# Framework-Oriented Approach to Ease the Development of Ambient Assisted-Living Systems (Enfoque Orientado al Marco para Facilitar el Desarrollo de Sistemas de Vida Asistida)

**Abstracto:**

Los sistemas de vida asistida ambiental (AAL), que incluyen dispositivos portátiles y domótica, podrían mejorar la calidad de vida de las personas frágiles. Como se analiza en este documento, la arquitectura típica de los sistemas AAL se basa en redes de sensores inalámbricos y puertas de enlace patentadas y puede ser compleja, costosa y apenas incluye dispositivos interoperables de múltiples proveedores. Las pasarelas emergentes basadas en Android pueden ayudar a reducir los costos, pero los problemas de flexibilidad e interoperabilidad aún permanecerán. Sobre la base de este hecho, se presenta el desarrollo de un sistema AAL mediante un enfoque orientado al marco. Gracias al uso del marco SAndroidE, el enfoque propuesto conduce a la reducción del esfuerzo de programación y mejora la interoperabilidad entre los diferentes proveedores, lo que garantiza altos niveles de personalización. Los resultados obtenidos en un caso de uso típico indican que el esfuerzo de programación para integrar dispositivos de múltiples proveedores se reduce en un factor de diez. El enfoque presentado allana el camino para la implementación de nuevos sistemas abiertos AAL de múltiples proveedores, también fomenta las arquitecturas redundantes y mejora la solidez de estas soluciones emergentes.

**Publicado en:** IEEE Systems Journal (Volumen: 13, Número: 4, diciembre de 2019)

**Página (s):** 4421 - 4432

**Fecha de publicación:** 16 de julio de 2019

**Información de ISSN:**

**Número de acceso de INSPEC:** 19157347

**DOI:** [10.1109/JSYST.2019.2924150](https://doi.org/10.1109/JSYST.2019.2924150)

**Editorial:** IEEE

**Agencia fundadora:**

**SECCIÓN I.**

## **Introducción y motivaciones**

En las últimas décadas, el promedio de vida ha aumentado, especialmente en los países desarrollados [1]. Esto se debe principalmente a los avances en el conocimiento y las tecnologías médicas, mejores condiciones de estilo de vida y una actitud más centrada en la prevención de enfermedades [2]. El aumento de la vida útil promedio es un logro notable, pero está poniendo a nuestro sistema de salud frente a los principales desafíos para mantener el nivel actual de servicios [3]. Una gran parte de los costos de atención médica está asociada con el aumento de la edad, y tiene su pico en los últimos años de vida [4]. Como ejemplo, un informe de la Organización Mundial de la Salud mostró que el dinero gastado en tres enfermedades crónicas principales (enfermedades cardíacas, derrames cerebrales y diabetes) en los países desarrollados se ha duplicado entre 2006 y 2015 [5].

La hospitalización es costosa; de hecho, además de los costos de las instalaciones (electricidad, alimentos, limpieza, etc.), generalmente los pacientes deben ser seguidos por personal especializado. Además, para garantizar una supervisión completa y un ajuste de la terapia, o en el caso de procedimientos particulares como la anestesia, los parámetros vitales se controlan continuamente mediante varios dispositivos médicos [6]. Se ha estimado que el costo promedio de la hospitalización de un paciente en los Estados Unidos es de aproximadamente $ 1900 por día [7]. Una posible forma de mitigar estos costos podría estar representada por dispositivos modernos de salud personal, es decir, instrumentos que el usuario pueda utilizar en un entorno doméstico, con supervisión remota del personal médico [8]. Para situaciones no críticas, los pacientes pueden seguir el proceso de rehabilitación directamente en sus propios hogares, con el apoyo de dichos dispositivos para el monitoreo de parámetros vitales. Además de la reducción de costos de hospitalización, las soluciones de asistencia en el hogar ofrecen una mejor calidad de vida, ya que el paciente puede continuar con su estilo de vida en su hogar [9]. La principal preocupación es el costo de los sistemas de automatización del hogar y la necesidad de un monitoreo en tiempo real de los parámetros vitales y fisiológicos del usuario, para permitir un diagnóstico rápido de situaciones críticas. Estos conceptos conducen al enfoque de vida asistida ambiental (AAL) [10]. AAL se refiere a tecnologías de vida asistida basadas en inteligencia ambiental e incluye dispositivos portátiles, así como infraestructuras domésticas. Este paradigma puede usarse para prevenir, curar, así como para mejorar el bienestar, el estilo de vida y las condiciones de salud de los adultos mayores [11], [12]. AAL es un campo amplio que incluye, por ejemplo, gestión de medicamentos y recordatorios [13], sistemas de respuesta a emergencias [14], monitoreo ambiental [15], monitoreo de museos [16], hogares inteligentes [17], [18], rehabilitación domiciliaria [19], y muchos otros. Estos servicios son factibles gracias a la disponibilidad de sensores de bajo costo e infraestructuras de tecnologías de información y comunicación (TIC). En los últimos años, las soluciones patentadas para la automatización del hogar, como Amazon Alexa [20] o Google Home [21], están ganando popularidad, pero son soluciones patentadas e impiden una fácil integración de dispositivos de diferentes proveedores. Se han propuesto varios sistemas sensoriales para el cuidado de la salud y AAL, que explotan diferentes tecnologías de comunicación inalámbrica, como Bluetooth (BT) [22], [23], Wi-Fi [24], Sistema Global para Comunicación Móvil (GSM) [25], hasta soluciones más recientes, como Wi-Fi directo [26]y Z-Wave [27]. Hoy en día, BT, gracias a la reducción de costos debido a la difusión generalizada de dispositivos de audio (como BT in-ear) y dispositivos habilitados para Internet de las cosas (IoT), es el estándar *de facto*, especialmente para dispositivos portátiles [28] - [29 ] [30] [31]. Se utilizan muchas otras tecnologías cableadas e inalámbricas para la detección y automatización en interiores o exteriores: KNX, Zigbee, BT 5.0 [33], bus WM, hasta LPWAN (como LoRaWAN [32]), específicamente diseñado para IoT. Se supone que esta última, que admite malla y características mejoradas en términos de alcance y rendimiento, será la tecnología líder en los próximos años. Hasta ahora, las arquitecturas para AAL implican el uso de una red inalámbrica de sensores [34] - [35] [36] con soluciones patentadas o personalizadas. Por el contrario, la disponibilidad de soluciones estándar de bajo costo, abiertas, escalables y altamente personalizadas es el desafío del próximo futuro. En 2008 se propuso un estándar para interconectar dispositivos médicos, CEN ISO/IEEE 11073 [8], pero todavía está infrautilizado [37]. Los sistemas complejos con varios dispositivos de diferentes proveedores a menudo se realizan mediante el uso de una potente unidad central. Una arquitectura menos pesada podría explotar los teléfonos inteligentes [38] como puertas de enlace. En general, una puerta de enlace incluye controladores adecuados para interconectar dispositivos que tienen diferentes interfaces de comunicación. Las puertas de enlace deben ser accesibles para todos los dispositivos que deben estar interconectados, es decir, dispositivos portátiles [39], dispositivos AAL [31]. En la Sección II se presenta un extenso análisis del estado del arte de los sistemas AAL para comparar enfoques basados ​​en puerta de enlace patentada y puerta de enlace basada en teléfonos inteligentes. En ambos casos, el esfuerzo de programación es el costo principal. Desarrollar una aplicación móvil (aplicación) es fácil hoy en día, pero administrar diferentes sensores y dispositivos de varios proveedores desde una misma aplicación única no es trivial. De hecho, se requiere un conocimiento profundo de las interfaces y protocolos de comunicación [como Bluetooth Low Energy (BLE)].

Nuestro grupo de investigación ha implementado un marco de Android novedoso, unificado y de código abierto, llamado SAndroidE (Sensores para Android Embedded) [40], con el propósito de facilitar la administración de dispositivos inalámbricos desde aplicaciones de Android (<https://github.com/SAndroidEOfficial>). SAndroidE está orientado a conectar dispositivos con estándares de comunicación de baja potencia, como el BLE ya mencionado, ya que el consumo de energía suele ser una de las principales limitaciones cuando se trata de dispositivos portátiles inalámbricos [36]. Con respecto al trabajo en [41], el marco se ha estructurado y extendido a más dispositivos y, en este documento, se utiliza efectivamente para administrar un sistema AAL complejo en un entorno Android. SAndroidE crea una sola aplicación de Android capaz de administrar fácilmente dispositivos BLE de múltiples proveedores, información basada en la web, así como sensores internos de teléfonos inteligentes. De hecho, siguiendo los paradigmas emergentes de IoT y Smart City, la presencia de dispositivos conectados aumentará en el futuro cercano. La posibilidad de explotar soluciones de hardware de múltiples proveedores para la implementación de nuevos sistemas será un factor clave para reducir los costos de implementación. En tales escenarios, acelerar el desarrollo de aplicaciones personalizadas facilitando los esfuerzos de integración del programador y reduciendo la tasa de error del proceso de codificación es una gran ventaja.

En este documento, un escenario AAL se considera un caso de uso significativo y un banco de pruebas de un sistema complejo; en particular, se ha considerado el problema del monitoreo continuo y remoto de un usuario mayor en la vida cotidiana. El sistema desarrollado emplea un conjunto de sensores/actuadores, como sensores de presencia, ambientales, de actividad física, botones BLE y una aplicación diseñada a propósito que explota el marco SAndroidE.

Las comparaciones teóricas y experimentales con los enfoques tradicionales han demostrado la validez y efectividad del enfoque propuesto.

**SECCION II.**

## **Estado del arte: arquitecturas**

### A. Portal de propiedad

En los últimos años, se han presentado varios trabajos sobre el paradigma AAL en la literatura. En [42], se propone una plataforma de tres capas: la capa de percepción, que contiene sensores dedicados a la recopilación de parámetros del usuario y del entorno; la capa de red y puerta de enlace, que transfiere datos de la anterior a la red; y, por último, la capa de aplicación integrada, cuyo papel cumple un servidor de alojamiento web, en el que los datos se almacenan y analizan para emitir advertencias cuando se detectan posibles situaciones peligrosas para los usuarios. La computadora personal de placa única un *Raspberry Pi 3* se utiliza como puerta de enlace, mediante un protocolo de transporte de telemetría de mensajes (MQTT) basado en Python. Las interfaces del dispositivo están diseñadas con un archivo XML. En este caso, la Frambuesa es la puerta de entrada principal; solo se pueden usar dispositivos y sensores conectados a Wi-Fi, mientras que los dispositivos comerciales necesitan interfaces específicas basadas en microcontroladores. Esta solución es versátil, pero la integración de sensores adicionales parece muy compleja.

En [43], se propone un marco de interacción humano-computadora. El usuario (generalmente una persona mayor) usa un dispositivo sensor inalámbrico en la muñeca, que adquiere parámetros físicos y envía datos a un servidor para su procesamiento. Un cuidador recibe información relacionada con los usuarios a través de una interfaz de usuario en un dispositivo portátil. Dicha comunicación es manejada por el marco SALSA, que define reglas, acciones, eventos e información de interés. En este caso, se admite un único dispositivo sensorial patentado y la solución general, aunque simple, no parece flexible.

En [44], se propone una arquitectura multinivel para manejar sistemas AAL. El nivel más bajo del sistema es el administrador de sensores, que se comunica con los sensores mediante la recopilación de datos sin procesar y el envío de mensajes de control específicos (por ejemplo, activación del modo de baja potencia del sensor). Después de una fase de preprocesamiento, las capas de *Reconocimiento de actividades* y el *Modelador ambiental,* son el núcleo del sistema. El primero explota una Red Bayesiana Dinámica para detectar qué tipo de actividad realiza el usuario, mientras que el segundo, entrenado con datos de sensores interiores y exteriores, intenta predecir los efectos de los actuadores en el medio ambiente. Esta solución es muy interesante, en el contexto del aprendizaje automático, pero parece bastante compleja con respecto a la gestión del sensor; en realidad, el sistema sensorial no está especificado y no se tiene en cuenta un sistema sensorial complejo de múltiples proveedores.

A partir de estos enfoques diferentes, todos con puertas de enlace patentadas, se pueden resumir las siguientes características y requisitos comunes.

1. Las infraestructuras de AAL a menudo son sistemas complejos, lo que implica diferentes fuentes de datos de diferentes sensores, a menudo de diferentes proveedores.
2. Las infraestructuras AAL generalmente requieren una unidad central que gestione y procese los datos provenientes del sensor.
3. El punto débil de las tres soluciones analizadas es la interfaz con el sistema sensorial: [42] se limita a MQTT y, por lo tanto, no es compatible de forma nativa con dispositivos conectados a BLE; [43] admite un solo sensor; y [44] no especifica un sistema sensorial real.

Los tres enfoques referenciados se centran en el alto nivel de la arquitectura y tienen sus límites en la interfaz de varios sistemas sensoriales y en el tiempo de desarrollo de software. Además, el enfoque de puerta de enlace propietaria es limitado, ya que no es un sistema estándar, como, por ejemplo, un conmutador, un enrutador o un dispositivo inteligente. Esto implica altos costos de mantenimiento.

### B. Gateway basado en Android

Para mantener bajo el costo general del sistema, un posible candidato como unidad central es un dispositivo inteligente. Los dispositivos inteligentes, los teléfonos inteligentes en particular, hoy en día son dominantes en nuestra vida cotidiana; tienen un amplio conjunto de interfaces inalámbricas, una capacidad computacional relativamente grande y son unidades de bajo costo. Otra característica importante es que alojan un sistema operativo (SO) ampliamente compatible y actualizado, diseñado específicamente para sistemas portátiles de bajo consumo y bajos recursos. En particular, Android es un sistema operativo de código abierto compatible con una amplia gama de dispositivos inteligentes. El entorno de desarrollo integrado gratuito (IDE), llamado Android Studio, facilita enormemente la programación de dispositivos inteligentes. Para facilitar el desarrollo de programas (llamados aplicaciones) en un entorno Android, se puede encontrar un amplio conjunto de aplicaciones de código abierto y ejemplos de código en Github; Además, una amplia gama de cursos y tutoriales están disponibles en línea. Los dispositivos inteligentes incluyen sensores integrados accesibles, en Android, mediante el llamado marco de sensores de Android (ASF), que se analizará en la siguiente sección.

A pesar de estas ventajas, el uso de un teléfono inteligente como puerta de enlace también puede tener algunas limitaciones, como recursos limitados para procesar una gran cantidad de nodos conectados e interfaces de comunicación limitadas a Wi-Fi y BLE. Además, el marco propuesto se limita a los teléfonos inteligentes Android; Esta es una buena opción para reducir costos y facilitar el desarrollo. Si los sensores usan otras tecnologías de comunicación, una buena opción es usar puertas de enlace adicionales para convertir y transmitir datos de los sensores a una interfaz compatible con Android. Una demostración de esto se da en la configuración descrita en la Sección VC, donde los sensores Z-Wave están conectados por medio de una Raspberry Pi, que actúa como una puerta de enlace y hace que los datos de los sensores estén disponibles para Android a través de una conexión Wi-Fi, por medio de un base de datos y un servicio web instalado en la Raspberry Pi.

Varios ejemplos de soluciones AAL basadas en teléfonos inteligentes Android están disponibles en la literatura.

En [44], se presentó un marco distribuido para extender las intenciones de Android (objetos de software utilizados para manejar el intercambio de datos entre aplicaciones); Es compatible con diferentes dispositivos externos conectados a BLE. La idea principal es que las aplicaciones relacionadas con diferentes dispositivos conectados a BLE envían Intentos serializados a dispositivos de red. Desafortunadamente, muchos proveedores de dispositivos BLE proporcionan aplicaciones sin interfaz de intención, y varias aplicaciones que se ejecutan al mismo tiempo implican un alto consumo de memoria y batería.

En [45], se desarrolló y presentó un middleware móvil como una solución a la barrera en la implementación de servicios AAL distribuidos. El núcleo de la arquitectura se basa en dos componentes que se ejecutan como servicios en segundo plano, proporcionando funcionalidades bajo demanda. Uno de ellos es el *Middleware*, que proporciona una interfaz común para aplicaciones de alto nivel con el fin de encontrar otros dispositivos/servicios en la red, recibir datos del consumidor y publicar datos como productor. El otro componente es el *CommunicationConnector*; basado en el protocolo MQTT, que se utiliza para desacoplar la lógica empresarial contenida en el nivel de middleware (manejo de la información de control y contexto) del mecanismo real utilizado para intercambiar mensajes. Además de las aplicaciones concurrentes, este enfoque presenta datos de diferentes maneras con respecto a ASF, lo que requiere más habilidades de programación.

Ambas soluciones mencionadas tienen como objetivo transferir todos los eventos de campo (muestras de sensores o eventos de actuación) a la nube, mediante métodos del sistema operativo (como en [44]) o aplicaciones de fondo adecuadas (como en [45]). Además, estas soluciones no explotan completamente la capacidad del dispositivo inteligente; Los servicios en la nube gestionan toda la elaboración de datos, mientras que los sensores integrados (p. ej., acelerómetros, micrófonos) y los actuadores (p. ej., vibración, altavoces) se ignoran.

En [46], los autores presentan sensores de kit de datos abiertos, un marco de nivel de aplicación para integrar sensores externos en Android. Proporciona un servicio de Android (básicamente una tarea en segundo plano) a cargo de ejecutar un administrador de sensores. Este administrador contiene los controladores necesarios para comunicarse con sensores externos y puede entregar las muestras recolectadas a la aplicación del usuario. Este enfoque crea un marco modular para agregar sensores, abstraer los canales de comunicación y hacer posible descargar nuevas capacidades de sensores de un mercado, en lugar de modificar las configuraciones del sistema operativo. Esta arquitectura puede verse como una versión de software de una placa de adquisición digital común que contiene sensores modulares. Acelera el desarrollo de aplicaciones: Los programadores implementan el protocolo de comunicación con el servicio en segundo plano sin conocimiento de controladores de sensores o canales y protocolos de comunicación. El principal inconveniente de este enfoque es que funciona a un nivel alto, enmascarando el sensor externo con un "controlador" o "placa de adquisición de software" adecuados; Esto significa que no se permite el acceso específico al dispositivo, que en cambio se necesita en sistemas AAL complejos.

**SECCION III.**

## **Estado del arte: enfoques de software**

Escribir una aplicación AAL no es una tarea fácil e implica muchas operaciones, incluida la gestión de sensores y actuadores integrados y externos; Esto es particularmente cierto considerando los recursos relativamente limitados (memoria, potencia computacional) de un teléfono inteligente. A pesar de que los dispositivos podrían usar la misma tecnología de comunicación, hoy en día generalmente están diseñados para ser interoperables solo con el ecosistema de dispositivos del proveedor (por ejemplo, iOS de Apple). Mientras desarrollan sus propias soluciones, los programadores deben manejar la pila de comunicación para cada dispositivo, desde la conexión de la capa física hasta la decodificación de los mensajes intercambiados.

El entorno Android simplifica la gestión de todas las funciones integradas del teléfono inteligente mediante la interfaz de programación de aplicaciones (API) proporcionada por el sistema operativo. Esta operación está ampliamente documentada por manuales y tutoriales en línea, además de la documentación oficial del desarrollador de Android. Los desafíos que plantea el uso de dispositivos externos pueden enfrentarse con tres enfoques diferentes.

### A. Enfoque de nivel de comunicación (CLA)

Este enfoque permite realizar aplicaciones desde cero proporcionando la interfaz más flexible y completa con dispositivos, pero requiere habilidades profundas de programación de comunicación de bajo nivel. Por ejemplo, de acuerdo con la guía de Android de conectividad Bluetooth, una aplicación que usa dos sensores BLE requiere que el programador:

1. manejar conceptos básicos de BLE (por ejemplo, perfil de atributo genérico (GATT), servicios y otros) con roles y responsabilidades;
2. obtener permisos de comunicación del sistema operativo;
3. habilitar y configurar la comunicación BLE;
4. encontrar dispositivos BLE;
5. conectarse al servidor GATT del dispositivo;
6. leer atributos BLE;
7. comenzar a recibir notificaciones del GATT.

### B. Enfoque orientado a aplicaciones (AOA)

Para evitar la complejidad mencionada anteriormente, algunos proveedores ofrecen una aplicación propietaria para exponer los datos del dispositivo [47]. Esta estrategia requiere que el usuario instale la aplicación del proveedor en el teléfono inteligente y explote las técnicas de intercambio de datos del sistema operativo (es decir, *intenciones de* Android o *proveedores de contenido*) para interactuar con la aplicación del proveedor y, en última instancia, con el dispositivo. Esto generalmente implica un alto uso de recursos (tiempo de CPU y memoria). Esto se debe a la naturaleza de propósito general de la aplicación, que incluye una gran cantidad de código y funcionalidades que a menudo no son necesarias para el objetivo del programador. Se necesita un código personalizado para extraer datos de las API de la aplicación o para manejar los mensajes de la aplicación. A menudo, la aplicación maneja solo un tipo de dispositivos, lo que impide compartir el canal de comunicación con otras aplicaciones que administran otros tipos de dispositivos. El desarrollo de AOA es más rápido con respecto al CLA, pero menos versátil. Otra desventaja es que el usuario tiene que instalar una aplicación para cada dispositivo, con todos los problemas que conlleva.

### C. Enfoque orientado al marco (FOA)

La tercera forma, orientada a facilitar el desarrollo pero, al mismo tiempo, aumentar la flexibilidad con respecto a AOA, es la FOA. En FOA, el fabricante proporciona un kit de desarrollo de software (SDK), que es una biblioteca de software, para acceder al dispositivo. Está más orientado a los programadores que CLA, y más flexible que AOA. Dependiendo de la implementación del SDK, los programadores podrían utilizar FOA para administrar más dispositivos al mismo tiempo. Un ejemplo interesante de FOA es el ASF, que proporciona una API estándar que permite a los programadores acceder directamente a los sensores integrados. En particular, la ASF proporciona lo siguiente.

1. Un administrador de sensores, que maneja la comunicación y el funcionamiento de los sensores del dispositivo. Proporciona acceso a todos los detalles de los sensores (precisión, tasas de adquisición, etc.).
2. Los objetos sensores, que son instancias de sensores específicos y proporcionan un mecanismo para determinar sus capacidades.
3. Eventos de sensores que contienen lecturas de sensores.
4. Detectores de eventos de sensores, para recibir los eventos de sensores y activar operaciones personalizadas.

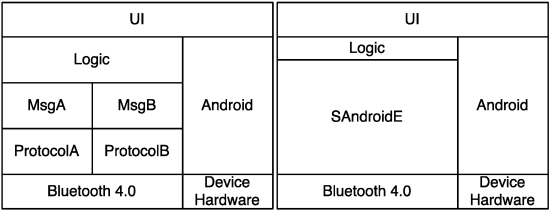
A diferencia de los SDK del proveedor, que están diseñados para administrar solo los productos propios del proveedor (si no solo una clase específica de productos), el ASF proporciona acceso a cualquier sensor existente en el dispositivo de destino, independientemente del fabricante. Esta arquitectura mejora la flexibilidad y la portabilidad del FOA. ASF está bien soportado en términos de tutoriales y códigos fuente disponibles, y por lo tanto, muchos programadores de aplicaciones están familiarizados con él.

**SECCION IV.**

## **Enfoque propuesto**

El enfoque propuesto para desarrollar sistemas AAL se basa en un enfoque FOA para la gestión de dispositivos externos (portátiles, interiores y exteriores) con una puerta de enlace basada en Android. Hay muchas aplicaciones dedicadas a la interfaz hombre-máquina, pero el objetivo del trabajo de investigación presentado es Aplicaciones para AAL que administra dispositivos externos de diferentes fabricantes, de forma automática, independientemente de la interacción con el usuario. En este escenario, que cumple con las arquitecturas de IoT, los dispositivos externos son normalmente accesibles de forma directa, por BLE o en la nube. El marco SAndroidE [48] ​​propuesto imita las API de ASF para ofrecer a los programadores el mismo enfoque conceptual empleado al desarrollar aplicaciones que utilizan sensores integrados.

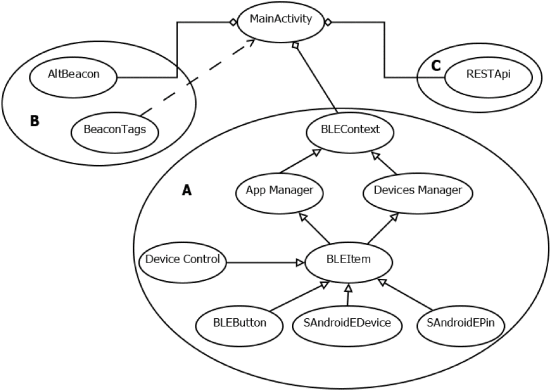
Al igual que con el kit de datos abiertos, un programador no tiene que lidiar directamente con el hardware del dispositivo y tampoco con los aspectos de comunicación. Las API de SAndroidE hacen un paso para permitir que el programador tenga control total de los dispositivos externos en cualquier ubicación del código de usuario, en lugar de tener un único punto de entrada dado por la interfaz de servicio. Este enfoque es más directo y versátil; el programador solo tiene que administrar sensores externos de manera similar a la que proporciona ASF para administrar sensores integrados. La figura 1 muestra la comparación entre los bloques en una aplicación de usuario sin y con el soporte de SAndroidE. Como se puede ver, SAndroidE ya implementa las pilas de comunicación dejando a los programadores solo el desarrollo de la interfaz de usuario (UI) y los bloques lógicos.

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/4267003/8910483/8764490/lenzi1-2924150-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/4267003/8910483/8764490/lenzi1-2924150-large.gif)

**Figura 1.** Bloques conceptuales de software que debe manejar un programador en el entorno de Android (izquierda) y (derecha) con el marco SAndroidE.

La figura 2 muestra un esquema general de los componentes principales del marco SAndroidE. La "actividad principal" de una aplicación puede referirse a tres familias de dispositivos.

1. Dispositivos conectables: son dispositivos BLE que necesitan una conexión BT permanente para funcionar (por ejemplo, algunos cinturones cardiovasculares, placas de microcontroladores como Arduino).
2. Dispositivos publicitarios: son dispositivos BLE que transmiten mensajes continuamente sin necesidad de establecer una conexión (por ejemplo, balizas BLE).
3. Dispositivos basados ​​en la nube: esta es una clase virtual de dispositivos que representan información accesible mediante cualquier API basada en la web.

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/4267003/8910483/8764490/lenzi2-2924150-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/4267003/8910483/8764490/lenzi2-2924150-large.gif)

**Figura 2.** Diagrama general de componentes de los componentes principales de SAndroidE.

La categoría de tipo A incluye todos los dispositivos conectados a BLE, incluidos los dispositivos programables y los kits de desarrollo o dispositivos comerciales no programables con interfaces BLE abiertas o accesibles desde la aplicación. El repositorio de SAndroidE Github proporciona una guía para hacer que el dispositivo externo conectado a BLE sea compatible con dispositivos de tipo A.

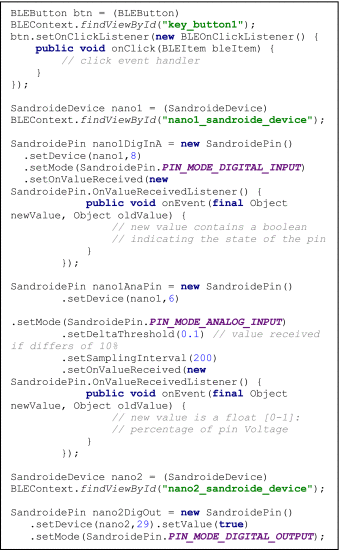
*BLEContext* es la clase principal donde se inicializa y mantiene la comunicación con dispositivos BLE. Es la "interfaz" entre el contexto de Android y el marco SAndroidE. Aquí, *AppManager* y *DevicesManager* se *crean* instancias y se ocupan de dispositivos con comunicación BLE abierta (enfoque CLA, por ejemplo, Arduino) o comunicación accesible desde la aplicación (enfoque AOA, por ejemplo, Botones Flic). *AppManager* y *DevicesManager* son responsables de la detección, *creación de* instancias y conexión a dispositivos BLE configurados cercanos. La interacción con el desarrollador se proporciona mediante objetos *BLEItem*; cualquier dispositivo conectado BLE normal se puede asignar a un elemento *BLE* específico (por ejemplo, simple*BLEButton*, que son representaciones virtuales de botones BLE físicos). *BLEButton*, *SAndroidEDevice* y *SAndroidEPin* son objetos que prácticamente representan dispositivos y sensores BLE. Este último representa un dispositivo físico programable como Arduino, Nano y Raspberry y sus respectivos pines, digitales o analógicos, de entrada, o salida. Para ser compatible con SAndroidE, cada clase de dispositivos BLE, primero debe describirse en un descriptor de dispositivo dedicado (archivo XML o JSON), destinado a asignar sus funcionalidades (es decir, servicios y características BLE) a uno de los *BLEItem*clases descritas anteriormente. Los descriptores para los dispositivos más utilizados, así como su firmware, ya se proporcionan con el marco en sí. Para identificar inequívocamente cada dispositivo con una ID de dispositivo única, se ha desarrollado una función SAndroidE, que permite que la aplicación de usuario se conecte a un dispositivo físico específico entre un conjunto de dispositivos con el mismo descriptor de dispositivo. La función de identificación se puede incluir directamente en la aplicación de usuario final, para permitir a los usuarios cambiar los dispositivos físicos. Alternativamente, la dirección/nombre de control de acceso a medios de mapeo (MAC) puede incluirse estáticamente dentro de la propia aplicación distribuida. Cada *BLEItem* tiene un *DeviceControl que* virtualiza el canal BLE. El *DeviceControl* está a cargo de los siguientes:

1. utilizando BLE GATT, la base para el diseño de cualquier sistema BLE, y definiendo la forma en que una aplicación de usuario (o cualquier dispositivo central) interactúa con un dispositivo final (dispositivo periférico), mediante una colección de servicios y características que exponen funcionalidades del dispositivo;
2. cargar el descriptor de dispositivo apropiado, que contiene la asignación de servicios BLE GATT, características y reglas de análisis para interpretar los datos recibidos o enviados al dispositivo;
3. conectarse al dispositivo, utilizando la dirección MAC asignada con el nombre especificado por el desarrollador en el código de la aplicación de usuario;
4. inicializando y configurando el dispositivo enviando los comandos apropiados como se define en el descriptor del dispositivo, como los comandos para configurar un pin como entrada o salida;
5. recibir y analizar comandos BLE del dispositivo y reenviar las acciones apropiadas al BleItem correspondiente, tal como se define en el descriptor del dispositivo; esto lleva los datos directamente a la devolución de llamada definida por el desarrollador para responder, por ejemplo, a un evento de presionar un botón o un cambio de valor de un pin de entrada;
6. enviar comandos al dispositivo, desde un conjunto de comandos disponibles definidos en el descriptor del dispositivo, como establecer un valor de pin, una matriz arbitraria de bytes o un umbral de pin.

Para usar un dispositivo de tipo A en una aplicación de usuario (por ejemplo, directamente en MainActivity, consulte la Fig. 2), después de la inicialización de *BLEContext*, el desarrollador de la aplicación de usuario debe seleccionar e instanciar un objeto de la clase *BLEItem* apropiada (por ejemplo, *BLEButton* o *SAndroidEPin*), también especificando el nombre correcto que identifica el dispositivo. El *administrador* apropiado crea el *DeviceControl* que lo vincula al *BLEItem*, personalizado a propósito por el archivo descriptor. Con SAndroidE, el desarrollador de la aplicación de usuario no requiere ninguna otra acción y los dispositivos externos se pueden administrar mediante los métodos proporcionados por *BLEItem*clase, tanto para recibir datos (mediante devolución de llamada, como para sensores integrados en ASF) como para enviar datos. Este último se usa tanto para configurar el *BLEItem* como para establecer el valor: *setAdvertisingInterval* establece el período de muestreo en milisegundos; *setMode* configura la polaridad de un pin; *setDeltaThreshold* establece la variación mínima para considerar un valor como modificado; *setValue* asigna un valor a un pin de salida; y *sendMessage* escribe una matriz de bytes en un dispositivo físico instanciado como un *SAndroidEDevice.*

La Fig. 3 indica las líneas de código fuente (SLOC [49]) para habilitar tres flujos de información: una entrada digital *"btn"* relacionada con el *botón BLE* "key\_button1;" una entrada digital *"nano1DigInA"* y una entrada analógica *"nano1AnaPin"* proveniente de un dispositivo tipo A "nano1\_sandroide\_device"; una salida digital *"nano2DigOut"* proveniente de un dispositivo tipo A "nano2\_sandroide\_device". Las líneas caracterizadas por *onEvent* son la devolución de llamada para administrar entradas (SLOC = 4). [La ecuación (1)](https://ieeexplore.ieee.org/document/#deqn1) es una fórmula para estimar SLOC debido a la habilitación y configuración de información proveniente de dispositivos tipo A, dondenorteUNAes el número de dispositivos físicos de tipo A, *NDigIn*, *NAnaIn* y *NOut* son el número de entradas digitales, entradas analógicas y salidas digitales o analógicas, respectivamente. Cabe destacar que SLOC es una estimación poco precisa del esfuerzo de programación, ya que depende del estilo de programación y de muchas otras cantidades que afectan. Además, ([1](https://ieeexplore.ieee.org/document/#deqn1)) es una estimación muy aproximada de SLOC, porque, por ejemplo, la Fig. 3 muestra que *"btn"* tiene solo cuatro (no seis) SLOC sobre devolución de llamada, mientras que *"nano1DigitalIn"* tiene siete (no seis) SLOC sobre inicialización y devolución de llamada.

S L O CUNA= N\_ A + 6 ⋅ ND i gyon + 9 ⋅ NA n a In + 4 ⋅ NO u t(1)

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/4267003/8910483/8764490/lenzi3-2924150-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/4267003/8910483/8764490/lenzi3-2924150-large.gif)

**Fig. 3.** Código fuente para habilitar la información proveniente de dispositivos tipo A.

Para mantener el marco ligero, se ha proporcionado un enfoque diferente para los dispositivos de tipo B y C. En particular, los dispositivos de tipo B utilizan la biblioteca AltBeacon ([https://altbeacon.org](https://altbeacon.org/)) para administrar etiquetas simples, así como balizas sensorizadas. Del mismo modo, los dispositivos de tipo C son accesibles solo si hay una conexión a Internet, e incluyen sensores reales con API web patentadas (por ejemplo, rastreadores de actividad), detección virtual desde servicios en la nube (por ejemplo, servicio de pronóstico del tiempo) e información de una web servidor (p. ej., humedad de un sensor web).

El uso de dispositivos de tipo B en una aplicación de usuario (p. Ej., Directamente en *MainActivity*, ver Fig. 2) es similar al uso de la biblioteca AltBeacon. El desarrollador necesita inicializar el *BeaconManager* y luego implementar una devolución de llamada para recibir eventos publicitarios de baliza. El uso de un dispositivo de tipo B requiere aproximadamente cinco SLOC para la inicialización de *BeaconManager*; para cada dispositivo hay un SLOC para la inicialización y cuatro SLOC para la devolución de llamada. Resumiendo, el SLOC para dispositivos de tipo B es 5 + 5 ·nortesi, dónde nortesi es el número de dispositivos de tipo B.

Para usar un dispositivo de tipo C en una aplicación de usuario, se debe crear una instancia de un objeto de la clase *RestAPI*, pasando el nombre del descriptor de API apropiado. Cada descriptor describe una API proporcionada por la nube, por ejemplo, API de Fitbit para rastreador de actividad, para facilitar las tareas de acceso a los datos recopilados del dispositivo. Antes de dar acceso a la información privada o confidencial del usuario, cada API de Cloud siempre solicita realizar un paso de autenticación, por ejemplo, la autenticación OAuth 2.0 para Fitbit. Este paso generalmente no es trivial cuando se implementa desde cero, pero es bastante fácil usando SAndroidE. Es muy difícil proporcionar una estimación SLOC para dispositivos de tipo C, debido a la multitud de enfoques de nube utilizados por diferentes fabricantes. En un caso de uso típico (Fitbit, número de pasos), inicialización y autenticación de un *RestAPI*requiere cinco SLOC, más los cuatro SLOC típicos para la devolución de llamada. Como una estimación aproximada, cada dispositivo de tipo C contribuye con cinco SLOC y cada llamada API de tipo C contribuye con aproximadamente cinco SLOC.

En conclusión, el marco SAndroidE simplifica el desarrollo de la aplicación de usuario, especialmente con dispositivos de tipo A, tanto en la identificación/configuración como en la operación de dispositivos físicos. Con los dispositivos de tipo C, se logra una buena simplificación en la fase de autenticación, mientras que en los dispositivos de tipo B la simplificación real, con respecto al uso de la biblioteca AltBeacon, es bastante modesta.

En referencia a ([1](https://ieeexplore.ieee.org/document/#deqn1)) y ([2](https://ieeexplore.ieee.org/document/#deqn2)) estima el SLOC total, dondenortesi y norteC son el número de dispositivos de tipo B y de tipo C, respectivamente, mientras que norteC A P Ies el número de llamadas de API de tipo C. Como ya se comentó para ([1](https://ieeexplore.ieee.org/document/#deqn1)) y también ([2](https://ieeexplore.ieee.org/document/#deqn2)) es solo una estimación aproximada del SLOC, y el SLOC en sí mismo es solo una indicación vaga del esfuerzo de desarrollo de software.

S L O C = S L O CUNA + 5 + 5 ⋅ Nsi+ 5 ⋅ NC+ 5 ⋅ NC A P I(2)

**SECCION V.**

## **Evaluación del marco en un caso de uso típico**

Para evaluar experimentalmente las ventajas de un enfoque FOA en el desarrollo de sistemas AAL complejos, el enfoque SAndroidE se ha experimentado en un caso de uso típico de asistencia en el hogar en comparación con los enfoques clásicos CLA o AOA.

### A. Descripción de la arquitectura

Como se detalla en las secciones anteriores, es difícil para las empresas de ingeniería jóvenes desarrollar y proporcionar sistemas AAL asequibles, debido a la gran variabilidad del escenario y los costos de desarrollo de software. En el enfoque FOA propuesto, el proveedor AAL diseña el sistema basado en lo siguiente.

1. Un servidor central realizado en PHP con MySQL, que actúa como proveedor de autenticación y sistema de enrutamiento. Los usuarios (pacientes) y los cuidadores se identifican por su dirección de correo electrónico. El proveedor de autorización permite la vinculación de los datos del usuario a cada cuidador que puede acceder a ellos por medio de API REST adecuadas. Aprovechando estos datos de autorización, el servidor configura automáticamente un agente de mensajes con canales adecuados para conectar instancias de usuarios con instancias autorizadas de cuidadores.
2. Una aplicación de usuario que adquiere datos de dispositivos portátiles y sensores domésticos, procesa datos y los envía al agente de mensajes del servidor, incluidas las notificaciones si se ha producido un evento.
3. Una aplicación de cuidador que notifica si ocurrieron eventos seleccionados; Estos eventos se pueden definir en la misma aplicación, estableciendo umbrales adecuados y configurables.

Es importante tener una aplicación de cuidador muy liviana (en términos de memoria, potencia, etc.) para limitar el uso de recursos del teléfono inteligente en caso de que el cuidador necesite ayudar a más de un paciente.

Desde la aplicación de usuario, es posible enviar una invitación para seguir un flujo específico de datos utilizando la dirección de correo electrónico del cuidador. La aplicación Caregiver recibe una notificación de la invitación o un correo electrónico con un enlace para descargar la aplicación Caregiver y configurar una cuenta en el sistema. Una vez que la invitación ha sido aceptada, comienza el flujo de datos. Desde la aplicación Caregiver, el cuidador puede suscribirse/darse de baja a las notificaciones y acceder voluntariamente a la última instantánea de los datos del paciente. No se almacenan series históricas o datos del paciente en el servidor central; de hecho, los datos del paciente se combinan con la identidad del usuario solo por medio de la configuración del agente.

La infraestructura de comunicación y las funciones de la aplicación Caregiver son las mismas para todos los sistemas AAL. La mayor parte del costo de desarrollo de la solución AAL personalizada está en la aplicación de usuario. Normalmente, la elaboración y la interfaz de usuario de la aplicación de usuario son bastante simples, mientras que la integración de dispositivos externos requiere habilidades avanzadas y un largo tiempo de desarrollo.

### B. Descripción del caso de uso

El caso de uso seleccionado es un sistema AAL diseñado para un usuario con enfermedad cardíaca, que vive solo y que es monitoreado periódicamente por personal médico. El usuario debe cumplir con una terapia médica, que incluye actividad física periódica. Al usuario no le gustan los sistemas implantables o portátiles y solo acepta usar un rastreador de actividad para el conteo diario de pasos y el monitoreo de la frecuencia cardíaca durante la actividad de ejercicio en bicicleta al aire libre y bajo techo. El usuario normalmente está en casa y un pariente, que no vive con ella, actúa como cuidador. El usuario acepta que se informa al cuidador cuando se detectan anomalías en el estilo de vida; Además, la información sobre la posición del usuario (en el hogar/fuera) y la adherencia a la terapia prescrita también se proporciona al cuidador. Particularmente,

1. Posición del usuario, en casa o en el exterior; notificación si está afuera por más de "2" h.
2. Número de pasos en el día, actualizados cada hora; notificación a las "18:00" si el número de pasos es inferior a "3000".
3. Número de minutos en el día en que la frecuencia cardíaca ha estado entre 94 y 132 lpm (intervalo de zona de frecuencia cardíaca *FatBurn* según la especificación de los rastreadores de actividad populares).
4. Número de minutos en el día en que la frecuencia cardíaca ha estado entre 132 y 160 lpm (intervalo de zona de frecuencia cardíaca *cardiovascular*).
5. Número de minutos en el día en el que el ritmo cardíaco ha sido de entre 160 y 220 lpm (*Pico* intervalo de zona de frecuencia cardiaca); notificación si la suma de minutos en la zona de frecuencia cardíaca *Cardio* y *Peak* es superior a "10" min.
6. Adherencia a la terapia, donde la terapia se identifica como un conjunto de elementos (hora, nombre) establecidos por el usuario. Si el dispensador de pastillas se mueve a la hora "± 30" min, no se genera ninguna alarma; de lo contrario, una alarma de Android normal le recuerda al usuario que tome el medicamento adecuado; notificación si el dispensador de pastillas está inactivo después de "2" h.
7. Número de veces que se presiona un botón de emergencia, con información de marca de tiempo para cada evento; Cada pulsación de botón genera una notificación.
8. Número de veces que el refrigerador está abierto; notificación a las "18:00" si es cero.
9. Número de activaciones de tres sensores de presencia (baño, comedor y dormitorio); notificación a las “18:00” si este número es cero.
10. Datos de la calidad del aire ambiental exterior (PM 2,5, NO 2, CO, CO 2, humedad y temperatura) muestreados cada 15 min. Esta característica es necesaria para los usuarios que deben limitar la actividad al aire libre (por ejemplo, ciclismo) en días con un alto nivel de contaminación (de color rojo si supera el "umbral").
11. Datos de calidad del aire interior [compuesto orgánico volátil (COV), CO 2, humedad y temperatura] muestreados cada 15 min; notificación si la concentración de CO 2 supera un "umbral".
12. Interruptor de luz y control de persiana eléctrica desde la cama.

Al usuario considerado no le gustan los teléfonos inteligentes, pero acepta usar uno con una interfaz hombre-máquina muy simplificada. El teléfono inteligente advierte al usuario si se detecta una anomalía en el uso del dispensador de pastillas. La descripción funcional se presenta para comprender el contexto de la aplicación de usuario (la aplicación de cuidador se refiere solo a los servicios web y está fuera de interés de este trabajo; por lo tanto, no se tiene en cuenta). Sin embargo, a continuación, el enfoque se establecerá solo en el esfuerzo de desarrollo para realizar el intercambio de datos entre cada dispositivo y la aplicación de usuario.

### C. Implementación del sistema AAL

El sistema AAL está compuesto por los siguientes dispositivos.

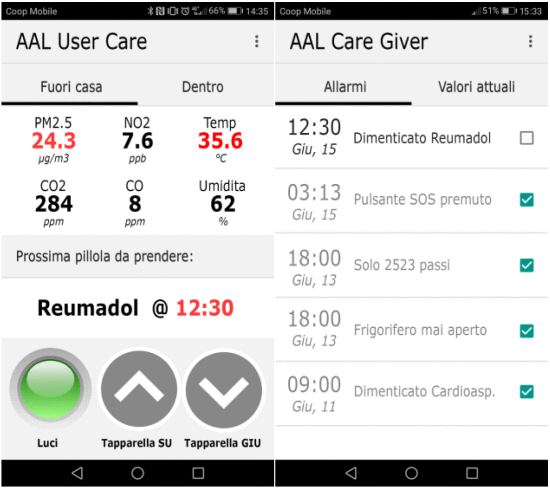
1. Un rastreador de actividad (dispositivo *Fitbit Charge* 2 — tipo C) que proporciona la cantidad de pasos diarios y la cantidad de minutos relacionados con los intervalos de frecuencia cardíaca mencionados anteriormente.
2. Un dispositivo a batería diseñado específicamente para controlar la calidad del aire exterior durante la actividad de ciclismo + ejercicio (tipo A con seis entradas analógicas). El sistema se basa en *RedBear Nano*, que es una placa compatible con *Arduino IDE*. Se han utilizado sensores adecuados, como *Shinyei PPD42NS* o *DD Scientific GS + 4CO* para detectar los parámetros ambientales exteriores mencionados anteriormente (no se proporcionan detalles sobre los circuitos de acondicionamiento específicos, ya que están fuera del alcance de este documento).
3. Tres sensores de presencia (*Everspring HSP02*), un sensor *Siegenia Sensoair* para COV y CO 2 en interiores y un sensor *Everspring ST814* para temperatura y humedad. Los sensores son comerciales y con interfaz Z-Wave. *El* software *Domoticz* se ha instalado en un *Raspberry Pi* 3 para que actúe como puerta de entrada para los sensores Z-Wave mediante una API de reposo (tipo C). Algunos de sus pines de entrada/salida de uso general se han utilizado para encender/apagar la luz en el dormitorio y para controlar el obturador eléctrico (tipo A con tres salidas digitales). En este caso, la *Raspberry* incorpora información de tipo A (pines de salida locales) e información de tipo C (sensores Z-Wave).
4. Una baliza BLE (USB *Gimbal*), instalada dentro de la casa, se usa para detectar si esta persona está en casa.
5. Dos sensores de movimiento *Estimote Nearable* para detectar si el dispensador de pastillas se ha movido y si se ha utilizado el refrigerador, respectivamente.
6. Un teléfono móvil normal con botón de pulsar para llamar es el dispositivo de emergencia preferido por el usuario considerado; el sistema AAL incluye también un botón BLE a prueba de agua (botón *Flic*), que se instalará en la ducha (tipo A con 1 entrada digital).

Resumiendo, el caso de uso seleccionado involucra dos dispositivos de tipo C (*Fitbit* y *Raspberry*) con nueve llamadas de tipo C (dos para *Fitbit* y siete para *Domoticz*), tres dispositivos de tipo B (una baliza y dos sensores de movimiento) y tres A- dispositivos de tipo (*Raspberry*, *RedBear Nano* y *Flic*) con un total de una entrada digital, seis entradas analógicas y tres salidas. Aplicando ([2](https://ieeexplore.ieee.org/document/#deqn2)), se espera un mínimo de 75 + 5 + 15 + 10 + 45 = 150 SLOC para la gestión de los dispositivos más el SLOC de la lógica implementada dentro de las devoluciones de llamada.

### D. Aplicaciones desarrolladas

Para comparar efectivamente los enfoques, se han desarrollado dos versiones de la misma aplicación de usuario. La primera versión, llamada SCRATCH, se ha desarrollado desde cero, utilizando aplicaciones y SDK proporcionados por cada fabricante de dispositivos, por lo tanto, utilizando los enfoques clásicos de CLA y AOA. La segunda versión, llamada SANDR, se ha desarrollado utilizando un enfoque FOA, explotando el marco SAndroidE, que abstrae de forma transparente la pila de comunicación [BT, Intentos, Proveedores de contenido y API de transferencia de estado representativa (REST)]. Durante la fase de desarrollo de la aplicación de usuario, se tendrá en cuenta el tiempo de desarrollo y el SLOC necesarios para la integración de cada dispositivo.

En la Fig. 4, se presentan las capturas de pantalla de la interfaz de usuario (en idioma italiano) de la aplicación de usuario (lado izquierdo) y la aplicación de cuidador (lado derecho). La aplicación de usuario incluye parámetros de calidad del aire interior y exterior, el siguiente medicamento que el usuario debe tomar, así como su hora programada y, en la parte inferior, los botones para activar las luces de la habitación y el obturador eléctrico. La aplicación Caregiver muestra una lista de notificaciones, la más reciente primero, con un texto descriptivo que indica el tipo de notificación. El cuidador puede confirmar la recepción de la notificación. El cuidador puede cambiar de pestaña para acceder a los últimos valores adquiridos de cada sensor.

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/4267003/8910483/8764490/lenzi4-2924150-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/4267003/8910483/8764490/lenzi4-2924150-large.gif)

**Fig.4.** Capturas de pantalla de la aplicación de usuario (izquierda) y la aplicación de cuidador (derecha).

El desarrollo de la aplicación de usuario se ha llevado a cabo desde el mismo programador en secuencia, primero la solución SCRATCH, luego la solución SANDR.

Cabe destacar que este es solo un caso de uso, pero es un método cuantitativo para evaluar los beneficios del enfoque propuesto, con respecto a los tradicionales, para desarrollar sistemas AAL que involucren dispositivos externos de diferentes proveedores.

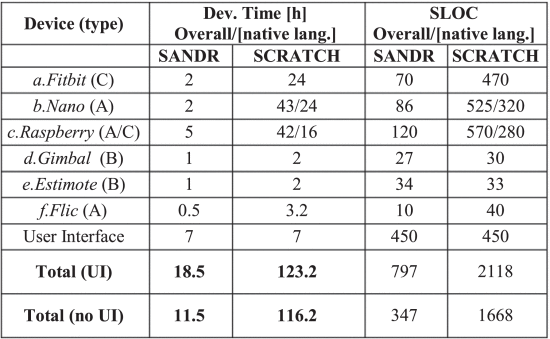
**SECCION VI.**

## **Resultados y discusión**

### A. Consideraciones sobre el tiempo de desarrollo

En la Tabla I, se informa el tiempo de desarrollo y el número de SLOC para las dos versiones de la aplicación de usuario mencionadas anteriormente. La Tabla I evalúa el esfuerzo requerido para incluir individualmente cada dispositivo externo identificado en la Sección VC.

**TABLA I** Evaluación y comparación del esfuerzo de desarrollo

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/4267003/8910483/8764490/lenzi.t1-2924150-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/4267003/8910483/8764490/lenzi.t1-2924150-large.gif)

*La* integración de *Fitbit* requirió que el desarrollador manejara la API REST HTTP de *Fitbit*. No se puede acceder directamente a los dispositivos *Fitbit* desde aplicaciones de terceros a través de BT; el único medio provisto por *Fitbit* es acceder a los datos de actividad a través de Internet desde los servidores en la nube de *Fitbit*. Esto implica pedirle al usuario que realice un paso de autenticación OAuth2.0 para autorizar a la aplicación de terceros a acceder a los datos del usuario. El proceso de autorización ha sido una tarea lenta durante el desarrollo del SCRATCH.

Finalmente, tomó 24 h de trabajo y SLOC = 470 para implementar SCRATCH, en comparación con 2 hy SLOC = 70 de la contraparte de SANDR.

Para la integración *Nano*, SCRATCH requirió aproximadamente 43 h de trabajo y SLOC = 525, en comparación con las 2 h y SLOC = 86 de la versión SANDR. La aceleración usando SAndroidE en un enfoque FOA es significativa, ya que SAndroidE proporciona un firmware predeterminado para *Nano* (como para cualquier otro dispositivo programable compatible, por ejemplo, *Arduino* y *Raspberry*); dicho firmware proporciona a la aplicación de usuario el *Nano* más comúncaracterísticas, como configurar un pin de salida digital, leer un pin de entrada e intercambiar datos arbitrarios. Por el contrario, en SCRATCH, el firmware debe desarrollarse desde cero, según el caso de uso, con la carga adicional de aprender un lenguaje de programación diferente (C en este caso), configurar el entorno de desarrollo de firmware y la cadena de herramientas, y desarrollar el firmware programa principal. Además del tiempo de desarrollo del firmware *Nano*, igual a 24 h, y el SLOC relacionado, igual a 320, SAndroidE también abstrae completamente la capa de comunicación, en este caso BLE, que debe manejarse y desarrollarse en el lado de Android, para el desarrollo de la versión SCRATCH.

Para *Raspberry Pi*, SCRATCH requirió aproximadamente 42 hy SLOC = 570, mientras que SANDR requirió 5 hy SLOC = 120. Al ser un dispositivo programable, el esfuerzo para establecer una comunicación y un protocolo de aplicación entre una aplicación de teléfono inteligente y una *Raspberry* es muy similar. con el mismo esfuerzo requerido para la integración de *Nano*, pero en un entorno diferente. Raspberry se ejecuta en Raspbian, que es una distribución de Linux, por lo tanto, el desarrollador debe tener conocimiento del sistema operativo Linux. Además, los desarrolladores necesitan conocimiento de al menos un lenguaje de programación compatible (NodeJS se ha utilizado en este trabajo) para construir la lógica de la aplicación del lado del dispositivo. En cuanto a *Nano*, los desarrolladores de SCRATCH también necesitan conocimiento del perfil BLE y GATT. La *frambuesa*la integración tomó 16 hy 280 SLOC en el lado del dispositivo. En cambio, en SANDR, SAndroidE proporciona el firmware para el dispositivo *Nano*, y el software *Domoticz* para administrar los sensores Z-Wave se puede descargar de Internet. El tiempo de instalación de ambos software ha sido descuidado.

Para la integración de balizas *Gimbal*, la diferencia en el tiempo y las líneas de código no es tan evidente, debido al uso de la biblioteca *AltBeacon*, que se encarga de la parte más compleja de tratar con balizas. La implementación de balizas en SCRATCH requirió 2 hy 30 SLOC frente a 1 hy 27 SLOC requeridos por SANDR; dichos valores son comparables, dado que SAndroidE usa AltBeacon internamente y lo extiende con la función de Etiquetado, que ayuda a identificar fácilmente los mensajes que provienen de una baliza específica. Por otro lado, hacer lo mismo en SCRATCH requería que el desarrollador identificara los mensajes de baliza por medio de una aplicación BLE Sniffer de terceros, para pasar correctamente su identificador a la biblioteca de escaneo AltBeacon.

Se pueden hacer consideraciones similares para *Estimote Nearable*. Estas son balizas especiales, que transmiten en el mensaje publicitario datos adicionales con respecto a las balizas de cardán, por ejemplo, aceleración y temperatura. Esto requirió el uso del Estimote SDK para analizar correctamente los datos. El esfuerzo para integrar *Estimote Nearables* en SCRATCH ha sido de 2 hy 33 SLOC frente a 1 hy 34 SLOC en SANDR.

*La* integración de *botones de Flic* se ha realizado utilizando el SDK de *Flic*, para recibir notificaciones de presión de botones a través de Intentos de Android por la aplicación oficial de *Flic*. El uso de *botones Flic* con SAndroidE es trivial y, por lo tanto, requiere 0.5 hy 10 SLOC, mientras que el uso del SDK de Flic en SCRATCH requiere 3.2 hy 40 SLOC.

La construcción de la interfaz de usuario, que es la misma para ambas versiones, requirió 7 hy 450 SLOC.

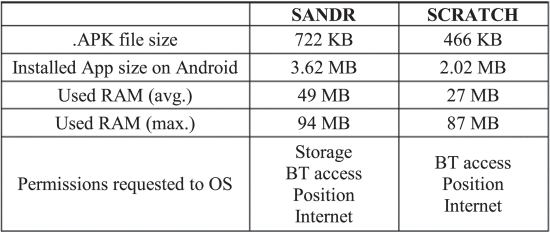
En resumen, SANDR usando SAndroidE en un enfoque FOA requirió al desarrollador una cantidad total de 18.5 hy 797 SLOC, frente a 123.2 h y 2118 SLOC requerido por SCRATCH, acelerando el desarrollo de un factor 123.2/18.5 = 6.7 × sobre el AOA clásico y enfoques de CLA. Teniendo en cuenta el tiempo necesario para desarrollar solo las características y no la interfaz de usuario, la aceleración es aún más significativa, alcanzando un factor de 116.2/11.5 = 10.1 ×. Con SCRATCH, el porcentaje de tiempo dedicado a escribir la lógica de la aplicación de usuario es mayor que en SANDR (100 × 116.2/123.2 = 94% vs 62%). El desarrollo de la lógica de la aplicación de usuario generalmente requiere más habilidades, depuración y pruebas que el desarrollo de la interfaz de usuario. También es evidente que el conocimiento de diferentes lenguajes de programación, entornos de desarrollo y sistemas operativos debe adquirirse antes de ser efectivo al escribir la aplicación final. En este trabajo específico, el desarrollador ya tenía un conocimiento parcial de todos los lenguajes de programación en cuestión; esto rara vez es el caso, por lo tanto, los factores de aceleración aquí descritos pueden considerarse como el peor de los casos.

### B. Consideraciones sobre RAM y espacio de almacenamiento

Para analizar de manera efectiva todas las compensaciones entre las dos versiones de la aplicación de usuario, se ha evaluado la ocupación de la memoria RAM y el espacio de almacenamiento. Las dos versiones han sido probadas en un Samsung Galaxy S7, con Android OS 6.0.1.

Como se ve en la Tabla II, tanto los archivos User Apps.APK como el espacio de almacenamiento ocupado cuando están instalados son más grandes para SANDR. Esto se espera dado que la biblioteca SAndroidE, que se incluye en SANDR, contiene una gran cantidad de código para facilitar al desarrollador y disminuir el tiempo de desarrollo. La ocupación de RAM se ha verificado utilizando la herramienta de evaluación de memoria de Android proporcionada por el sistema operativo. SANDR usa más RAM, 49 MB en promedio, contra 27 MB, pero con una ocupación máxima de RAM de 94 MB contra 87 MB. También se espera este comportamiento, dado que escribir una aplicación desde cero da la oportunidad de optimizar cada aspecto. En cualquier caso, la ocupación de la memoria es algo que posiblemente podría optimizarse en el marco SAndroidE con futuras actualizaciones.

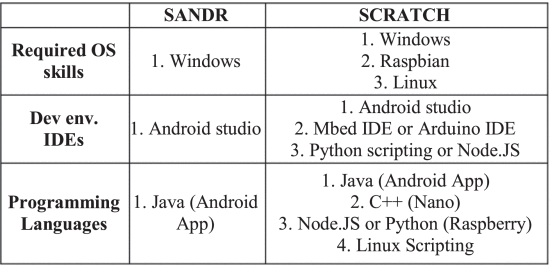
**TABLA II** Comparación de los parámetros peculiares de la aplicación de usuario

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/4267003/8910483/8764490/lenzi.t2-2924150-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/4267003/8910483/8764490/lenzi.t2-2924150-large.gif)

C. Consideraciones sobre las habilidades requeridas

Para concluir la comparación entre las dos aplicaciones de usuario desarrolladas, en la Tabla III se presenta el esfuerzo en términos de habilidades requeridas para el desarrollador.

**TABLA III** Comparación de las habilidades requeridas por la aplicación del usuario

[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/4267003/8910483/8764490/lenzi.t3-2924150-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/4267003/8910483/8764490/lenzi.t3-2924150-large.gif)

Para desarrollar SANDR, es suficiente el conocimiento del entorno de Windows, Android Studio y el lenguaje Java. Por el contrario, el desarrollo de SCRATCH requiere el conocimiento de diferentes sistemas operativos, IDE y lenguajes de programación, para hacer frente a la necesidad de desarrollar firmware personalizado y software local para dispositivos *Nano* y *Raspberry*. Por lo general, un solo desarrollador no tiene todas estas competencias, por lo que puede ser necesario un equipo de desarrolladores.

### D. Consideraciones sobre el sistema AAL realizado

El sistema AAL presentado se ha realizado y proporcionado a una persona de 74 años, con dos hijos, uno que vive cerca y otro que vive a más de 100 km de distancia.

El usuario anciano dio acceso completo a los datos personales al hijo más cercano y solo un acceso limitado a medicamentos y pasos cuentan para el otro hijo. Durante los 15 días de uso, después de la primera semana de ajuste del usuario al sistema, el hijo vivo más cercano ha detectado las siguientes notificaciones:

1. tres anomalías en la toma de medicamentos;
2. una pulsación de botón de emergencia;
3. cuatro alertas "fuera de casa".

Los datos de frecuencia cardíaca resultaron ser poco confiables, debido a la mala adherencia del Fitbit a la muñeca del usuario. Por esta razón, se ha agregado un cinturón de cardio al sistema AAL. Agregando a ambas aplicaciones de usuario, el cinturón cardio Polar H10, que es compatible con SAndroidE, tomó 2 hy 40 SLOC adicionales en SANDR, mientras que tomó 16 hy 140 SLOC en SCRATCH. Obviamente, si se debe utilizar un dispositivo que aún no es compatible con SAndroidE, primero será necesario agregarlo al marco, utilizando las pautas proporcionadas en el Wiki de Github de SAndroidE.

En conclusión, debe tenerse en cuenta cómo el desarrollo de la aplicación de usuario con SAndroidE tomó un tiempo de desarrollo general de menos de 30 h, lo que hace que sea asequible construir y proporcionar sistemas AAL personalizados. Agregar el cinturón cardiovascular, requirió solo 2 horas más, lo que dejó el costo de desarrollo de software en el mismo orden de magnitud que el costo del hardware.

### E. Notas sobre confiabilidad y resolución del enfoque propuesto

El caso de uso descrito ha sido diseñado a propósito para proporcionar un sistema realista para las pruebas de evaluación y comparación y, por lo tanto, para validar el enfoque propuesto y resaltar las posibles ventajas con respecto a los enfoques tradicionales. La confiabilidad del sistema depende principalmente del hardware (los componentes empleados, como sensores y teléfonos inteligentes) y del software (la biblioteca SAndroidE y la aplicación de usuario). Con respecto a los aspectos de hardware, debe tenerse en cuenta que, con el marco propuesto, se facilita la redundancia del sensor, porque muchos sensores, incluso de diferentes fabricantes, podrían emplearse sin aumentar significativamente la complejidad de la aplicación. Además, gracias al amplio uso de dispositivos como teléfonos inteligentes o tabletas, especialmente en el hogar, también es posible hacer que las funciones de la puerta de enlace de datos sean completamente redundantes [35], de una manera muy simple, instalando la misma aplicación de Android con la misma configuración de sensores en múltiples dispositivos. Cabe señalar que las funcionalidades de la puerta de enlace se activan automáticamente solo en un dispositivo a la vez; de hecho, las conexiones Bluetooth requieren solo un maestro por cada esclavo. Cuando el maestro sale del rango del sensor, se puede habilitar automáticamente una conexión con un nuevo dispositivo maestro. Esto se logra dentro de un período de tiempo, que depende del sensor comercial utilizado, pero generalmente es inferior a 10 s. Por lo tanto, cualquier otro dispositivo maestro, como otro teléfono inteligente o tableta, es BLE para ofrecer funcionalidades de puerta de enlace de respaldo, reemplazando al maestro conectado previamente. La redundancia de la puerta de enlace se ha probado experimentalmente con SAndroidE utilizando sensores Bluetooth, un teléfono inteligente (puerta de enlace predeterminada), y una tableta (puerta de enlace de respaldo) en el mismo entorno. Cuando se apaga el teléfono inteligente, el cambio de puerta de enlace se produce con un tiempo promedio de 4,3 s durante 10 pruebas, con un mínimo de 3,5 sy un máximo de 8,2 s. En principio, es posible extender la redundancia de la puerta de enlace también a sensores que no sean Bluetooth compatibles con SAndroidE, implementando funciones de modificación de puerta de enlace más avanzadas, directamente dentro de la aplicación del usuario.

Con respecto a los aspectos del software, es un hecho que la confiabilidad de la aplicación de usuario depende en gran medida de la implementación y, por lo tanto, de la habilidad de programación del desarrollador y del sistema y contexto específicos. Sin embargo, con la capacidad del desarrollador igual, se ha verificado que la reutilización del código y el uso de componentes estándar, como SAndroidE, ayuda a escribir software o aplicaciones más confiables, como también se reconoce en [46]: “bien entendido y Las piezas estándar ampliamente probadas ayudan a mejorar el mantenimiento y la confiabilidad ".

Considerando este hecho, es interesante evaluar el proceso para garantizar la confiabilidad dentro del proceso de desarrollo del marco. El marco SAndroidE en sí es desarrollado de manera colaborativa por una comunidad con especial atención a la producción de código que sea confiable en el tiempo. El código fuente se revisa y mejora continuamente y la comunidad ayuda a depurar la biblioteca, detectando y resolviendo errores de programación. Además, ambas solicitudes de nuevas características y contribuciones de código fuente, que se incluirán en la versión oficial de SAndroidE, se evalúan exhaustivamente de manera democrática, a través de un sistema de votación que permite la aceptación de contribuciones solo cuando se alcanza un quórum de opinión positivo, que actualmente corresponde al 10% del tamaño de toda la comunidad.

En aras de la exhaustividad, se ha llevado a cabo una comparación de fiabilidad entre las aplicaciones SCRATCH y SANDROIDE. En particular, las dos aplicaciones se han ejecutado durante un período de 1 mes, con dos teléfonos inteligentes y dos conjuntos iguales de sensores en el mismo entorno doméstico. Las dos aplicaciones han recopilado registros de errores para evaluar la cantidad de fallas. Se ha verificado que la aplicación SANDR informó un total de dos errores críticos que conducen a la interrupción de la ejecución de la aplicación; Por otro lado, SCRATCH App reportó siete errores críticos. A partir de un análisis de registro, también ha sido posible verificar que los errores SCRATCH estaban relacionados principalmente con la conexión Bluetooth, que debía implementarse desde cero desde el desarrollador, mientras que los errores SANDR estaban relacionados con los errores de la interfaz de usuario. Incluso si a pequeña escala, la prueba permitió verificar que,[40].

Desde el punto de vista de la resolución, tenemos que distinguir entre la resolución de la estimación de la cantidad de interés y la resolución de tiempo de la detección del evento. De hecho, la situación puede ser diferente si se consideran los sensores integrados o externos. De hecho, en el caso de sensores integrados, los datos se obtienen por medio de la ASF. En cuanto a la resolución de la cantidad de interés, debe destacarse que la ASF generalmente adopta el tipo flotante Java, que está codificado con el estándar de coma flotante IEEE 754; por lo tanto, la resolución depende solo del rendimiento del sensor. Es interesante observar que estas consideraciones se aplican tanto a los sensores integrados como a los externos.

Por el contrario, con respecto a la resolución de detección de eventos, el sistema operativo Android introduce una limitación cuando se trata de sensores integrados. De hecho, el ASF permite al desarrollador establecer el retraso de datos (que se puede suponer que representa la velocidad de muestreo de los sensores integrados) entre 20 ms, 60 ms, 200 ms o "lo más rápido posible". Sin embargo, como lo indica la documentación oficial de Android: “El retraso que especifique es solo un retraso sugerido. El sistema Android y otras aplicaciones pueden alterar este retraso ". Este problema afecta a cualquier aplicación que se ejecute en un dispositivo Android, utilizando sensores integrados, independientemente del enfoque de implementación (CLA, AOA o FOA). Por otro lado, cuando se emplean sensores externos como en el caso de uso descrito, el ASF no se utiliza, los datos se transfieren mediante una conexión BLE. Según la especificación BLE, hasta 80 B de datos cada 7. Se pueden transferir 5 ms. Incluso si solo se considera una muestra (4 B como en el estándar IEEE 754) por paquete BLE, la frecuencia de muestreo efectiva es compatible con la opción de 20 ms proporcionada por la ASF. Una comparación detallada entre el rendimiento de SAndroidE y el ASF está fuera del alcance de este documento, como ya se ha descrito en un trabajo anterior[51].

**SECCION VII.**

## **Conclusión**

Los nuevos sistemas AAL podrían beneficiarse de arquitecturas abiertas de múltiples proveedores, a fin de disminuir el costo de soluciones altamente personalizadas. En este trabajo, una arquitectura FOA, basada en una versión mejorada del marco SAndroidE, ha sido explotada y probada en un caso de uso típico de asistencia en el hogar, comparando los resultados con un enfoque basado en CLA. Los resultados indican una reducción del esfuerzo de programación en el orden de un factor diez. La única competencia requerida es la programación Java en Android, incluso para dispositivos externos que normalmente requieren habilidades avanzadas de programación, no solo a nivel Android, sino también en el desarrollo/modificación del firmware del dispositivo.

### RECONOCIMIENTO

Los autores desean agradecer al Ing. Angelo Vezzoli por el soporte técnico durante la experimentación del sistema.