las conclusiones en la Sección 4.

2. Metodología

La arquitectura sigue la división conceptual de bloques propuesta en X73PHD (Figura 2) que incluye a groso modo la capa de aplicación, la capa de intercambio y la capa de transporte. Esta última soporta los protocolos Bluetooth usando el perfil de salud personal (*Health Device Profile* – HDP), USB usando la clase de dispositivo de salud personal (*Personal Healthcare Device Class* – PHDC) y ZigBee usando el perfil de salud personal (*Health Care Profile* – HCP). Internamente la capa de intercambio se divide en la librería de patrones [12] que contiene los patrones necesarios para la manipulación de APDUs y el núcleo X73 que se encarga de la síntesis y el análisis de los APDUs. Tanto el núcleo X73 como la librería de patrones pueden implementarse de muy diversas maneras. En lo restante de esta sección se presentan aquellas que el grupo considera más relevantes considerando requerimientos de recursos, tiempo de desarrollo y consumo de potencia.

2.1. Hilo único usando llamadas no bloqueantes

Esta solución emplea un buffer para recepción de APDUs y otro para transmisión de las mismas. En el caso del buffer de recepción, se emplea para almacenar un APDU hasta que éste está completo, llamando en ese instante a la función del núcleo X73 que se encarga de procesarlo. Similarmente, en el caso del buffer de transmisión, cuando se necesita enviar un APDU, éste llama a una función del núcleo X73 que sintetiza el APDU y lo encola en este buffer. El procesamiento de un APDU cambia el estado de la máquina de estados finitos (*Finite State Machine* – FSM), informa a la capa de aplicación y, en caso de ser un servicio que requiere una respuesta, genera un nuevo APDU. En la Figura 3 se muestra un ejemplo de implementación con parte del patrón de petición de asociación y alguna de las funciones del núcleo X73 encargadas del procesamiento del mensaje de respuesta de la petición de asociación. Un buen número de agentes pueden implementarse siguiendo este modelo, que además de ser sencillo en su comprensión no supone mayor complejidad a la hora de implementarlo en código.

Sin embargo, se pueden encontrar situaciones en las que por limitaciones de memoria este tipo de implementación no es factible. De hecho es necesario un análisis de la RAM de la que dispone el dispositivo. En el caso más general será necesario una cantidad mínima recomendada de RAM de 8 y 63 KB para los buffers de recepción y transmisión, respectivamente. En el caso de dispositivos con poca RAM se debe recurrir a otro tipo de técnicas como las que se presentan a continuación.

2.2. Implementación basada en un sistema operativo en tiempo real (Real Time Operating System – RTOS)

En este caso, la capa de intercambio se ejecuta como varios procesos independientes. En concreto se asignan dos procesos a cada canal X73. Uno de ellos se ocupará de la síntesis y transmisión de APDUs salientes. El otro de la recepción y análisis de APDUs entrantes. Estas tareas procesan los APDUs por partes (byte a byte) bloqueándose en caso necesario. De esta forma se reduce considerablemente la RAM dedicada a los buffers de transmisión y recepción del caso 2.1.

Al mismo tiempo que la tarea de recepción recibe y analiza un APDU, ésta extrae la información necesaria para su posterior procesamiento (Figura 4). Una vez ha recibido la APDU por completo, se envía una notificación al módulo FSM. El módulo FSM a su vez realiza los cambios de estado necesarios, notifica a las tareas de la capa de aplicación y, si es necesario, solicita a la tarea de envío que transmita un nuevo APDU. Dicha solicitud lleva consigo toda la información necesaria para que la tarea de envío pueda sintetizar el APDU. Tanto la tarea de envío como la de recepción emplean la librería de patrones para el procesamiento de los APDUs.

```
const byte_t pttrn_assoc_req[] = {
    0xE2 0x00  // APDU CHOICE Type (AarqApdu)
    0x00 0x32  // CHOICE.length = 50
    0x80 0x00 0x00 0x00  // assoc-version
    ...
}
const byte_t pttrn_assoc_rsp_0[] = {
    0x00, 0x2C              //   CHOICE.length = 44
}
...
matchAssocResp(APDU_buff * apdu) {
    x73_memcmp(apdu->buff,pttrn_assoc_rsp_0,
               sizeof(pttrn_assoc_rsp_0));
    ...
    x73_recv16(result);
    x73_ready(ST_RECV_ASSOC_RESP, result);
    return result;
}
```

Figura 3. *Ejemplo de implementación: patrón de petición de asociación y función de procesado de respuesta de la petición de asociación*

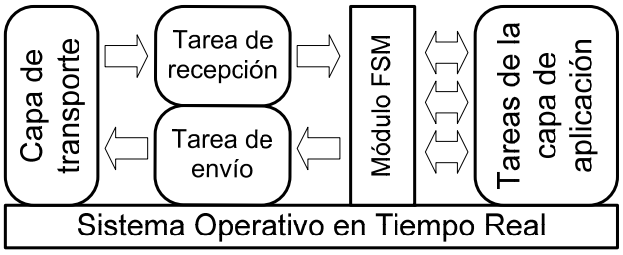

Figura 4. *Implementación usando un RTOS*

Nótese que X73 impone que los cambios de estado debidos a la transacción de un APDU deben ser transaccionales (todo o nada). De esta forma las tareas de recepción y transmisión deben de ser programadas de una forma particular. La tarea de recepción debe notificar al módulo FSM únicamente cuando se recibe una APDU completa (o si se produce un error en la recepción), mientras que la tarea de transmisión debe de tomar toda la información para sintetizar la APDU de una sola vez, justo antes de enviar la APDU.

El uso de un RTOS tiene ciertos inconvenientes que en algunos casos son irrelevantes pero en otros tienen una importancia crítica. Estos son el tamaño del programa (*footprint*), debido a que comúnmente un RTOS incorpora gran cantidad de funcionalidades, y el aumento del consumo de energía, debido a los procesos en segundo plano, necesarios para el RTOS como por ejemplo el mantenimiento del reloj de sistema. Ciertamente, dependiendo del acceso que se tenga a la configuración/código del RTOS estos inconvenientes se puede reducir e incluso eliminar.

2.3. Implementación basada en tareas colaborativas

En el caso anterior se ha incorporado un RTOS para proporcionar servicios de procesado paralelo (múltiples tareas) y temporización, necesarios para la implementación de X73. La siguiente aproximación consiste en realizar una implementación a medida (*ad hoc*) de estos dos servicios, teniendo como objetivo la reducción del consumo de procesamiento y memoria. Esto es posible debido a que las tareas tienen una naturaleza colaborativa (las tareas ceden el procesador voluntariamente a otras tareas), es decir no necesitan un algoritmo de planificación de procesador (*multithreading scheduler*) preventivo (*preemptive*) sino colaborativo (*collaborative*).

Esta aproximación emplea la misma separación de tareas propuesta en la aproximación usando un RTOS (Figura 5). La diferencia más significativa es el reemplazo de estos dos servicios mencionados por servicios a medida. Dada la sencillez de estos dos servicios esta solución resulta muy versátil y puede ser fácilmente integrada en cualquier entorno de desarrollo. De hecho esta implementación puede incluso ser empleada junto con un sistema operativo.

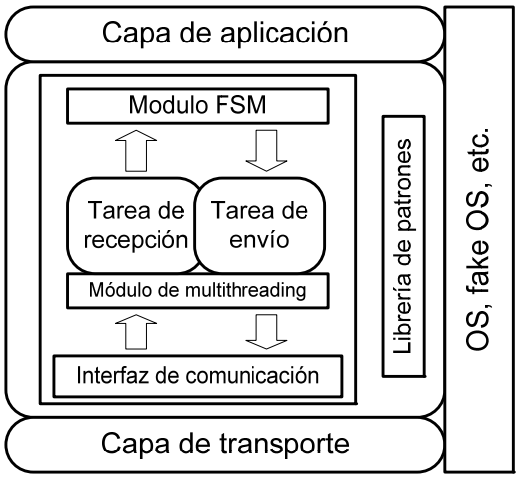

Figura 5. *Implementación usando una estrategia multihilo colaborativo*

2.4. Implementación basada en código interpretado

Hasta ahora las soluciones propuestas se compilan partiendo de un lenguaje genérico (en la mayor parte de los casos C) y el programa se ejecuta nativamente en lenguaje máquina. En la siguiente propuesta se da un paso en otra dirección. La idea consiste en usar una serie de instrucciones de alto nivel (Figura 6) en forma de código interpretado (al estilo de una máquina virtual especializada – *Virtual Machine*, VM) las cuales representan a cada una de las funciones de alto nivel bloqueantes. Por ejemplo, en el caso más general el código que se implementa nativamente incluye lazos, saltos y bifurcaciones, acceso a variables, operaciones matemáticas, llamadas a funciones externas, etc. La parte codificada en código interpretado incluye el envío y comparación de un patrón, recepción de bytes en un buffer, etc. Las instrucciones en código interpretado, con un diseño óptimo, permiten la reducción del tamaño total de memoria empleada por el programa. Puesto que existe un compromiso entre velocidad de ejecución y tamaño del programa, el diseño del código interpretado puede depender del dispositivo a implementar.

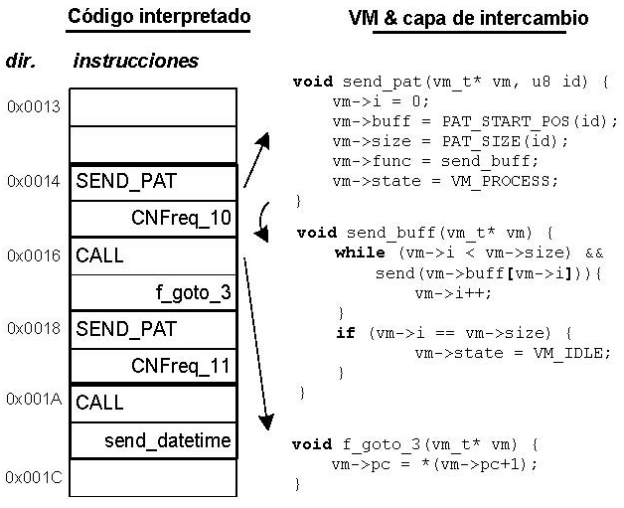

Figura 6. *Implementación basada en código interpretado*

3. Discusión

Una comparativa cualitativa entre las diferentes propuestas recogidas en este artículo se presenta en la Tabla 1. Los cuatro enfoques aparecen en el mismo orden en que han sido expuestos en el trabajo. En la tabla, más cruces significan que la aplicación consume más recursos (memoria, procesador, tiempo de desarrollo o consumo de energía). Los valores de esta tabla son el resultado de varias implementaciones experimentales. Sin embargo como los autores no tienen resultados cuantitativos estos deben ser tomados como indicativos.

La Tabla 1 muestra las diferencias entre las posibles implementaciones. Como se puede observar, el enfoque que requiere menos memoria es el basado en código interpretado. Sin embargo este enfoque es el más costoso con respecto al tiempo de desarrollo. El uso de procesador y consumo de energía es inferior en tanto en multithread *ad hoc* como en el monoproceso. Este último enfoque es la técnica con la que es más fácil implementar el código. El más versátil entre los cuatro enfoques parece ser el multithread *ad hoc*. Se están realizando algunos estudios por nuestro grupo para reducir el tiempo de desarrollo de esta solución. Las alternativas aquí expuestas han sido elegidas como las más factibles de una colección que incluye tanto técnicas que se han estudiado pertenecientes a proyectos a los que el grupo ha tenido acceso como a propuestas individuales de los miembros del grupo. Aunque estas técnicas cubren las aplicaciones más comunes, puede haber otros enfoques diferentes que pueden ser más adecuados. Por lo tanto, cada caso debe ser estudiado a fondo por el desarrollador de dispositivos inalámbricos.

Técnica	Necesidades de memoria	Necesidades de Procesador	Tiempo de desarrollo	Consumo de energía
2.1	++++	+	+	+
2.2	+++	++	++	++
2.3	++	+	+++	+
2.4	+	++++	+	++

Tabla 1. *Comparativa de las alternativas propuestas en este artículo*

4. Conclusiones

En este trabajo se han presentado cuatro propuestas diferentes que permiten implementar la norma X73 en sistemas inalámbricos con requerimientos de LV-LP basados en microcontroladores, utilizando para ello la metodología de los patrones. Se muestran las diferencias entre ellos en forma cualitativa y se concluye que hay un compromiso entre recursos de memoria, uso de procesador, tiempo de desarrollo y consumo de energía, lo que indica que la idoneidad de cada uno para implementar X73PHD en un dispositivo específico depende de sus características. El trabajo realizado marca una vía que facilita la adopción e implementación del estándar por parte del fabricante.

Agradecimientos

Este trabajo ha sido parcialmente subvencionado por los proyectos TIN2008-00933/TSI y TIN-2009-08414 de la Comisión Interministerial de Ciencia y Tecnología (CICYT) y Fondos Europeos para el Desarrollo Regional (FEDER), y TSI-020302-2009-7 y TSI-020302-2009-89 Plan Avanza I+D del Ministerio de Industria, Turismo y Comercio.

Referencias

[1] Monteagudo JL y Moreno O, *ehealth for patient empowerment in europe*, 2009, http://ec.europa.eu/information/society/newsroom/cf/itemdetail.cfm?item_id=3448 (Consultada:04/10)

[2] Carroll R, Cnossen R, Schnell M, y Simons D, *Continua: An interoperable personal healthcare ecosystem*, IEEE Pervasive Computing, vol. 6, no. 4, pp. 90–94, 2007.

[3] Wartena F, Muskens J, y Schmitt L, *Continua: The impact of a personal telehealth ecosystem*, Libro de Actas - International Conference on eHealth, Telemedicine, and Social Medicine, eTELEMED, 2009, pp. 13–18.

[4] Galarraga M, Serrano L, Martinez I, Toledo PD, y Reynolds M, *Telemonitoring systems interoperability challenge: An updated review of the applicability of iso/ieee 11073 standards for interoperability in telemonitoring*, Libro de Actas – Annual Int Conf IEEE EMB, 2007, pp. 6161–6165.

[5] Clarke M *et al*, *Developing a standard for personal health devices based on 11073*, Libro de Actas – Annual Int Conf IEEE Eng in Medicine and Biology (EMBC), 2007, pp. 6174–6176.

[6] Lutter N, et al, *Point of care: New connectivity standards and novel technologies in intensive care*, International Journal of Intensive Care, vol. 12, no. 4, pp. 175-185, 2005

[7] Schmitt L, Falck T, Wartena F, y Simons D, *Novel iso/ieee 11073 standards for personal telehealth systems interoperability*, 2007 Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability - HCMDSS-MD PnP '07. IEEE, 25-27 Junio 2007 2008, pp. 146–8.

[8] Wang WS, *Bluetooth: A new era of connectivity*, IEEE Microwave Magazine, vol. 3, no. 3, pp. 38–42, 2002.

[9] Kinney P, *Zigbee technology: Wireless control that simply works*, Communications Design Conference, 2003.

[10] Warren S, Lebak J, y Yao J, *Lessons learned from applying interoperability and information exchange standards to a wearable point-of-care system*, 1st Transdisciplinary Conference on Distributed Diagnosis and Home Healthcare, vol. 2006, 2006, pp. 101–104.

[11] Martínez-Espronceda et al, *Implementing iso/ieee 11073: Proposal of two different strategic approaches*, Annual Int Conf IEEE Eng in Medicine and Biology (EMBC) - Proceedings, 2008, pp. 1805–1808.

[12] Martínez-Espronceda et al, *Standard-Based Homecare Challenge: Advances of ISO/IEEE11073 for u-Health*, Handbook of Digital Homecare, 2009, pp. 179–202, doi:10.1007/978-3-642-01387-4.

[13] Martínez-Espronceda et al, *Intensa: Heart failure patient's follow-up system using the iso/ieee11073 standard*, Libro de Actas - 9th International Conference on Information Technology and Applications in Biomedicine, ITAB, 2009.

[14] *X73 agent and manager implementation source-code*, World Wide Web electronic publication, 2009, http://www.freescale.com/webapp/sps/site/prodsummary.jsp?code=MEDICALUSB, (consultado: Agosto 2010)