PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Isabel Bebelú Crisólogo Bohorquez

Carioca: Framework para la configuración de ambientes inteligentes utilizando dispositivos con recursos limitados

DISSERTAÇÃO DE MESTRADO

Tesis presentada como requisito parcial para obtener el título de Mestre por el programa de Pos-Graduación en Informática de la PUC-Rio.

Orientador: Prof. Hugo Fuks

Río de Janeiro

Junio del 2013

Pontifícia Universidade Católica
do Rio de Janeiro

Isabel Bebelú Crisólogo Bohorquez

Carioca: Framework para la configuración de ambientes inteligentes utilizando dispositivos con recursos limitados

Tesis presentada como requisito parcial para obtener el título de Magíster por el programa de Pos-Graduación en Informática de la PUC-Rio. Aprobada por la Comisión Examinadora que suscribe a continuación.

Prof. Hugo Fuks
Orientador
Departamento de Informática – PUC-Rio

Prof. Noemi de La Rocque Rodriguez
Departamento de Informática – PUC-Rio

Prof. Alberto Barbosa Raposo
Departamento de Informática – PUC-Rio

Prof. José Eugenio Leal
Coordinador Sectorial del Centro Técnico Científico da PUC-Rio

Rio de Janeiro, 21 de Junio de 2013

Isabel Bebelú Crisólogo Bohorquez

Graduada en Ingeniería Electrónica por la Pontificia Universidad Católica del Perú (PUCP), Perú. Su área de investigación actual es la Web de las Cosas y Redes de Sensores Inalámbricos. Trabajó en el área de seguridad electrónica y de automatización de sistemas de aire acondicionado y refrigeración. Desarrolló, como proyecto de maestría, un framework para ambientes inteligentes.

Datos Bibliográficos

Crisólogo Bohorquez, Isabel Bebelú

Carioca: Framework para la configuración de ambientes inteligentes utilizando dispositivos con recursos limitados / Isabel Bebelú Crisólogo Bohorquez; orientador: Hugo Fuks. – 2013.

81 f: il.(color); 30 cm

Dissertação (Mestrado)–Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2013.

Inclui bibliográfica.

1. Informática – Teses. 2. Framework. 3. Web das Coisas. 4. Redes de Sensores Sem Fio. 5. Prototipação. I. Fuks, Hugo. II. Pontifica Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

A mis padres,
María y Aurelio,
por su apoyo incondicional

Agradecimientos

A mi asesor, Profesor Hugo Fuks, por el apoyo, incentivo y por las horas dedicadas para la realización de este trabajo.

A la PUC-Rio por acogerme en esta mi nueva casa de estudios y a la agencia CAPES por el apoyo financiero para la realización de este proyecto.

A la Profesora Noemi Rodriguez, por permitir que utilice sus tarjetas electrónicas.

A Adriano Branco, por su valiosa orientación y sugerencias durante todo el desarrollo de este trabajo y por proporcionar su software que contribuyó bastante para el enriquecimiento de este trabajo.

A mis padres María y Aurelio, por creer en mí, por apoyarme, por su amor incondicional y porque son ejemplo de trabajo, dedicación y perseverancia. A mis hermanos Ricardina, Liliana y Alaín, por comprender mi ausencia durante todo el tiempo que me dedique a mis estudios.

A Edward y Katia por su apoyo y sus sugerencias por el desarrollo de este trabajo.

Y por último, a mi amigo Henry, que desde que se enteró de mi trabajo, de forma desinteresada, acompaño mi proyecto dando sugerencias y me respaldo con su experiencia.

Resumen

Crisólogo Bohorquez, Isabel Bebelú; Fuks, Hugo. **Carioca: Framework para la configuración de ambientes inteligentes utilizando dispositivos con recursos limitados.** Rio de Janeiro, 2013. 81p. Tesis de Maestria - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

En este trabajo presentamos un framework para disponibilizar “cosas” a la Web haciendo uso de una red de sensores inalámbricos con nodos de recursos limitados siguiendo el paradigma de la Web de las Cosas (Web of Things). Nos enfocamos en los tipos de nodos que no utilizan dirección IP y que poseen una capacidad de memoria limitada. Ofrecemos una herramienta reutilizable y configurable para escenarios de aplicación dentro da área de Smart Home que, en conjunto con la red de sensores, posibilita monitorear e intervenir en el ambiente físico a través de la Web. Para la implementación de aplicaciones con la red de sensores inalámbricos utilizamos la herramienta TERRA la cual nos ofrece configuración y programación remota de los nodos de la red. Realizamos una evaluación funcional y un caso de estudio. Con la evaluación del sistema se buscó probar que los objetivos planeados inicialmente se reflejan en nuestro software. El caso de estudio fue realizado considerando un usuario programador del framework. La principal contribución de este trabajo es ofrecer un framework para el monitoreo de nodos sensores, incluyendo la adecuación del framework para otras aplicaciones de ambientes físicos. Esa contribución es basada en un abordaje que posibilita la programación distribuida de nodos a partir de la herramienta TERRA en dispositivos de recursos limitados. Este trabajo presenta el proceso da prototipación del framework, los casos de aplicación y las dificultades que se presentaron.

Palabras clave

Web de las Cosas, Framework, Red de Sensores Inalámbrica, TERRA.

Resumo

Crisólogo Bohorquez, Isabel Bebelú; Fuks, Hugo. **Carioca: Framework para a configuração de ambientes inteligentes utilizando dispositivos com recursos limitados.** Rio de Janeiro, 2013. 81p. Dissertação de Mestrado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Neste trabalho apresentamos um framework para disponibilizar “coisas” na Web fazendo uso de uma rede de sensores sem fio com nós de recursos limitados seguindo o paradigma da Web das Coisas (Web of Things). Nós focamos nos tipos de nós que não utilizam endereçamento IP e que possuem uma capacidade de memoria limitada. Oferecemos uma ferramenta reutilizável e configurável para cenários de aplicação dentro da área de Smart Home que, em conjunto com a rede de sensores, possibilita monitorar e intervir no ambiente físico através da Web. Para a implementação de aplicações na rede de sensores sem fio utilizamos a ferramenta TERRA a qual nos oferece configuração e programação remota dos nós da rede. Realizamos uma avaliação funcional e um caso de estudo. Na avaliação do sistema buscou-se provar que os objetivos planejados inicialmente se refletem no nosso software. O caso de estudo foi realizado considerando um usuário programador do framework. A principal contribuição deste trabalho é oferecer um framework para o monitoramento de nós sensores, incluindo a adequação do framework para outras aplicações de ambientes físicos. Essa contribuição é baseada numa abordagem que possibilita a programação distribuída de nós a partir da ferramenta TERRA em dispositivos de recursos limitados. Este trabalho apresenta o processo da prototipação do framework, os casos de aplicação e as dificuldades que se apresentaram.

Palavras chave

Web das Coisas, Framework, Redes de Sensores Sem Fio, TERRA.

Abstract

Crisólogo Bohorquez, Isabel Bebelú; Fuks, Hugo. **Carioca: Framework for the configuration of smart environments using resource-constrained devices.** Rio de Janeiro, 2013. 81p. MSc. Dissertation - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

In this work we introduce a framework in order to have available "Things" in to the Web making use of a wireless sensor network with limited resource nodes following the paradigm of the Web of Things. We focused on those type of nodes that do not have an IP address and have capacity of limited memory.

We offer one tool that will be reusable and configurable for scenarios inside the area of Smart Home that united with the wireless sensor network make possible to monitor and intervene in to the physical environment through the Web. For the implementation of wireless sensor networks applications we used the TERRA tool which offers remote configuration and remote programming of the nodes. A functional evaluation and a case study were realized. In evaluating the system we looked for prove that the initial planned objectives was reflected in our software. The case study was realized for the programmer user of the framework. The most important contribution of this work is offering a framework for the monitoring of this sensor nodes and the adequacy of the framework to other applications of physical environments. These contributions were based on an approach that makes possible the distributed programming of nodes using TERRA in devices of limited resources.

This work shows all the process of prototyping the framework, the application cases, and the difficulties presented.

Keywords

Web of Things, Framework, Wireless Sensor Network, TERRA.

Tabla de contenido

Lista de figuras

Lista de tablas

1
Introducción

El objetivo de este capítulo es presentar una visión general de la investigación realizada para el proyecto. Abarca la motivación del tema de estudio, se definen los objetivos, el alcance y el cronograma del proyecto.

La organización de este documento fue dividida en seis capítulos. El capítulo 2 define el estado de arte del trabajo en el contexto actual, describiendo los principales conceptos a ser usados en la tesis: Web de las Cosas, redes de sensores inalámbricos y el hardware de los nodos. El capítulo 3 describe el framework Carioca. El capítulo 4 muestra el proceso de prototipación utilizado y los métodos y características de cada iteración. El capítulo 5 presenta las pruebas realizadas y los resultados obtenidos. El capítulo 6 expone las conclusiones del trabajo resaltando las principales contribuciones y limitaciones, y los posibles trabajos futuros.

1.1.Motivación

Para capacitar el espacio físico de modo que tendremos un ambiente inteligente es necesario un nuevo conjunto de hardware y software. Todo ello es posible gracias al desarrollo de la tecnología: la miniaturización de la electrónica, redes inalámbricas más fiables y los smartphones que nos posibilita tener acceso al Internet desde cualquier lugar. De las tecnologías citadas, el internet conecta computadoras a una red global. En esta red, la World Wide Web (Web) proporciona una plataforma global para el almacenamiento de información, intercambio de recursos, publicación de servicios, etc. La web evolucionó con el paso de los años. Al principio estaba formada por un entorno estático, se trataba básicamente de páginas que sufrían pocas actualizaciones y no tenían mucha interacción con el usuario. Poco a poco se ha producido una transición hacia lo que se denomina Web 2.0. El término Web 2.0 es usado para describir sites que usan la tecnología mucho más allá que paginas estáticas como anteriormente sucedía [1]. Tim O'Reilly y John Battellepor en "Web Squared: Web 2.0 Five Years On" [2] señalan que actualmente, la Web está aplicándose en áreas que no se habían predicho en el 2004, como dispositivos móviles y objetos

conectados a internet, en efecto, actualmente la web está dominando todos las áreas antes no imaginadas. Paralelamente a los avances de la web, también se desarrolló la computación ubicua con la ayuda de sensores, actuadores y dispositivos informáticos pequeños instalados en las personas y en el ambiente. Estas dos tendencias se empezaron a juntar denominándolo "Internet de las Cosas" [3] que crea una red de sensores y actuadores conectados al internet. La "Internet de las Cosas" es una red mundial de comunicación que se está expandiendo y está evolucionando con ayuda de tecnologías como redes de sensores inalámbricos y RFID. Con estas tecnologías, el mundo físico se convierte en "integrable" con los ordenadores, y posibilita que el estado de los objetos y su entorno sea accesible a los sistemas de software. El "Internet de las Cosas" evolucionó a lo que es llamado la "Web de las Cosas" [4] que utiliza el estándar HTTP como protocolo de aplicación y no solo como un protocolo de transporte para proporcionar conexión de sensores con el internet. La tendencia es que las cosas cotidianas se conecten cada vez más al internet. Una estimación reciente de Ericsson dice: "Para el 2020 habrán 50 billones de dispositivos conectados al internet de forma inalámbrica" [5]. Según Cearley [6] todo va conectarse a internet, incluyendo cameras, micrófonos, realidad aumentada, edificios y sensores en todos los lugares, en muchos casos, ya se hace presente. El Internet de las Cosas va conducir nuevos productos, como los basados en uso seguro o de políticas fiscales. También levantará nuevas cuestiones. "Estamos en un momento en que ya no es exagerado imaginar que mucho de lo que utilizamos esté ligado al internet" [6].

Un elemento clave de la "Web de las Cosas" es la habilidad para desplegar sensores con flexibilidad y movilidad. Una tecnología importante aquí es la tecnología de redes de sensores inalámbricos conocido como WSN (Wireless Sensor Network) [7].

Las redes de sensores inalámbricas tienen aplicaciones industriales, ya que reducen la brecha entre los sistemas tradicionales de empresas y el mundo real. Sin embargo, cada aplicación en particular es compleja en esfuerzo, técnicamente, y en tiempo que impide a los usuarios crear pequeñas aplicaciones ad-hoc mediante redes de sensores. Soluciones tradicionales de integración proponen implementar una arquitectura alternativa donde los nodos de sensor son accesibles usando los principios del REST [8]. Con este enfoque, los nodos pasan a formar parte de una "Web de las Cosas" e interactuar con ellos así como componer sus servicios con las existentes, llega a ser casi tan fácil como navegar por la web.

La red de sensores inalámbricos es una tecnología promisora, gracias a sus innumerables aplicaciones benéficas. La tendencia reciente de “Web of Things” es aprovechar tecnologías web sustanciales y kits de herramientas que simplifican enormemente la tarea de desarrollo de aplicaciones WSN. Sin embargo, el servidor web y la comunicación HTTP dificultan la portabilidad de aplicaciones de WSN y gestiones de recursos del nodo ya que algunas series de estos dispositivos no cuentan con dirección IP para el acceso directo de sus recursos como sensores y actuadores. Tanto, la comunicación HTTP y WSN presentan individualmente sus dificultades.

Uno de los desafíos presentes son los sistemas operativos para redes de sensores inalámbricos, los cuales no ofrecen opciones de reconfiguración de los nodos a distancia, dificultando ajustes en la programación después de estos dispositivos ser instalados en el ambiente físico obligando al desmontaje de estos nodos. Esos ajustes suelen ser desde configuraciones mínimas hasta la reprogramación completa de los nodos.

Teniendo en cuenta estas dificultades en WSN, la herramienta TERRA [9] ofrece una programación en lenguaje de alto nivel, una configuración y programación a distancia evitando así la recuperación de todos los nodos para su reprogramación. TERRA es compatible con tres familias de nodos ofreciéndonos la posibilidad de implementar aplicaciones indiferentes al hardware de los nodos.

Diversos trabajos están siendo dedicados para conectar cosas a la web usando redes de sensores inalámbricos, entre los más destacados está el framework Californium [10] con el software para sus nodos Erbium [11] basados en el protocolo de comunicación CoAP [12]. Este framework utiliza lo más último en tecnología de redes de sensores inalámbricos WSN: el estándar 6LoWPAN [13] y tiene dirección IPv6 [14] siempre con el protocolo CoAP de base para su comunicación. El Californium necesita de motes con memoria suficiente para soportar el IPv6 y el CoAP. Este framework abarca desde servidor, software para la WSN y un plugin en navegador Web especial para gerenciar los nodos; lo tomaremos como ejemplo de desarrollo de framework para nuestro proyecto por ser uno de los más completos actualmente.

En este trabajo se presenta un framework configurable para su comunicación con aplicaciones Smart Home y que posibilite múltiples tipos de aplicaciones para ambientes físicos como oficinas, hogares, clínicas, etc. Proponemos que los objetos cotidianos, se conecten a la Web utilizando URI estandarizado y configurables que describan por si solos el tipo de recurso que

se está conectado a la Web así como también la acción que se pretende hacer en los dispositivos y así facilitar al programador WEB la construcción de aplicaciones Smart Home. El foco principal de nuestro trabajo es basado en esa idea, el cual es posible utilizando tecnologías como WSN con la herramienta TERRA y la utilización del estándar HTTP para conectar cosas como artefactos eléctricos a la WEB.

El objetivo principal de nuestro trabajo es integrar la red de sensores inalámbrica con nodos limitados y disponibilizar los datos de los actuadores y sensores al usuario mediante el protocolo HTTP siguiendo como modelo el framework Californium. Nuestro Framework ofrece una interfaz Web de configuración y un modelos de URI configurable y estandarizado, que son recursos que el Californium no ofrece a los usuarios. Se tiene presente que el hardware de los nodos son más limitados que los usados por el Californium, sin embargo ofrecemos una alternativa para que nuestro sistema cumple de forma similar el propósito del Californium el cual es conectar cosas cotidianas a la WEB.

1.2.Definición del proyecto

En esta sección se define los objetivos, el alcance y el cronograma del proyecto.

1.2.1.Objetivos

Se definieron el objetivo general y los específicos del proyecto.

Objetivo general

Implementar un framework para capacitar espacios físicos en ambientes inteligentes con dispositivos de bajos recursos.

Objetivos específicos

- Reconfigurar ambientes físicos usando las tarjetas Mica2 con el concepto del Web de las Cosas;
- Demostrar que con dispositivos de recursos limitados también es posible desarrollar aplicaciones para disponibilizarlos al internet;
- Hacer un framework reutilizable para otras aplicaciones dedicadas a ambientes físicos;

- Ofrecer un framework que posibilite el monitoreo de ambientes físicos a través de una interfaz web para usuarios y una URL estandarizada para otras aplicaciones web; y
- Programar de forma distribuida los puertos de entrada y salida de los nodos usando la herramienta TERRA.

1.2.2.Alcance

A continuación se mencionan los puntos tomados en cuenta para delimitar el alcance del proyecto:

- TERRA se usará para la programación de los nodos;
- El framework será dedicado solo para espacios físicos y no para otro tipo de aplicaciones;
- Se utilizará la tarjeta Mica2 para nuestras pruebas entre la comunicación inalámbrica de los nodos y nuestro framework;
- Nuestro sistema desarrollado es mono-hilo (monothread); y
- Se proveerá de un manual de usuario para la configuración del framework.

1.2.3.Cronograma del proyecto

A continuación se muestran las actividades para el desarrollo de la tesis de maestría. Así como la duración de cada una de ellas. La Figura 1 muestra las actividades a un alto nivel. El trabajo se dividió en 4 principales actividades: El estudio del marco conceptual, el framework, casos de aplicación y el desarrollo del documento.

Actividades	2012					2013				
	Ag	Set	Oct	No	Di	En	Feb	Mar	Abr	May
Definición del tema										
Estudio del marco conceptual, WDvm y TERRA										
Planificación y especificaciones de prototipación del framework										
Implementación del framework										
Implementación de casos de uso										
Pruebas y resultados										
Desarrollo del documento										

Figura 1 Cronograma de actividades

Como se observa en el cuadro de cronogramas, el tema de maestría se definió en enero, a partir de allí se definen las especificaciones del framework y se empezó la implementación.

Este documento detalla el proceso seguido para desarrollar el Framework, la evaluación funcional y un caso de estudio.

2
Marco teórico

En este capítulo se definen los conceptos principales a ser utilizados en este documento: Web de las Cosas, redes de sensores inalámbricos, el software TERRA y RESTful. Se hace una descripción del módulo Mica2 el cual forma nuestra red de sensores inalámbricos, además se presenta una sección de trabajos relacionados y el framework Californium, que será utilizado como modelos para nuestro prototipo. Por último se describe algunos escenarios del proyecto.

2.1.Web de las Cosas

En la coyuntura actual existe una necesidad, a ritmo acelerado, de que las "cosas" se conviertan en dispositivos "inteligentes" conectados a la Web para compartir datos y, además, estos dispositivos inteligentes se les programe para tomar decisiones basadas en sus propios datos y datos de otros fuentes. Es así como la “Web de las Cosas” [4], [15], surgió con una visión en la que los dispositivos cotidianos se integren a la Web a través del uso de estándares WEB, Por ejemplo, en una red de sensores, cada sensor tiene un URI (que es un recurso) y es posible consultar su lectura (valor que se obtiene mediante el acceso a dicho recursos).

Gracias a este enfoque, las cosas físicas son accedidas mediante un URI, permitiendo el uso de toda la gama de tecnologías web. El paradigma de la "Web de las Cosas, ha llevado a la implementación de frameworks basados en la Web creando una capa de servicio que permite virtualizar objetos instalados en espacios físicos y ofrecerlos a las aplicaciones Web y usuarios. El uso de la Web como plataforma de alojamiento y la exposición de objetos conectados, ofrece múltiples beneficios tecnológicos algunos de los cuales incluyen flexibilidad (agnóstico aplicación), la alta disponibilidad, el despliegue y el uso de protocolos de comunicación estandarizados.

En la Web de las Cosas, sensores y actuadores desempeñan un papel central, ya que constituyen la interfaz física entre el mundo digital y real: Posibilitan capturar y modificar los aspectos del mundo real, en tiempo real. La

idea es ofrecer al usuario Web la capacidad de componer servicios basados en objetos físicos para un usuario final.

Tomando como base este concepto, se implementa el framework Carioca para la integración de los dispositivos físicos al internet.

2.2.Redes de sensores inalámbricas (WSN)

Una red de sensores inalámbricos (Wireless Sensor Network o WSN) [16] es una red que consiste en una gran cantidad de pequeños dispositivos, autónomos, distribuidos físicamente, llamados nodos de sensores, instalados alrededor de un fenómeno para ser monitoreado, con la capacidad de almacenar y comunicar datos en una red de forma inalámbrica. Un sistema WSN incorpora un gateway que provee conectividad inalámbrica de regreso al mundo de cables y nodos distribuidos. El protocolo inalámbrico de uso depende de los requerimientos de la aplicación. Algunos de los estándares disponibles incluyen radios de 2.4 GHz basados en los estándares IEEE 802.15.4 o IEEE 802.11 (Wi-Fi) o radios propietarios, los cuales son regularmente de 900 Mhz.

La carga de trabajo de redes de sensores inalámbricos es de detección pertinente de información por los nodos, recoger la información, interpretarla, grabarlo y comunicar a una estación de base. Con el fin de realizar dichas actividades cada nodo sensor está formado por algunos sensores, un transceptor de radio y un microcontrolador. Como fuente de energía por lo general tienen una batería.

Las redes de sensores están formadas por un grupo de sensores con ciertas capacidades sensitivas y de comunicación inalámbrica los cuales permiten formar redes Ad Hoc sin infraestructura física preestablecida ni administración central. A continuación describimos los elementos que constituyen una WSN:

SENSORES: De distintos tipos y tecnologías los cuales toman del medio la información y la convierten en señales eléctricas;

NODOS: Toman los datos del sensor a través de sus puertas de datos, y envían la información a la Base Estación;

GATEWAY: Elemento para la interconexión entre la red de sensores y una red TCP/IP;

BASE ESTACIÓN: Comunicador de datos entre la red de nodos y el ordenador; y

RED INALÁMBRICA: Basada en el estándar 802.15.4 ZigBee, IEEE 802.11 o radios de 900 Mhz.

Las redes de sensores es un concepto relativamente nuevo en adquisición y tratamiento de datos con múltiples aplicaciones en distintos campos tales como entornos industriales, domótica, entornos militares, detección ambiental. Sin embargo la red de sensores se resume a dos típicos casos:

Caso 1.

Posibilitar interacción entre los dispositivos del mismo ambiente, es decir, a partir de uno, activar o desactivar el otro. Por ejemplo, un interruptor y un foco. A partir del interruptor se encienda o apague el foco. Como ambos dispositivos no cuentan con servidor web integrado, se les proveerá a cada uno de los módulos.

Caso 2.

En el siguiente caso, con la integración de servidores para cada dispositivo, solo se exportará su funcionalidad elemental utilizando los recursos REST. La lógica de la aplicación de dispositivos se ejecuta en un servidor de aplicación. Por ejemplo, una sala de presentación inteligente donde el estado de cada dispositivo podrá ser observado en la Web y se hará uso una aplicación lógica para que la sala de presentación, se acondicione a las necesidades del presentador sin su intervención en la hora programa.

En nuestro tema de tesis, utilizaremos nuestra red de sensores inalámbricos siguiendo el modelo del caso 2.

Respecto al hardware, una WSN está formada por un conjunto de dispositivos denominados nodos sensores. De estos nodos sensores encontramos actualmente en el mercado una gran variedad de tipos de distintos fabricantes. El tipo de mota que se ha utilizado para este proyecto han sido las motas Mica2 (900 Mhz). A continuación una descripción de este hardware.

2.2.1.Mica2

El Mica2 [17] (ver Figura 2) es la tercera generación de módulos utilizado para permitir el bajo consumo de energía, comunicación por radio y redes de sensores. En el Mica2 se ejecuta el TinyOS que es un sistema operativo de

código abierto fabricado especialmente para su uso en redes de sensores inalámbricos.

Mica2 tiene un conector que proporciona un gran número de pines de E/S y opciones de expansión, el Mica2 es un nodo para cualquier tipo de aplicaciones. En la extensión se conectan tarjetas de sensores, como por ejemplo el MTS300CA que tiene sensor de luz, temperatura, micrófono y un pequeño altavoz. Estos módulos son conectados a detectores de movimiento, de puerta-ventana y sensores aplicados a la construcción de sistemas de seguridad.

Además, el Mica2 es capaz de recibir y enviar mensajes a otros Mica2 incluidos los ordenadores.

2.2.2.Plataforma Mica2 wireless sensor

Está basado en el microcontrolador Atmel ATmega 128L. El ATmega 128L es de baja potencia y vienen en tres modelos según su banda de frecuencia RF: el MPR400 (915 MHz), MPR410 (433 MHz), y MPR420 (315 MHz) [18]. Los nodos usan el Chipcon CC1000 para la modulación de la frecuencia. Todos los modelos utilizan el ATMEGA128L y una radio frecuencia ajustable con rango extendido. La frecuencia de los MPR4x0 y MPR5x0 son compatibles y se comunican entre sí. El Mica2 tiene un conector con 51 pines de expansión que son compatibles con entradas analógicas, Digital I/O, I2C, SPI, UART [17]. Para el desenvolvimiento de este proyecto se hará uso de esta interfaz para monitorear sensores y controlar actuadores. En la Figura 3 se muestra el diagrama de bloques del Mica2.

Figura 2 Modulo del Mica2

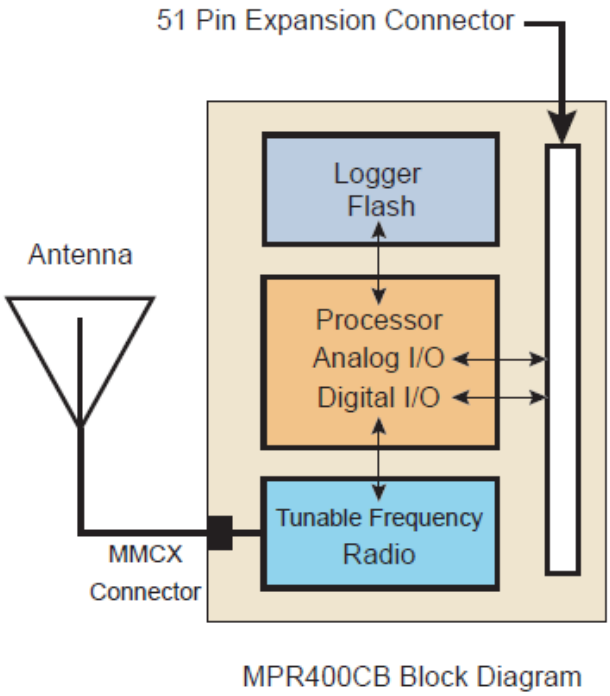

Figura 3 Diagrama de bloques del Mica2

Las necesidades que tiene un nodo de una WSN son distintas a las que tiene cualquier otro dispositivo como una PC, por ello los nodos de una WSN tienen sus propios sistemas operativos. Los sistemas operativos para WSN difieren respecto a los de propósito general, tanto debido a los requisitos especiales de las aplicaciones en las que se usan, como a las restricciones de recursos encontradas en las plataformas hardware utilizadas. Entre los más conocidos son el TinyOS y el Contiki. Para nuestro proyecto usaremos el TERRA basado en el TinyOS. A continuación una descripción de este software.

2.2.3.TERRA

Diversos autores han propuesto soluciones para actualizar remotamente las aplicaciones de redes de sensores inalámbricos como por ejemplo en [19] el cual es implementado como una máquina virtual dinámicamente extensible, que posibilita ejecutar un scripts de alto nivel escritos en código byte portátil. Una propuesta similar es del WDvm (Wireless sensor network Dynamic virtual machine) [20]. El WDvm es una plataforma que proporciona una máquina virtual que se ejecuta sobre TinyOS y que se instala con diferentes componentes prefabricados. Es una herramienta tiene un lenguaje de programación de bajo nivel tipo assembler y es cargada remotamente a la VM. Los comandos proporcionados por el sistema abstraen el código de programación del nivel de

los nodos. La abstracción permite que se implementen programas de una forma más abstracta y sean utilizados en una grande variedad de aplicaciones. También proporciona una programación y reconfiguración remota evitando así la desinstalación de los motas para su programación. Una característica de esta herramienta es que todos los nodos de la red de sensores son programados con el mismo programa.

Inicialmente se utilizó el WDvm para la implementación de nuestro tema de tesis, sin embargo como todo software, evoluciona ofreciendo algunas funciones adicionales surgiendo así una nueva versión: un WDvm mejorado llamado TERRA [9].

El TERRA es un sistema basado en máquinas virtuales configurables de aplicación específica con un lenguaje de programación reactiva que se analiza estáticamente para evitar la ejecución infinita y conflictos de memoria. Este enfoque posibilita la flexibilidad de cargar remotamente código en los nodos.

El sistema utiliza el lenguaje de programación Céu [21] el cual es de alto nivel y es compilada para a lenguaje assembler para después ser cargada remotamente a la VM. TERRA simplifica el desarrollo de nuevas aplicaciones (Wireless Sensor Network) combinando un conjunto de componentes de alto nivel (WD) con un nuevo lenguaje de programación (Céu).

El TERRA mantiene las características del WDvm, por ejemplo los componentes son los mismos, ambos ofrecen una programación a distancia, son basados en el TinyOS, la programación de los nodos es distribuida, y su MV funciona en varios tipos de nodos como Mica2, Micaz y TelosB. Sin embargo el gran diferencial entre ambos es el lenguaje de programación. TERRA usa un lenguaje más fácil de utilizar y permite algunas construcciones más complejas que el WDvm. Por otro lado, un programa assembler en TERRA tiende a ser mayor que el equivalente en WDvm, pues TERRA precisa incluir más controles para atender esos nuevos tipos construcciones.

Operaciones Locales

El TERRA ofrece operaciones locales de lectura de sensores, la energía de la batería, configuración de los LEDs de la tarjeta, y el acceso a otros dispositivos conectado en los pines de entrada y salida del nodo. Esta herramienta será ampliamente usada para el uso de lectura (sensores) y escritura (actuadores) en los periféricos. Para eso se hará uso de las operaciones que dan acceso a las entradas y salidas del nodo. Actualmente TERRA tiene implementado las siguientes operaciones de acceso:

Tabla 1 Operaciones de acceso a los periféricos

Instrucción	Descripción
TEMP	Sensor de Temperatura
PHOTO	Sensor de Luminosidad
LEDS	Leds de la tarjeta
VOLT	Sensor del voltaje de la batería
PORT_A	In/Out digital del pin 1
PORT_B	In/Out digital del pin 2
INT_A	Pin 1 de interrupción
INT_B	Pin 2 de interrupción

Los pines I/O (puerto A y B) son digitales y tienen que estar configurado como entrada o salida antes del uso. La lectura de un puerto de entrada es utilizada como lectura de un sensor y la escritura de 1/0 un pin de salida activa o desactiva el dispositivo conectado. Los pines habilitados como interrupciones nos posibilitan realizar diferentes aplicaciones para eventos inesperados.

De este modo, TERRA nos ofrece operaciones locales importantes para el desenvolvimiento del proyecto, además, la arquitectura del TERRA flexibiliza la integración con nuevos componentes de la VM y eso posibilita la utilización de esta VM para otros tipos de dispositivos.

2.3.La Web orientada al Framework

Para la interconexión física de los objetos, se utiliza la Web como capa de aplicación y comunicación. Así, aprovechamos los conceptos existentes para integrar los objetos cotidianos. Esta sección, propone que el protocolo de la Web sea utilizado para que los dispositivos se comuniquen mediante el mismo idioma con cualquier recurso en la Web. A continuación, se hace una descripción del protocolo HTTP y el URI, conceptos que nos serán útiles para construir una infraestructura para la interacción con los dispositivos en la Web.

2.3.1.Protocolo HTTP

De acuerdo con Fielding [22], el HTTP o “protocolo de transferencia de hipertexto” (HiperText Transfer Protocol), utiliza el modelo cliente-servidor, como la mayoría de los protocolos de red, basándose en el paradigma de solicitud y

respuesta. Un solicitante (cliente) establece una conexión con otro programa receptor (servidor) y le envía una petición en forma de método, una URI, y una versión de protocolo seguida de los modificadores de la petición de forma parecida a un mensaje MIME, información sobre el cliente y al final un posible contenido. El servidor contesta con una línea de estado que incluye la versión del protocolo y un código que indica éxito o error, seguido de la información del servidor en forma de mensaje MIME y un posible contenido.

2.3.2.URI

Un identificador uniforme de recursos o URI (Uniform Resource Identifier) es una cadena de caracteres que identifica inequívocamente un recurso (servicio, página, documento, dirección de correo electrónico, etc.). Normalmente estos recursos son accesibles en una red o sistema.

Nosotros utilizaremos el URI para identificar recursos como sensores y actuadores, también, para identificar acciones que deseamos ejecutar en estos recursos.

Las URIs de los recursos deben poseer una correspondencia intuitiva con el recurso que identifican y su estructura debe tener una forma previsible. Por ejemplo, cuando se observa el URI http://www.puc-rio.br/laboratorio/temperatura el cliente esperará que el dato de temperatura sea accesible en el directorio “laboratorio/temperatura” y que esa temperatura sea correspondiente al ambiente “laboratorio” (Ver Figura 4).

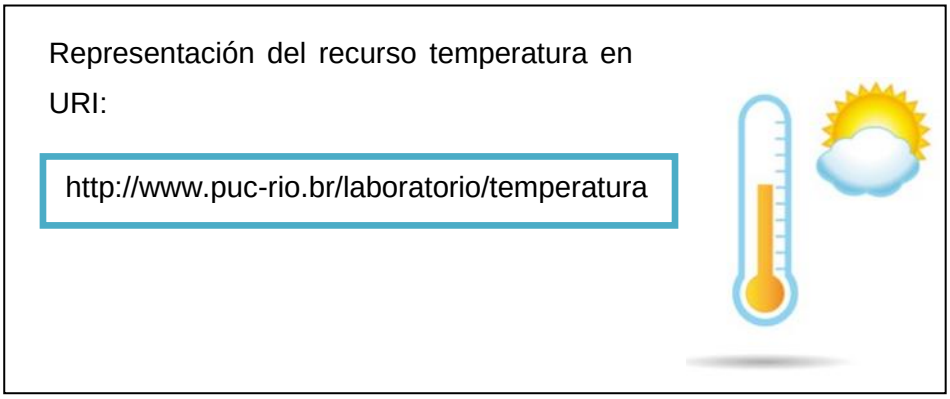

Figura 4 URI de la temperatura

En la Web, el cliente tiene acceso a recursos a través de por lo menos una URI, así la información es transferido entre el mismo y un servidor [23]. Con estos conceptos claros, en el capítulo 3 vamos a esclarecer con más detalle como pretendemos usar el URI.

2.4.Escenarios

Las situaciones hipotéticas descritas a continuación serán utilizadas como base para el desarrollo del Framework. Se presentan dos escenarios que ponen en destaque la necesidad del usuario en diferentes ambientes.

Escenario 1:

> En un centro clínico, una María es enfermera y necesita que el sistema Smart Home instalado en el ambiente le facilite información sobre la temperatura y humedad en la sala de cuidados intensivos de una forma constante sin la necesidad que ella este solicitando la información constantemente. Por lo tanto, el sistema devuelve los resultados correspondientes y estos datos en tiempo real son monitoreados por María desde un computador. Sin embargo, ella no siempre está frente al monitor, es por ello que, el sistema fue programado para que si la temperatura es mayor de 21 grados centígrados, entonces, se active una alarma en los corredores de la clínica, así ella u otro funcionario tomara decisiones al respecto.

Escenario 2:

> Pensemos ahora en el caso un ambiente totalmente distinto como una fábrica, en esta, la temperatura es también muy importante, por lo tanto, es un recurso monitoreado contantemente, y caso la temperatura pase de los 35 grados centígrados, se activará una alarma de incendio.

En el primer escenario, un sistema (o una aplicación) necesita que los datos se actualicen en tiempo real sin necesitad que el usuario lo solicite. En el segundo escenario, el control de temperatura es importante en una clínica como en una fábrica, el ambiente cambia pero la función de control de temperatura es la misma. Es algo que se debe consideras para la implementación de nuestro framework.

Escenario 3:

> María en casa tiene un bebe de cuatro años, mientras ella cocina su hijo está jugando en el jardín. Ella tiene miedo que su hijo abra la puerta de la calle y salga mientras que ella está ocupada, es por eso que mediante el sistema Smart Home sabrá si ese tipo de eventos inesperados suceden sin la necesidad de monitorear el estado de la puerta cada intervalo de tiempo puesto que durante ese intervalo existe la posibilidad de que suceda ese escenario.

En este escenario, un usuario quiere ser informado cuando suceden eventos inesperados. Para la implementación de nuestro framework se abordara estos casos y se propone una solución.

2.5.Trabajos relacionados

Entre los trabajos referentes a ambientes inteligentes tenemos el de Lidan [24] el cual no utiliza framework ni red de sensores inalámbricos. Es una propuesta que tiene por objetivo el diseño, construcción y evaluación de un ambiente inteligente para hacer presentaciones, haciendo que todas las tecnologías de la dispositivos envueltos para este fin sean invisibles para el presentador, es decir, para eliminar las dificultades de configuración y compatibilidad para que el usuario sea capaz de hacer su presentación. El autor presentó una solución específicamente para el escenario de espacios para presentaciones.

Entre los trabajos relacionados, tenemos el de Ostermaier [25], el cual, propone un framework para el emergente Web de las Cosas. Se compone de varios módulos para la integración de las cosas a la Web, incluyendo sus sensores y actuadores. En este concepto los objetos del mundo real son introducidos en el WWW mediante la representación como recursos Web, que se acceden usando APIs basados en los principios REST. Los objetos del mundo real incluyendo sus sensores y actuadores son expuestos en la red mediante URLs permitiendo nuevos escenarios de aplicación. Utiliza los métodos PUT, GET, POST y DELETE, que se usan para crear, leer, actualización y eliminación de datos. Cuando fue publicado ese trabajo, los autores trabajaban para

proporcionar conceptos unificados para interactuar con los objetos del mundo real. También estaban enfocados en la seguridad, pues es un tema importante, para ellos el tradicional username/contraseña es un esquema de autenticación que consideran adecuado para una Web de las Cosas. Del mismo modo, ellos no tienen desarrollado un mecanismo de descubrimiento de elementos en la Web de las Cosas, sin embargo consideran que es necesario. Este mismo grupo de trabajo desarrolló el Californium [10] que es tomado como modelo de framework para nuestro proyecto.

Entre otros trabajos el de Kamilaris [26] se enfoca también en la Web de las Cosas, propone un avance considerable respecto el anterior trabajo. El autor propone un framework para dispositivos embebidos introduciendo el concepto de red de sensores inalámbricos. Para su propuesta utiliza el mote TelosB. Es un framework donde todos los recursos tienen un URI y para la manipulación de esos recursos son usados los métodos PUT/GET. El autor propone un sistema "Ad Hoc Device Discovery" pues considera que la detección de dispositivos es un tema crucial en las redes Ad Hoc. El Ad Hoc Device Discovery es un proceso dinámico que consiste en el descubrimiento de nuevos dispositivos integrados y registran su información básica. En resumen, el autor tiene claro las tres posibilidades de interacción con diferentes dispositivos de información que cubren los casos de uso más importantes que deben ser abarcados por el framework:

- Ad-Hoc interacción. Los usuarios tienen acceso directamente a los sensores y actuadores (en el modelo de solicitud y respuesta);
- Monitorización. Los datos de los dispositivos son monitorizados a cada intervalo de tiempo (por ejemplo medidores inteligentes que miden el consumo de energía de los aparatos eléctricos); y
- Sistemas basados en evento. Los eventos que se producen en los nodos se envían esporádicamente al servidor cuando algo importante sucede (por ejemplo, la indicación de incendio).

Además, proponen una lista de requisitos que la mayoría de los frameworks existentes para dispositivos embebidos consideran necesarios en entornos ubicuos a gran escala, abiertos y heterogéneos:

- Exportar datos a las aplicaciones Web en formatos estándar;
- La interoperabilidad de la Interfaz de programación que proporciona las primitivas a los usuarios finales con poca experiencia en programación para realizar algunas tareas avanzadas;

- Concurrente, soporte multiusuario a través de la Web; y
- Acceso uniforme para dispositivos embebidos heterogéneos. El espacio ubicuo se convierte en una nube donde se acceden individualmente a cualquier dispositivo de una manera estandarizada.

El autor también se basa en los cuatro conceptos básicos para una Arquitectura orientada a los recursos [27]:

- **Recursos.** Un recurso en un ROA es algo lo suficientemente importante como para ser referenciado y vinculado con otros recursos. Se puede ver como una entidad que proporciona algún servicio;
- **Su Nombre**. Un recurso tiene que ser direccionable, es decir, se necesita tener un URI (Uniform Resource Identifier) que identifica de forma exclusiva. Este concepto se conoce como el requisito de direccionamiento de RESTful API;
- **Los vínculos entre ellos**. Gracias a la estructura de los hiperenlaces de la Web, los recursos son relacionados y conectados entre sí. Este es el requerimiento para la conexión de RESTful API; y
- **Los recursos son representan utilizando diversos formatos.** Para la descripción del recurso el autor utiliza el formato XML.

El mismo autor un año después (2011) produce el documento “HomeWeb: An Application Framework for Web-based Smart Homes” [28] acrecentando el protocolo IPv6 al documento anterior. En este nuevo proyecto, se ejecutan aplicaciones IPv6 directamente en los nodos. Se cree que la tecnología de internet, utilizando el IPv6, se convertirá en el futuro estándar en la automatización del hogar. Además, en ese proyecto, utiliza la red de sensores inalámbricos y el sistema también es “Ad Hoc Device Discovery”. Su propuesta de framework es orientado a una aplicación funcional, basado en la Web para hogar inteligente. El autor aprovecha los recientes avances en las tecnologías como IPv6 y 6LoWPAN para mejorar la funcionalidad del framework con los dispositivos directamente habilitados en la Web. El autor desarrolló una interfaz gráfica Web para la automatización del hogar y ayudar a los residentes a visualizar el contexto ambiental. Esta aplicación cliente fue implementada en

JavaScript, utilizando el Google Web Toolkit (GWT6). Se comunica con el framework de aplicaciones a través de su interfaz RESTful.

Diferente a lo que anteriormente fue descrito como trabajos relacionados, se destaca el framework Californium. Su diferencial con el trabajo de Kamilaris es el protocolo Constrained Application Protocol (CoAP). En la sección 2.6 se explica detalladamente este framework.

2.6.Framework Californium

Fue pensado para sistemas con redes de sensores inalámbricos de bajo consumo de energía. Está basado en el protocolo de internet IPv6 juntamente con el último estándar de bajo consumo de energía inalámbrico: 6LoWPAN [13] y CoAP [12].

El CoAP es un protocolo de software que es utilizado en los dispositivos electrónicos para comunicarse interactivamente vía Internet. Es un protocolo para sensores de baja potencia, interruptores, válvulas y componentes similares que tienen que ser controlados o supervisados remotamente, por una red estándar de Internet. CoAP es un protocolo para el nivel de aplicación que está destinado para su uso en dispositivos de recursos limitados con acceso a internet tales como algunos nodos de WSN. CoAP está diseñado para traducirse a HTTP para la integración simplificada con la Web, también cumple con requerimientos especializadas como el soporte multicast.

EL Californium es un framework CoAP implementado en JAVA, está diseñado como back-end cliente - servidor y como un proxy CoAP.

Californium tiene como objetivo blindar a los usuarios los detalles del CoAP con el fin de proporcionarles un framework que interactúe con los puntos finales (endpoints) CoAP o proporcionar servicios específicos. Con eso, los usuarios no tienen que lidiar con las partes internas como retransmisiones de mensajes y manejo de observación [29].

El cliente está habilitado para ejecutar las operaciones RESTful: GET, POST, PUT o DELETE en los objetos, para eso necesita especificar el URI de la tarjeta endpoints de destino. Después de la ejecución de la petición, la respuesta resultante es procesada de forma síncrona o asíncrona.

La configuración de un servidor CoAP con el Californium sólo requiere definir sus recursos para providenciar subclases que implementen un controlador de peticiones GET, POST, PUT y/o DELETE. Por lo tanto, no se requiere el envío de mensaje personalizado.

El Framework Californium cumple tres funciones diferentes:

- CoAP servidor en el back-end;
- El cliente para CoAP; y
- El proxy HTTP-CoAP.

Respecto al protocolo CoAP, el modelo de interacción es semejante al modelo cliente/servidor de HTTP. Un pedido CoAP es equivalente al de HTTP y es enviado por un cliente para solicitar una acción (usando un código de método) en un recurso (identificado por un URI) en un servidor. El servidor envía una respuesta con un código de respuesta, lo que también, incluir una respuesta del recurso.

Para el acceso a los recursos CoAP vía HTTP, se utiliza un proxy que mapea las solicitudes HTTP a CoAP.

EL CoAP está basado en UDP (User Datagram Protocol) [30]. CoAP se basa en el intercambio de mensajes cortas que son transportados sobre el estándar UDP (es decir, cada mensaje CoAP ocupa una sección de datos de un datagrama UDP). También soporta el uso de multicast.

El Californium propone que los aparatos físicos sean parte significativa del Internet de las Cosas. Estos aparatos serán capaces de comunicarse, exportar sus datos a interfaz de usuario y ofrecer un API abierta con todas sus informaciones y funcionalidades, como por ejemplo, el tiempo de ejecución de la máquina de lavar ropa, el estado del sensor de luz, de la tv, o el control de sus luces. Una refrigeradora inteligente, por ejemplo, podría avisar cuando hay producto a punto de estragar [31].

2.7.Erbium

Érbium (Er) es un REST Engine de bajo consumo de energía y que incluye una implementación CoAP para el sistema operativo Contiki [32]. El Erbium soporta draft-ietf-core-coap-03, draft-ietf-core-coap-12, y draft-ietf-core-coap-13, juntamente con la transferencia por bloques y el observador [33].

Los dispositivos que ejecutan el Erbium (Er) REST Engine exportan su funcionalidad sin acoger cualquier lógica de la aplicación. Por lo tanto, sirven para múltiples aplicaciones que se ejecutan en el back-end sin reprogramar [10].

El Erbium funciona en tarjetas que soportan el sistema Operativo Contiki, en otros sistemas operativos no funciona.

2.8.Copper (Cu)

Otra forma que tiene el usuario para interactuar directamente con los dispositivos es usando el Copper (Cu) [34]. Este plugin es una extensión para el navegador Web Mozilla Firefox, por lo tanto el usuario interactúa usando la Web. La extensión registra un controlador de protocolo CoAP. Así, un URI CoAP se introduce en la barra de direcciones para acceder a la Web del nodo.

2.9.Californium y Erbium

En [10] el Erbium es ejecutado en la tarjeta Tmote Skys (actualmente llamada TelosB). A esta tarjeta se conectan dispositivos para exporta sus funcionalidades a través de una API RESTful. La lógica de la aplicación se separa y se ejecuta como 'APP' en la nube, es decir, en cualquier equipo con conectividad IP y es en ese equipo que el Californium debe estar ejecutándose.

En la Figura 5 se muestra un ejemplo de objetos interactuando con el Erbium, Californium y el Cooper. Los objetos cotidianos como interruptor, foco, enchufe y máquina de lavar son conectados a tarjetas (como el TelosB) que en conjunto con el Erbium (Er), exportan sus datos y funcionalidades posibilitando que el usuario final obtenga estos datos e interfiera directamente en estos objetos a través del Cooper (Cu) o también, a través de una aplicación Web basada en el Californium (Cf) con el cual diferentes dispositivos como tablets accederán a los objetos mediante el protocolo HTTP.

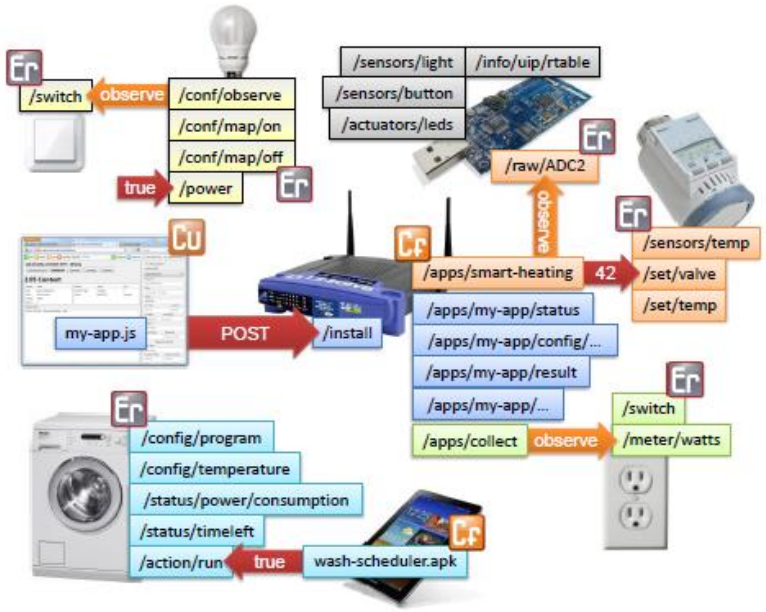

Figura 5 Prototipo del Copper (Cu), Californium (Cf), y Erbium (Er) [34]

2.10.Interacción del usuario con los nodos

El framework Carioca ofrece al usuario dos formar de tener acceso a los dispositivos conectados a los nodos: a través del plugin Copper (Cu) o por medio de una aplicación Web. La Figura 6 ilustra las dos formar de acceso a los dispositivos con sus respectivos protocolos usados.

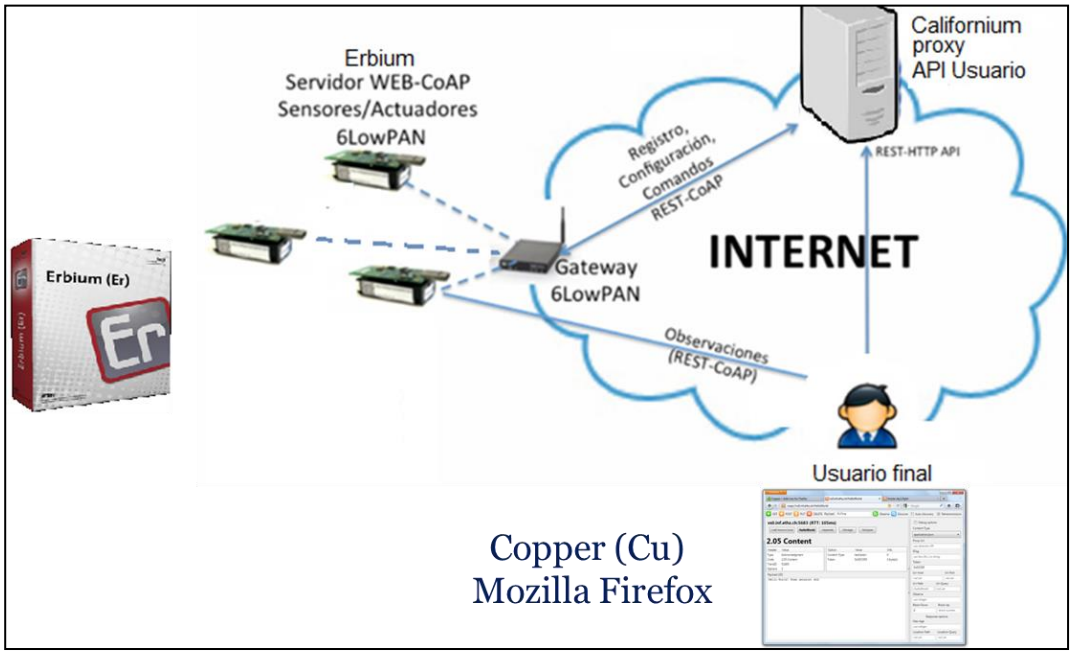

Figura 6 Visión general de framework Californium

A través del plugin Copper (Cu)

El usuario instala el plugin en el navegador Firefox y a través de este interactúa directamente con el dispositivo conectado al nodo, es decir, hace lectura de los datos de los sensores y activa o desactiva los actuadores. El protocolo de comunicación entre el nodo y el navegador Firefox es el CoAP (Ver Figura 6).

A través del Californium (Cf)

La Figura 6 ilustra también el camino de comunicación de la red de nodos con una aplicación Web. Como la red de nodos tiene el protocolo de comunicación CoAP, el proxy del Californium se encarga de transformar el protocolo CoAP a HTTP y viceversa. Las aplicaciones clientes utilizan el protocolo HTTP que funciona como un protocolo de Request-Response en el modelo computacional cliente-servidor, donde el servidor es el Californium.

3
Descripción y características del Framework Carioca

En este capítulo se hace una descripción del Framework, las partes que la componen y sus características. Esta sección abarcará el protocolo de comunicación entre el framework, los nodos y la aplicación Web. El prototipo de framework para la Web de las Cosas no se basa en una infraestructura centralizada. Este prototipo hace hincapié en la integración de sensores y actuadores a la Web.

3.1.Descripción del framework Carioca

El Carioca es un framework para dispositivos limitados como el Mica2 para integrar cosas cotidianas a la Web. Para ellos, se toma como modelo de ejemplo el framework Californium. El Californium propone un framework basado en el protocolo CoAP para integrar los dispositivos a internet usando módulos como el TelosB y Micaz. La idea central es hacer un framework que sea adaptable para todo tipo de aplicaciones enfocadas a convertir espacios físicos en ambientes inteligentes.

Nuestra propuesta es que el framework Carioca adaptable a diferentes ambientes físicos como por ejemplo: salas de reuniones, Hogares, laboratorios, taller de manufactura, ambientes clínicos, etc.

En todos estos ambientes citados tienen dispositivos como interruptor de luz, aire acondicionado, televisor, ventilador, plancha, motor, etc. Todos esos dispositivos necesitar ser prendidos para funcionar y apagados para dejar de funcionar. También necesitas ser censados para saber el estado del dispositivo o también para saber la lectura de algunas variables, como de luminosidad, temperatura, humo, humedad. En el caso de la luz de cualquier ambiente, nuestro sistema debe ser capaz de censar la luminosidad, y ese dato ser enviado a la aplicación Smart Home.

Se pretende que el usuario programador haga configuraciones que no alteren la programación interna de nuestro prototipo (Framework Carioca). Para eso se pretende tener una interfaz donde será permitida la configuración del Framework con la aplicación Smart Home.

El usuario tiene la posibilidad de integrar su propia aplicación Smart Home con el servidor Carioca y desarrollar su programa particular en el nodo todo eso siguiendo los parámetros fijados por el framework y el protocolo de comunicación. La Figura 7 ilustra el servidor Carioca y el nodo cada uno compuesta por capas. El framework consta de dos partes: el servidor y la red de sensores inalámbricos.

3.1.1. Servidor Carioca

Tiene como función recibir los pedidos de la aplicación Smart Home del ambiente inteligente. Ellos se comunican usando el protocolo HTTP. A la vez, tiene un proxy el cual se encarga de transformar los pedidos de la aplicación en una cadena de bytes para ser procesados por un programa en JAVA y enviar esta cadena de bytes a la red de sensores inalámbricos. Del mismo modo, el servidor recibirá datos de la red de sensores, la procesará y envíará la respuesta a la aplicación Web.

3.1.2. Nodo Carioca

Los nodos tienen instalado el TERRA para la ejecución de aplicaciones dentro de los módulos. Para el intercambio de mensajes con el servidor Carioca, se utiliza un mote llamado Base Estación (Base Station) el cual es conectado a la estación servidor mediante un cable USB y cuya función es ayudar en el intercambio de mensajes con los nodos vía radio.

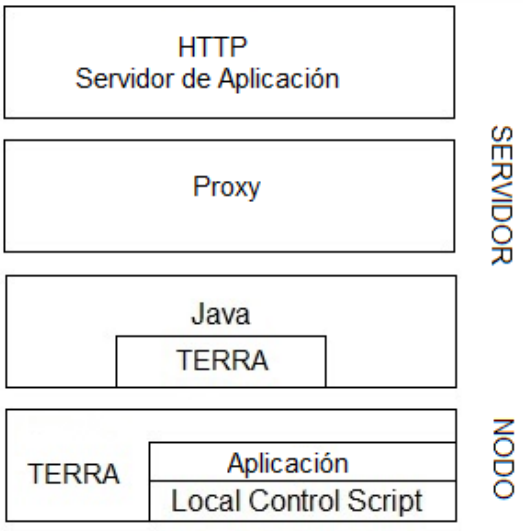

Figura 7 Arquitectura del Carioca

3.1.3.Características

Entre las características del Carioca esta que cada recurso tenga su propio URI. El formato del URI debe ser estandarizado de modo que se le facilite al usuario programador Web la construcción de aplicaciones Smart Home. Además, con el proxy se pretende mapear el URI del recurso para convertirlo en una cadena de bytes siguiendo un protocolo propio. De esta forma, se pretende disponibilizar los recursos en Web.

Respecto al protocolo usado en el Carioca, la red de nodos utiliza un protocolo propio del TERRA donde los datos son transportados en una cadena serial de bytes hasta llegar al servidor Carioca. El servidor Carioca se encargará de transformar esa cadena de bytes al protocolo HTTP para que los datos de los dispositivos (sensores y actuadores) estén disponibles en la Web en URI's estandarizados por el framework Carioca para que aplicaciones WEB externas al framework los utilicen en aplicaciones Smart Home.

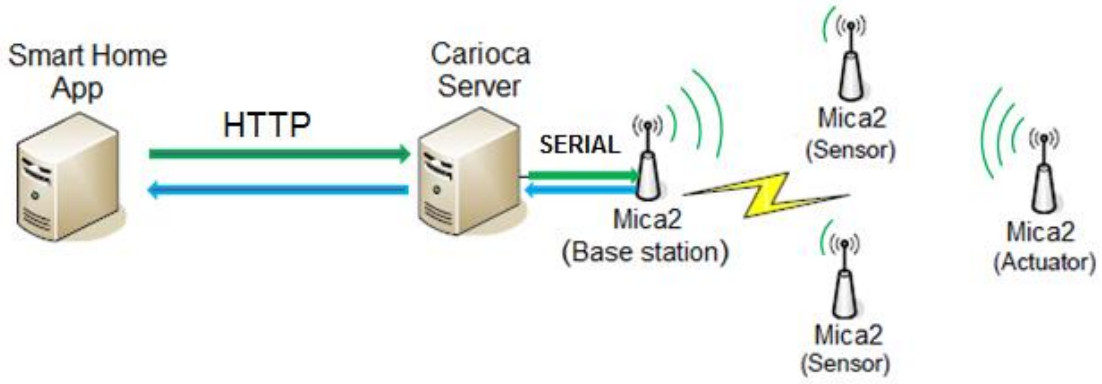

Figura 8 Visión general del framework Carioca

En la Figura 8 se ilustra una visión general de nuestra framework, una aplicación Smart Home que se comunica con el Servidor Carioca vía HTTP y este se comunica con la red de sensores inalámbrica compuesta por módulos Mica2.

3.2.Definición de la arquitectura del nodo

Un nodo de sensor inalámbrico debe contener el programa específico para ese nodo, hacer censar el ambiente donde está instalado y activar el actuador cuando sea el caso. Todos los nodos se comunican con la PC mediantes el uso de una "Base Estación". La comunicación Base Estación - nodo es vía radio frecuencia.

En la Figura 9 se muestra el diagrama de un nodo sensor/actuador. Cada nodo consta de un módulo Mica2 y una tarjeta que sirve de interfaz para conectar los sensores y actuares al Mica2.

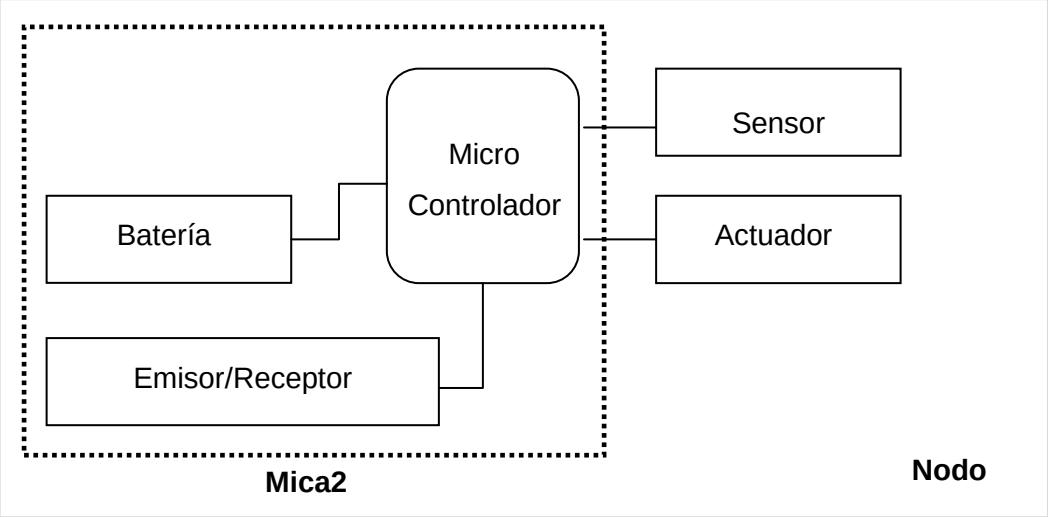

Figura 9 Diagrama del nodo

3.2.1. Visión en conjunto de los nodos

En la Figura 10 se muestra la típica red de sensores y sus componentes para el desarrollo del tema de tesis. En el lado izquierdo son los nodos de sensores, que generalmente consiste de un nodo mote y una placa de Sensor/Actuador para tareas de detección y control. El nodo tiene una CPU 8 bits, la memoria 8-256K, y un radio de baja velocidad y un sistema operativo. El sistema operativo está instalado en la memoria flash del Mica2. Una placa de Sensor/Actuador está unida al módulo del Mica2 por medio del periférico de extensión. La Base Estación es también un mote, que está conectado mediante un cable USB a un PC. El programa que se ejecuta en la Base Estación es para:

recibir mensajes del servidor y difundirlo a la red de sensores inalámbricos; y

recibir datos vía radio frecuencia de la red de nodos para el servidor.

El servidor Carioca con su aplicación sirve como un centro de análisis de datos y de control.

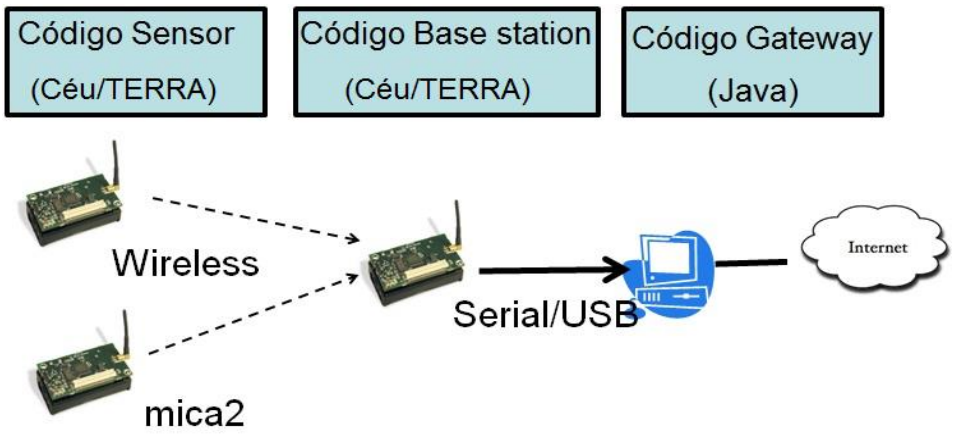

Figura 10 Visión general de los nodos

Como se ve en la ilustración, esa es la configuración para la red de sensores inalámbricos.

3.3.Aspectos del framework Carioca

En esta sección, se detalla los aspectos a tomar en cuenta antes de la implementación de framework, como la estandarización del URI, el orden de los byte para la comunicación con la red de nodos y la comunicación con la aplicación Smart Home.

3.3.1.Estandarización del URI

La estandarización del URI es una etapa crucial, pues con ella se comunicará el framework y las aplicaciones Smart Home. También, con el URI se definirá el recurso (actuador y sensor) y la acción que se pretende realizar.

El Framework es para ambientes físicos, por lo tanto vamos a considerarlo en la formación del URI.

La construcción del URI para nuestro framework tendrá los siguientes parámetros:

- **El Protocolo.** Utilizado para acessar al recurso ("http://");
- **El Dominio.** Es la dirección del computador servidor, Por ejemplo utilizaremos la dirección http://**www.midominio.br**;
- **Ambiente físico.** Es para que a través del URI el usuario entienda que es allí donde está el recurso. Con la definición del ambiente, es posible tener el mismo recurso en diferentes ambientes. http://www.midominio.br/**ambiente/**;

- **Recurso.** Identifica el nombre del actuador o sensor que se pretende identificar. Recurso es temperatura, humedad, motor, ventilador, interruptor, etc. Y el nombre del recurso estará en esa posición del URI http://www.midominio.br/ambiente/**recurso**/ ; y por ultimo
- **Acción.** La "acción" se ubica después del nombre del recurso. http://www.midominio.br/ambiente/recurso/accion. Una petición conteniendo esta estructura será ejecutada según la acción pedida. Dicha acción es para lectura de datos (sensor) o para interferir en el recurso (actuador).

De esta forma, se estandariza el URI para que las aplicaciones Web intercambien datos con el Framework:

http://www.midominio.br/ambiente/dispositivos/accion

El trecho "recurso" es utilizado para identificar variables físicas como temperatura, luminosidad, humedad. También son objetos que se quieren controlar o monitorizar como motor, interruptor, ventilador, etc. Dichos recursos será conectado en un puerto de la tarjeta Mica2.

La "acción", se implementará inicialmente para tres tipos de requerimientos de los recursos. En la tabla 2 se describen las acciones que tendrán el framework Carioca y algunos ejemplos de nombres que dependerán del recurso.

Tabla 2 Acciones que ofrece el Carioca para los recursos

Acción	**Nombre**
activa recurso	On, ligar, abrir, prender, open
Desactiva recurso	Off, desligar, cerrar, apagar, close
Leer estado o valor del recurso	Get, leer, estado, status

En principio esas tres acciones son las que se implementarán. En el capítulo 4 agrandaremos este URI estandarizado con otros modificadores de objeto como "flujo", "Interrupción" y "temporización".

El URI también debe ser configurable, es decir, el usuario programador, debe adjudicar cada nombre del "ambiente", "recurso" y "acción" con el identificador que desee y que tenga relación con el recurso.

Datos adquiridos del URI

El URI tiene información que el servidor Carioca necesita para formar una cadena de bytes que serán enviadas a la red de sensores inalámbricos. La Tabla 3 relaciona cada parte de la estructura del URI: el ambiente, el recurso y la acción, con dato especificando el nodo como su identificador, su periférico y un dato procesado por la aplicación TERRA.

Tabla 3 Interpretación de la estructura del URI

De: URI	Para: Mica2
www.midominio.br/**ambiente**/recurso/....	Id del Mica2
www.midominio.br/ambiente/**recurso/....**	Puerto del Mica2
www.midominio.br/ambiente/dispositivo/**accion**	Id de la acción

En un ambiente es posible tener instalado más de un nodo, por lo tanto cuando se configura un ambiente en el framework, se debe relacionar con el identificador (id) del nodo, así el framework tendrá una relación de nodos relacionados a un ambiente.

Un “recurso” es actuador o sensor y se conecta a un pin del nodo. En la Tabla 4 lo llamaremos puerto del nodo.

La “acción” está relacionada a un dato que será el identificador de la acción. Este dato es procesado por el software del nodo.

3.3.2.Formato de mensaje: Del framework para la WSN

Es una cadena de bytes de información que circula entre el framework y la red de nodos. El mensaje tiene un tamaño fijo y tiene un orden que se debe seguir para que el mensaje entre el servidor y la red de sensores fluya correctamente. Contiene informaciones como el id del nodo, pin, código de la acción, mensaje e id del mensaje.

De la Tabla 4, los campos del mensaje del Carioca se definen como sigue:

- **ID acción**: de 8 bits. Es un número que identifica la acción;
- **ID recurso:** de 8 bits. Es un número que identifica el recurso;
- **Dato:** de 16 bits. Es el espacio para enviar al servidor la lectura de recursos (ejemplo: el valor de la temperatura, o el estado de un actuador: 1 o 0);
- **ID nodo:** de 8 bits. Cada nodo tiene un numeró identificador, y es en esta posición que se especifica; y

- **ID usuario:** de 8 bits. Número que identifica quien está haciendo el pedido desde la Aplicación Smart Home.
- **ID extra:** de 8 bits. Es el que tiene el ID del modificador de objeto: el flujo, Interrupción en el nodo o temporización en el nodo.

Tabla 4 Formato de mensaje entre la red de sensores y la Estación Base

ID acción	ID recurso	Dato	ID nodo	ID de usuario	ID extra
8 bits	8 bits	16 bits	8 bits	8 bits	8 bits

En el framework Californium, los datos enviados a los nodos tienen la siguiente estructura:

Tabla 5 Formato de mensaje en Californium

IP del Nodo	Puerto	ID del recurso	Mensaje

Como se observa, los datos del nodo Californium tienen el IPv6 como identificador del nodo los demás datos son similares al Carioca.

Tabla 6 Descripción del trecho “acción”

Acción	Nombre (por defecto)	Para: ID acción
Lectura de recurso	get	2
Ligar	on	1
Desligar	off	0

En la Tabla 6 relacionamos un número con la acción, así la cadena de bytes enviadas desde el servidor Carioca a la red de sensores tiene un número que identifica la acción.

La acción es configurable, sin embargo ofrecemos el GET, ON y OFF como nombres por defecto y que pueden ser cambiados por el usuario.

3.3.3.Actualización de datos en aplicación Web

Tradicionalmente, una página Web tiene que enviar una solicitud al servidor para recibir nuevos datos, es decir, la actualización de datos en la aplicación Web es solo si este le solicita al servidor.

En nuestro caso, el framework Carioca maneja datos en tiempo real, y estos datos son requeridas por aplicaciones Smart home. La actualización de los

datos en la aplicación son importantes pues son variables de monitoreo continuo, por ese motivo se procuró una tecnología que actualice los datos en la aplicación Web desde el servido, así encontramos el Server Sent-Events.

3.3.3.1.Protocolo tradicional

El modelo clásico de aplicaciones Web funciona de esta forma: La mayoría de las acciones del usuario en la interfaz disparan un requerimiento HTTP al servidor web. El servidor efectúa un proceso (recopila información, procesa números, hablando con varios sistemas propietarios), y le devuelve a una página HTML al cliente.

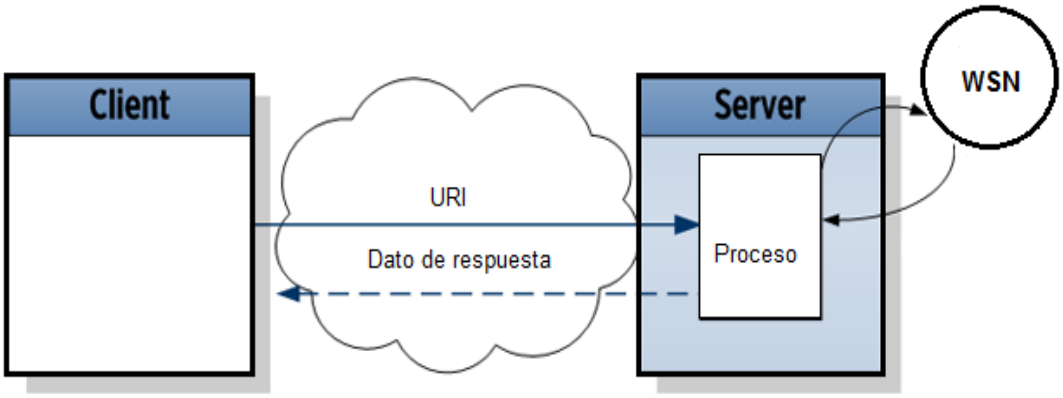

Figura 11 Comunicación tradicional

En la Figura 11 se ilustra esta comunicación. El usuario envía a través de navegador HTTP una solicitud al servido con un ID del recurso y el estado actual. El servidor procesa el URI recibido y solicita el dato a la red de nodos, cuando el servidor recibe el dato de la red de sensores, este lo procesa e lo envía al cliente. De lado del cliente, el dato se actualizar en la interface.

3.3.3.2.Server Sent-Events

El Server Sent-Events [36] es una tecnología para ofrecer notificaciones push de un servidor para un cliente del navegador en forma de eventos DOM [37]. El Server-Sent Events EventSource API ahora está siendo estandarizado como parte del HTML5 [38] por la W3C.

El Server-Sent Events (SSE) es un estándar que describe como los servidores inician la transmisión de datos para los clientes una vez que una conexión de cliente inicial fue establecida. Son comúnmente usados para enviar actualizaciones de mensajes o corrientes continuas de datos para un cliente del

navegador y proyectado para mejorar nativas, cross-browser de streaming a través de un API JavaScript llamado EventSource, a través del cual un cliente solicita una URL específica, a fin de recibir un flujo de eventos. El Server-Sent Events soporta los siguientes navegadores: Chrome, Firefox, Opera y Safari.

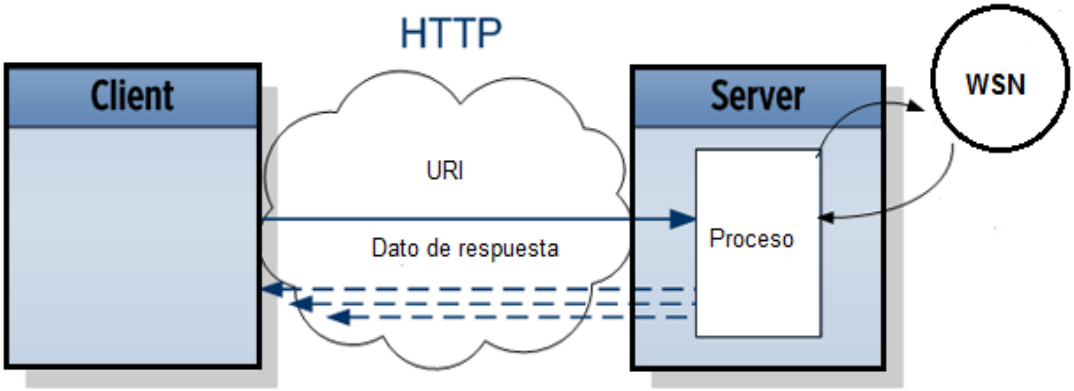

Figura 12 Cliente – servidor con Server-Sent Events

En una comunicación usando SSE (ver Figura 12), el servidor envía datos para su aplicativo, sin la necesidad de hacer un pedido inicial. En otras palabras, las actualizaciones son transmitidas del servidor para el cliente. SSE abre un canal unidireccional entre el servidor y el cliente.

3.4.Diferencias entre el Californium y Carioca

El framework Carioca y Californium tienen sus diferencias, sin embargo tienen el mismo objetivo: colocar objetos a la Web. En esta sección nos enfocaremos en las diferencias entre los dos. En la Figura 13 se observa los dos framework Carioca y Californium.

En la ilustración se observan el sistema que se ejecuta en los nodos también se muestra los tres bloques que constituyen el Carioca y en el caso del Californium un bloque de nombre Proxy, el bloque “Aplicación HTTP” que se comunica con el Californium y el “Cliente CoAP” que representa aplicaciones CoAP y el Copper. A continuación una descripción de cada bloque:

Bloque Nodo.

El Carioca y el Californium tienen una estructura para sus nodos. En el caso del Carioca, se tiene el TERRA que se encarga de proveer un ambiente de programación de alto nivel para desarrollas programas que se comunicará con la

Base Estación. El Californium tiene el Erbium que tiene la misma función que el TERRA.

Bloque Servidor.

El carioca tiene un bloque llamado “JAVA” y es allí donde se reciben y se envían datos para la red de nodos, ese bloque contiene librerías del TERRA. También se tiene el bloque “Proxy” quien es el encargado de transformas una cadena de datos seriales (provenientes de la red de nodos) al protocolo HTTP y viceversa. El Californium cuenta con un “Proxy” para transformar datos que están en protocolo CoAP (provenientes de la red de nodos) al protocolo HTTP, este proxy es importante porque hace de puente entre los nodos y una aplicación cliente de protocolo HTTP. En el Carioca tenemos por último el bloque “HTTP Servidor de Aplicación” que representa la interfaz Web que es la una opción de configuración del Carioca.

PLUGIN

Solo el Californium tiene un Plugin para el navegador Mozilla llamado COPPER. En la figura 13 está representado como “Cliente CoAP” y no requiere de un proxy para comunicarse con la red de nodos porque el protocolo del COPPER es el CoAP.

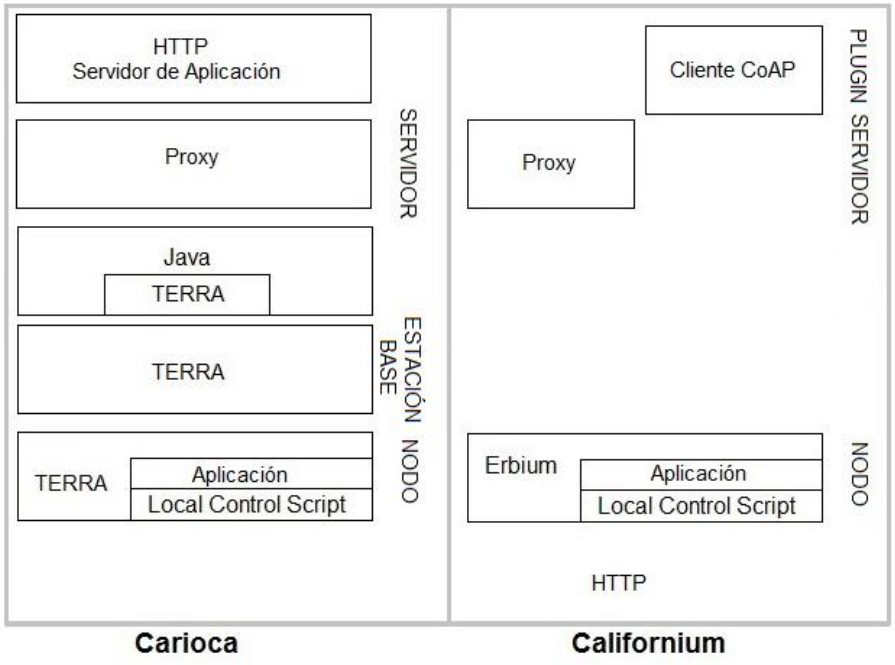

Figura 13 Comparativo entre los frameworks Carioca y el Californium

A seguir una lista de las diferencias entre un sistema y otro.

Software en los nodos

Mientras que en el Californium es el Erbium basado en el sistema operativo Contiki, en el Carioca tenemos el TERRA basado en el TinyOS. La función de ambos es de procesar una aplicación en el nodo.

En el caso del Erbium, tiene una aplicación que el autor espera ser usado sin modificar el código. En el Terra, es posible modificar el código siempre que se respete el orden de los bytes de comunicación entre el nodo y el servidor. Además, el TERRA, nos ofrece una programación a distancia sin necesidad de desinstalar los nodos del ambiente físico.

Protocolo

El Californium está basado en el protocolo CoAP (Constrained Application Protocol) tanto el servidor de aplicaciones como en los nodos.

En el caso del Carioca, se utiliza el protocolo HTTP entre una aplicación Smart Home y el servidor Carioca. Para la comunicación entre el servidor y la red de nodos se utiliza el modelo de parametrización de los datos en el nivel de transporte. Este protocolo que llamaremos de CcoAP (Constrained Constrained Application Protocol).

Hardware del nodo

Las tarjetas usadas en el Californium soportan dirección IPv6, utilizan el protocolo CoAP y tienen el sistema operativo Contiki. En el Carioca se utiliza las tarjetas Mica2, solo permite el sistema operativo TinyOS y no tienen dirección IP.

URI

Mientras que en el Californium cada nodo tiene su propio URI (por ejemplo coap://example.com:436/sensors/temp?min=10), en el Carioca con el servidor se le proporciona un URI (www.localhost:8080/lightswitch/on).

URI configurable

Es una ventaja del Carioca la opción de configurar los nombres de sus recursos y acciones. Porque por ejemplo para un foco, sus acciones serian: PRENDE, APEGA. Ya para un recurso como “puerta” tiene más sentido poner de nombre a sus acciones: ABRE y CIERRA. Esta configuración no ofrece el Californium.

Lógica de aplicación en el nodo

El Californium tiene un programa padrón para sus nodos. Ese programa no se modifica y su funcionamiento consiste en esperar una petición externa, lo ejecuta y devuelve el resultado, por lo tanto la aplicación lógica de los recursos debe realizarse en la aplicación cliente y no en los nodos.

El Carioca ofrece la posibilidad reprogramar los nodos y sobre todo, es posible desarrollar aplicación lógicas de los recursos dentro de los nodos (ver sección 5.4.4) y también en la aplicación cliente (ver sección 5.4.3).

Actualización de datos en la aplicación cliente.

El Carioca y el Californium ofrecen una comunicación de pregunta respuesta Cliente – Servidor. Donde el cliente hace una pedido al servidor, este ejecuta y devuelve.

El Carioca también ofrece la opción de actualizar los datos de la aplicación cliente usando Server Sent-Events. Para su uso, es solo agregar al final del URI del recurso, el nombre FLUJO.

Manipulación de recursos

En el Californium, se ejecuta las operaciones RESTful: GET, POST, PUT o DELETE para adquirir valor del recurso o para cambiar el estado de un actuador. Con el Carioca, estas operaciones están en el URI y hasta el momento tenemos definido tres operaciones: GET, ON y OFF. En el capítulo 4 se profundiza cada uno.

Proxy

En el Californium el proxy es usado para mapear las solicitudes HTTP a CoAP y transformar los recursos CoAP en formatos HTTP haciendo de puente entre los dos tipos de protocolos.

En el caso del Carioca, el proxy se encarga traducir el URI (por ejemplo http://www.puc-rio.br/lightswitch/ligar) a una cadena de bytes para ser procesados y encaminados al nodo correspondiente.

Ad Hoc Device Discovery

El Californium tiene esta característica. En nuestro caso de ambientes inteligentes no es una prioridad debido a que usamos cosas físicas que no se desplazan dentro del ambiente físico.

3.5.Discusión

La tecnológica y el protocolo usado en el Californium es diferente al Carioca, sin embargo esto no es un impedimento para que los dos framework cumplan la misma función.

El principal diferencial de nuestra propuesta en relación a los trabajos mencionados es que trabajamos con dispositivo más restricto y limitados como el Mica2. Eso desenvuelve una serie de limitaciones por ejemplo: nuestros nodos no funcionan como servidores Web, por lo tanto no se tiene acceso como cliente servidor y no usamos el protocolo IPv6. El protocolo CoAP es otra restricción que tiene el Mica2 pues este nodo no cuenta con memoria suficiente. Con el Carioca se pretende demostrar que con tarjetas de recursos limitados se logra el mismo objetivo. Las limitaciones de la tarjeta no limitan lo que el Carioca tiene para proporcionar.

El Carioca ofrece un framework adaptable a diferentes escenarios y tipos de uso. Ofrece la opción de comunicarse con aplicaciones clientes por medio de URI estandarizado y configurable, y además una programación a distancia de los nodos. También ofrece dos tipos de actualizaciones de datos en las aplicaciones cliente, y una interfaz Web para configurar el Framework, todo esto el Californium no proporciona.

3.6.Cuadro comparativo entre Californium y Carioca.

En el capítulo 2 se describió los diferentes framework y el Californium. En este capítulo se presentó el Carioca con más detalles así como las diferencias entre ellos. En esta sección se hace un resumen entre los dos, debido a que el Californium es tomado como modelo para nuestro prototipo de framework. El siguiente cuadro muestra las diferencias entre los dos Frameworks:

Tabla 7 Comparativo entre los frameworks Californium y Carioca

	Californium	**Carioca**
Basado en protocolo	CoAP	HTTP
Proxy	CoAP/HTTP	HTTP
Ad Hoc Device Discovery	Si	No
Tarjeta con dirección IP	IPv6	No (tarjeta limitada)
URI del recurso	coap://[aaaa::0212:7402:0002:0202]:5683/light	http://www.casa.br/laboratorio/light/get

Radio	IEEE 802.15.4/ZigBee	900 MHZ
Base Estación	No necesita	Si necesita.
Plataforma de los nodos	Erbium	TERRA
Protocolo nodo - servidor	CoAP	Serial
Dispositivos	TelosB	Mica2
Plugin	Copper (Cu) Mozilla Firefox	No necesita

La principal diferencia entre ambos proyectos es el protocolo de comunicación, el Californium está basado en el CoAP y los dispositivos usados tienen la capacidad de comunicarse en ese protocolo. En nuestro caso los dispositivos son más restrictos limitándonos a un protocolo propio serial. Nuestros dispositivos no tienen la capacidad de memoria suficiente para usar el protocolo CoAP.

A pesar de la limitación, nos proponemos a hacer un framework Carioca con la misma función del Californium: el de conectar cosas a la Web.

4
Prototipos

En este capítulo vamos a presentar nuestra prototipación descrita en el capítulo 3. En la sección 4.1 se presenta una visión general de la arquitectura de Framework Carioca. En 4.2 se describe las clases que componen el servidor Carioca. De los protocolos de comunicación entre nodos y framework. Enseguida, en la sección 4.3 abordaremos el flujo de datos del framework. En la sección 4.4 se presentará el protocolo de comunicación del Carioca con la Base estación y con la aplicación Cliente. En la sección 4.5 se describe el diseño de circuito para conectar cosas. En 4.6 se hace una descripción de los comandos usados en el URI. En 4.7 y 4.8 vamos a presentar la interfaz Web que el framework ofrece para que el usuario programador configure su aplicación con el Carioca y la interfaz que se generan después de configurar el Framework. Por ultimo en la sección 4.9 implementamos el programa para los nodos.

4.1.Visión Operacional general

j Nuestro framework Carioca está dividido en dos componentes importante: en el servidor Carioca y la red de sensores inalámbricos.

Nuestra implementación en el servidor Carioca consiste en desenvolver un proceso que interpreta el URI de la aplicación Web y transformarlo en una cadena serial de bytes. Luego, procesar esa cadena de bytes usando un proceso en JAVA e intercambiar mensaje con la red de sensores inalámbricos.

En la Figura 14 tenemos el diagrama con los módulos utilizados en el framework. En la ilustración, en el bloque Servidor, el “Mapeo URI” es un proceso que se encarga de recibir las URI de las aplicaciones Smart Home y las descompone para ser procesadas por el “Control de sistema”. El bloque “Interfaz Web” se refiere a una interfaz propia del framework donde el usuario también tiene acceder a los recursos sin una aplicación Web. El URI de este interfaz no es mapeada por el “Mapeo URI”. La información proveniente de estos dos bloques es procesada por el “Control de sistema” y es quien envía los datos a la red de sensores por medio de una Base Estación.

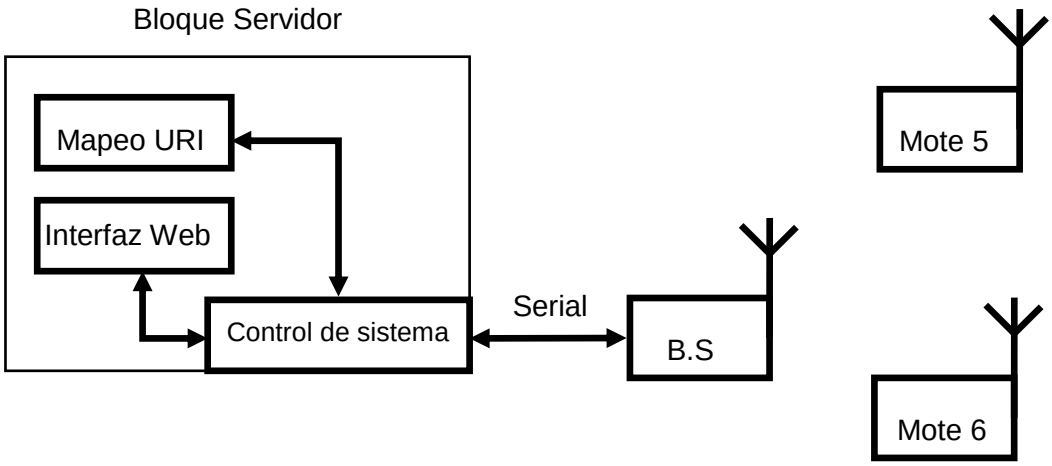

Figura 14 Diagrama de bloques del framework

En el Bloque Servidor, además de esos bloques que describimos, se tiene otros procesos. El framework internamente tiene clases y otros procesos en JSP y HTML que se implementan en esta etapa del documento. A continuación con más detalles se presenta un diagrama de clases genérico.

4.2.Diagrama de clases del Carioca

Se utilizó un diagrama de clases (ver figura 15) para representar los objetos del framework y tener una visión del paso de mensajes entre ellos y los procesos que forman parte del framework.

A continuación se presenta un listado de estos objetos del Framework y sus definiciones:

- Aplicación Cliente: representa la aplicación Smart home;
- MapeoURI: es una clase JAVA que captura los URI de la aplicación cliente;
- EnvioRecibo: es una clase JAVA que es el comunicador entre el servidor y la red de sensores;
- Sala: los ambientes ingresado en el sistema;
- Nodo: tiene el identificador de la placa;
- Recurso: son los sensores y actuadores que fueron configurado en el sistema;
- Acción: contiene todas las acciones que se realizan a los recursos;
- ProcessConfig: esta clase JAVA se encarga de guardar en un archivo texto los datos ingresados en la configuración el framework: sala, nodo, recursos y acciones;

- InterfaceWeb: esta representa la interfaz que el framework ofrece para el usuario, y no es la aplicación Smart Home; y
- TERRA: representa el software de la Base Estación. Esta se comunica con el EnvioRecibo.

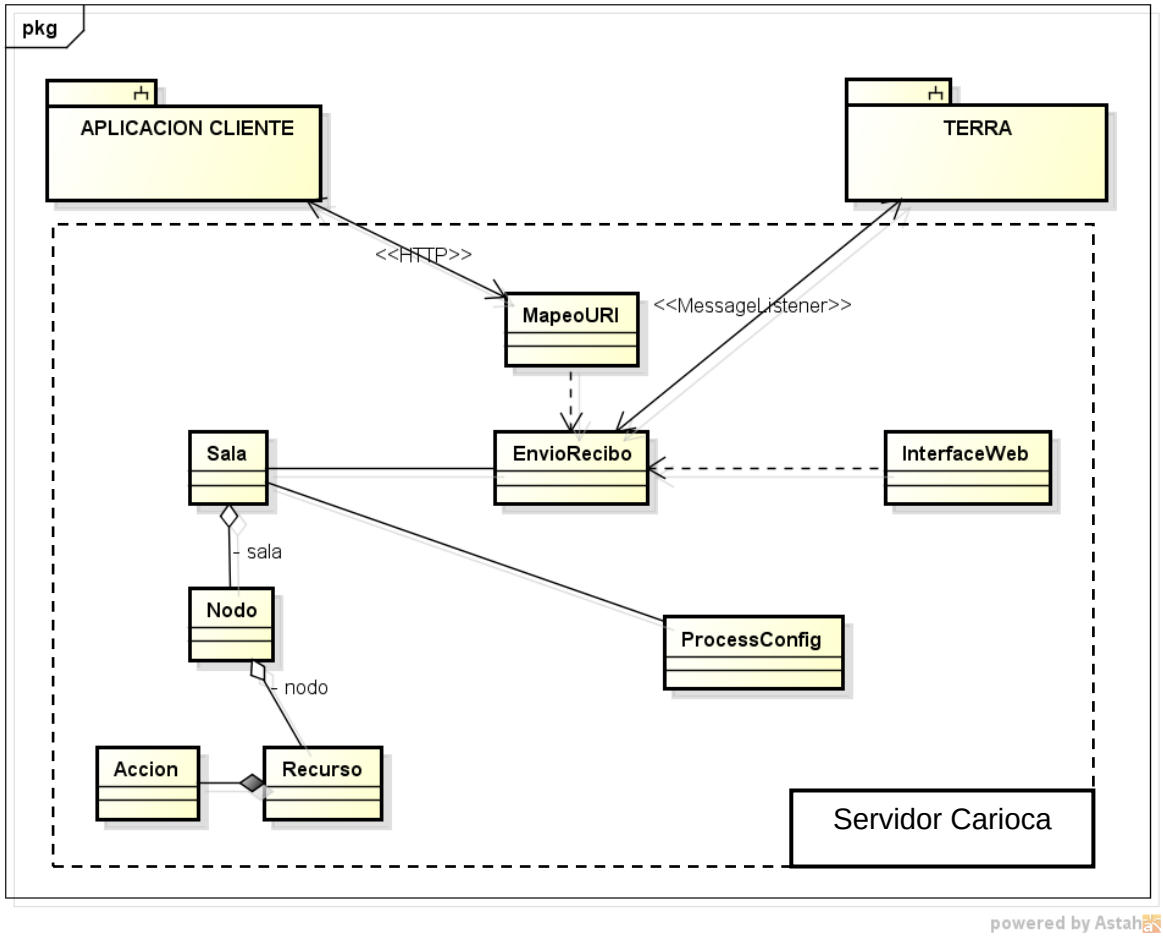

Figura 15 Diagrama de clases del framework Carioca

En la ilustración, la InterfaceWeb no es un objeto JAVA, sin embargo se agregó en el diagrama para mostrar que es parte de lo que ofrece el servidor Carioca. En el caso de “Aplicación cliente” y TERRA, no forma parte del servidor, sin embargo estos interactúan directamente con el servidor Carioca.

4.3.Flujo de datos del Framework

El flujo de datos se muestra en la Figura 16 y muestra los componentes que forman parte de este diagrama.

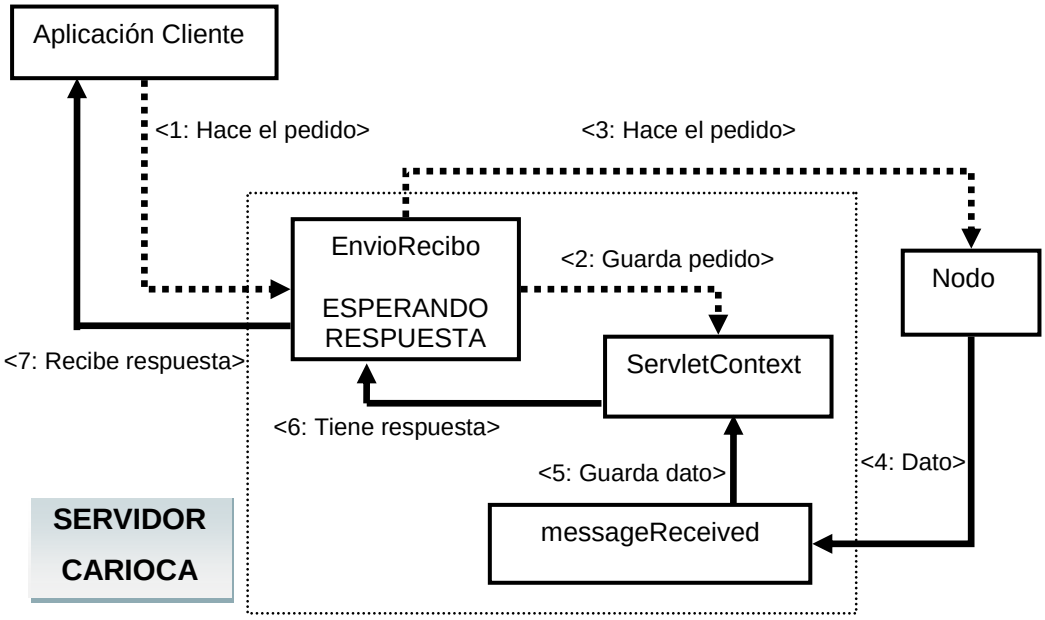

Figura 16 Diagrama de bloques del flujo de datos

La aplicación cliente hace un pedido de dato al servidor, este procesa el pedido hasta llegar a la clase “EnvioRecibo”, dentro de este programa se tiene el “ServletContext” en él se crea la variable con el identificador del nodo (idnodo), el identificador del recurso (idrec) y el estado que se encuentra el proceso, inicialmente será "procesando". Con eso, la clase “EnvioRecibo” periódicamente revisará si la variable "procesando" cambio a "terminado".

A la vez, guardado la variable en el “ServletContext”, el programa “EnvioRecibo” hará una solicitud de dato a la red de nodo. Cuando el nodo tenga la respuesta, esta será guardada en el “ServletContext” correspondiente a la variable ingresada inicialmente. Y allí cambiará el estado de "procesando" a "terminado". Cuando la aplicación “EnvioRecibo” lea que en “ServletContext” el estado de la solicitud es "terminado", la clase toma la respuesta del “ServletContext” y lo enviara a la aplicación cliente.

```
String variable = "resp." + idnodo + "." + idrec;
getServletContext().setAttribute(variable + ".estado", "procesando");
String estado = "procesando";
```

Figura 17 Código que atribuye el estado "procesando" en ServletContext

En la Figura 17 ejemplificamos como se guarda las variables y el estado "procesando" en el “ServletContext”.

4.4.Protocolos de comunicación

Para la implementación del framework Carioca, se estandarizará la comunicación entre cada capa del sistema. La comunicación entre nodos es vía wireless, y tiene un protocolo de comunicación definido en el TERRA. La comunicación entre el Base Estación y el servidor así como del servidor a la aplicación Web serán explicados a continuación.

4.4.1.Protocolo Base Estación – servidor

La Base Estación recibe datos de todos los nodos y usando un programa “EnvioRecibo” lee los datos y los pone a disposición del servidor Carioca. Para que se logre la comunicación del servidor Carioca con la red de nodos, este programa utiliza la librería del TinyOS y la librería del TERRA.

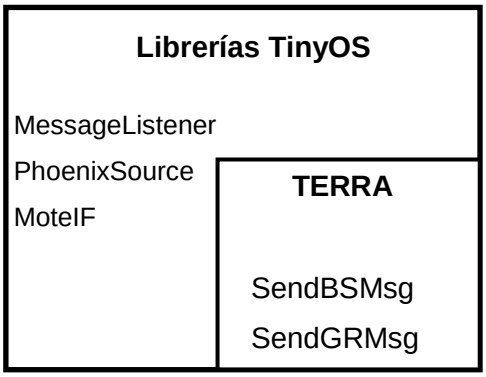

Figura 18 Librerías del TinyOS y del TERRA usadas en el Carioca

En la Figura 18, el SendGRMsg y SendBSMsg son clases personalizadas que nos proporciona el TERRA para la comunicación con el Framework. Sin ellos sería imposible logran una transferencia de datos entre el servidor Carioca y la red de nodos.

Ahora, dado que se tiene las librerías listas para entablar comunicación, describimos los datos que recibimos de la red de nodos a través de la Base estación. A continuación la Tabla 8 nos muestra una relación de datos que llegan a la Base Estación. De estos datos el “Data” (ver Tabla 4) proporciona a servidor datos de los recursos y es este dato que procesa y envía a la aplicación cliente.

Tabla 8 Datos envidos de los nodos para la Base Estación

Descripción	Representación
Indica el id del nodo que envía el dato	Sender
Indica el evento que se ejecutó en el nodo	EventID
Sirve para que evitar dobles lecturas	Seq
Cadena de Bytes que es enviado desde el nodo al servidor (ver Tabla 4)	Data

Estos datos son recibidos en el método “messageReceived”.

La Tabla 9 describe datos que son ingresados en el método “PruebaEnviar” y son envidados del servidor a la Base Estación. Estos datos son de configuración de protocolo del TERRA y con ellos envía la cadena de bytes de datos que el nodo procesará en su aplicación.

Tabla 9 Datos envidos del servidor los nodos

Descripción	Representación
Es el id del nodo	ReqMote
Es un id del mensaje cambia en cada envió de dato	ReqSeq
Normalmente de valor 1.	MaxHops
Número del evento que se quiere ejecutar en el nodo.	Evt.ID
Normalmente de valor 1.	Gr.ID
Normalmente de valor 1.	Gr.Par
Para enviar dato a un nodo en específico.	Target
De 16 bytes. Dato recibido y enviado del nodo a la Base Estación. (ver Tabla 4)	Data(max 16)

De esta información enviada a los nodos, el “Target” y “Data” fueron previamente adquiridos de la aplicación cliente.

4.5.Proceso de prototipación de sensor e actuador

Tener el control con el Mica2 de dispositivos de mayor potencia como luces, motores, puertas, y muchos más es una de las aplicaciones más interesantes y útiles. Pero es peligroso cuando los voltajes de línea de alimentación están siendo controlados. Existen diferencias significativas en el

control de alimentación de CA en comparación con DC. Esta sección cubrirá el circuito diseñado para disponibilizar dispositivos para que el Mica2.

4.5.1.Circuito del actuador

El nodo, tendrá conectado una tarjeta sensor/actuador. El siguiente circuito corresponde al actuador que será conectado al conector del Mica2. La idea de este circuito es, además de servir como IO del módulo Mica, separar la parte lógica con el circuito de fuerza, por ese motivo el propone el siguiente circuito electrónico.

El circuito tiene un optoacoplador el cual es un aislamiento eléctrico entre los circuitos de entrada y salida. Mediante el optoacoplador, el único contacto entre ambos circuitos es un haz de luz. Este aislamiento es útil en aplicaciones de alta tensión en las que los potenciales de los dos circuitos difieren en varios voltios. En nuestro circuito, el circuito de entrada es el conector I/O con el Mica2 y el circuito de salida contiene el relé y el conector del actuador. Para separar el circuito de entrada y de salida, cada una tiene su fuente independiente y no comparten tierra. El circuito de entrada es alimentado con 2.8vdc y el circuito de salida es energizada con 5vdc.

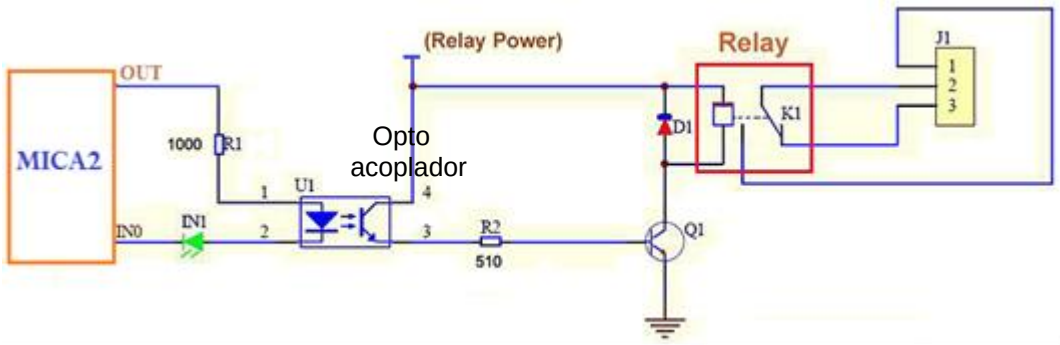

Figura 19 Circuito del actuador

En el conector J1 de la Figura 19, se conectará el dispositivo. Dependiendo del dispositivo conectado, el circuito tendrá variaciones.

4.6.Recursos del framework Carioca

En el sección 3.3.1, específicamente en la Tabla 2 se listó las acciones que el framework ofrece para los recursos: GET, ON, y OFF. Estas acciones fueron implementadas en esta sección, Además de estos comandos, el framework

implementó los modificadores de las acciones. En la Tabla 10 se enumera los tres modificadores.

Tabla 10 Lista de los modificadores de objetos

Modificador de objeto	Nombre
Indica que se utiliza el Server Sent-Events para la actualización de datos en la aplicación cliente.	flujo
Indica que cualquier cambio de estado del recurso produce una interrupción en la ejecución del programa del nodo.	interruptor
Indica que periódicamente se hace una lectura a un recurso y esta aplicación se hace en el nodo.	Int_temporizada

Estos modificadores se ubicarán en la parte final del URI, por ejemplo:

http:///localhost:8080/ambiente/dispositivo/get/flujo
http:///localhost:8080/ambiente/dispositivo/get/interrupcion
http:///localhost:8080/ambiente/dispositivo/get/int_temporizada

Se utilizan solo cuando se requiere una lectura del recurso. Para estos tres modificadores, no se implementó la opción que el usuario defina los nombres de cada uno, esos tres nombre son fijos y no cambiables. Con estos modificadores, pretendemos incrementar las formas de obtener la lectura del recurso por diferentes caminos. Así incremente el tipo de aplicaciones que serán desarrolladas con el carioca. Haciendo un comparativo con el Californium, este no ofrece estas opciones. Por lo tanto el Carioca proporciona opciones que otros no ofrecen.

4.7.Interfaz de configuración del Carioca

Implementado como el archivo “configuración.jsp”. Nos posibilita configurar cualquier tipo de aplicación cliente con el framework Carioca, de esta forma ambas aplicaciones se comunicaran de forma harmoniosa. Esta configuración debe ser realizada por el usuario programador framework. En la Figura 20 mostramos la interfaz implementada.

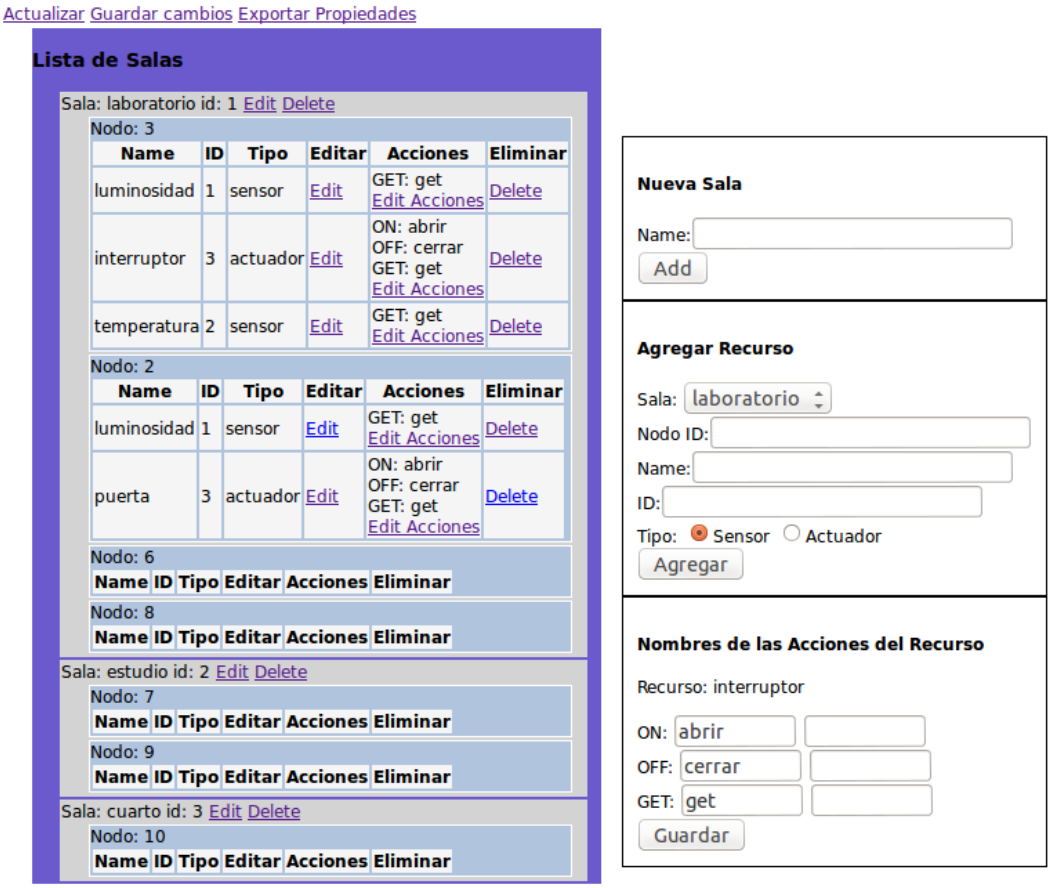

Figura 20 Interfaz de configuración

En la parte derecha de la figura se agrega los datos para la configuración. Estos datos requeridos son:

- Nombre de la sala: aquí se agregó de ejemplo laboratorio, estudio y cuarto. Después de ingresar estos datos se va la parte de “Agregar Recurso”.
- Agregar Recurso: aquí de ingresa el identificador del nodo (Nodo ID), el nombre del recurso, el identificador del recurso (ID) y se selecciona si es un recurso sensor o actuado. El siguiente paso es agregar acciones a los recursos.
- Nombres de las acciones del recurso: dependiente del tipo de recurso seleccionando (sensor o actuador) previamente, cuando se quiera editar los nombres de las acciones, se ofrecerá opción de agregar el nombre de la acción GET como de la Figura 20 si el recurso fue configurado como SENSOR o, caso el recurso haya sido configurado como ACTUADOR, se agregará los nombres de las acciones para GET; ON y OFF.

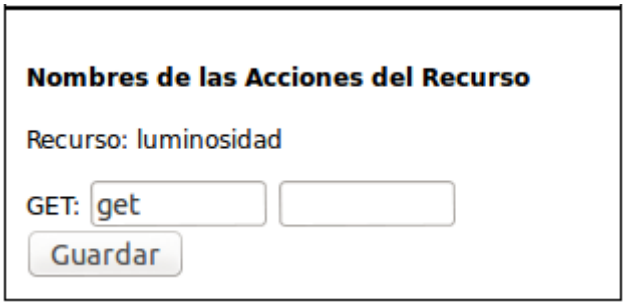

Figura 21 Configurar recurso del tipo sensor.

En el caso del recurso INTERRUPTOR (ver en la Figura 20 el recurso correspondiente a sala laboratorio) los nombres configurados para las acciones ON, OFF y GET son: abrir, cerrar y obtener, respectivamente.

El framework Carioca ofrecer agregar más de un nombre para la misma acción. Como se observa en la Figura 21, el usuario tiene la opción de agregar dos nombres en la acción GET.

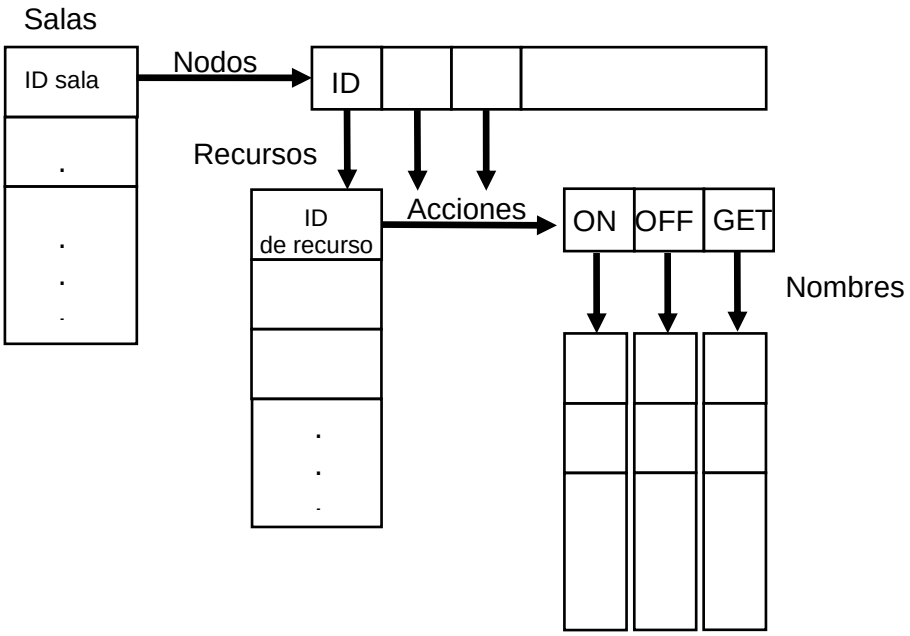

Figura 22 Estructura datos ingresados en el Framework

La figura 22 grafica la estructura de datos, en ella se aprecia una fila de salas, cada uno con un identificador, cada sala tiene su lista de nodos asociados con su identificador. Un nodo tiene una lista de recursos con su ID y por ultimo cada recurso tiene una lista de nombres que el usuario ingresa para identificar las tres acciones que ofrece el Framework.

Después de haber ingresado todos los datos requeridos en la interfaz “configuración”, el usuario debe guardar estos datos, para eso, en la parte

superior hay un Link de “guardar cambio”, se selecciona y los datos son guardado en un archivo llamado “datos.properties”.

4.7.1.Grabar datos de configuración

Para guardar los datos ingresados en la interfaz de configuración (ver sección anterior) se implementó un SERVLET de nombre “ProcessConfig”. En este programa, se crea un archivo de nombre “datos.properties”, y es escrito allí los datos ingresados: sala, nodos, recursos, tipo de recurso, nombre de las acciones. En este archivo texto también se guardan datos que el usuario no ingresa pero que el framework interpreta, como el número de salas que fueron registradas en el sistema, y la cantidad de nodos que se tiene registrados para una misma sala.

```
FileOutputStream fos = new
FileOutputStream(getServletContext().getRealPath("/WEB-
INF/datos.properties"));
System.out.println(getServletContext().getRealPath("/WEB-
INF/datos.properties"));
PrintWriter pw = new PrintWriter(fos);
pw.println("#lista de salas");
pw.println("salas = " + nroSalas);
….
```

Figura 23 Guarda en archivo los datos ingresados en “configuración”

```
# lista de salas
salas = 3
sala.1.name = laboratorio
sala.2.name = estudio
sala.3.name = cuarto


# lista de nodos de la primera sala
sala.1.nodos = 4
sala.1.nodo.1.name = nodo1
sala.1.nodo.1.id = 3
sala.1.nodo.1.recursos = 3

sala.1.nodo.1.rec.1.name = luminosidad
sala.1.nodo.1.rec.1.id = 1
sala.1.nodo.1.rec.1.tipo = sensor

sala.1.nodo.1.rec.2.name = interruptor
sala.1.nodo.1.rec.2.id = 3
sala.1.nodo.1.rec.2.tipo = actuador

sala.1.nodo.1.rec.3.name = temperatura
sala.1.nodo.1.rec.3.id = 2
sala.1.nodo.1.rec.3.tipo = sensor
#########################
```

Figura 24 Archivo texto datos.properties

La Figura 23 muestra un trecho de la implementación. En él se observa la creación del archivo, también tiene una línea de código que se encarga de imprimir la ubicación del archivo y a continuación, el código que escribe los datos en el archivo.

Finalizado la ejecución del programa “ProcessConfig” se tiene el archivo “datos.properties” como se observan en la Figura 24.

4.7.2.Exportar propiedades

En la interfaz de configuración (ver figura 20), en la parte superior, se tiene la opción “Exportar Propiedades”. Se implementó para que el usuario descargue el archivo “datos.properties”.

4.8.Interfaz interna del Carioca

Después que los datos fueron ingresados en la interfaz de “configuración”, el framework genera una interfaz Web interna como de la Figura 25.

Figura 25 Interfaz Web interna del carioca

En la ilustración, se observa en la primera página tiene la relación de salas, en la segunda se ve los recursos relacionados a la sala “laboratorio”. Por último en la tercera página de la ilustración se observa las tres acciones habilitadas para el recurso “interruptor” que es actuador.

4.9.Programación en TERRA

Para que la red de nodos interactúe con el servidor Carioca se implementa un programa en Céu. Este programa tiene como datos de entrada los mencionados en la Tabla 4. Esta aplicación para los nodos consiste en que cada vez que llega una cadena de bytes del servidor Carioca, este ejecuta y responde a servidor con una respuesta.

A continuación se presenta el diagrama de flujo del programa que se implementó para los nodos. En el programa, los recursos de temperatura y luminosidad son proporcionados por las tarjetas MDA100CA y MTS300CA que tienen sensores de estos recursos. Entonces usamos cualquiera de estas tarjetas como sensor de estos recursos.

El recurso actuador está refiriéndose a cualquier objeto que se conecta en los pines de I/O del nodo.

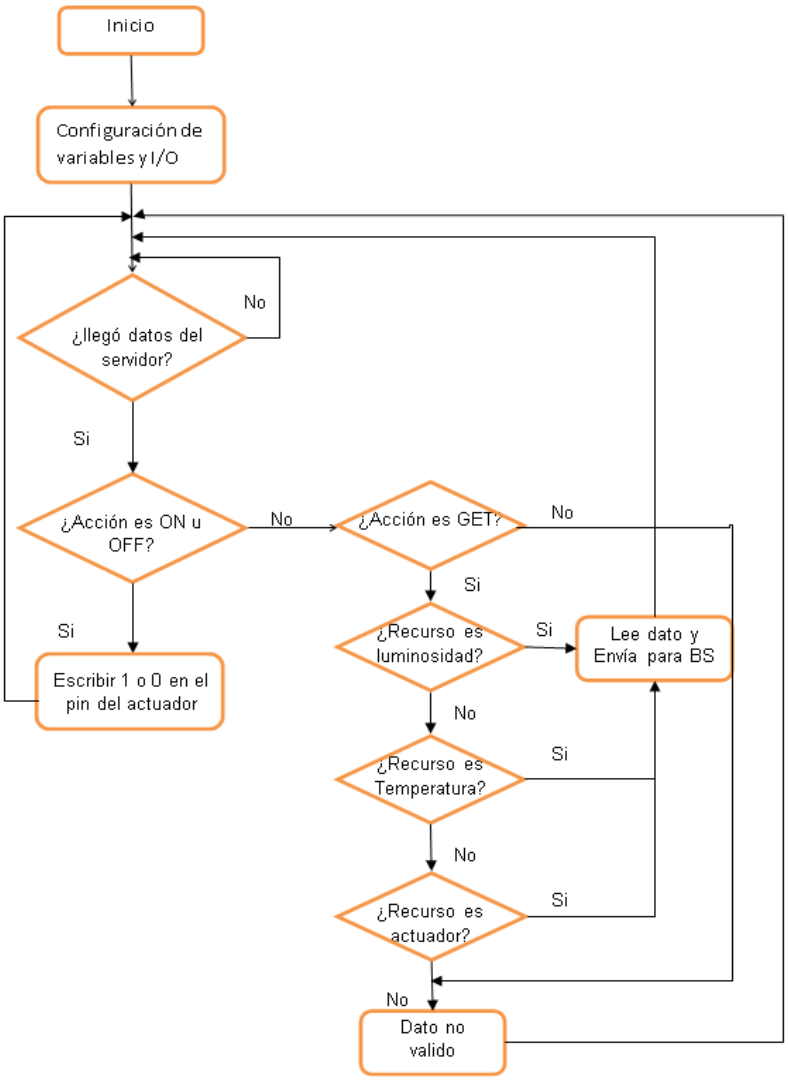

Figura 26 Diagrama de flujo del programa en TERRA

5
Pruebas y análisis de resultados

En esta sección, se realiza una evaluación de nuestro framework. Inicialmente, se describe los usuarios del sistema, luego las pruebas de evaluación detallando que se quiere probar con el sistema. También se hace una descripción de la instalación del sistema experimental para las pruebas. Después presentamos una serie de diferentes casos de pruebas que hemos ejecutado con el fin de demostrar la funcionalidad del Carioca. Nuestra evaluación fue dividida en dos partes. En la primera ejercitamos nuestro modelo de configuración de URI de nuestro framework. En La segunda parte, hacemos una evaluación de los diferentes recursos de programación ofrecidas por el Framework y los nodos, para el desarrollo de aplicaciones de usuario.

5.1.Usuarios del sistema.

El Framework abarca diferentes tecnologías, redes de sensores inalámbricos, lenguaje de programación web, lenguaje de programación Céu y uso de la interfaz Web. En todo ese contexto se define que nuestro sistema tiene tres tipos de usuario: un usuario programador Carioca, un usuario Web y por último un usuario final. A continuación se hará una breve descripción de cada uno.

El usuario programador Carioca

Es aquel que sabe programar los nodos, por lo tanto debe tener conocimientos del lenguaje Céu. Dicho usuario debe definir los URI de cada recurso, el nombre que será usado y los comandos de acciones con las que se identificará lo que se requiere del recurso. Opcionalmente debería saber instalar los nodos en el ambiente físico por lo tanto, debe tener conocimientos de electrónica.

A continuación una ilustración de lo que debe ser programado y configurado por el usuario programador Carioca.

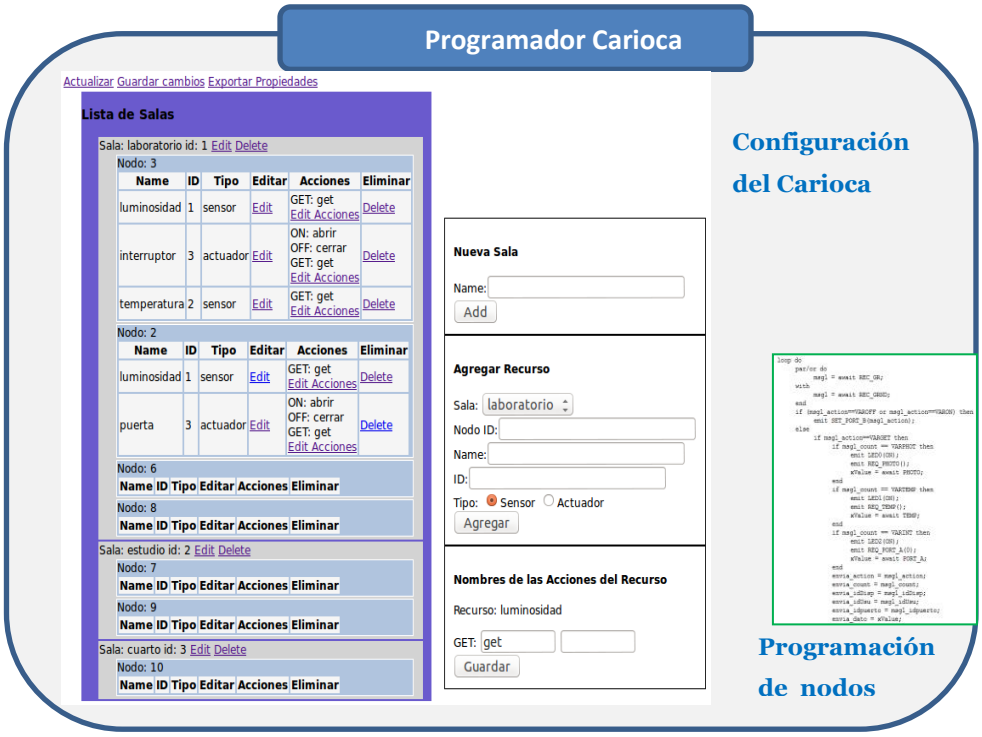

Figura 27 Programador Carioca

La figura 27 muestra las dos cosas que debe hacer el programador Carioca: la configuración en la interfaz del Carioca y la programación que se debe hacer para los nodos.

El usuario Programador Web

Es aquel que se encarga de desarrollar las aplicaciones Web para el usuario final. Este usuario utiliza los URI que el usuario programador Carioca definió y con ello desarrollará una aplicación para el monitoreo e intervención en el ambiente físico.

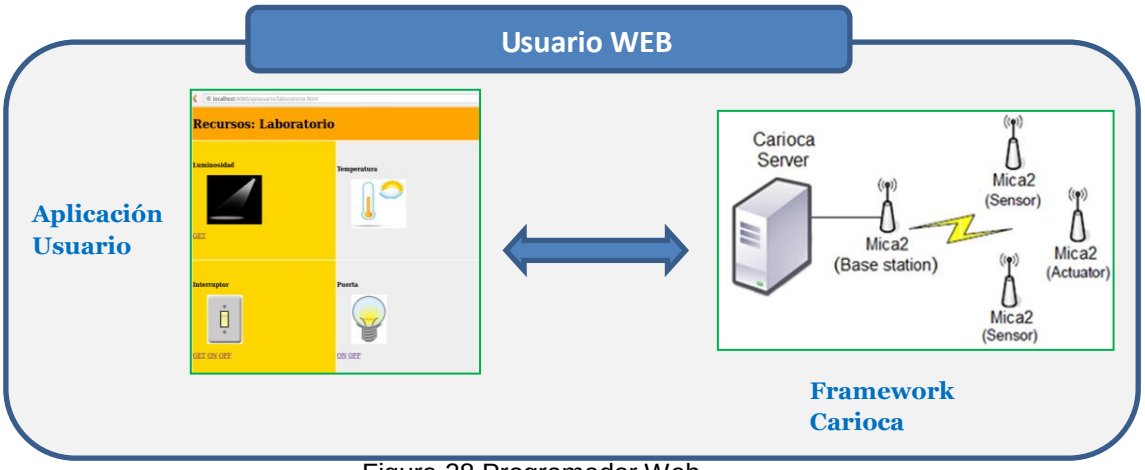

Figura 28 Programador Web

La figura 28 ilustra que la aplicación cliente se comunica con el framework Carioca por medio de URIs.

El usuario final

Es aquel que usa la interfaz que el programador Web desarrolló para él.

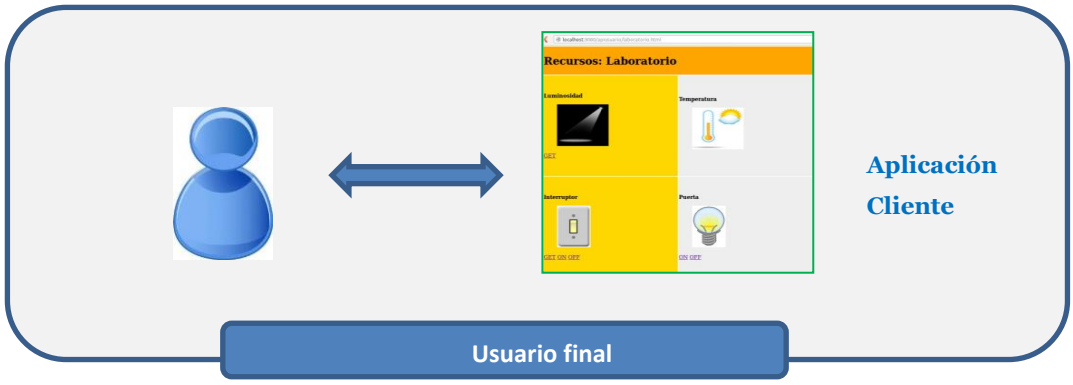

Figura 29 Ilustración del usuario y la aplicación cliente

En la Figura 29 se ilustra un usuario interactuando en una aplicación cliente.

Para la implementación de nuestras pruebas de evaluación, suponemos que inicialmente un ingeniero electrónico definió e instaló los nodos en el ambiente físico. Este especialista deja especificado en que pines del nodo conectó los sensores y actuadores. Con estos datos, el usuario programador Carioca configura el framework.

5.2.Pruebas de evaluación

Como se definió en la sección anterior, existen tres tipos de usuarios:

- Usuario final;
- Usuario programador Web; y
- Usuario programador Carioca.

Para cada usuario se requiere realizar testes especifico mencionados en la Figura 30, sin embargo para nuestro framework nos enfocaremos en el “programador Carioca” por lo tanto en este documento nuestros testes van dirigidos a este usuario que hace uso directo del framework a quien se le

ofrece diferentes tipos de usos y situaciones como se visó en la sección 2.4.

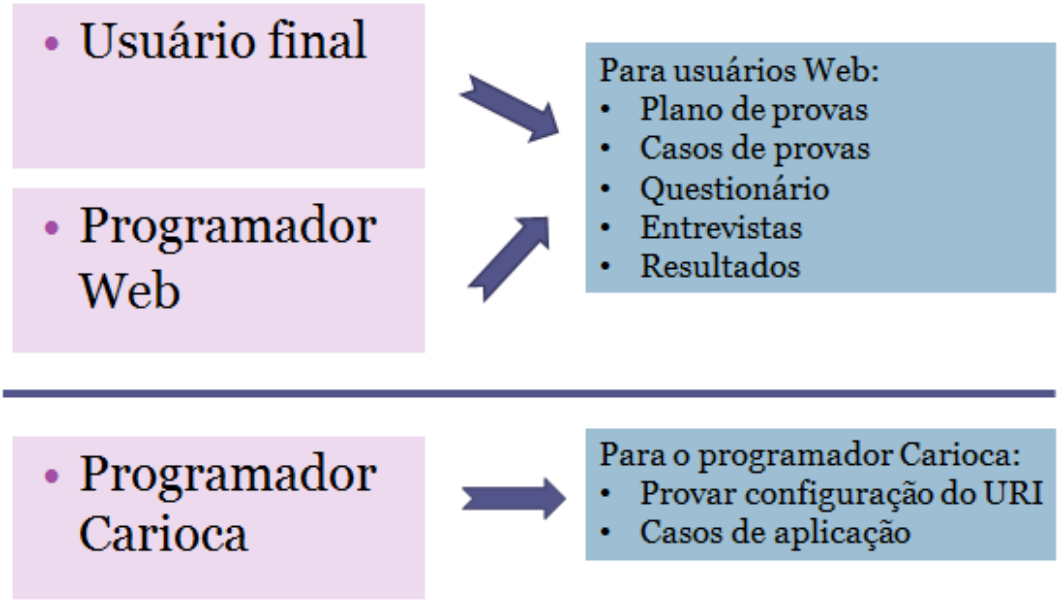

Figura 30 Usuarios y sus respectivos testes de evaluación

Nuestras pruebas de evaluación están dirigidos a situaciones de uso del Framework para demostrar que se adecua al ambiente físico y a los requerimientos del usuario. En este caso quien desarrolló el framework, ofrecerá cuatro tipos aplicaciones:

- Una aplicación que demuestra como configurar el Framework a través de su interfaz Web de nombre “Configuración”;
- Una aplicación cliente que reciba la actualización de datos usando el “Server Sent-Events” y la actualización tradicional;
- Una aplicación Web que utilice los datos proporcionados por el Framework para hacer sistemas inteligentes en el ambiente físico. Como sonar una alarma si la temperatura pasa del umbral permitido; y
- Hacer una aplicación lógica con los recursos pero programando en los nodos. De este modo la aplicación lógica no sucedería en la Web cliente y si en el nodo.

Con estas cuatro pruebas de uso, probaremos que nuestro Framework es versátil para diferentes tipos de escenarios.

5.3.Instalación del sistema

Nuestro experimento se basa usar con los nodos Mica2 para las pruebas del Framework. El Carioca es instalado en un ordenador portátil (Intel Core i3). A

su alrededor, en un radio de 30 cm, desplegó dos motas Mica2, dentro de ellas se ha instalado con el software TERRA. Un Mica2 fue conectado en un puerto USB de la computadora portátil para servir como una Base Estación, para recibir Paquetes vía Wireless desde/hacia el portátil a través del puerto USB.

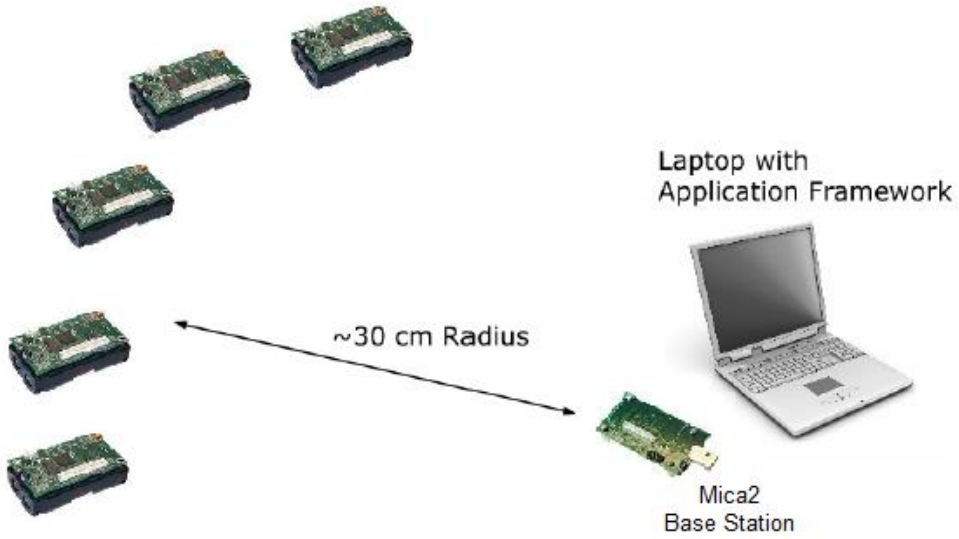

Figura 31 Instalación experimental para la evaluación del framework Carioca

Con los dispositivos instalados se implementan los casos de prueba.

5.4.Casos de prueba

Las aplicaciones que realizaron con el framework Californium en el documento [10] se basan en probar la herramienta con varios Skys Tmote utilizándolos como sensores, routers y termostatos. Las pruebas del Californium se basaron también en utilizar otras familias de tarjetas como el Poggs para realizar mediciones de electricidad. A la vez, utilizaron el Erbium (Er) para exportar los datos sensados por las tarjetas. Resumiendo, las casos de prueba proporcionadas por los autores del Californium consisten en que el framework es utilizado por dispositivos heterogéneos de diferente fabricantes y dichas tarjetas están dirigidas al área del Internet de las Cosas.

En este documento utilizamos el Mica2 para realizar las pruebas de nuestro Framework Carioca. Primero se presenta una aplicación con la que pretendemos mostrar la configuración que el programador Carioca (El usuario que programa directamente el Framework) debe realizar. Las demás aplicaciones presentadas en esta sección tienen como propósito probar la versatilidad del framework en adecua a diferentes tipos de usos en el área del

Smart Home. A continuación se explicaran cuatro tipos de aplicaciones del Framework Carioca.

5.4.1.Aplicación 1 – Configuración del framework Carioca

Se configura para el ambiente Laboratorio.

Tabla 11 Caso de uso: configuración del Framework

PRUEBA	configurar el framework y el URI para una aplicación cliente
OBJETIVOS	Este caso de prueba tiene como objetivo demostrar la configuración del Framework. Además lista los URI definidos para este ambiente.
ACTIVIDAD	Ingresamos los datos: Sala, Recurso, Tipo de recursos, ID nodo e ID del recurso. Por ultimo ingresamos nombres de las acciones. Ver Tabla 12 con las especificaciones.
RESULTADO	Ingresado todos estos datos (ver Figura 32), se genera las páginas de interfaz del Carioca (ver Figura 25). Con esa configuración también se crean las URIs que serán aceptadas por el Framework para monitorear e interactuar con los recursos desde una aplicación cliente. Las URI se listan en la Tabla 12.

Tabla 12 Datos ingresados en la interfaz de configuración.

Recursos	**ID recurso**	**ID nodo**	**Tipo**	**Acción**			**Tipo de dato**
				GET	**ON**	**OFF**	
Temperatura	2	3	Sensor	Status Lectura	__	__	Flujo
Luminosidad	1	3	Sensor	Status Valor	__	__	__
Interruptor	3	3	Actuador	Estado	Activar Ligar	Desactivar Desligar	__
Ventilador	3	2	Actuador	Estado	Prender	Apagar	Flujo

Tabla 13 Lista de URIs aceptadas por el Framework para los recursos

Recurso	Acción	URI
Temperatura	Leer valor	www.localhost:8080/laboratorio/temperatura/status/flujo www.localhost:8080/laboratorio/temperatura/lectura/flujo
Luminosidad	Leer valor	www.localhost:8080/laboratorio/luminosidad/status www.localhost:8080/laboratorio/luminosidad/lectura
Interruptor	Activa	www.localhost:8080/laboratorio/interruptor/activar www.localhost:8080/laboratorio/interruptor/ligar
	Desactiva	www.localhost:8080/laboratorio/interruptor/desactivar www.localhost:8080/laboratorio/interruptor/desligar
	Leer estado	www.localhost:8080/laboratorio/interruptor/estado
Ventilador	Prender	www.localhost:8080/laboratorio/ventilador/prender
	Apagar	www.localhost:8080/laboratorio/ventilador/apagar
	Leer estado	www.localhost:8080/laboratorio/ventilador/estado/flujo

En la tabla 13 tenemos el listado de todas los URI que una aplicación cliente de ambiente “Laboratorio” utilizará para comunicarse con el framework. La siguiente aplicación utilizará los URI de la tabla.

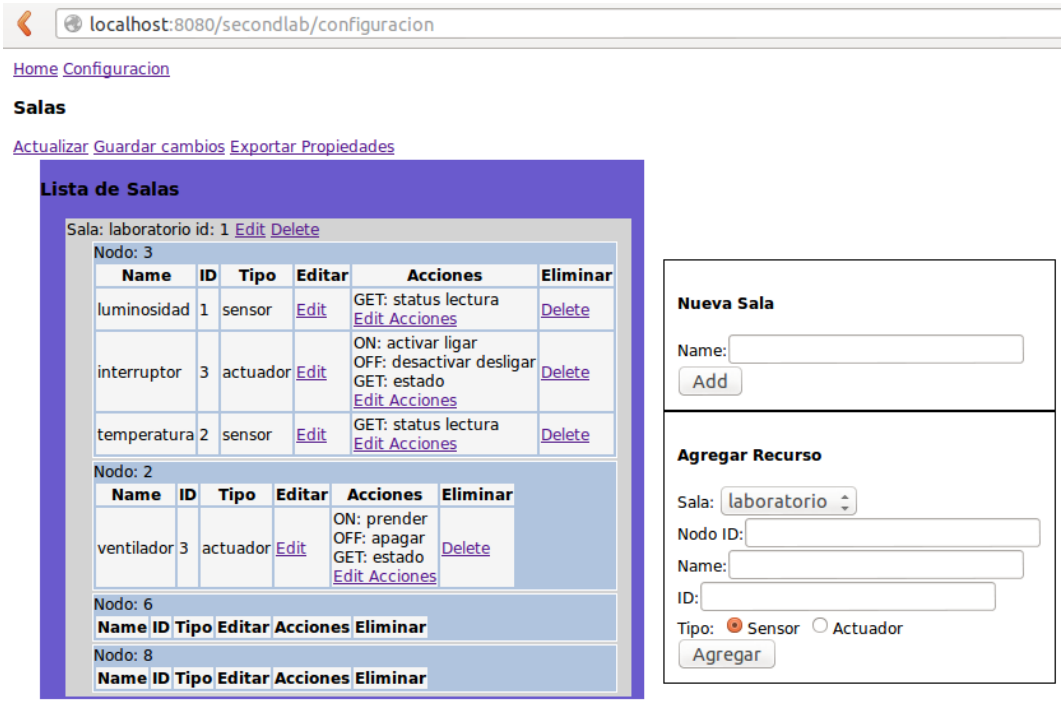

Figura 32 Datos ingresados y grabados en la interfaz “Configuración”

En la figura 32, se observa los datos ingresados en la interfaz Web de Carioca.

5.4.2.Aplicación 2 – Aplicación cliente

Esta aplicación solo es un espejo en tiempo real de los recursos. Desde la aplicación cliente tiene la opción de cambiar el estado de los recursos instalados en el ambiente físico y también muestra en tiempo real el estado de los recursos si fueron modificados directamente en el ambiente inteligente.

Figura 33 Aplicación cliente

Tabla 14 Aplicación cliente de monitoreo de recursos

PRUEBA	Hacer una aplicación cliente para monitorear los recursos
OBJETIVOS	Este caso de prueba tiene como objetivo mostrar los dos tipos de actualización de los datos en la aplicación cliente. Y para este caso, se propone una programación en el nodo de pedido y respuesta.
ACTIVIDAD	Hacemos una aplicación cliente que tiene cuatro recurso: temperatura, luminosidad, interruptor y ventilador. La temperatura y luminosidad son variables que serán solo de lectura de datos. El interruptor y el ventilador son dos recursos que desde la aplicación cliente se podrá modificar su estado o también, se hará una lectura del estado en tiempo real. Utilizamos la configuración hecha en la aplicación 1. Hacemos un programa para los nodos. Consiste en recibir un pedido del servidor, el nodo ejecuta y devuelve el resultado

	inmediatamente.
RESULTADO	En la Figura 33 se ve la aplicación cliente con los cuatro recursos. Además, en la Figura 26 muestra el programa que se ejecuta en los nodos.

En la Tabla 13 se observa para los recursos temperatura y ventilador, en la acción de leer estado de estos recursos, tiene al final de su URI la palabra “flujo”. Este comando indica que la actualización del dato es tratado como eventos periódicos enviados e incentivados por el servidor Carioca. Entonces para estos dos recursos usamos el Server Sent-Events.

En el caso del recurso TEMPERATURA, el dato se actualiza cada tres segundos. Y no requiere que el usuario de la interfaz cliente solicite la actualización del dato. Con el recurso VENTILADOR, en caso que alguien en el ambiente físico, cambien el estado del recurso, este será actualizado automáticamente. También ofrece la opción de cambiar el estado del recurso apretando uno de los link mostrado en la figura: el ON u OFF.

Para los recursos LUMINOSIDAD e INTERRUPTOR, el programador Web, habilito los links GET, ON y OFF para que el usuario, cambien el estado de los recursos o en el caso del GET, solicite el dato del recurso. Los otros recursos con URI sin "flujo", hace un pedido desde el cliente y espera hasta que el servidor envié el dato.

5.4.3.Aplicación 3 – Aplicación lógica del lado del cliente

Se hace una programación en el lado del cliente usando los datos de los recursos proporcionados por el Framework para hacer una aplicación inteligente.

Si la temperatura en el laboratorio supera los 21°C entonces se activa una alarma sonora.

Tabla 15 Lógica en la aplicación cliente

PRUEBA	Hacer una aplicación lógica en el lado del cliente
OBJETIVOS	Este caso de prueba tiene como objetivo usar los recursos diponibilizardos por el framework Carioca para hacer aplicaciones clientes con lógica.
ACTIVIDAD	De la anterior aplicación cliente, utilizamos la temperatura para que cada vez el dato llegue al cliente, este lo compare como

	21°C y caso sea superior a este valor, entonces se activará una alarma sonora. En la interfaz de configuración, se ingresa un recurso llamado alarma, considerado del tipo actuador. El URI de estos dispositivos está en la Tabla 16. El programa para los nodos es el mismo que el programa usado en la aplicación 2.
RESULTADO	En la Figura 31 se ve la aplicación cliente con los dos recursos. Además, respecto la aplicación 2, se agregó un código pequeño de comparación de temperatura.

La interfaz cliente con los dos recursos: temperatura y alarma se ver en la siguiente Figura 34:

Figura 34 La interfaz de la aplicación cliente

A continuación los URI que se usaron para esta aplicación.

Tabla 16 Recursos para la aplicación de control de temperatura

Recurso	Acción	URI
Temperatura	Leer valor	www.localhost:8080/laboratorio/temperatura/lectura/flujo
Alarma	Activa	www.localhost:8080/laboratorio/alarma/activar
	Desactiva	www.localhost:8080/laboratorio/alarma/desactivar
	Leer estado	www.localhost:8080/laboratorio/alarma/estado

La parte lógica que monitora si la temperatura es mayor que 21°C se hace desde la aplicación cliente, esta recibe el dato de temperatura cada tres segundos y lo compara con 21°C, si es menos que ese valor, no sucede nada. Cuando pasa de este valor, se hace un pedido al URI:

www.localhost:8080/laboratorio/alarma/activar

Se Activa una alarma sonora, indicando así que la temperatura pasó del umbral. En esta aplicación se utiliza el Server-Sent Events, el cual actualiza el dato de temperatura como un evento programado, por lo tanto el servidor pide cada tres segundos el dato al nodo, este ejecuta el pedido y entrega el dato al servidor, este a su vez envía el dato a la aplicación cliente. Resumiendo, cada tres segundos el servidor hace un pedido de dato de temperatura a la red de nodos. En esta aplicación se utiliza el programa que es el estándar programado para ser reconfigurado para otros abordajes (ver Figura 24).

Esta aplicación nos muestra que con el framework, es posible hacer diferentes aplicaciones lógicas, usando los datos de los recursos disponibilizados por el Framework.

Comparando la aplicación 2 y la 3, vemos que el Framework ofrece diferentes tipos de aplicaciones haciéndolo una herramienta versátil.

5.4.4.Aplicación 4 – Aplicación lógica del lado del nodo

Ahora proponemos la misma aplicación pero con otro camino de solución. Hacer una aplicación cliente para una clínica. Si la temperatura supera los 21°C entonces se activa una alarma sonora.

Tabla 17 Lógica de recursos en el nodo

PRUEBA	Hacer una aplicación lógica en el nodo
OBJETIVOS	Este caso de prueba tiene como objetivo demostrar que la aplicación lógica de los recursos también se hace dentro del nodo y no necesariamente en la aplicación cliente. Además, demostrar que las mismas aplicaciones sirven para diferentes escenarios. En este caso es para una clínica.
ACTIVIDAD	En la interfaz de configuración, se edita el nombre de sala: de LABORATORIO a CLÍNICA. Se ingresa un recurso llamado alarma, considerado del tipo

	actuador. El URI de estos dispositivos está en la Tabla 18. El programa para los nodos varía respeto el programa padrón que se usó para las otras aplicaciones. Ahora agregamos un código para que se ejecute una lectura periódica de 3 minutos al recurso Temperatura. Y cuando la temperatura pasa de 21°C envía al servidor este dato. Avisando así que paso del umbral.
RESULTADO	En la aplicación cliente se notifica que la temperatura pasó del umbral programado.

Tabla 18 URI del recurso usando para esta aplicación

Recurso	Acción	URI
Temperatura	Leer valor	www.localhost:8080/clinica/temperatura/lectura/ Int_temporizada

La Tabla 18 muestra el URI que la aplicación usuario utilizará. En esta se observa el comando “Int_temporizada” al final de URI. Este comando es procesado por el servidor Carioca e indica que el recurso “temperatura” será leído periódicamente dentro del nodo.

El umbral de temperatura es ingresado en la aplicación del nodo y el tiempo que periódicamente se lee el dato de la temperatura es ingresado también en la aplicación nodo.

La propuesta de aplicación es de modificar el programa del nodo usado en las anteriores aplicaciones y que desde allí periódicamente se obtenga la temperatura y se compare con el umbral que es 21°C. Si la temperatura pasa del umbral, entonces se activa la alarma y la red de nodos informa al servidor que la temperatura es mayor que la permitida y este a su vez notifica a la aplicación cliente. Recordemos que con el “Int_temporizada” la aplicación cliente y el servidor Carioca abren un hilo de comunicación usando el Server Sent-Events.

En resumen, toda la lógica de control de temperatura es realizada dentro del nodo. La aplicación cliente solo notifica al usuario final que la temperatura pasó del umbral y que se activó una alarma sonora.

Falto implementar una aplicación con el comando “interrupción” sin embargo con las pruebas realizadas se demuestra la versatilidad del framework Carioca.

6
Conclusiones y trabajos futuros

En esta sección se presentan las conclusiones de nuestra experiencia en el desarrollo de framework Carioca así como los trabajos futuros.

6.1.Conclusiones

En este trabajo, implementamos un framework para ambientes inteligentes y utiliza una red de sensores inalámbricos para que en ellas se conecten cosas cotidianas. Asimismo, dicha red de sensores utiliza como plataforma de hardware el Mica2.

El framework está basado en el paradigma de la Web de las Cosas. Con este concepto, los objetos se integran a la Web ofreciendo así sus recursos y datos para quien lo requiera.

El framework Carioca ofrece un URI estandarizado y configurable, dejando a criterio del usuario los nombres de los ambientes, recursos y acciones siempre siguiendo el orden establecido para su correcto mapeo en el lado del servidor. Además, tiene una interfaz Web, para que el usuario programador configure el framework y así, se comunique con la aplicación cliente.

Con el sistema presentado fue posible implementar tres aplicaciones con diferentes abordajes y mostramos que una misma aplicación es utilizada en escenarios diferentes. Para la implementación de estas aplicaciones basta que el usuario tenga conocimientos de programador Web. En una de las aplicaciones implementadas, proponemos que la aplicación lógica, que tradicionalmente se hace en la aplicación cliente, se realice en los nodos.

Identificamos también algunas limitaciones en nuestro trabajo:

- La evaluación de nuestra sistema fue hecho por un usuario, y este es alguien que sabe desarrollar aplicación Web, sabe programar los nodos, conoce de electrónica y sabe configurar el Framework; y
- No se logró implementar el circuito 4.5.1. por lo tanto se hicieron testes haciendo lectura y modificando el estado de los pines del nodo.

Las principales contribuciones de este trabajo son:

- Con dispositivos de recursos limitados como el Mica2 también se conectan cosas cotidianas a la Web;
- Se ofrece otro tipo de solución a las ya existentes; y
- Construcción de un framework adaptable para diferentes escenarios.

6.2.Trabajos futuros

Entre los trabajos futuros, es demostrar que es posible adaptarlo a otras áreas que no necesariamente sean ambientes físico. Por ejemplo usar nuestra implementación para monitorear adultos mayores o paciente clínicos, donde los signos vitales son monitoreados y en caso que algún signo vital pase del límite de lo normal, se active una alarma o se llame a la ambulancia.

Usando el mismo concepto para darle al framework aplicaciones en otras áreas, por ejemplo monitorear los signos vitales de pacientes clínicos:

www.clinica.com.br/cuidadosintensivos/pulsodejuanArce/sensar

Con este URI identificamos que el paciente se encuentra en la sala de cuidados intensivos, su nombre es Juan Arce y se está midiendo su pulso.

Es posible usar el framework otro tipo de aplicaciones por ejemplo para cuidar adultos mayores en ambientes inteligentes, por ejemplo:

www.localhost:8080/MariaRojas/ubicacion/sensar

En este caso, lo que vamos a monitorear no es un ambiente y si una persona, para saber lo que está haciendo en un ambiente: caminando, durmiendo, sentado en el sofá, agachado, etc. En el URI de ejemplo se solicita la ubicación de la persona en el ambiente físico.

Los ejemplos de monitoreo de personas clínicas y de adultos mayores, son citados para mostrar que con el Framework es posible adecuarse a otras necesidades, no necesariamente en monitorear ambientes físico. En este trabajo no se implementarán estos dos ejemplos, sin embargo consideramos importante mencionarlos para posibles trabajos futuros.

Son diferentes tipos de aplicaciones que son realizables con el Framework que necesariamente no es ambientes inteligente. Otras posibles aplicaciones son en la agricultura, monitorear sembrío o animales de granja.

Respecto al Framework y los recursos que ofrece, continuación se listan los posibles incrementos que podrían mejorar la funcionalidad del sistema:

- Hacer pruebas con otras tarjetas distintas a la Mica2;
- Ofrecer un framework multi-hilos;
- Incrementar comandos en el URI que permitan desenvolver aplicaciones lógicas en el lado del framework y nodo;
- Hacer una red social de dispositivos, es decir, que un nodo interactúe con otro autónomamente; y
- Como trabajos futuros se incluirá el “Ad Hoc Device Discovery” el cual sirve para descubrir dispositivos de formar autónoma.

Este trabajo utiliza la herramienta TERRA para la programación en los nodos. Del mismo modo, esperemos que en un futuro próximo utilicen el framework para futuras aplicaciones.

7
Referencias Bibliográficas

[1] O'REILLY, T. **What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software.** Design, v. 65, p. 17–37, 2007.

[2] O'REILLY, T.; BATTELLE, **J. Web Squared : Web 2 . 0 Five Years On.** Proc of the 6th Annual Web, v. 20, p. 1–15, 2009.

[3] MA, J. H. et al. **Research challenges and perspectives on Wisdom Web of Things (W2T).** The Journal of Supercomputing, 2010.

[4] DILLON, T. S. et al. **Web of Things as a Framework for Ubiquitous Intelligence and Computing.** Lecture Notes in Computer Science, UBIQUITOUS INTELLIGENCE AND COMPUTING, v. 5585, p. 2–13, 2009.

[5] ERICSSON. **More than 50 billion connected devicesEricsson white paper**,2011. Disponíble en: <http://www.ericsson.com/res/ docs/whitepapers /wp-50-billions.pdf> Acceso en: 20 de Febrero del 2013.

[6] THIBODEAU, P. **Diez tecnologías que estarán en alta en el 2013, según Gartner. Computerworld/EUA.** Disponible en: <http://cio.uol.com.br/ tecnologia/2012/10/26/dez-tecnologias-que-estarao-em-alta-em-2013-segundo-o-gartner/>. Acceso en: 17 de Febrero del 2013.

[7] HILL, J. L.; CULLER, D. E. **Mica: a wireless platform for deeply embedded networks.** IEEE Micro, v. 22, 2002.

[8] FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. University of California, Irvine, 2000.

[9] BRANCO, A.; RODRIGUEZ, N.; SANT'ANNA, F.; IERUSALIMSCHY, R. **Terra: Flexibility and safety in WSNs.** Manuscript in preparation. PUC-Rio. 2013.

[10] KOVATSCH, M.; MAYER, S.; OSTERMAIER, B. **Moving Application Logic from the Firmware to the Cloud: Towards the Thin Server Architecture for the Internet of Things.** Proceedings of the 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS 2012). Palermo, Italy, July 2012.

[11] KOVATSCH, M., DUQUENNOY, S., DUNKELS, A. **A Low-Power CoAP for Contiki**. Proceedings of the 8th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2011). Valencia, Spain, October 2011.

[12] SHELBY, Z.; STUREK, D.; FRANK, B. **Constrained Application Protocol (CoAP)**. Disponíble en: <http://tools.ietf.org/html/draft-ietf-core-coap-08>. Acceso en: 11 de Diciembre del 2012.

[13] SHELBY, Z.; BORMANN, C. **6LoWPAN : The Wireless Embedded Internet**. Disponível en: <www.6lowpan.net>. Acceso en: 15 de enero del 2013.

[14] **IPv6.** Disponible en: <http://pt.wikipedia.org/wiki/IPv6>. Acceso en: 15 de enero del 2013.

[15] WILDE, E. **Putting Things to REST**, UCB iSchool Report 2007-015, School of Information, UC Berkeley, 2007.

[16] CULLER, D.; ESTRIN, D.; SRIVASTAVA, M. **Overview of Sensor Networks.** Computer, v. 37, p. 41–49, 2004.

[17] CROSSBOW. **Mica2** datasheet. Product folder, 2004.

[18] CROSSBOW. **MPR/MIB Manual de user's: Wireless Sensor Networks**. Disponible en: <www.db.ics.uci.edu/pages/research/quasar /MPRMIB %20Series%20User%20Manual%207430-0021-06_A.pdf>. Acceso en: 12 de enero del 2013.

[19] BALANI, R.; HAN, R.; RENGASWAMY, K.; TSIGKOGIANNIS, I.; SRIVASTAVA, M. **Multi-level Software Reconfiguration for sensor networks.** In Proceedings of the 6th ACM & IEEE International Conference on Embedded Software, EMSOFT '06, pages 112 - 121, New York, USA, 2006.

[20] BRANCO, A.; RODRIGUEZ, N. **Determining the boundary cost and flexibility in wireless sensor networks.** Monografia em Ciência da Computação. PUC-Rio. 2012.

[21] Ceú: **The Programming Language**. Disponible en: http://www.ceu-lang.org/. Acceso en: 1 de mayo del 2013.

[22] FIELDING, R.; GETTYS, L.; MOGUL, J.; FRYSTYK, H.; MASINTER, L.; LEACH, P.; T. BERNERS-LEE. **RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1**. Society, p. 1–114, 1999.

[23] MAYER, S. **Deployment and Mashup Creation Support for Smart Things**, Institute for Pervasive Computing Department of Computer Science ETH Zurich, 2010. Disponible en <http://www.vladtrifa.com /files/publications/Mayer10.pdf>. Acceso en: 02 de mayo de 2013.

[24] LIDAN HU. **An Intelligent Presentation System**. Master of Science Thesis Stockholm, Sweden, 2008.

[25] OSTERMAIER, B.; SCHLUP, F.; ROMER, K. **WebPlug: A framework for the Web of Things.** Proceedings of the First IEEE International Workshop on the Web of Things (WOT2010). Mannheim, Germany, March 2010.

[26] TRIFA, V.; KAMILARIS, A.; PITSILLIDES, A. **The Smart Home meets the Web of Things.** International Journal of Ad Hoc and Ubiquitous Computing, 2011.

[27] RICHARDSON, L.; RUBY, S. **RESTful Web Services**. O'Reilly, 2007.

[28] KAMILARIS, A.; TRIFA, V.; PITSILLIDES, A. **HomeWeb: An application framework for Web-based smart homes.** 18th International Conference on Telecommunications, p. 134–139, 2011.

[29] PAULI, D.; OBERSTEG, D. **Californium: A CoAP Framework in Java.** The Institute for Pervasive Computing, Department of Computer Science, ETH Zurich, 2010.

[30] **UDP (User Datagram Protocol).** Disponible en: <http://pt.wikipedia.org /wiki/User_Datagram_Protocol>. Acceso en: 16 de Diciembre del 2012.

[31] **Californium: A CoAP Framework in Java (L).** Disponible en: <http://www.vs.inf.ethz.ch/edu/abstract.html?file=/home/webvs/www/htdocs/edu/theses/mkovatsc-californium>. Acceso en: 11 de Octubre del 2012.

[32] **Contiki.** Disponible en: <http://www.contiki-os.org/#about>. Acceso en: 18 de Diciembre del 2012.

[33] **Erbium (Er): REST Engine and CoAP Implementation for Contiki.** Disponible en: <http://people.inf.ethz.ch/mkovatsc/erbium.php>. Acceso en: 5 de Diciembre del 2012.

[34] KOVATSCH, M. **A User-Centered Application Layer for the Internet of Things.** Institute for Pervasive Computing, ETH Zurich, Switzerland, 2011.

[35] KOVATSCH, M. **Demo Abstract: Human–CoAP Interaction with Copper**. Proceedings of the 7th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS 2011). Barcelona, Spain, June 2011.

[36] **Server-sent events**. Disponible en: <https://developer.mozilla.org/en-US/docs/Server-sent_events > Acceso en: 3 de Abril del 2013.

[37] **Eventos DOM**. Disponible en: <http://en.wikipedia.org/wiki/DOM_ events>. Acceso en: 3 de Abril del 2013.

[38] **HTML5**. Disponible en: <http://www.w3.org/html/wg/drafts/html/master/> Acceso en: 3 de Abril del 2013.