PROTOTIPO DE SOFTWARE ORIENTADO AL SERVICIO DE MANEJO DE DATOS Y ADMINISTRACIÓN DE CAJAS NEGRAS DESTINADAS A TELEMETRÍA Y TELECONTROL.

PAULA FERNANDA RUEDA ROMÁN

CARLOS ALBERTO CONTRERAS HERNÁNDEZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS
BUCARAMANGA
2014

PROTOTIPO DE SOFTWARE ORIENTADO AL SERVICIO DE MANEJO DE DATOS Y ADMINISTRACIÓN DE CAJAS NEGRAS DESTINADAS A TELEMETRÍA Y TELECONTROL.

PAULA FERNANDA RUEDA ROMÁN
CARLOS ALBERTO CONTRERAS HERNÁNDEZ

Trabajo de grado para optar al título de Ingeniero de Sistemas

Director
Msc. Fernando Antonio Rojas Morales
Magíster en Informática

Codirector
Jaime Antonio Rueda Rivera
Ph.D en Ingeniería de Telecomunicaciones

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS
BUCARAMANGA
2014

DEDICATORIA

A Dios que me ha dado la vida, una muy feliz llena de infinitas bendiciones,
A los mejores: mis padres! los ángeles que Dios me regaló y me han apoyado siempre,

A mis hermanos y mi familia entera, que me llenan a diario de su cariño,

A ti, Yeferson Toloza, que me has acompañado tanto y sigues llenando de alegría mis días,

A Carlitos, que me contagió con su amistad y conocimiento para lograr esta meta,

A mis amigos, cómplices infaltables en todo momento.

Paula Fernanda Rueda Román

Dedico este trabajo a mi abuela Rosita, quien me inculcó desde pequeño el ser un hombre de bien, me llenó de su amor incondicional y que por cosas de la vida no me verá cumplir este sueño ya que ahora está en la paz señor y a mi madre quien es la persona que me dio la vida, que más me ha cuidado a lo largo de los años y que más me ha apoyado para cumplir mis metas. Las amo.

Carlos Alberto Contreras Hernández

AGRADECIMIENTOS

A la Universidad Industrial de Santander, por estos años de formación, aprendizaje y experiencias vividas.

Al profesor Fernando Antonio Rojas Morales, por su apoyo para la realización de este proyecto.

Al Ingeniero Jaime Antonio Rueda Rivera, por su orientación y experiencia que permitieron dar inicio a la idea que hoy es una realidad mediante este prototipo.

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

GLOSARIO

- **Firmware:** Bloque de instrucciones de máquina para propósitos específicos que establece la lógica de nivel más bajo que controla los circuitos electrónicos de un dispositivo de cualquier tipo.
- **Caja negra:** cualquier elemento que es estudiado desde el punto de vista de las entradas que recibe y las salidas o respuestas que produce, sin tener en cuenta su funcionamiento interno.
- **Telemetría:** Sistema de medida de magnitudes físicas que permite transmitir esta a un observador lejano.
- **Telecontrol:** Envío de indicaciones a distancia mediante un enlace de transmisión. Ejemplo, a través de cables, radio, dirección IP, etc.
- **Middleware:** Software de Intermediación para las comunicaciones y de apoyo a los procesos relacionados con la gestión y administración, así como las funciones específicas del sistema.
- **Rail Road Diagram:** Diagrama utilizado para representar gramáticas libres de contexto.
- **Análogo:** Magnitudes que varían con el tiempo en forma continua. Ejemplo: distancia, temperatura.
- **Digital:** Magnitudes discretas que no reflejan la variación continua a través del tiempo, solo percibe picos (altos y bajos), presencia o ausencia.
- **Framework:** Marco de trabajo que consta de una estructura en conceptual y tecnológica de soporte definido con módulos de software que sirven para la organización y desarrollo de software.

RESUMEN

TÍTULO: PROTOTIPO DE SOFTWARE ORIENTADO AL SERVICIO DE MANEJO DE DATOS Y ADMINISTRACIÓN DE CAJAS NEGRAS DESTINADAS A TELEMETRÍA Y TELECONTROL.*

AUTORES: PAULA FERNANDA RUEDA ROMÁN **
CARLOS ALBERTO CONTRERAS HERNÁNDEZ**

PALABRAS CLAVES: Telemetría, Protocolo, Servidor, RTU, FDD, TCP/IP, Ruby, Rails.

DESCRIPCIÓN:

Este proyecto surge de la necesidad de una herramienta de fácil configuración e implementación para la administración de los datos provenientes de dispositivos microcontroladores ubicados remotamente, destinados a telemetría y telecontrol, conectados a internet a modo de unidades de recolección de datos (RTU), se propone entonces en primer lugar el diseño de un protocolo de libre distribución llamado 'Remote Telemetry Unit Protocol - RTUP' que funcione como soporte a la transmisión de los datos captados por estos dispositivos emisores hacia y en segundo lugar un receptor tipo servidor denominado RTUServer encargado de almacenar procesar y administrar la interacción de los usuarios con los emisores.

El prototipo cuenta además con un componente de visualización que permite a los usuarios tener constante administración de sus dispositivos mediante la observación de su estado, últimas interacciones y envío de comandos con ejecución inmediata.

Las herramientas utilizadas para el desarrollo del prototipo son Ruby como lenguaje de programación, SQLite3 como manejador de base de datos, EventMachine para los eventos relacionados con la red y Rails como marco de trabajo incluyendo su librería Active Records para el mapeo relacional de objetos.

El proceso de validación del prototipo se realizó mediante pruebas funcionales de la capa de comunicación de red y pruebas de lógica y persistencia con el fin de comprobar el funcionamiento general del prototipo, estas estuvieron compuestas por pruebas de fase de autenticación y comandos iniciales y pruebas para comandos de atención.

---

*Trabajo de Grado: Modalidad de investigación.

**Facultad Ingenierías Físico-Mecánicas. Escuela Ingeniería de Sistemas e informática. Director Fernando Antonio Rojas Morales.

ABSTRACT

TITLE: SOFTWARE PROTOTYPE ORIENTED TO DATA HANDLING SERVICE AND MANAGEMENT OF BLACK BOXES FOR TELEMETRY AND REMOTE CONTROL PURPOSES.*

AUTHORS: PAULA FERNANDA RUEDA ROMÁN **
CARLOS ALBERTO CONTRERAS HERNÁNDEZ**

KEY WORDS: Telemetry, Protocol, Server, RTU, FDD, TCP/IP, Ruby, Rails.

DESCRIPTION:

This Project comes from the need of an easy configuration and implementation tool for data handling coming from microcontrolers devices with a remote location, telemetry and remote control purposes, those plugged to the internet in as remote telemetry units (RTU). First It is proposed the design of an open source protocol which works as a support to the transmission of data captured by those issuing devices to a receiver named as RTUServer in charge of store, process and manage the interaction between users and devices.

Furthermore the prototype has got a visualization component which allows to the users have continuous management of their devices through the status observation, last interactions and command sending with immediate execution.

It has been used some tools to develop the prototype, those are Ruby as a programming language, SQLite3 as a relational database management system, EventMachine library used for network events and Rails as a framework including its library Active Records used for object relational mapping.

The validation process of this prototype was made by functional testing of network layer and logic and persistence testing in order to prove the general operation, those tests were composed by authentication phase and initial commands tests and solicitude commands tests.

_____

*Thesis: Investigation mode.

**Physical Mechanical College. Systems Engineering and Computer School. Director Fernando Antonio Rojas Morales.

INTRODUCCIÓN

La tendencia impuesta durante los últimos años en la industria colombiana hacia la automatización, control y monitoreo, ha conducido a la implantación de soluciones de ingeniería que aborden estas necesidades de manera directa y faciliten las actividades de los usuarios, permitiendo un uso más eficiente de recursos y aumentando la productividad en los procesos. Sin embargo, se han acogido prácticas que aunque proporcionan el objetivo de transmisión de datos a un observador lejano y permiten el control del dispositivo remotamente desde la Internet, en realidad no ofrecen grandes prestaciones y a la vez implican gastos que podrían considerarse innecesarios.

Se propone entonces una evolución de este sistema, mejorando ampliamente las prestaciones, reduciendo costos específicos por unidad de dispositivo recolector utilizada, conservando siempre su objetivo original de comunicación y control, y aplicando en este caso la arquitectura cliente-servidor que centraliza un núcleo de comunicación que brinda amplias posibilidades de implementación de nuevas funcionalidades de seguridad, estadísticas, almacenamiento de datos, etc.

1. DESCRIPCIÓN DEL PROYECTO

1.1. DESCRIPCIÓN DEL PROBLEMA Y JUSTIFICACIÓN

Las soluciones implantadas en el entorno nacional a la tendencia de la automatización, consisten de manera general en la conexión a la red de datos de dispositivos electrónicos programados de manera embebida mediante un firmware (combinación de memoria persistente, código de programa y datos almacenados), permitiendo así, un control básico y comunicación individual sobre estos dispositivos a través de una conexión común en Internet, y a través de un computador remoto, que ejecuta una aplicación de escritorio capaz de trabajar bajo el mismo protocolo.

Este sistema soluciona la necesidad más básica de “Monitoreo y control remoto”, pero por varias razones no puede considerarse una solución robusta; En primer lugar se requiere de una dirección IP pública para cada dispositivo, debido a que estos dispositivos trabajan en modo servidor, dirección que debe mantenerse estática permanentemente, ya que el operario de la aplicación de escritorio, se conecta al dispositivo a través de ésta, en consecuencia, cuando el número de dispositivos aumenta, aspectos como el costo y la complejidad de administración también aumentan generando reducción en los beneficios de uso.

En segundo lugar, actualmente es un hecho que la mayoría de los sistemas de telemetría y telecontrol que se implementan son personalizados, no solo por su hardware (Dispositivos implicados en la implementación), también el protocolo de comunicación, el firmware e indiscutiblemente la aplicación del lado del cliente que se emplea para controlar el dispositivo; al carecer de una plataforma de fácil implementación en el mercado, cuando una nueva solución de este tipo es requerida, los desarrolladores deben pensar nuevamente en todos los aspectos y generar soluciones únicas para cada cliente, para cada tarea, e incluso para cada nuevo proyecto solicitado por un mismo cliente, a pesar de que en muchos casos compartan cierta similitud.

En tercer lugar la lógica que se aplica en los sistemas conocidos a la fecha, requiere de un operario que esté constantemente conectado al dispositivo captador de datos y actuador para su normal funcionamiento, en caso de ausencia de dicho operario, el dispositivo podría considerarse desperdiciado, ya que es el operario quien se encarga de la mayor parte de la gestión al

sistema, además de llevar por su parte un histórico de los cambios. En conclusión, la mayoría de las funcionalidades establecidas para estos sistemas las debe ejecutar un operario, estamos hablando entonces, de un grado poco considerable de autonomía, aspecto muy ventajoso en estos sistemas para la gestión y administración, como para las funcionalidades específicas.

En cuarto lugar y paralelo al ítem anterior, en la lógica de un sistema de telemetría y telecontrol que carezca de un “middleware”, es irrelevante contemplar la posibilidad de que un dispositivo interaccione, influya o determine el actuar de otro que se relacione con él, física o lógicamente, característica indispensable en un sistema de telemetría robusto, ya que la ausencia de interacciones entre los componentes hardware del sistema limita el alcance y la utilidad de un sistema de este tipo.

Analizando esta situación, sobresale la posibilidad de optimizar lo que se tiene, centralizando las comunicaciones de estos dispositivos electrónicos a través de un servidor capaz de gestionar el flujo de datos provenientes de los dispositivos de telemetría y ejercer el tele-control, aplicando la arquitectura cliente servidor mediante un núcleo de comunicación que permita la interacción entre dispositivos por paso de mensajes y facilitar la gestión y seguimiento de los mismos sin necesidad de la presencia de un operario, que cuente con un componente de almacenamiento de datos, además, de proveer una interfaz agradable e intuitiva que facilite que los usuarios accedan a las funcionalidades propuestas.

Es importante aclarar que el actual proyecto se orienta a generar una solución genérica aplicable a escenarios y dispositivos electrónicos que cuenten con soporte para programación embebida y con conexión a la red de datos mediante una interfaz de red genérica Ethernet que implemente el conjunto de protocolos para la World Wide Web en especial TCP/IP.

1.2. OBJETIVOS

1.2.1. Objetivo General

Construir un prototipo software que aplique la arquitectura Cliente-Servidor y permita la captación, interpretación, y almacenamiento de datos a cajas negras de telemetría y la administración de estas a los usuarios.

1.2.2. Objetivos Específicos

- Diseñar un protocolo orientado a telemetría y telecontrol que normalice la comunicación entre los dispositivos y el prototipo propuesto, basando su estructura en comandos y parámetros complementarios a estos.

- Implementar un componente que permita aceptar conexiones para la recepción de paquetes mediante Sockets TCP/IP y la interpretación de estos utilizando el protocolo anteriormente propuesto.

- Implementar un componente que almacene los datos y permita realizar operaciones CRUD (Crear, Obtener, Actualizar y Borrar) sobre estos.

- Implementar un componente que permita a los usuarios la visualización de los datos correspondientes a su caja negra y la administración de determinados parámetros de la misma.

- Realizar el diseño y la implementación de los componentes propuestos, aplicando la metodología FDD (Desarrollo Dirigido por Características) para organizar este proceso en áreas específicas, actividades y características con el propósito de llevar a cabo un desarrollo ágil.

- Validar el prototipo final mediante pruebas de conectividad y simulación de paquetes, con el fin de determinar la correspondencia del prototipo con el uso previsto.

2. MARCO TEÓRICO

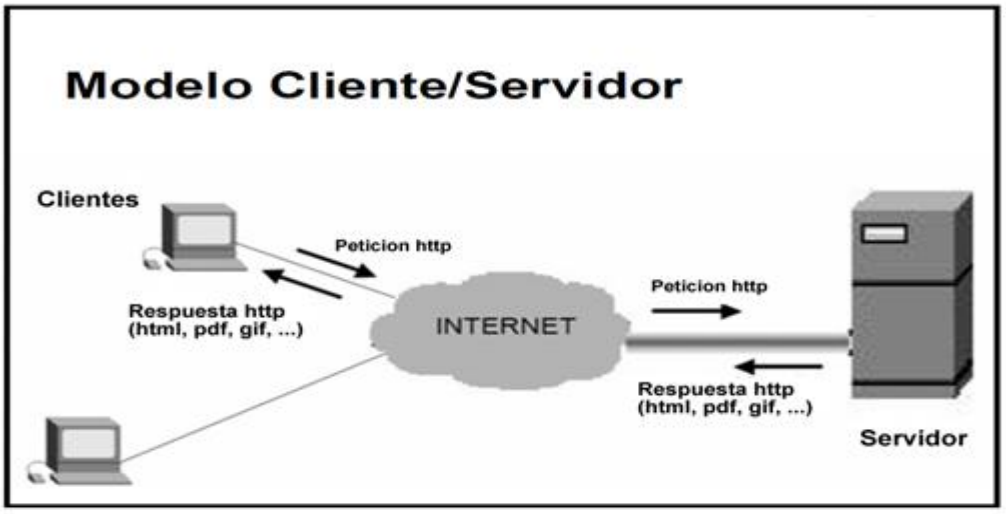
En este capítulo se abarca la fundamentación teórica necesaria para comprender el desarrollo del prototipo propuesto. Inicialmente se abarcan conceptos generales sobre la arquitectura cliente servidor, los sistemas embebidos y el protocolo TCP/IP y posteriormente se describen conceptos relacionados con las herramientas y modelos utilizados para la implementación.

2.1. ARQUITECTURA CLIENTE SERVIDOR

La arquitectura Cliente/Servidor es un modelo distribuido en el que se dividen y especializan las tareas entre los proveedores de servicios o recursos y los demandantes, llamados servidores y clientes respectivamente, de forma que la tarea que cada uno de ellos realiza se efectúe con la mayor eficiencia posible y permita simplificar las actualizaciones y mantenimiento del sistema.

Los clientes (generalmente distribuidos geográficamente) envían uno o varios mensajes realizando así la petición de un determinado servicio al programa servidor, que les da respuesta proveyendo el servicio.

Figura 1. Modelo Cliente - Servidor



Fuente: Arquitectura Cliente/Servidor - Universidad de las Américas Puebla.

Características principales de la arquitectura Cliente – Servidor:

-En un sistema distribuido cada máquina puede cumplir el rol de servidor para algunas tareas y el rol de cliente para otras.

-Las tareas del cliente y del servidor tienen diferentes requerimientos en cuanto a recursos de cómputo como velocidad del procesador, memoria, velocidad y capacidades del disco.

-La relación establecida puede ser de muchos a uno, en la que un servidor puede dar servicio a muchos clientes, regulando el acceso a recursos compartidos.

-Se presentan procesos activos y pasivos, los primeros provienen de los clientes debido a que estos realizan las peticiones y proporcionan la interfaz entre el usuario y el resto del sistema; los pasivos provienen del servidor, debido a que este espera las peticiones de los clientes y a su vez funciona como motor de software que maneja los recursos compartidos como las bases de datos.

-El cliente realiza funciones como administrar interfaz de usuario, interactuar con el usuario, procesar la lógica de la aplicación y hacer validaciones locales, generar requerimientos de bases de datos y recibir los resultados del servidor.

-El servidor realiza funciones como procesar los requerimientos de bases de datos que hacen los clientes, formatear datos para transmitirlos a los clientes, procesar lógica de la aplicación y realizar validaciones a nivel de bases de datos.

---

[1] Arquitectura Cliente Servidor. Capítulo 5. Universidad de las Américas Puebla. 2008. [En línea] [Consultado 22/03/2014]. Disponible: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/marquez_a_bm/capitulo5.pdf

2.2 SISTEMAS EMBEBIDOS

Un sistema embebido es un circuito electrónico computarizado que está diseñado para cumplir una labor o satisfacer una función específica en un producto, por ejemplo un reloj, un control de temperatura en electrodomésticos, una alerta sobre eventos inesperados, etc. [2]

Un microprocesador embebido es aquel que está dedicado a controlar una función específica y arranca por sí mismo sin requerir la intervención humana y tiene contenidos propios con sus programas de operación. [3]

Un sistema microprocesador consta de tres partes: la unidad central de procesamiento (CPU), la cual reconoce y ejecuta las instrucciones de un programa. Ésta es la parte que usa el microprocesador, las interfaces de entrada y salida, para manejar las comunicaciones entre la computadora y el mundo exterior; el término puerto se usa para la interfaz, y la memoria es donde se almacenan instrucciones de programas y datos. [3]

Dentro de un microprocesador, las señales digitales se mueven a lo largo de los buses que son trayectorias paralelas para transmisión de datos paralelos en lugar de datos en serie.

Los microprocesadores que contienen memoria y varios arreglos de entrada y salida en un mismo chip se llaman microcontroladores.

Un microcontrolador es una pequeña computadora que tiene residente en su memoria la inteligencia artificial, secuencias y algoritmos de un sistema embebido, es decir, un microcontrolador es el cerebro de un sistema electrónico que está contenido (“embebido”) dentro de un equipo que incluye partes mecánicas y electromecánicas.

---

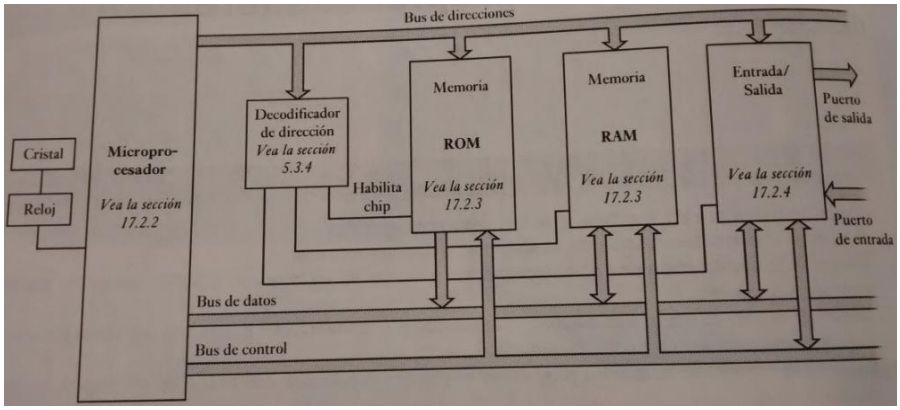
[2] GALEANO, Gustavo. Programación de Sistemas Embebidos en C. Colombia. Alfaomega, 2009. P.2.

[3] W. Bolton. Mecatrónica. Sistemas de control electrónico en la Ingeniería Mecánica y eléctrica. Quinta edición. México D.F. Alfaomega, 2013. Cap 17. P.360 - 374

Para que un microprocesador pueda funcionar como un sistema aplicado al control, son necesarios chips adicionales, por ejemplo, dispositivos de memoria para almacenar programas y datos, así como puertos de entrada/salida para permitir que se comunique con el mundo exterior y reciba señales desde él.

El microcontrolador integra en un chip de microprocesador con memoria, interfaces de entrada /salida y otros dispositivos periféricos como temporizadores, de los que hablaremos posteriormente.

Figura 2. Forma general de un sistema microprocesador



Fuente: Sistemas de control electrónico en la Ingeniería Mecánica y eléctrica. [3]

CPU:

El microprocesador es la unidad de procesamiento central CPU. En esta parte se procesan los datos, se traen instrucciones y datos. Las partes que forman un microprocesador son: Unidad aritmético-lógica (ALU), responsable de llevar a cabo la manipulación de los datos, los registros que son los datos internos que la CPU suele utilizar, estos se mantienen temporalmente mientras se ejecutan las instrucciones, Unidad de control que determina la temporización y secuencia de las operaciones, genera señales utilizadas para traer del a memoria una instrucción del programa y ejecutarla. Las operaciones pertenecientes a los microprocesadores se reconocen por la cantidad de ciclos que se requieren para ejecutarlas. La cantidad, la dimensión y el tipo de registros varía de un microprocesador a otro. Los registros más comunes son: Registro acumulador, Registro de estado, Registro contador del programa o apuntador de instrucciones, Registro de direccionamiento de memoria (MAR), Registro de instrucciones (IR), Registros de propósito general y Registro de apuntador de la pila (SP).

Memoria:

Guarda datos binarios y toma la forma de uno o varios circuitos integrados. Los datos puedes ser códigos de instrucciones de un programa o números con los que se realizan operaciones. El tamaño de la memoria depende de la cantidad de líneas del bus de direcciones. Los elementos de la unidad de memoria están formados por grandes cantidades de celdas de memoria, cada una guarda un bit 0 o 1.

Las celdas de memoria se agrupan por localidades, cada una de estas puede guardar una palabra. Para acceder a una palabra almacenada, se identifica cada localidad por una dirección única. Así, en un bus de dirección de 4 bits se pueden identificar 16 direcciones diferentes, cada una capaz de guardar un byte (8 bits).

La capacidad de la unidad de memoria está establecida por la cantidad de localidades de memoria disponibles, 1K es 2^{10}= 1024 localidades, por lo que una memoria de 4K tiene 4096 localidades.

Hay varios tipos de unidad de memoria: ROM, que es una memoria de solo lectura, allí se guardan los datos de manera permanente, PROM, que es una memoria ROM programable, EPROM, que es una memoria ROM borrable y programable, es decir que se puede programar y posteriormente permite modificar, EEPROM, que es una memoria PROM eléctricamente borrable. Es una memoria parecida a la EPROM pero para el borrado se utiliza una alta carga de voltaje y RAM, que es una memoria de lectura/escritura de acceso aleatorio. Aquí se guardan los datos temporales, es decir, los datos que se utilizando para realizar operaciones.

Operación entrada/salida

Esta operación se define como la transferencia de datos entre el microprocesador y el mundo exterior. Los dispositivos periféricos o más comúnmente llamados “actuadores” se refieren a las piezas de equipo que intercambian datos con un sistema de microprocesador. Estos dispositivos se conectan a través de circuitos de interfaz debido a que las velocidades y características de estos pueden ser muy diferentes a las del microprocesador; dichos circuitos sirven principalmente para sincronizar la transferencia de datos entre el microprocesador y el dispositivo periférico. En las operaciones de entrada, el dispositivo de entrada coloca los datos en el registro de datos del circuito y estos datos permanecen ahí hasta que

los lee el microprocesador, de manera opuesta en las operaciones de salida el microprocesador coloca los datos en el registro hasta que el dispositivo los lee.

Un microcontrolador común tiene terminales para la conexión externa de entradas y salidas, alimentación eléctrica y señales de reloj y de control. Las conexiones de entrada/salida se agrupan en unidades denominadas puertos de entrada/salida, Por lo general, estos puertos tienen ocho líneas para poder transportar una palabra de datos de 8 bits. Para una palabra de 16 bits utilizan dos puertos, uno para transmitir los 8 bits inferiores, y otro para los 8 bits superiores. Los puertos pueden ser sólo entrada o sólo salida, o programables para funcionar como entrada o salida [3].

2.3 PROTOCOLO TCP/IP

Este protocolo surge de la necesidad de conectar múltiples redes en una manera sólida.[4] Su primera definición fue en 1974, realizada por Vinton Cerf y Robert Kahn, en donde se planteó como un método de comunicación entre redes de computadores a través de empaquetamiento de datos, posteriormente y luego de diversas definiciones, en 1988, el Departamento de Defensa de los Estados Unidos (DoD) implementa funcionalmente el protocolo [5], con el propósito de que las conexiones se mantuvieran en las máquinas de origen y destino, aunque las máquinas intermedias dejaran de funcionar repentinamente, declarándolo así como protocolo estándar para las comunicaciones entre redes militares.

El protocolo TCP/IP está organizado en capas, niveles o módulos encapsulados que realizan tareas precisas en un orden específico, de manera que cada uno es independiente del otro. En la figura 3 se pueden visualizar estas capas con los protocolos utilizados en cada una:

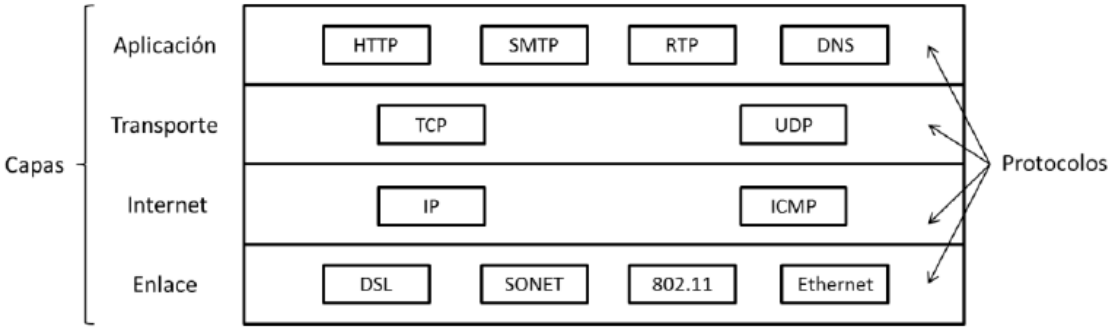
---

[3] W. Bolton. Mecatrónica. Sistemas de control electrónico en la Ingeniería Mecánica y eléctrica. Quinta edición. México D.F. Alfaomega, 2013. Cap 17. P.360 – 374

[4] A. Tanenbaum. Redes de Computadoras. México D.F. Pearson, Prentice Hall. 2010. Cap 1. P.41.

[5] J. Postel. «RFC 760» DoD Standard Internet Protocol. Enero 1980. [En línea] [Consultado 22/03/2014]. Disponible: http://tools.ietf.org/html/rfc760

Figura 3. Capas y protocolos en TCP/IP.



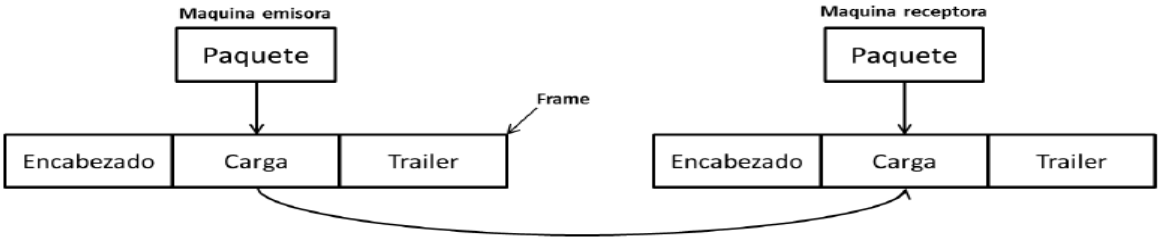
Fuente: A. Tanenbaum. Redes de Computadoras. [4]

2.3.1 Capa de enlace de datos

Capa que tiene como función principal tomar los paquetes provenientes de la capa de red para que puedan ser transmitidos entre emisores y receptores [4]. Es capaz de brindar una interfaz de servicio adecuadamente definida, manejar las faltas y/o errores que se presenten en la transmisión y administrar el flujo de datos de manera que la velocidad de envío se mantenga constante y no se vea afectada por la diversidad en las capacidades de emisores y receptores.

Para lograr lo anteriormente mencionado, esta capa realiza un proceso de encapsulamiento de los paquetes en tramas. Como se puede visualizar en la Figura 4 cada una está compuesta por un encabezado, un campo de carga para almacenar el paquete y un campo final de trama.

Figura 4. Capa de enlace de datos.



Fuente: A. Tanenbaum. Redes de Computadoras. [4]

[4] A. Tanenbaum. Redes de Computadoras. México D.F. Pearson, Prentice Hall. 2010. Cap 3. P.183 - 190.

Es importante mencionar que aunque los paquetes son teóricamente transmitidos entre las capas de internet, con la ayuda de las tramas de las capas de enlace del emisor y receptor respectivamente, realmente para la transmisión realizan el recorrido completo hasta la parte física de cada uno.

2.3.2 Capa de internet

También llamada capa de red, tiene como función principal llevar los paquetes desde el origen hasta el destino, es decir es la principal capa encargada de la transmisión [4]. En general esta capa le provee servicios a la capa de transporte. Para lograrlo, esta capa debe:

- Encaminar los paquetes (encapsulados en esta capa en una forma específica llamados datagramas) por la mejor ruta para que lleguen a su destino. Esta ruta puede ser calculada mediante los algoritmos de encaminamiento o enrutamiento.
- Enviar los datagramas las veces q sea necesario, esto debido a que para seguir la ruta adecuada mencionada en el ítem anterior, puede existir más de un dispositivo enrutador que debe recibir el datagrama antes de poder entregarlo al destino final [6].
- Realizar el encaminamiento y envío a través de los datagramas individualmente (es decir elegir una ruta específica para cada uno, en donde se corre el riesgo, para cuando son paquetes divididos, de que algunos no lleguen adecuadamente y deban re-enviarse para obtener el paquete completo, método conocido como no orientado a la conexión) o mediante una estrategia llamada circuitos virtuales y/o orientada a la conexión (que evita la selección de múltiples rutas y permite que todos los paquetes identificados con el mismo circuito sean enviados todos por la misma ruta ideal inicial).

---

[4] A. Tanenbaum. Redes de Computadoras. México D.F. Pearson, Prentice Hall. 2010. Cap 5. P.345 - 360.

[6] J. Postel. «RFC 1122» Comunication Layers. Octubre 1989. [En línea] [Consultado 22/03/2014]. Disponible: http://tools.ietf.org/html/rfc1122

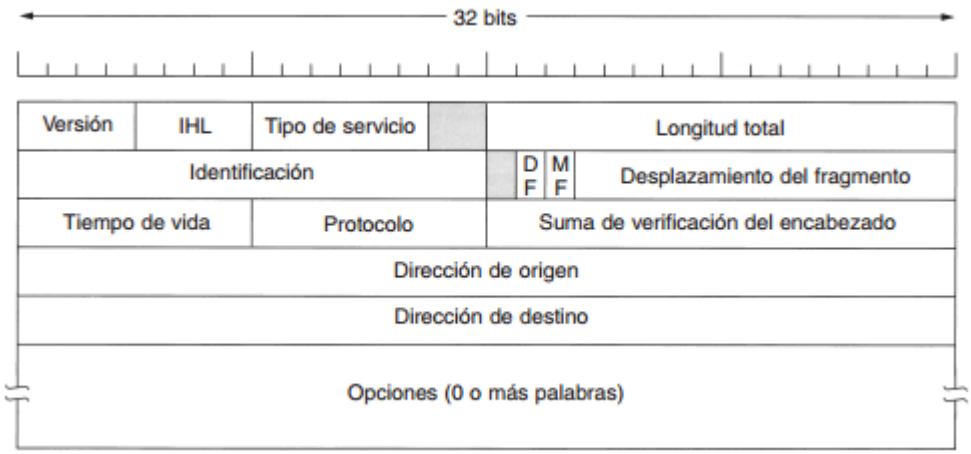
A continuación se presenta una breve explicación del protocolo más importante utilizado en esta capa y sobre el que trabaja el protocolo propuesto en este proyecto (RTUP): Internet Protocol (IP.

2.3.2.1 Protocolo IP

Este protocolo específico de capa de internet, tiene como finalidad transportar los datagramas a través de un conjunto de redes interconectadas, es decir, moverlos desde una capa de internet de una máquina hacia la capa de internet de otra máquina las veces que sea necesario hasta llegar al destino (estas capas residen en internet a través de puertas de enlace y hosts, es decir, el enrutamiento debe basarse en direcciones de internet) [7].

Los datagramas utilizados en este protocolo son llamados datagramas IP y están compuestos por dos partes: encabezado (con parte fija de 20 bytes y parte opcional de longitud variable, ver Figura 5) y texto.

Figura 5. Encabezado IP.



Fuente: A. Tanenbaum. Redes de Computadoras. [4]

---

[4] A. Tanenbaum. Redes de Computadoras. México D.F. Pearson, Prentice Hall. 2010. Cap 5. P.345 - 480.

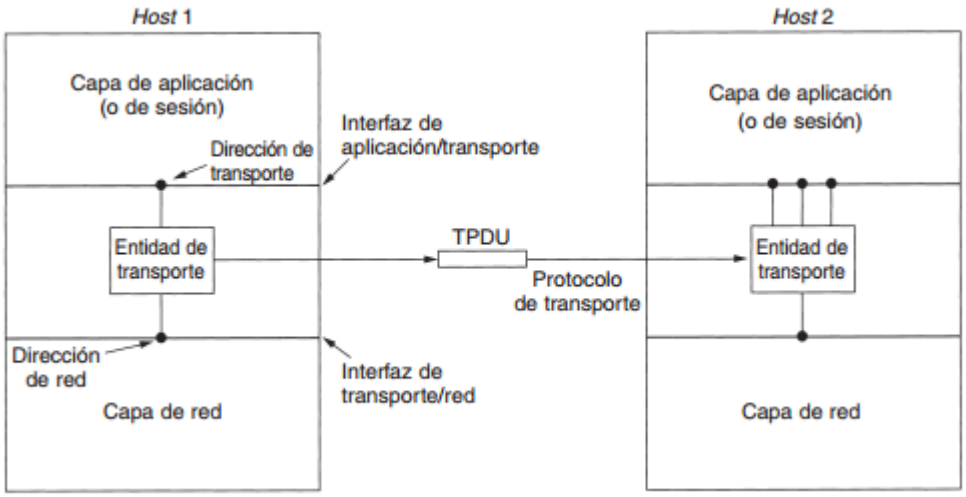
[7] J. Postel. «RFC 791» Internet Protocol. Septiembre 1981. [En línea] [Consultado 22/03/2014]. Disponible: http://tools.ietf.org/html/rfc791

2.3.3 Capa de transporte

Capa que tiene como función principal la eficiencia, confiabilidad y economía del transporte de los datos del origen al destino final, sin tener en cuenta la red física que se esté usando. Esta capa se considera como el corazón de la jerarquía de protocolos [4].

Para lograr su funcionalidad (proveerle servicios a la capa de aplicación), la capa de transporte utiliza los servicios proporcionados por la capa de internet. En la siguiente figura se puede observar la interacción que hay entre las tres capas superiores del modelo TCP/IP:

Figura 6. Capas de internet, transporte y aplicación.



Fuente: A. Tanenbaum. Redes de Computadoras. [4]

[4] A. Tanenbaum. Redes de Computadoras. México D.F. Pearson, Prentice Hall. 2010. Cap 5. P.345 - 480. Cap 6. P.481.

De la misma manera que en la capa de internet, la funcionalidad de la capa de transporte se puede realizar con método orientado o no a la conexión (ambos similares a los explicados en la capa de internet); en cualquiera de los casos se contará con un proceso compuesto por tres fases: establecimiento, transferencia de datos y finalización [4]. La razón por la que estos métodos existen tanto en capa de internet como en esta capa es que en la primera, el código es ejecutado directamente en los equipos enrutadores, los cuales en la mayoría de los cuales no se tiene acceso, por lo que sería prácticamente imposible garantizar la eficiencia y/o recuperación de datos perdidos, es por eso que surge la necesidad de crear una capa inferior, en donde se ejecute el código directamente en las máquinas de los usuarios, y así se mejore la calidad del transporte de datos.

A continuación se presenta una breve explicación del protocolo más importante utilizado en esta capa y sobre el que (al igual que IP Protocol) trabaja el protocolo propuesto en este proyecto (RTUP): Transmission Control Protocol (TCP); además se menciona el protocolo User Datagram Protocol (UDP) utilizado en este nivel como el método no orientado a la conexión.

2.3.3.1 Protocolo TCP

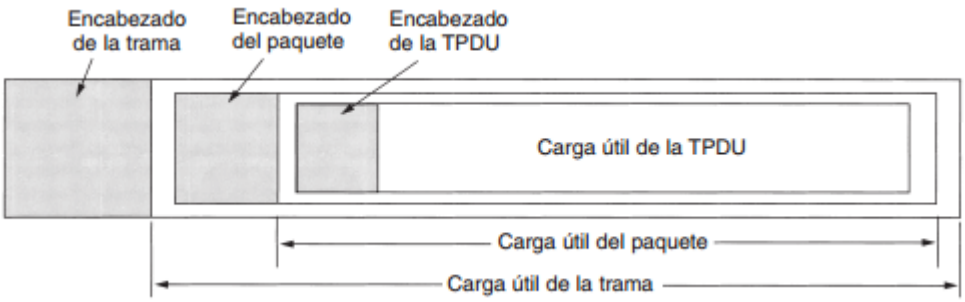
Protocolo de capa de transporte que tiene como funcionalidad proporcionar una entrega fiable orientada a la conexión de transporte a través una red no necesariamente confiable [6].
En esta capa el nivel de encapsulamiento es mayor (ver figura 7), ya que las tramas creadas a nivel de enlace de datos, que están a su vez encapsuladas en datagramas a nivel de internet, en este punto serán encapsuladas en ‘contenedores’ llamados segmentos o técnicamente TPDU (Unidades de datos del protocolo de transporte).

---

[4] A. Tanenbaum. Redes de Computadoras. México D.F. Pearson, Prentice Hall. 2010. Cap 6. P.482

[6] J. Postel. «RFC 1122» Comunication Layers. Octubre 1989. [En línea] [Consultado 22/03/2014]. Disponible: http://tools.ietf.org/html/rfc1122

Figura 7. Encapsulamiento a nivel de enlace, internet y transporte.



Fuente: A. Tanenbaum. Redes de Computadoras. [4]

En el momento en que el servidor y el cliente crean puntos terminales se obtiene el servicio TCP a través de los comúnmente llamados sockets.

Cada uno tiene un número asignado que es la dirección IP del host y además un número de 16 bits llamado puerto. El servicio es plenamente establecido cuando se conectan los sockets de las máquinas emisoras y receptoras. Los sockets son muy útiles debido a que pueden utilizarse a la vez para más de una conexión. Hay puertos asignados para representar servicios estándares:

Tabla 1. Listado puertos asignados.

Puerto	Protocolo	Uso
21	FTP	Transferencia de archivos
23	Telnet	Inicio remoto de sesión
25	SMTP	Correo electrónico
69	TFTP	Protocolo de transferencia de archivos trivial
79	Finger	Búsqueda de información sobre un usuario
80	HTTP	World Wide Web
110	POP-3	Acceso remoto al correo electrónico
119	NNTP	Noticias USENET

Fuente: A. Tanenbaum. Redes de Computadoras. [4]

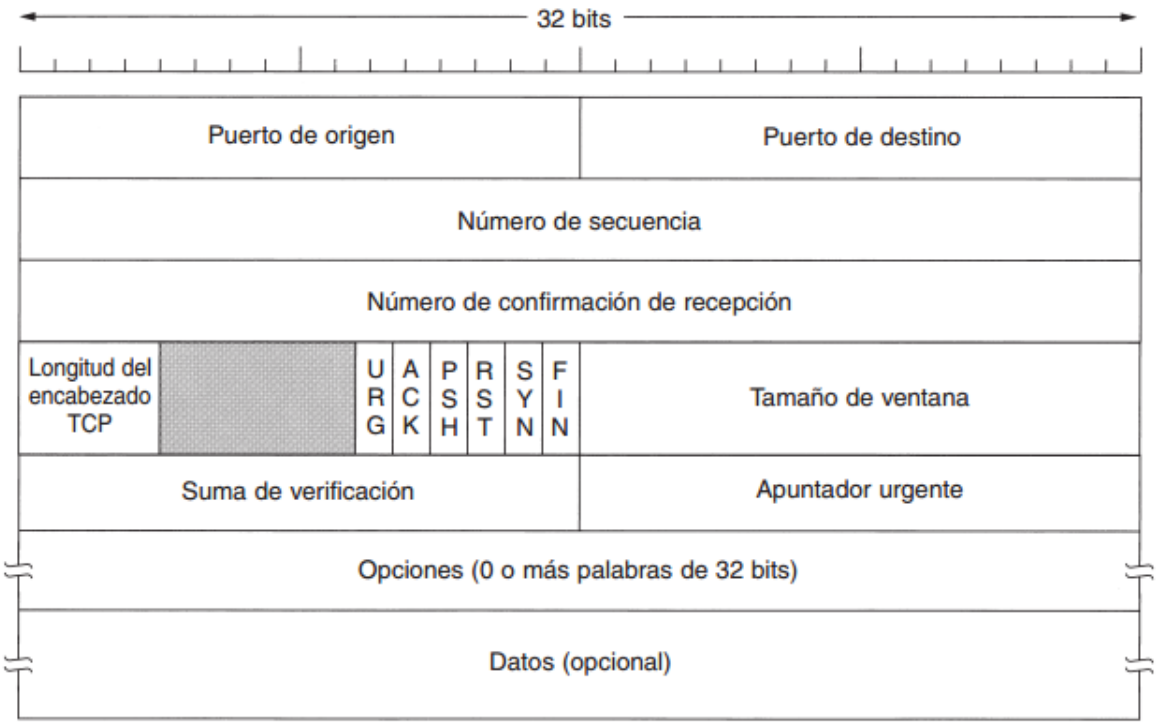
[4] A. Tanenbaum. Redes de Computadoras. México D.F. Pearson, Prentice Hall. 2010. Cap 6. P.482.

El encapsulamiento (segmentos) de los datagramas de esta capa tienen un encabezado particular, ya que está compuesto de 20 bytes de la siguiente manera [4]:

- Puertos de origen y destino: son los identificadores de la conexión, ya que dan nombre a los puntos terminales.

- Número de secuencia y número de confirmación: tienen 32 bites de longitud, el primero se refiere al orden de los paquetes que fluyen de extremo a extremo en la conexión; el segundo, se utiliza para confirmar que ya se recibieron los bytes y tiene como valor el siguiente byte esperado.

Figura 8. Encabezado del segmento TCP.



Fuente: A. Tanenbaum. Redes de Computadoras. [4]

[4] A. Tanenbaum. Redes de Computadoras. México D.F. Pearson, Prentice Hall. 2010. Cap 6. P.482

- Longitud del encabezado: en este campo se visualizan la cantidad de palabras de 32 bits que están contenidas en el encabezado TCP.
- Campo vacío: este espacio no es utilizado para ninguna finalidad.
- Indicadores: son 6 campos de longitud 1 bit cada uno, se utilizan para reportar estados de la comunicación, por ejemplo: ACK se encuentra en estado 1 cuando el número de confirmación de recepción es válido.
- Campo que indica la cantidad de bytes que se pueden enviar (empezando por el byte del que se recibí la confirmación).
- Suma de verificación: campo de 16 bits utilizado para brindar mayor confiabilidad. Es una suma de confirmación de errores en el encabezado y los datos.
- Opciones: en este campo se pueden incluir características que no han sido especificadas en los demás campos del encabezado.

2.3.3.2 Protocolo UDP

Protocolo que ofrece un servicio mínimo de transporte, ya que no garantiza la entrega de los datagramas, y les da a las aplicaciones acceso directo al datagrama de la capa de internet [6].

Este protocolo de capa de transporte no es orientado a la conexión ya que no mantienen conectados los dos extremos de la comunicación mientras el transporte se realiza, simplemente expulsa y acepta datagrama provenientes de la red [8].

2.3.4 Capa de aplicación

Esta es la capa superior del modelo TCP/IP, es en esta capa en donde ocurre la interacción con el usuario y en donde se da el acceso a los diversos protocolos que operan en las capas inferiores de la red.

---

[6] J. Postel. «RFC 1122» Comunication Layers. Octubre 1989. [En línea] [Consultado 20/07/2014]. Disponible: http://tools.ietf.org/html/rfc1122

[8] J. Postel. «RFC 1180» A TCP/IP Tutorial. Septiembre 1991. [En línea] [Consultado 22/03/2014]. Disponible: http://tools.ietf.org/html/rfc1180

Tiene como funciones principal identificar la disponibilidad de comunicación del destino presentado por el origen y decidir los recursos para que dicha comunicación pueda darse.

Esta capa tiene muchos protocolos implementados, algunos de los más comunes son: FTP, HTTP, TELNET, SMTP.

2.4 PATRÓN REACTOR

Es un patrón de programación concurrente (simultaneidad en la ejecución de múltiples tareas) para el manejo de solicitudes de servicios entregadas de manera simultánea a un administrador mediante una o más entradas. El administrador de los servicios separa todas las solicitudes y las responde de manera sincronizada a los respectivos administradores de cada una [9].

La implementación de este patrón consta de tres partes: primero los recursos, que pueden ser cualquier medio que proporcione entradas y/o salidas el sistema, segundo lo que podría llamarse demultiplexor síncrono de eventos que usa un bucle para bloquear los eventos hasta que se pueda dar respuesta una solicitud, momento en el que la tercera y última parte iniciaría su función, el despachador entrega ordenadamente los recursos al administrador de cada solicitud. Este último podría considerarse como una cuarta parte de este patrón, que consta de una aplicación con sus respectivos recursos para realizar la solicitud.

Este patrón contribuye directamente al funcionamiento efectivo del prototipo propuesto, ya que se encarga de la administración de las conexiones múltiples para las que está diseñado, soluciona los inconvenientes presentados con la concurrencia y orden de atención de solicitudes, por lo tanto es escogido para el desarrollo del componente relacionado con la conexión y capa de red.

Existen muchas implementaciones del patrón reactor para diversos lenguajes de desarrollo, en el capítulo eferente al desarrollo de la metodología se mencionan algunas y se explica con mayor detalle la escogida para este proyecto.

---

[9] Ruby on Rails Guides. «Object Relational Mapping». [En línea] [Consultado 31/05/2014]. Disponible: http://guides.rubyonrails.org/active_record_basics.html#object-relational-mapping

2.5 MAPEO RELACIONAL DE OBJETOS - ORM

ORM es una técnica utilizada para conectar los objetos de una aplicación con las tablas del sistema administrador de base de datos. La principal ventaja de usar ORM es que los objetos de la aplicación pueden ser actualizados, almacenados, leídos y/o eliminados fácilmente sin necesidad de usar sentencias directas de SQL y con menos código del generalmente utilizado para acceder a una base de datos [10].

Podría decirse que lo que ocurre cuando se usa ORM es la creación de una base de datos orientada a objetos virtual sobre la base de datos relacional real, de tal manera que se puedan utilizar las características propias de la orientación a objetos, brindando un alto nivel de facilidad en la manipulación de datos.

En comparación con las técnicas comunes de acceso a bases de datos, ORM reduce en gran parte la cantidad de código utilizado y presenta un alto nivel de abstracción ya que no permite visualizar lo que realmente está sucediendo en el código de la implementación, sin embargo, por expresarlo de alguna manera, tener restringido este acceso no es un aspecto que cause imprevistos o dificultades para el desarrollo de este prototipo, por el contrario contribuye a la construcción optimizada y basada en estándares existentes, por lo tanto ORM es la técnica escogida para implementar el componente relacionado al almacenamiento de datos, tema que será abordado en el capítulo de desarrollo de la metodología.

2.5 CONCEPTO 'INTERNET OF THINGS' (IoT).

El internet de las cosas es un concepto que hace referencia a como los objetos cotidianos pueden comunicarse a través de la interconexión digital con internet. Los objetos están compuestos de tecnología embebida para interactuar con estados internos y también con el ambiente externo, aspecto que inevitablemente ha hecho que cambien las decisiones, la forma en que se toman e incluso quien lo hace [11].

---

[10] Ruby EventMachine. «Reactor Pattern». [En línea] [Consultado 24/04/2014]. Disponible: https://www.igvita.com/2008/05/27/ruby-eventmachine-the-speed-demon/

[11] Cisco. «Internet of Things». [En línea] [Consultado 31/05/2014]. Disponible: http://www.cisco.com/web/solutions/trends/iot/overview.html

Los parámetros a resaltar de este concepto son la cantidad de datos e información generada a partir de este alto nivel de conectividad que a su vez implica eficiencia y seguridad

Partiendo de este concepto y los avances evidenciados hasta hoy, se propone por analistas de tecnología y visionarios del tema que el momento en que la cantidad de objetos conectados sea mayor que la de personas, no está lejos.

Involucramos este tema debido a la relación directa que se tiene con el prototipo desarrollado, las RTU hacen parte del conjunto de objetos interconectados que están comunicándose y generando información a cada momento.

3. METODOLOGÍA

Existe una gran variedad de metodologías que pueden abordarse para el desarrollo general de software, las relacionadas al modelo en cascada, en donde se abordan fases como análisis de requerimientos, diseño y documentación, implementación y desarrollo, pruebas y despliegue final, de manera que cuando una fase está completamente terminada se puede dar paso a la siguiente, y las relacionadas con el modelo ágil, en donde aunque se mantiene el concepto de fases, se incursiona en el desarrollo iterativo, permitiendo regresar a las fases iniciales, reduciendo así las debilidades del modelo inicial en donde cuando se llegaba a las fases finales, se descubrían errores cometidos en las fases iniciales, produciendo para corregirlos altos costos de dinero y tiempo.

Para el desarrollo del prototipo propuesto en este proyecto, se escogió la metodología ágil FDD (Feature Driven Development) o desarrollo orientado por características que a continuación será explicada de manera más detallada junto con los resultados obtenidos en cada etapa.

3.1 METODOLOGÍA FDD (Feature Driven Development)

Metodología ágil introducida en Singapur en el año 1997 por Jeff De Luca y Peter Coad, propone una división del desarrollo en 'características' pequeñas que son implementadas de manera rápida e independiente, brindando entregables tangibles y de calidad durante todo el proceso de desarrollo [12].

La metodología FDD propone algunas buenas prácticas para tener en cuenta cuando durante todo el proceso de desarrollo, entre ellas están: Modelado de objetos de dominio que consiste en la construcción de diagramas de clases que representan los tipos de objetos y sus relaciones, desarrollo por características y asignación de equipos a cada una, aspecto que se detalla más adelante en los procesos específicos propuestos por la misma metodología, inspecciones regulares utilizadas para garantizar la calidad del diseño e implementación,

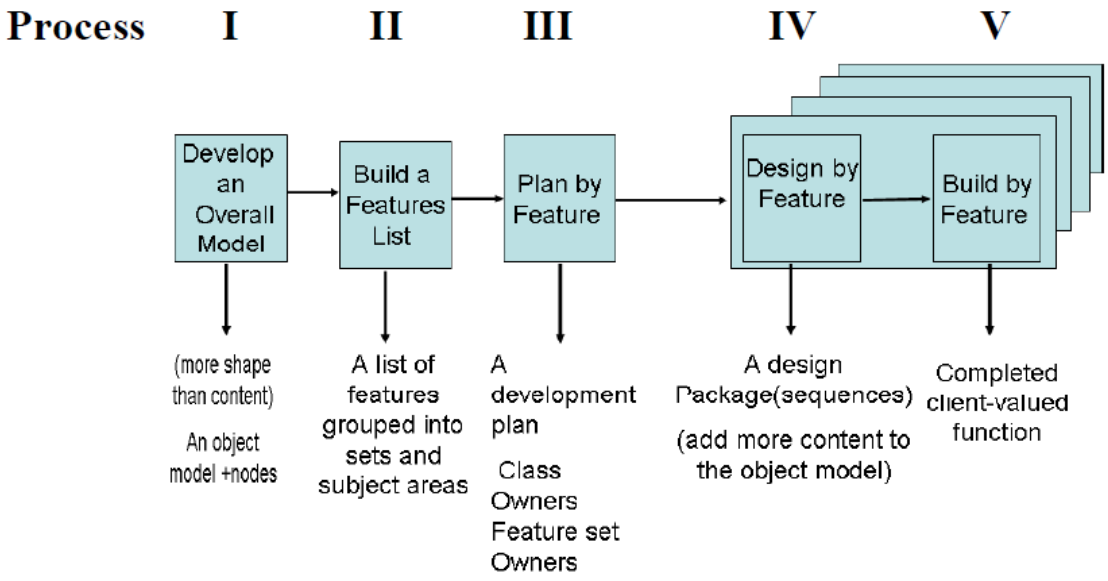
---

[12] S. Goyal. «Major Seminar on Feature Driven Development» Technical University Munich. Agosto 2007. [En línea] [Consultado 08/03/2014]. Disponible: http://csis.pace.edu/~marchese/CS616/Agile/FDD/fdd.pdf

manejo de configuraciones que hace referencia al control de las distintas versiones que puedan llegar a crearse del prototipo, reporte y visibilidad de resultados durante todo la construcción para mantener actualizados sobre avances y cambios a todos los interesados, entre otras.

Para el adecuado desarrollo de software y aplicando en lo mayor posible todas las buenas prácticas ágiles mencionadas, la metodología FDD propone los siguientes procesos:

Figura 9. Proceso Metodología FDD.



Fuente: S. Goyal. Major Seminar on FDD. [12]

3.1.1 Desarrollo de un modelo general

Paso inicial en el que los miembros del equipo de dominio, desarrollo y arquitectos presentan inicialmente el alcance del sistema y su contexto, de manera que posteriormente se planteen las áreas del problema y un modelo propuesto para cada una de estas áreas que luego serán puestos a discusión y el modelo correcto

---

[12] S. Goyal. «Major Seminar on Feature Driven Development» Technical University Munich. Agosto 2007. [En línea] [Consultado 08/03/2014]. Disponible: http://csis.pace.edu/~marchese/CS616/Agile/FDD/fdd.pdf

para cada área del problema es escogido, formando así el modelo general. Tanto las pequeñas áreas como el modelo completo propuesto en este proceso, pueden ser modificados al transcurrir el desarrollo si se encuentran inconsistencias.

Los criterios de entrada para este proceso inicial son los roles de expertos de dominio y desarrolladores escogidos.

Los criterios de salida son diagrama(s) de clase(s) que se centran en la forma del modelo, métodos y atributos identificados y asociados con las clases, diagramas de secuencia si se considera necesario, notas del modelo general con el fin de tener presente que aspectos fueron considerado para construirlo.

3.1.2 Construcción de lista de características

Es aquí en donde se identifican las características que dan soporte a los requerimientos del sistema. Se elabora entonces el listado de las características basado en la definición de característica como una función preciada para el cliente expresada en la forma <acción><resultado><objeto>. Posteriormente se agrupan las características por modelos de dominio, una actividad particular del negocio generalmente es reflejada en conjunto o categoría específica de características. Finalmente el listado es revisado para completarlo si es necesario [13].

Los criterios de entrada para este proceso son nuevamente los expertos de dominio, desarrolladores y además documentación relacionada con requerimientos existentes y/o especificaciones funcionales.

Los criterios de salida son el listado de áreas temáticas o “Subject areas”, para cada área un listado de “Business activities” relacionadas y finalmente para cada actividad la característica que la satisface: “Features list”.

3.1.3 Plan por característica

Durante este proceso en primer lugar se establece la secuencia de desarrollo teniendo en cuenta la prioridad de cada característica y posteriormente a cada grupo de características se le asigna uno o varios responsables [13].

---

[13] F. Palmer. «A Practical Guide to Feature-Driven Development». Prentice Hall 2002. [En línea] [Consultado 08/03/2014]. Disponible: http://www.nebulon.com/articles/fdd/download/fddprocessesA4.pdf

El criterio de entrada para este proceso es la lista de características totalmente definida.

El criterio de salida es la asignación del orden, la duración de desarrollo y el propietario a cada característica del listado definido durante el proceso anterior.

3.1.4 Diseño por característica

Se identifican las clases relacionadas con las características que inician la secuencia de desarrollo, especificando así lo que se va a desarrollar en la etapa. A continuación cada responsable elabora el diseño de los modelos y métodos correspondientes y necesarios para el desarrollo de su característica asignada.

El criterio de entrada para este proceso es la planeación por característica finalizada.

El criterio de salida es un paquete de diseño compuesto de un documento de especificación de detalles y requerimientos (si se considera necesario), diagramas de secuencia, modelo de objetos con las clases y atributos actualizados con lo que se haya considerado necesario añadir desde la última versión que se propuso en el proceso inicial.

3.1.5 Construcción por característica

Se implementan los detalles señalados en la etapa anterior, se escribe el código correspondiente y luego se inspecciona correctamente hasta su aprobación.

El criterio de entrada para este proceso es el diseño completo por cada característica.

Los criterios de salida son las clases y métodos de la característica ya implementados.

Los procesos 4.1.4 y 4.1.5 son repetidos con todas las características hasta completar el desarrollo total del sistema; es importante resaltar que el proceso de documentación se realiza me manera paralela a las fases indicadas anteriormente.

---

[13] F. Palmer. «A Practical Guide to Feature-Driven Development». Prentice Hall 2002. [En línea] [Consultado 08/03/2014]. Disponible: http://www.nebulon.com/articles/fdd/download/fddprocessesA4.pdf

4. PROTOCOLO PARA UNIDADES REMOTAS DE TELEMETRÍA V. 1.0 (Remote Telemetry Unit Protocol – RTUP)

4.1 Introducción

RTUP es un protocolo basado en comandos para comunicaciones en red, trabaja en la capa de aplicación (definida en el modelo de la pila de protocolos TCP/IP), funciona sobre TCP en capa de transporte y está planteado para ser tenido en cuenta como un estándar de comunicación bidireccional para sistemas de telemetría o control remoto que se implementan en base a la arquitectura Cliente – Servidor.

Figura 10. Ubicación RTUP en la pila de protocolos.

Aplicación	RTUP
Transporte	TCP
Red	IP

4.2 Lineamientos

El protocolo RTUP surge tras la iniciativa de introducir la arquitectura Cliente – Servidor en la implementación de sistemas de telemetría y telecontrol que se realiza mediante el uso de Microcontroladores conectados a Internet a modo de unidades de recolección de datos (RTU), de las cuales se desprende una red de sensores y actuadores que se encargan de recolectar datos o realizar diversas acciones respectivamente, surge entonces la necesidad del diseño de un protocolo para soportar labores como la transmisión de datos captados por este tipo de emisores hacia un receptor que se encarga de almacenar, procesar los datos y orquestar el funcionamiento de los emisores enviando órdenes a estos.

Dada la gran diversidad de fabricantes y modelos de este tipo de dispositivos, se espera que el protocolo RTUP soporte funciones básicas comunes a todos los diferentes tipos de unidades remotas de telemetría. Un protocolo no puede estar ligado a características específicas de un determinado fabricante o modelo de dispositivo, en base a este requisito, es que el problema se aborda desde el enfoque de la teoría cajas negras (Black-Box Theory), en donde se analiza cada elemento que conforma un sistema como una “caja negra”, lo cual significa que se

desconoce su funcionamiento interno (esto es lo que difiere de un modelo y fabricante de dispositivo a otro), y por el contrario se conocen las entradas que la caja negra debe recibir y las salidas que debe generar, es decir, las unidades remotas de telemetría, así como el servidor central al que estas se consideraron cajas negras para el diseño de este protocolo (las entradas y salidas se refieren a los paquetes que en conjunto constituyen la especificación del protocolo RTUP).

Es mediante el anterior análisis que se concluyen tres premisas que constituyen los lineamientos sobre los cuales se diseñó el protocolo RTUP: la primera se trata de las funciones que el sistema debe realizar las cuales se enumeran y se describen; la segunda es, que a pesar de que exista gran variedad de unidades remotas de telemetría que se diferencian por fabricante y modelos, el protocolo debe ser universal a estos dispositivos, explotando las funciones y características comunes a todos ellos; la tercera y última es que estos dispositivos como miembros del sistema, estarán conectados a Internet y se pretende que ellos se conecten a un servidor central el cual también tiene funciones definidas.

4.3 Motivación

El protocolo descrito en este capítulo hace parte de la solución para la implementación del prototipo de software (del que habla este documento) orientado a solucionar la necesidad de un servidor central para sistemas de telemetría y control remoto con base en transmisión de comandos cargados de argumentos a través de Internet, y por tanto este protocolo hereda la mayoría de los requisitos iniciales, reunidos en la fase de recolección de requisitos para este proyecto y aunque el protocolo aquí descrito constituye una pieza clave para el desarrollo del prototipo software mencionado, se propone a este protocolo como uno universal, genérico y abierto a diversas implementaciones.

Además de las anteriores consideraciones, el diseño de la presente especificación de protocolo, se ve influenciado por la metodología(se detallará posteriormente) que sigue el proyecto anteriormente mencionado, esta metodología es FDD (Feature Driven Development) o “Desarrollo Orientado por Características”, se trata de una metodología ágil para abordar problemas complejos de desarrollo de software; la influencia que ejerce FDD en el diseño radica en el concepto conocido como “Feature” o “Característica” que la metodología introduce como: Una definición en términos del cliente final sobre una funcionalidad específica con la cual debe contar el sistema. El diseño del protocolo RTU, trata de posibilitar las siguientes características:

- Transmisión de magnitudes físicas desde una unidad remota de telemetría hacia un servidor central.

- Transmisión de texto plano o bajo el formato JSON desde una unidad remota de telemetría hacia un servidor central.
- Transmisión de órdenes desde el servidor hacia una unidad remota de telemetría que desencadenen acciones en ella.
- Transmisión de configuraciones en ambas direcciones para hacer seguimiento e identificar cada una de las unidades remotas de telemetría.
- Transmisión de alertas desde una unidad remota de telemetría hacia el servidor.
- Autenticar la identidad de la unidad remota de telemetría antes de aceptar la interacción de esta con el servidor.

4.4 Modelado del contexto del problema

Tomando como base que la implementación aplicará la arquitectura cliente – servidor, se espera que los clientes tengan la capacidad de buscar y conectarse a un servidor; este servidor será único, un ente central en el sistema, no obstante como ya se definió con anterioridad, tanto el servidor como los clientes serán considerados “Cajas Negras”; aquí existe una distinción importante: en el contexto del problema, las cajas negras de tipo cliente serán principalmente emisores de datos, los cuales representan objetos del mundo físico, y cuyos datos representan magnitudes medidas o captadas mediante sensores; además de esta característica (considerada principal) estos clientes también pueden contar con la capacidad de llevar a cabo determinadas acciones mediante diversos actuadores (por ejemplo encender/apagar luces, encender/apagar alarmas, abrir/cerrar puertas, abrir/cerrar válvulas, entre otras).

Por su parte la caja negra de tipo servidor, cubre funciones muy diferentes, ya que esta principalmente se dedicará a recibir datos y almacenarlos para cualquier posterior procesamiento que desee realizarse, además de recibir datos, el servidor debe ser capaz de enviar órdenes a los clientes para que estos realicen determinadas acciones, otra función del servidor es el seguimiento de los clientes en diferentes aspectos, primero debe ser capaz de identificar cada cliente que se conecta a él y relacionarlo con una estructura abstracta, para mantener el estado actual de cada cliente, y de esta manera llevar un control preciso del funcionamiento de cada uno, también en el proceso de asociar un cliente con una estructura abstracta debe realizar una autentificación para que se valide que cliente es efectivamente el que desea conectarse y no un ente externo tratando de acceder al sistema de manera no autorizada, es decir, no todo cliente necesariamente debe ser atendido, solo aquellos que pasen la fase de validación del lado del servidor, comprobando la identidad que estos presentan al tratar de conectarse.

En base a las anteriores consideraciones, se establece un modelo de comunicación conformado por comandos que se cargan de argumentos; se consideran preestablecidos los comandos que solucionan los requisitos bases mencionados antes, pero no se sesgará la posibilidad de implementar más comandos en el futuro; cada comando se identifica por un número entero, el primer comando será identificado como el comando cero, el segundo será el comando uno, y así sucesivamente; dado que existen dos tipos de actores en el modelo de comunicación (Clientes, Servidor), existirá el concepto de "sentido" en la definición de los comandos, ya sea un comando emitido por un cliente que se dirige hacia el servidor o viceversa. Los comandos se cargan de argumentos o parámetros en el sentido de que además de un identificador de comando, en cada descripción de los comandos se definirá que parámetros debe incluir y como deben ser incluidos, finalmente se opta por ajustar la sintaxis de los comandos al formato ligero para intercambio de información conocido como JSON (JavaScript Object Notation) el cual se puede considerar intuitivo para el ser humano y como formato estándar al referirnos a acciones de análisis y generación por parte de máquinas.

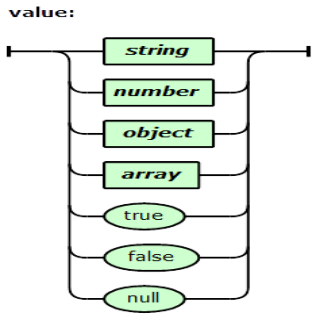
4.5 Generalidades

Dado que la sintaxis para el protocolo RTU utiliza el estándar JSON, se contempla a continuación una corta introducción a este formato.

El formato JSON abarca cinco conceptos, los cuales se explican mediante una breve descripción y un "Rail Road Diagram", estos diagramas suelen ser utilizados para describir gramáticas de libre contexto (como lo son la mayoría de lenguajes de programación o los formatos de transmisión de información), la idea base es que el diagrama tiene un punto inicial, un punto final y una serie de vías o rieles que permiten llegar desde el punto inicial hasta el final, existen diversas vías para este fin, y son precisamente estos rieles los que denotan todas las posibilidades que se pueden presentar en la gramática respecto a como avanza el diagrama, es decir hay vías únicas y hay vías opcionales, las vías únicas serían caracteres o porciones de estructura obligatorias, mientras que las vías que se bifurcan son mutuamente excluyentes, donde solo una puede ser recorrida en la tarea de alcanzar el punto final.

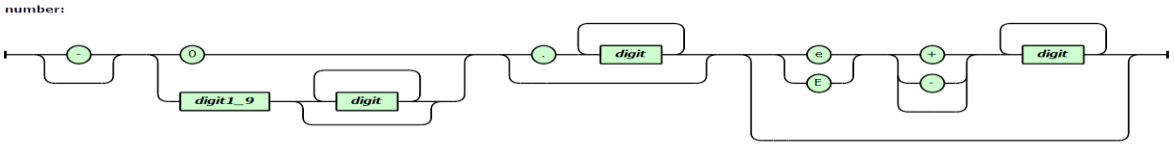
- **JSON Value:** Se considera que un "Valor" en el formato JSON puede ser uno de los siguientes: un número, una cadena de texto, "true" que significa verdadero, "false" que significa falso, "null" que significa vacío, un JSON array o un JSON object, que son estructuras para almacenar colecciones de datos.

Figura 11. Formato JSON Value.



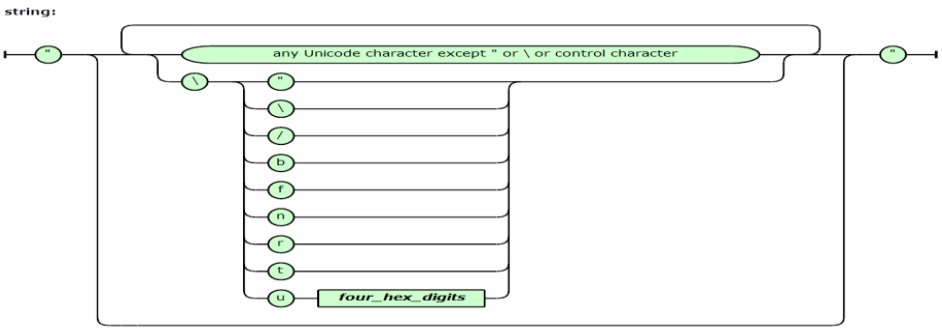
- **JSON Number:** Este es el formato para describir números reales, el estándar se describe mediante el siguiente Rail Road Diagram, el cual indica que la función de describir números en forma exponencial también es soportada.

Figura 12. Formato JSON Number.



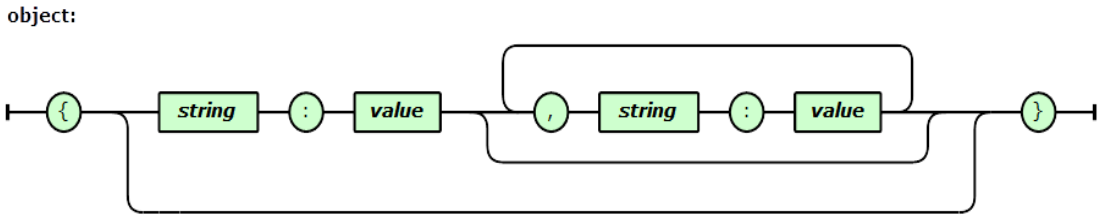
- **JSON String:** Las cadenas de texto en la sintaxis de JSON se consideran secuencias de cero o más caracteres unicode, se delimita entre dobles comillas, existe un caracter de control "\", que permite insertar caracteres especiales, como el salto de línea, tabulación, entre otros.

Figura 13. Formato JSON String.



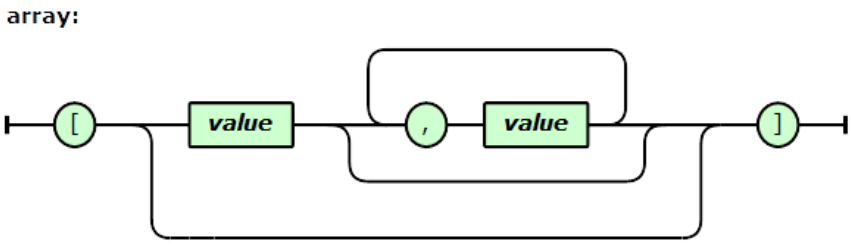
- **JSON Object:** Es la primera de dos estructuras primordiales en el formato JSON, esta se compone de una secuencia de parejas que a su vez se componen de dos elementos, un JSON string a modo de llave y un JSON value a modo de valor para esa llave, la secuencia se delimita entre los caracteres “{” y “}” y cada pareja se separa de la siguiente mediante una coma “,” y a su vez las llaves se separan de los valores mediante dos puntos “:”.

Figura 14. Formato JSON Object.



- **JSON Array:** Es la segunda estructura primordial en el formato JSON, esta se compone de una secuencia de JSON values que se delimita mediante los caracteres “[“y“]”, también cada miembro de la secuencia se separa de los siguientes mediante una coma “,”.

Figura 15. Formato JSON Array.



4.6 Especificación de comandos para RTUP.

Tabla 2. Resumen de comandos (incluye sentido).

Identificador de Mensaje	Descripción	Emitido Por	
		RTU	Servidor
0	Finaliza la conexión	X	X
1	Saludo	X	
2	Transmisión de entradas digitales	X	
3	Transmisión de salidas digitales	X	X
4	Transmisión de entradas análogas	X	
5	Transmisión de salidas análogas	X	X
6	Transmisión de texto plano o JSON	X	X
7	Transmisión de variables del RTU	X	X
8	Transmisión de distribución de pines	X	
9	Transmisión de código de Alerta	X	
10	Solicitudes		X
11	Reconocimiento del saludo		X
12	Reconocimiento de autenticación		X
13	Autenticación	X	

Además de estos mensajes, se implementó un mecanismo de comunicación interna entre los usuarios(a través de la interfaz web Telemetry Web App, que se muestra posteriormente en el capítulo 5) y el prototipo propuesto RTUServer, esto mediante un nuevo mensaje con identificador 14 que permite a los usuarios desencadenar un envío de un mensaje a los dispositivos RTU, es decir, el usuario emite este comando hacia nuestro servidor RTUServer para que así mismo este envíe el respectivo comando al dispositivo, por ejemplo un comando solicitando la transmisión de salidas digitales o análogas, o la transmisión de variables del dispositivo.

4.7 Sintaxis.

La sintaxis RTU para todo comando consiste en un JSON object que contiene dos parejas, la primera con llave “cmd” y el segundo con llave “arg”, cada comando se diferencia de otro en sintaxis por los valores que se asignan a estas llaves, en el caso de “cmd” el valor debe ser un número entero, tal como lo indica la anterior tabla, en el caso de “arg” el valor asignado debe ser un JSON Object. Es importante aclarar que en este capítulo se presenta una sintaxis que es orientada a ser leída y entendida por humanos, pero para la implementación del protocolo no se requieren espacios, o saltos de líneas ni tabulación en los paquetes, solo se presenta de esta manera para que sea más clara la estructura de cada paquete.

{“cmd”:***command_number***, “arg”:***JSON_Object***}

Comando 0: Finaliza la conexión

Este comando indica que la otra parte cerró la conexión, en términos de la pila de protocolos este comando haría las veces de un “acknowledgement” o reconocimiento, comando cuya intención es dar a conocer el cierre del canal de comunicación voluntario por parte de uno de los extremos, en cuanto al JSON object que se pasa como argumento en “arg”, este contiene una única pareja con llave “msg” y valor asignado de tipo JSON_String, el valor representa el mensaje que explica en términos técnicos el por qué el otro extremo cerro la conexión.

{

“cmd”:0,

“arg” :{

“msg”: ***JSON_String***

}

}

Comando 1: Saludo

Es el primer comando que espera recibir el servidor de parte de un cliente que intente conectarse, dado que técnicamente todas las conexiones son aceptadas

por la capa de transporte de la pila de protocolos TCP/IP, esto siempre y cuando la conexión no vaya en contra de alguna regla preestablecida en el firewall, es por esto que es necesario controlar el acceso de los clientes en la capa de aplicación, dado que el servicio que provee el servidor central de un sistema de telemetría está orientado a atender y asignar recursos a aquellos dispositivos que se encuentran registrados para dicho fin, no toda conexión debe ser aceptada y por ninguna razón se deben asignar recursos para atender a clientes que no estén registrados o que no cumplan con las reglas del protocolo. El comando de saludo es el primero de cuatro pasos que se deben realizar en orden para llevar a cabo un control de acceso mediante autenticación, este comando pasa como argumento el código serial del dispositivo al servidor, este código serial es un identificador único que se asigna a cada RTU, coincidiendo el serial, el servidor podrá determinar si este dispositivo es efectivamente un cliente registrado y activo en el sistema; la sintaxis del JSON object asignado como argumento en el comando es la siguiente(ver comandos 11,12 y 13):

```
{
        “cmd”:1,
        “arg” :{
                “serial”: JSON_String
                }
}
```

Comando 2: Transmisión de entradas digitales

Este comando es idéntico en estructura al comando 3, no obstante existen importantes diferencias con respecto a la lógica de que modelan, este comando es transmitido por las RTU, nunca por el servidor, y es usado para reportar el estado de las entradas digitales, se considera una entrada o salida digital a aquel terminación física presente en el RTU que puede generar o leer dos posibles estados, como lo son: presencia o ausencia de voltaje, lo que se traduce en la lógica del sistema a “0” para la ausencia y en “1” para presencia de voltaje, además de estos dos estados se considera un tercero que tiene que ver con la abstracción de entradas o salidas que no estén en uso o de las cuales no se desea o no se puede reportar su estado, este sería el estado representado por un 2. En cuanto a la estructura, el JSON object pasado como argumento, se compone de una pareja con llave “values” cuyo valor es un String simulando la estructura interna de un JSON Array que contiene un elemento tipo entero por cada entrada

que exista en la RTU, el sistema cuenta con una distribución de pines que se explicará más adelante y es la clave para saber que pin es el que reporta cada valor ya sea 0,1 o 2.

{

“cmd”:2,

“arg” :{

“values”: JSON_String

}

}

La estructura del JSON_string debe ser la siguiente: “[value0, value1,..., valueN]”

Comando 3: Transmisión de salidas digitales

Este comando puede ser transmitido por la RTU o por el servidor dado que pueden presentarse ambos casos, que el servidor esté ordenando a la RTU establecer cierto estado o que la RTU esté reportando el estado actual de sus salidas digitales, se siguen los mismos lineamientos del comando 2 (ver comando 2).

{

“cmd”:3,

“arg” :{

“values”: JSON_String

}

}

La estructura del JSON_string debe ser la siguiente: “[value0, value1,..., valueN]”

Comando 4: Transmisión de entradas análogas

Una entrada análoga es una terminación física en la RTU la cual puede leer una diferencia de potencial (Voltaje) y a partir de esta lectura puede establecer una

escala numérica que generalmente se indica mediante un número de bits, por ejemplo los sensores comúnmente establecen diferencias de potencial de entre 0 a 5 voltios en una escala de 10 bits, es decir 1024 posiciones (números enteros entre 0 y 1023), entre más alta sea la escala mayor sensibilidad en el sensor; esta lectura en escala puede transformarse en magnitudes físicas mediante fórmulas de conversión que cada fabricante establece así como la sensibilidad de sus sensores, entonces la función de las entradas análogas es la conexión de sensores a las RTU, para captar medidas de magnitudes físicas, claro está, la compatibilidad de un sensor a una entrada análoga depende de si ambos pueden tener la misma escala de sensibilidad y trabajar bajo los mismos rangos de voltaje, por ejemplo, si una entrada análoga soporta voltajes entre 0 y 5 voltios, y un sensor reporta entre 0 y 12 Voltios estos son incompatibles; por otra parte si un sensor reporta una sensibilidad de 12 bit, es decir 4096 posiciones, pero si la entrada análoga no puede detectar tal precisión también se hacen incompatibles.

Todo lo anterior para explicar la estructura del argumento para este paquete, en donde se envía un JSON Array compuesto de JSON Objects cada uno con las llaves “index” asignada con un número entero, que indica que entrada es la que reporta el estado, “d_value” asignada con un número entero que indica el valor en escala numérica que reportó el sensor, “m_value” asignada con un número decimal de coma flotante, que representa una magnitud física obtenida mediante una fórmula sobre el valor numérico reportado por el sensor, “units” asignada con un JSON string, que representa las unidades de la magnitud reportada, estas unidades se encuentran preestablecidas en el sistema, por tanto las abreviaturas deben concordar con la medida; el usuario podrá definir nuevas unidades de medida en cualquier momento, pero la definición debe hacerse anterior a la puesta en marcha de envíos de estas nuevas unidades ya que si el servidor no logra reconocer la abreviatura de las unidades, no almacenará la información reportada por la RTU y además la conexión será cerrada inmediatamente.

Tabla 3. Unidades predefinidas.

Símbolo	Unidad
C	Grados Centígrados
F	Grados Fahrenheit
cm	Centímetros
M	Metros
db	Decibeles
%	Porcentaje

{

“cmd”:4,

“arg”:{

“inputs”: [

{

“index”: ***integer_number,***

“d_value”: ***integer_number,***

“m_value”: ***float_number,***

“units”: ***JSON_String***

},

{

“index”: ***integer_number,***

“d_value”: ***integer_number,***

“m_value”: ***float_number,***

“units”: ***JSON_String***

},...,

{

“index”: ***integer_number,***

“d_value”: ***integer_number,***

“m_value”: ***float_number,***

“units”: ***JSON_String***

}] } }

Comando 5: Transmisión de salidas análogas

Este comando puede ser usado en ambas vías, ya sea emitido por el servidor o emitido por la RTU, según el emisor cambia la intención, si es emitido por el servidor implica la orden de establecer el estado indicado en “arg” en las salidas análogas indicadas por los índices, en este caso el servidor transmite las llaves “Index” y “d_value” para que la RTU fácilmente pueda identificar la salida análoga que se desea establecer con un valor diferente, el servidor no transmite llave “units” ya que las unidades no se necesitan para ejecutar dicha acción, el servidor tampoco transmite la llave “m_value” ya que esto implicaría una doble acción por parte de la RTU, debería descifrar que valor digital corresponde al valor de magnitud, y aplicar el cambio en la salida, ese es el funcionamiento por defecto, no obstante esta funcionalidad podría ser implementada en futuras versiones; por su parte si el comando es emitido por la RTU representa el estado actual de las salidas análogas, en este caso todas las llaves serán transmitidas. Este comando sigue los mismos lineamientos que el comando 4 (ver comando 4).

{

“cmd”:5,

“arg”:{

“outputs”: [

{

“index”: ***integer_number,***

“d_value”: ***integer_number,***

“m_value”: ***float_number,***

“units”: ***JSON_String***

},

{

“index”: ***integer_number,***

“d_value”: ***integer_number,***

“m_value”: ***float_number,***

“units”: ***JSON_String***

},...,

{

“index”: ***integer_number,***

“d_value”: ***integer_number,***

“m_value”: ***float_number,***

“units”: ***JSON_String***

}

]

}

}

<u>Comando 6: Transmisión de texto plano o JSON</u>

Este comando es incluido como una medida para dejar abierta la posibilidad de que las RTU no solo transmitan magnitudes físicas, en caso de que las RTU transmitan variables, nombres, o cualquier otra estructura de datos diferente a las que se contemplan como posibles de recolectar y transmitir mediante salidas y entradas digitales o análogas, por ejemplo una RTU podría transmitir la identidad de un empleado que recientemente accedió a un espacio restringido usando como llave su biometría, se aplican dos llaves “is_json” asignada como un 'true' o 'false' dependiendo si este texto que se transmite esta formateado a la sintaxis de JSON o no, y “data” asignada a un JSON_String, en este punto es necesario aclarar que un JSON_String es diferente al tipo String en la mayoría de los lenguajes de programación en el sentido del límite de caracteres, en la mayoría de lenguajes de programación el tipo string se limita a 255 caracteres, no obstante un JSON_String en teoría podría ser infinito, esto es necesario tenerlo en cuanta al momento de interpretar los paquetes, este comando puede ir en ambas vías dado que tanto la RTU puede reportar texto, como el servidor enviar texto a la RTU para que esta lo interprete y ejecute acciones ajenas a lo relacionado a las entradas y salidas, por ejemplo mostrar cierta información en una pantalla.

{

“cmd”:6,

“arg” :{

“is_json”: ***boolean,***

“data”: ***JSON_String***

}

}

<u>Comando 7: Transmisión de variables del RTU</u>

Cada RTU debe contar con ciertas variables predefinidas, mediante este comando la RTU puede transmitir los valores de estas variables o el servidor puede indicarle al RTU que establezca sus variables a nuevos valores, las variables obligatorias con las que cada RTU debe contar son las siguientes:

- **rtu_owner:** Nombre de la persona o entidad la cual se considera que es el propietario de la RTU.
- **contact_name:** Nombre de la persona encargada de administrar el dispositivo, generalmente se entenderá que esta persona es quien tiene acceso al software, hardware y firmware de la RTU.
- **contact_phone:** Número telefónico del administrador del dispositivo.
- **contact_email:** Dirección de correo electrónico de la persona encargada de la administración del dispositivo.
- **rtu_name:** Nombre que identifica de manera particular al dispositivo, debería servir a manera de mnemotecnia para su administrador referirse a la RTU.
- **rtu_serial:** Numero serial que se asigna a la RTU para identificarla de manera única en el sistema.
- **rtu_lat:** latitud de las coordenadas geográficas que representan la ubicación del dispositivo sobre el globo terráqueo.
- **rtu_lon:** longitud de las coordenadas geográficas que representan la ubicación del dispositivo sobre el globo terráqueo.

- **rtu_model:** Número del modelo o identificación del micro-controlador o board que se emplea como RTU.
- **rtu_manufacturer:** Se refiere al nombre de la empresa que fabricó el micro-controlador o board que se emplea como RTU.
- **firmware_version:** Versión del firmware instalado en el micro-controlador programable.
- **power_supply:** Tipo de conexión que usa la RTU para abastecer la energía, generalmente se indicará el voltaje y amperaje, en algunos casos también el tipo de conector.
- **data_link:** Tipo de conexión que usa la RTU para acceder a Internet, generalmente será una conexión de área local (LAN), con tecnologías como ADSL, DSL o Cable Coaxial, pero podría ser WiFi, 3G GSM, 4G LTE, o Bluetooth según sea el caso.
- **report_interval:** Es un tiempo en segundos que indica cada cuanto es necesario enviar un reporte completo del estado actual de las entradas y salidas con las que cuenta la RTU (comandos 2,3,4,5).

La sintaxis del comando es la siguiente:

{

“cmd”:7,

“arg” :{

"rtu_owner": ***JSON_String,***

"contact_phone": ***JSON_String,***

"contact_email": ***JSON_String,***

"rtu_serial": ***JSON_String,***

"rtu_lat": ***JSON_String,***

"rtu_lon": ***JSON_String,***

"rtu_model": ***JSON_String,***

"rtu_manufacturer": ***JSON_String,***

"firmware_version": ***JSON_String,***

"power_supply": ***JSON_String,***

"data_link": ***JSON_String,***

"report_interval": ***JSON_String***

}

}

<u>Comando 8: Transmisión de distribución de pines</u>

Además de las variables involucradas en el anterior comando, cada RTU debe contar con la distribución de sus pines en 4 variables, “analog_input”, “analog_output”, “digital_input”, “digital_output”, tal como su nombre lo indica, estas variables permiten saber que pin corresponde a que función de las 4 presentadas, el formato que se espera que esta distribución sea transmitida es como un JSON_Array por cada variable, es así como por ejemplo, si la variable fuera “digital_input”:[12,14,21,22] el servidor podría saber que esa RTU cuenta con 4 entradas digitales y que los pines 12, 14, 21 y 22 están asignados a estas, también es así como cuando el servidor cuando recibe un comando 2 que reporta algo como esto: “values”:[1,1,2,0], el servidor sabrá que el pin 12 reporta presencia de voltaje, que el pin 14 reporta presencia de voltaje, que el pin 21 no reportó, y que el pin 22 reportó ausencia de voltaje. Los elementos del array deben ser enteros positivos.

{

“cmd”:8,

“arg” :{ “analog_input”: ***JSON_String,***

“digital_input”: ***JSON_String,***

“analog_output”: ***JSON_String,***

“digital_output”: ***JSON_String***} }

La estructura del string debe ser la siguiente: ***“[Integer, Integer,..., Integer]”***

Comando 9: Transmisión de código de alerta

La transmisión de alertas es una característica esencial en un sistema de telemetría, una alerta es un reporte de un evento inesperado. Las alertas, a diferencia de un reporte de rutina implican la toma de acciones urgentes, un reporte de alerta se compone de 3 llaves: “code” o código de alerta asignada con el código preestablecido para la alerta como un JSON_String, “details” o detalles, asignada con un JSON string que se utiliza para enviar información adicional y “is_json” como variable indicando si el valor relacionado en los detalles mantiene o no el formato JSON.

{

“cmd”:9,

“arg”:{

“code”: ***JSON_String,***

“details”: ***JSON_String***,

“is_json”: ***Boolean***

}

}

Comando 10: Solicitudes

Las solicitudes son comandos cuya acción recíproca es transmitir diversa información, en este caso se contemplan 6 solicitudes por defecto para el protocolo, pero dado que estas están pensadas mediante un sistema de códigos, queda abierta la posibilidad de implementar nuevas solicitudes, lo que convierte a este comando en una puerta para la implementación de nuevos comandos. Las solicitudes tienen dos llaves: “code” o código de la solicitud que asigna a un JSON_String, y “params” o parámetros de la solicitud, que se asigna a un JSON_Array, es por esto que las variantes de las solicitudes dependen del código de esta; la llave “params” es opcional en todos los casos. Sintaxis general:

{

“cmd”:10,

“arg” :{

“code”: ***JSON_String,***

“params”: ***JSON_Array***,

}

}

Los posibles code predeterminados son:

- DI: para solicitar el estado actual de las entradas digitales, en el caso de esta solicitud el JSON_Array se espera que sea vacío {} o que no sea transmitida la llave “params”.
- DO: para solicitar el estado actual de las salidas digitales, al igual que en el caso de las entradas digitales se espera un JSON_Array vacío o que no sea transmitida la llave “params”.
- AI: para solicitar el estado actual de las entradas análogas, en esta variante si la llave “params” es transmitida se espera un arreglo de índices de entradas análogas de las cuales se desea transmitir su estado, ejemplo: [0, 3,5] en este caso el índice se refiere a la configuración lógica de las entradas, no al número del pin físico en la RTU.
- AO: para solicitar el estado actual de las salidas análogas, el valor de la llave “parms” al igual que en la variante anterior podría ser un vector con los índices de las salidas análogas que se desea sean transmitidas al servidor central.
- VARS: para solicitar la transmisión de las variables de la RTU (ver comando 7), la llave “params” puede ser transmitida con un arreglo de los nombres de las variables que se desea que sean transmitidas al servidor central, ejemplo: [“rtu_lat”, “rtu_lon”, “rtu_elv”], para este valor en la llave “params” se transmitirían solo los valores para las variables especificadas.
- PIND: para solicitar la transmisión de la distribución lógica de los pines físicos de la RTU (ver comando 8), el valor de la llave “params” puede contener un arreglo de cual distribución se desea que sea transmitida a el servidor central, ej.: [“alanog_input”, analog_output”] lo que resultaría en la transmisión de únicamente las distribuciones especificadas.

Comando 11: Reconocimiento del saludo

Este comando es la respuesta que el servidor envía a un cliente para el comando de saludo (ver comando 1), en este comando se le informa al cliente si es aceptado para seguir al paso de autentificación o si es rechazado y por tanto se espera que el servidor cierre el canal de comunicación; para una respuesta positiva la llave “response” es asignada a un numero 1, para una respuesta negativa la llave “response” se asigna al número 0.

{

“cmd”:11,

“arg”:{

“response”: ***boolean***

}

}

Comando 12: Reconocimiento de la Autenticación

Este comando es la respuesta que el servidor ofrece al comando de autenticación (ver comando 13), este comando tiene dos finalidades, la primera es dar a conocer al cliente que su llave de conexión si es válida, la segunda es enviar una nueva llave de conexión al cliente, se usa como modo de implementación de un mecanismo de asignación de llaves de conexión automatizado, donde según el serial del dispositivo, el servidor puede decidir si enviar una nueva llave de conexión al RTU o simplemente aceptar la actual. Consta de dos llaves: “response” que puede asignarse con un 1 para una respuesta positiva a la intención de autenticación o con un 0 para una repuesta negativa a la intención de autenticación, y la llave “new_key” que de estar presente se tratará de la nueva llave de conexión asignada al RTU.

{

“cmd”:12,

“arg” :{

“response”: ***boolean***

“new_key”***: JSON_String***

} }

<u>Comando 13: Autenticación</u>

Este comando es el segundo paso en el proceso de control de acceso a los recursos del servidor, una vez que el dispositivo logre superar la etapa de reconocimiento del serial, deberá emitir este comando como segunda interacción con el servidor; si este no es el segundo comando emitido, el servidor deberá enviar un código de error al RTU mediante el comando 0 e inmediatamente cerrar la conexión; este comando solo cuenta con la llave “key” que es asignada a un JSON_String que se trata como la llave de conexión del RTU y el servidor deberá confirmar si esta es válida o no y retornar una respuesta a la RTU mediante el comando 12(ver comando 12).

{

“cmd”:13,

“arg” :{

“key”***: JSON_String***

}

}

<u>Comando 14: Disparador de otros comandos (Para atención de usuarios)</u>

Este comando está orientado a extender la función del servidor para soportar la atención a usuarios, ya que son los usuarios los que tienen la capacidad de disparar el envió de ciertos comandos hacia las RTU(comandos 0, 3, 5, 6, 7 y 10), por ejemplo, el comando 10 (Solicitudes) es el que permite al usuario enviar solicitudes para que las RTU respondan y el usuario termine enterándose mediante los cambios registrados por el servidor en la base de datos de eso que el solicito, también cuando se quiere establecer cierta salida ya sea digital o análoga a determinado valor(comandos 3 y 5), esta es una función que el usuario debe desencadenar, es importante asegurar que el servidor sea capaz de autenticar las credenciales del usuario(alias y frase de seguridad) y que una vez se establezca su validez, al usuario se le permita disparar el comando que se envió como parte del argumento; para tal efecto el usuario debe conocer la estructura de dichos comandos y conocer la identidad de la RTU a la cual desea enviar alguno de los

comandos anteriores. En resumen el comando 14 se conforma de tres partes: la primera se trata de las credenciales del usuario, la segunda consta de un identificador de la RTU a la cual desea enviar una comando y finalmente la estructura completa del comando que desea enviar, así:

{

“cmd”:14,

“arg”:{

“user”: {

“login_name”: ***JSON_String,***

“passphrase”: ***JSON_String***

},

“rtu_id”: Integer,

“trigger_cmd”: JSON_object

}

}

RTU_id hace referencia al campo llave primaria del dispositivo en la base de y datos y debe ser un número entero.

El json object asignado a la clave “trigger_cmd” debe seguir la misma estructura general de otros comandos, por ejemplo un JSON Object válido seria el siguiente:

{

“cmd”:0,

“arg”:{

“msg”: JSON_String

} }

5. DESARROLLO DEL PROYECTO

En este capítulo se aborda el desarrollo del proyecto en cada una de las fases propuestas por la metodología FDD, explicadas en el capítulo anterior.

Para empezar se hace referencia al diseño del modelo general, los conceptos tenidos en cuenta en este punto y el resultado propuesto. Posteriormente se relaciona el listado de características, que como se indicó anteriormente, será el centro del desarrollo del prototipo. A continuación se podrá visualizar el plan general asignado a cada característica identificada. Finalmente se presenta el diseño y la implementación realizada de las características de que dan como resultado el prototipo propuesto para este proyecto.

En cada proceso se presentan diversas tareas propuestas por la metodología para lograr el o los criterios de salida mencionados para cada una en el capítulo anterior.

5.1 Desarrollo del Modelo General

5.1.1 Formación del equipo de modelado

A continuación se muestra la distribución de roles en el equipo:

Tabla 4. Distribución de roles.

Rol	Responsable	Función
Programador Líder	Carlos Contreras	Desarrollador con experiencia en diversos proyectos, concluye las decisiones finales sobre implementación.
Programadores/Modeladores	Carlos Contreras, Paula Rueda.	Desarrolladores generales de las características del sistema, diseñadores del modelo general y demás modelos individuales presentados, serán también los propietarios asignados de las clases en el proceso 3.
Experto de Dominio	Jaime Rueda	Miembro con alto conocimiento y dominio sobre el tema, puede contextualizar y explicar a los integrantes del equipo sobre el propósito del sistema.
Arquitecto Líder/Director del Proyecto	Fernando Rojas	Guía principal y aprobador en las sesiones de diseño del sistema.

5.1.2 Desarrollo del tutorial del dominio

Esta guía o tutorial para establecer el dominio del sistema es proporcionada por el experto de dominio, para este caso específico contamos con una breve descripción que contextualiza al equipo sobre el alcance, necesidades y funcionalidades esperadas.

La guía proporcionada por el experto para establecer el dominio del sistema está expresada de manera explícita en el capítulo correspondiente a la descripción del proyecto.

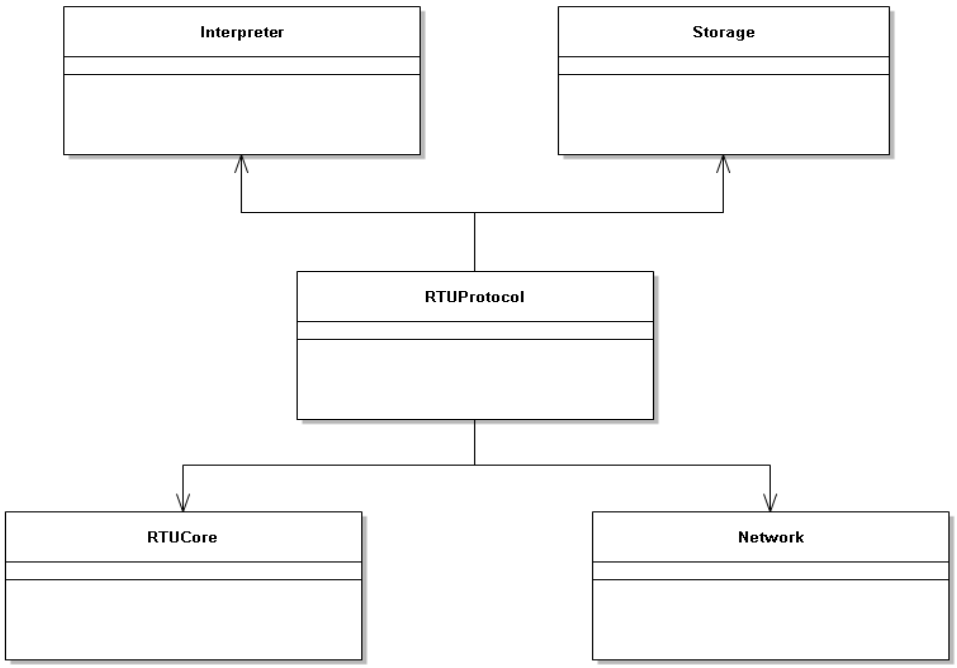
5.1.3 Documentos de estudio

La documentación de interés para el desarrollo del prototipo propuesto en este proyecto trata principalmente sobre la arquitectura cliente servidor, sistemas embebidos (definición y composición), protocolo TCP/IP y los protocolos relacionados y/o similares al protocolo RTUP (planteado por los autores en el capítulo 4), toda se encuentra debidamente especificada en el capítulo 2 de este documento.

5.1.4 Desarrollo del modelo

Se propusieron diversas opciones como modelos iniciales, la que presentó mayor apoyo por parte de los integrantes del equipo fue:

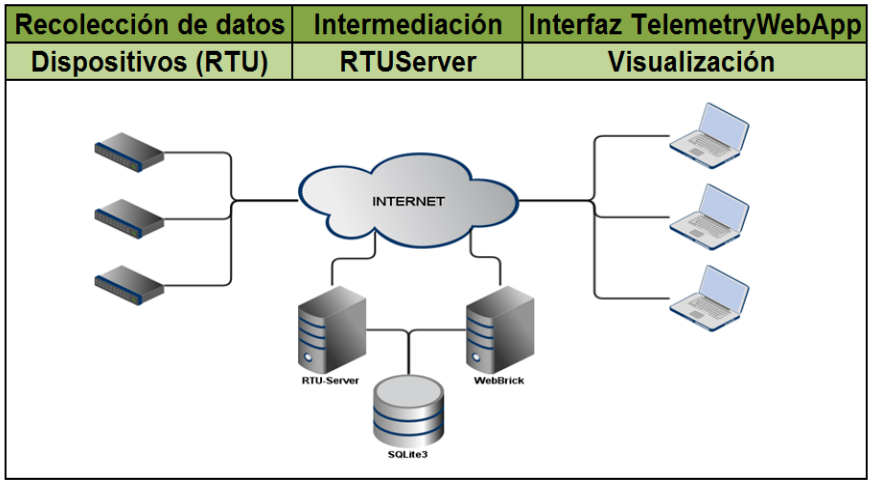
Figura 16. Overal Model Inicial.



5.1.5 Revisión y corrección del modelo

Esta tarea se aplica en los momentos en que luego de creado el modelo general inicial, se consideró necesario agregar y/o eliminar aspectos que no fueron adecuadamente expresados, además se plasmó a manera de gráfico la arquitectura general del sistema propuesto, cuya versión final luego de correcciones se plasma a continuación y en la siguiente página se visualizan las correcciones finales realizadas al modelo general de clases, de manera que se puede comparar con el modelo propuesto inicialmente.

Figura 17. Arquitectura general del sistema.



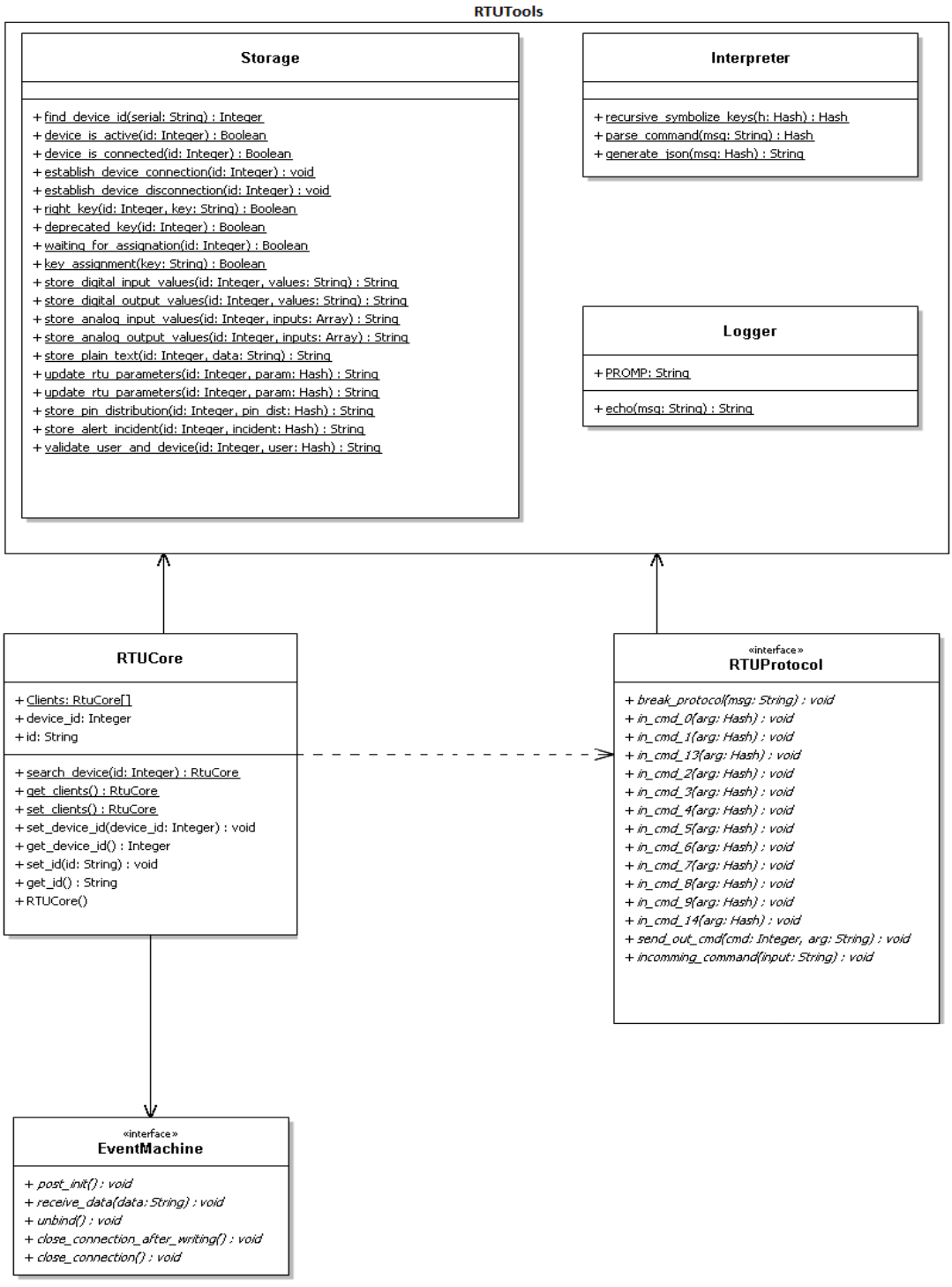
5.1.6 Escritura de apuntes del modelo

Notas sobre los conceptos tenidos en cuenta para la construcción del modelo general, de manera que se puedan conservar como una referencia de este proceso de la metodología. Una de las actividades realizadas para iniciar la construcción del modelo fue una lista inicial de conceptos involucrados en el sistema, de esta manera se fueron identificando aquellos que podrían ser clases principales del sistema, quizás atributos o simplemente características importantes en el prototipo más no literalmente plasmables en el modelo, así:

Tabla 5. Lista inicial de conceptos involucrados en el sistema.

Usuario	Prototipo	Interfaz
Caja Negra	Software	Administradores
Plataforma	Middleware	Módulo
Datos	Registro	Sensor
Servidor	Paquete	Relé
Archivos planos	Telemetría	Alarmas
Dispositivos	Protocolo	Control

Figura 18: Overal Model Final



5.2 Construcción de lista de características

5.2.1 Formación del equipo de creación de la lista de características

Para esta tarea inicial del segundo proceso, la metodología establece que el equipo encargado de crear la lista de características está compuesto por los programadores jefes establecidos en el proceso número 1. De acuerdo a la tabla mostrada en ese proceso, para el presente proyecto el equipo general de desarrollo es reducido ya que está compuesto por los dos autores, por lo que ambos tomaron responsabilidad en la construcción de la lista de características.

Tabla 6. Distribución de roles proceso 2.

Rol	Responsable	Función
Responsable construcción lista de carcterísticas.	Carlos Contreras(Programador Líder). Paula Rueda.	Desarrollo del proceso 2 descrito en la metodología, arrojando como salida final la lista de características del proyecto en curso.

5.2.2 Construcción de la lista de características

Para esta tarea la metodología establece que se debe descomponer de manear funcional el dominio establecido por los expertos en el primer proceso. En primer lugar se descompone en áreas temáticas que están compuestas por actividades de negocio que a su vez están compuestas por características que representarán un paso a realizar en cada actividad.

Subject area 1: *Formular y diseñar el Protocolo.*

Business Activity 1.1: Realizar la formulación conceptual

Feature 1.1.1: Investigar documentación relacionada.

Feature 1.1.2: Definir requisitos y características del protocolo.

Business Activity 1.2: Diseño de comandos

Feature 1.2.1: Definir la funcionalidad de los comandos.

Feature 1.2.2: Definir la estructura, sintaxis y sentido de cada comando.

Subject area 2: *Implementar componente de Conexión.*

Business Activity 2.1: Definir el patrón que se utilizará para la conectividad del prototipo (actividades relacionadas con las capas inferiores a la capa de aplicación TCP/IP).

Feature 2.1.1: Investigar documentación relacionada.

Feature 2.1.2: Seleccionar la opción que presente mejores beneficios para implementación del prototipo propuesto.

Business Activity 2.2: Implementar la opción seleccionada en la característica anterior.

Feature 2.2.1: Incluir la(s) librería(s) e implementar los métodos necesarios para la finalidad propuesta.

Subject area 3: *Implementar componente de Almacenamiento.*

Business Activity 3.1: Elegir la herramienta que se utilizará para el manejo de los datos en el prototipo.

Feature 3.1.1: Investigar documentación relacionada.

Feature 3.1.2: Seleccionar la opción que presente mejores beneficios para implementación del prototipo propuesto.

Business Activity 3.2: Crear el modelo relacional.

Feature 3.2.1: Diseñar de la estructura de tablas y relaciones.

Business Activity 3.3: Implementar el componente que realice las operaciones CRUD (Create, Read, Update, Delete).

Feature 3.3.1: Implementar el mecanismo para realizar la operación de lectura de registros de la base de datos.

Feature 3.3.2: Implementar el mecanismo para realizar la operación de escritura de registros de la base de datos.

Feature 3.3.3: Implementar el mecanismo para realizar la operación de sobre escritura o actualización de registros de la base de datos.

Feature 3.3.4: Implementar el mecanismo para realizar la operación de eliminación de registros de la base de datos.

Subject area 4: *Construir componente de Visualización.*

Business Activity 4.1: Definir aspectos generales.

Feature 4.1.1: Definir el medio por el cual se dará la interacción entre el usuario y el prototipo.

Feature 4.1.2: Definir las funcionalidades del componente.

Business Activity 4.2: Construir las funcionalidades detectadas.

Feature 4.2.1: Implementar las funcionalidades.

Subject area 5: *Realizar el proceso de validación del prototipo.*

Business Activity 5.1: Realizar pruebas de nivel unitario.

Feature 5.1.1: Definir casos de acuerdo al nivel de prueba relacionado.

Feature 5.1.2: Realizar casos de prueba definidos.

Feature 5.1.3: Documentar los resultados obtenidos.

Business Activity 5.2: Realizar pruebas de nivel de comportamiento.

Feature 5.2.1: Definir casos de acuerdo al nivel de prueba relacionado.

Feature 5.2.2: Realizar casos de prueba definidos.

Feature 5.2.3: Documentar los resultados obtenidos.

5.3 Plan por característica

5.3.1 Formación del equipo de planeación por característica

Para esta tarea inicial del tercer proceso, la metodología establece que el equipo encargado de crear el plan por característica está compuesto por el director del proyecto y los programadores jefes establecidos en el proceso número 1. Al igual que en el proceso inmediatamente anterior, debido a que el equipo del presente proyecto es reducido, ambos tomaron responsabilidad en la construcción del plan por característica.

Tabla 7. Distribución de roles proceso 3.

Rol	Responsable	Función
Responsable planeación por carcterística.	Fernando Rojas. Carlos Contreras(Programador Líder). Paula Rueda.	Desarrollo del proceso 3 descrito en la metodología, arrojando como salida final la planeación por característica del proyecto en curso.

5.3.2 Determinación de la secuencia de desarrollo.

Para esta tarea, la metodología propone asignar una fecha compuesta por mes y año para la finalización de cada conjunto de características. En la tabla 6 se puede visualizar la planeación realizada:

Tabla 8. Secuencia de desarrollo.

Subject Areas	Business Activities	Features	Fecha Terminación
Formular y diseñar el protocolo	Realizar la formulación conceptual	Investigar documentación relacionada	Marzo-2014
		Definir requisitos y características del protocolo	Marzo-2014
	Diseño de comandos	Definir la funcionalidad de los comandos.	Marzo-2014
		Definir la estructura, sintaxis y sentido de cada comando.	Marzo-2014
Implementar componente de Conexión	Definir el patrón que se utilizará para la conectividad del prototipo (actividades relacionadas con las capas inferiores a la capa de aplicación TCP/IP).	Investigar documentación relacionada.	Abril-2014
		Seleccionar la opción que presente mejores beneficios para implementación del prototipo propuesto.	Abril-2014
	Implementar la opción seleccionada en la característica anterior.	Incluir la(s) librería(s) e implementar los métodos necesarios para la finalidad propuesta.	Abril-2014
Implementar componente de Almacenamiento	Elegir la herramienta que se utilizará para el manejo de los datos en el prototipo.	Investigar documentación relacionada.	Mayo-2014
		Seleccionar la opción que presente mejores beneficios para implementación del prototipo propuesto.	Mayo-2014
	Crear el modelo relacional.	Diseñar de la estructura de tablas y relaciones.	Mayo-2014
	Implementar el componente que realice las operaciones CRUD.	Implementar el mecanismo para realizar la operación de lectura de registros de la base de datos.	Mayo-2014
		Implementar el mecanismo para realizar la operación de escritura de registros de la base de datos.	Junio-2014
		Implementar el mecanismo para realizar la operación de sobre escritura o actualización de registros de la base de datos.	Junio-2014
		Implementar el mecanismo para realizar la operación de eliminación de registros de la base de datos.	Junio-2014
Construir componente de Visualización.	Definir aspectos generales.	Definir el medio por el cual se dará la interacción entre el usuario y el prototipo.	Julio-2014
		Definir las funcionalidades del componente.	Julio-2014
	Construir las funcionalidades detectadas.	Implementar las funcionalidades.	Agosto-2014
Realizar el proceso de validación del prototipo	Realizar pruebas de nivel unitario.	Definir casos de acuerdo al nivel de prueba	Septiembre-2014
		Realizar casos de prueba definidos.	Septiembre-2014
		Documentar los resultados obtenidos.	Septiembre-2014
	Realizar pruebas de nivel de comportamiento.	Definir casos de acuerdo al nivel de prueba	Septiembre-2014
		Realizar casos de prueba definidos.	Septiembre-2014
		Documentar los resultados obtenidos.	Septiembre-2014

5.3.3 Asignación de las ‘Business Activities’ a los programadores Jefe.

Según la documentación de la metodología, es aquí en donde se realiza la distribución de las ya listadas ‘Business Activities’ entre los programadores jefe, esto de acuerdo a su complejidad y equilibrio de las responsabilidades que han tenido en las etapas anteriores.

Debido a la composición reducida de los equipos de diseño, desarrollo y ejecución general de este proyecto, para este paso de la metodología, todas las Business Activities’ listadas en los pasos anteriores fueron asignadas al único programador jefe mencionado.

5.3.4 Asignación de las clases a los desarrolladores.

La metodología establece que para este paso final de la planeación, ahora deben ser distribuidas las clases entre el equipo de desarrollo, contemplando la posibilidad de que un desarrollador sea propietario de más de una clase.

Retomando las razones mencionadas en el paso anterior, todas las clases fueron asignadas a los dos desarrolladores existentes.

5.4 Diseño y Construcción por característica

Estas son las fases finales de la metodología, que se establecen de forma iterativa, de manera que el equipo desarrollador pueda escoger una a una cada característica e iniciar a desarrollarla, construir los diagramas necesarios, iniciar la implementación de clases y métodos correspondientes hasta que se considera esa característica completa y se puede dar continuación a las demás del listado.

A continuación se relaciona el proceso realizado con las principales características del listado realizado para este proyecto:

5.4.3 Formular y diseñar el protocolo.

El diseño y desarrollo de esta característica (formulación conceptual y diseño de comandos) se explica detalladamente en el capítulo 4 de este documento (ver capítulo 4).

5.4.4 Implementar el componente de conexión.

Como se mencionó en el marco teórico de este documento, el patrón o técnica escogida para esta etapa de la implementación fue el llamado ‘Reactor Pattern’ o Patrón Reactor. Se realizó una investigación sobre las diversas implementaciones de este patrón, las principales encontradas son:

- Apache MINA: es un framework que ayuda a los usuarios a desarrollar fácilmente aplicaciones de red de alta escalabilidad y rendimiento. Este framework proporciona una API abstracta, dirigida por eventos y asincrónica [14].
- xSocket: es una librería basada en NIO fácil de usar para la construcción de aplicaciones de red de alto rendimiento y escalabilidad. Esta librería apoya de una forma intuitiva las aplicaciones tanto del lado del cliente como del lado del servidor y encapsula conceptos como la programación NIO de bajo nivel, la administración del conjunto de conexiones y la detección del tiempo de espera de conexión [15].

  xSocket está diseñada para construir componentes cliente servidor de alto desempeño como un servidor SMTP, proxy, o componentes cliente servidor que están basados en un protocolo propio.

- Netty: es un framework NIO de cliente servidor que permite el desarrollo fácil de aplicaciones de red como servidores de cliente, agiliza la programación a nivel de red como TCP y/o sockets UDP [16].

  Netty utiliza como lema 'Rápido y fácil', argumentando que los desarrollos con menos dificultad no son obligatoriamente más propensos a sufrir necesidades de mantenimiento o problemas de desempeño.

- POE (Perl Object Environment): como su nombre lo indica es la implementación del patrón reactor para el lenguaje de desarrollo Perl.

- Twisted: es un motor de red orientado a eventos escrito en el lenguaje Python que opera con licencia de código abierto. Twisted respalda protocolos comunes de red como POP3, IMAP, SSHv2, DNS [17].

---

[14] Apache Software Foundation. «Apache MINA». 2003-2012. [En línea] [Consultado 20/04/2014]. Disponible: http://mina.apache.org/

[15]«Socket». [En línea] [Consultado 20/04/2014]. Disponible: http://xsocket.org/

[16]The Netty Project. «Netty». 2014. [En línea] [Consultado 20/04/2014]. Disponible: http://netty.io/

[17] Twisted Matrix Labs. «Twisted». 2014. [En línea] [Consultado 20/04/2014]. Disponible: https://twistedmatrix.com/trac/

- Event Machine: es una librería de procesamiento por eventos sencilla y rápida para programas realizados en el lenguaje Ruby y proporciona entradas y salidas por eventos usando el patrón reactor [18].

  Esta librería está diseñada para ser usada en los entornos más exigente de producción, que implican alta escalabilidad, rendimiento y estabilidad, además permite a los desarrolladores enfocarse en la lógica de la aplicación que están diseñando y dejar a un lado los problemas relacionados con la red.

  Como es visible, todas las opciones presentadas contemplan beneficios y funcionalidades similares, que podrían ayudarnos a cumplir la idea de utilizar el patrón reactor, sin embargo, teniendo en cuenta que Ruby es un lenguaje de código abierto, inclinación presentada desde el inicio de este prototipo, Event Machine es la opción escogida para implementar el componente de conexión del que se habla en esta característica.

  Para la implementación de esta librería, luego de contar con la instalación de Ruby, la instalación de Event Machine se puede realizar en un paso sencillo con la sentencia gem install eventmachine.

  El núcleo de Event Machine es llamado el Reactor, es allí en donde ocurre todo el procesamiento de la librería, el código que se cree de la aplicación va a enlazarse en el reactor mediante conexiones de socket, temporizadores y otros medios, por lo que a manera de conclusión se puede decir que la aplicación completa está envuelta dentro de un bloque de ejecución de esta librería (EM) que se ejecutará infinitamente hasta que el método 'stop' sea llamado [19].

---

[18]«Event Machine». [En línea] [Consultado 20/04/2014]. Disponible: http://rubyeventmachine.com/

[19] Git Hub Wikis. «Event Machine Introduction». 2014. [En línea] [Consultado 20/04/2014]. Disponible: https://github.com/eventmachine/eventmachine/wiki/General-Introduction

5.4.5 Implementar el componente de almacenamiento.

Como se mencionó en el marco teórico de este documento, el patrón o técnica escogida para esta etapa de la implementación fue el llamado 'Mapeo Relacional de Objetos - ORM'.

La implementación de ORM para el lenguaje Ruby es llamado 'Active Record', es la capa del sistema encargada de la lógica y los datos de la aplicación. Active Record es considerada la M en el Modelo Vista Controlador (MVC) ya que facilita la creación y el uso de objetos cuyos datos necesitan ser almacenados de manera persistente en la base de datos [20].

Active Record es la implementación del patrón que lleva su mismo nombre, que a su vez, es la descripción del sistema de mapeo relacional de objetos (ORM) mencionado anteriormente, este sistema tiene como teoría que el aseguramiento de la lógica de que el acceso de los datos hace parte del objeto mismo, educará a los usuarios de cada objeto en cómo escribir y leer o asignar y obtener desde la base de datos; podríamos decir entonces que Active Record es un librería de implementación de ORM.

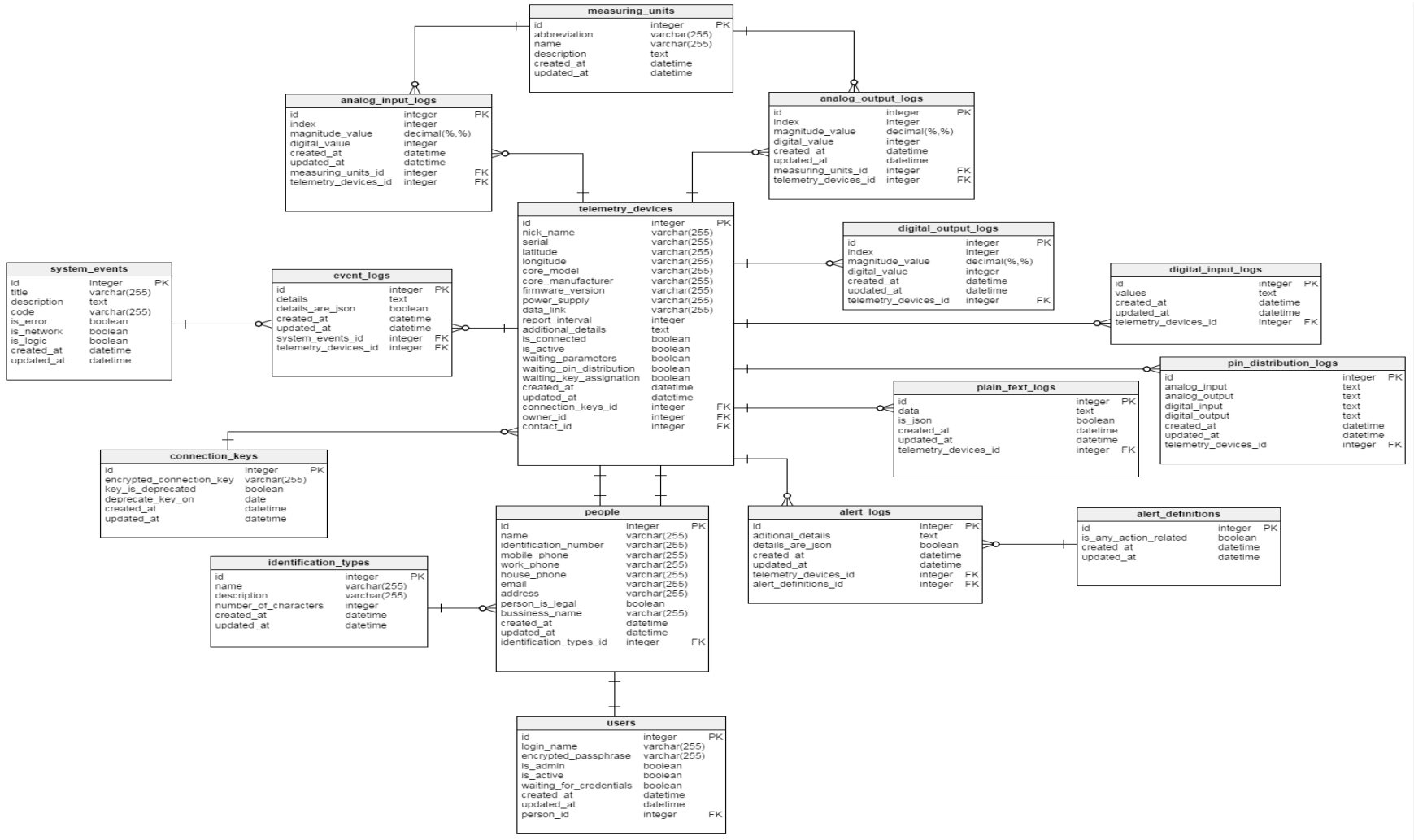
Este marco de trabajo proporciona la capacidad de representar modelos con sus datos, así como también las asociaciones y jerarquías de herencia entre los mismos, además y como características importantes aprovechadas en gran magnitud en el desarrollo de este proyecto, permite validar los modelos antes de que se almacenen y realizar las operaciones generales de bases de datos de una manera orientada a objetos, razón por la cual aunque este componente se refiere a almacenamiento y base de datos, se menciona constantemente el término objetos.

Continuando ahora con la característica asociada a la creación del modelo relacional, para realizarlo se utilizó la herramienta web 'Vertabelo', a continuación se muestra el modelo:

---

[20] Ruby on Rails Guides. «Active Record Basics». [En línea] [Consultado 21/06/2014]. Disponible: http://guides.rubyonrails.org/active_record_basics.html#what-is-active-record-questionmark

Figura 19. Diagrama Entidad- Relación.



En cuanto a las operaciones CRUD (Create, Read, Update, Delete), Active Record crea de manera automática los métodos que permiten leer y manipular los datos almacenados en sus tablas [21]:

- Crear (C): Los objetos mediante active records pueden ser creados mediante un hash, un bloque o tener sus atributos establecidos manualmente luego de su creación. El nuevo método ('crear') retornará un nuevo objeto mientras lo guarda en la base de datos.
- Leer (R): Active Record ofrece una API para poder acceder a los datos que están almacenados en la base de datos.
- Actualizar (U): Para el proceso de actualización, se realiza la búsqueda del objeto que se quiere modificar, se realizan las modificaciones que se quieren a sus atributos y se guardan nuevamente en la base de datos.
- Eliminar (D): Para el proceso de eliminación, debe realizarse en primero lugar y al igual que en el anterior proceso, la búsqueda del objeto que desea eliminarse, se destruye a través de un método propio llamado 'destroy' que lo elimina automáticamente de la base de datos.

Como se mencionó anteriormente, Active Records brinda una función realmente útil al momento de almacenar registros en la base de datos, nos referimos puntualmente a las validaciones, estás permiten garantizar que los datos que serán almacenados son totalmente válidos. Estas pueden realizarse mediante métodos predefinidos que ayudan a corroborar los datos, por ejemplo 'uniqueness' método que como su nombre lo indica verifica que el registro sea único, 'numericality' método que verifica que los registros sólo sean numéricos, 'length' método que verifica que la longitud del registro está dentro de los parámetros especificados (mínimo, máximo, entre un rango), 'presence' método que verifica que determinados registros o atributos no se encuentran vacíos, 'absence' método que verifica que los atributos señalados se encuentran vaciós, etc. Para nuestro caso puntual, se realizaron validaciones como los valores 0 y 1 en entradas digitales, formato JSON en los comandos enviados por la RTU, cantidad de valores de entradas reportados con respecto a la cantidad de entradas habilitadas para esa RTU, etc.

---

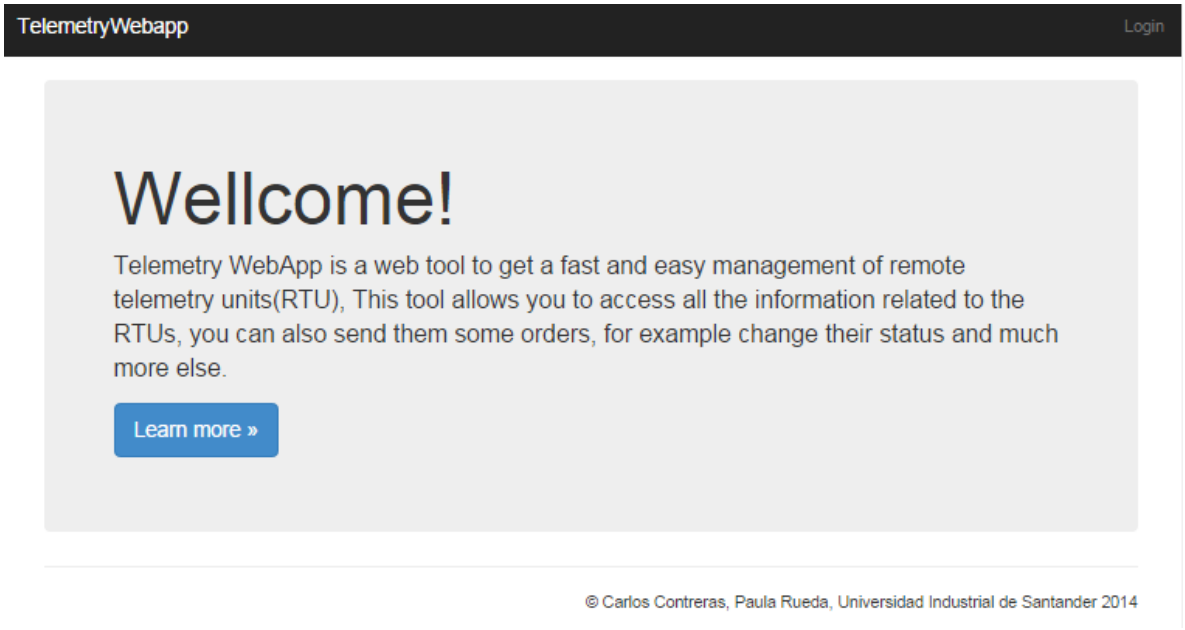
[21] Ruby on Rails Guides. «CRUD – Reading and Writing Data». [En línea] [Consultado 22/06/2014]. Disponible: http://guides.rubyonrails.org/active_record_basics.html#crud-reading-and-writing-data

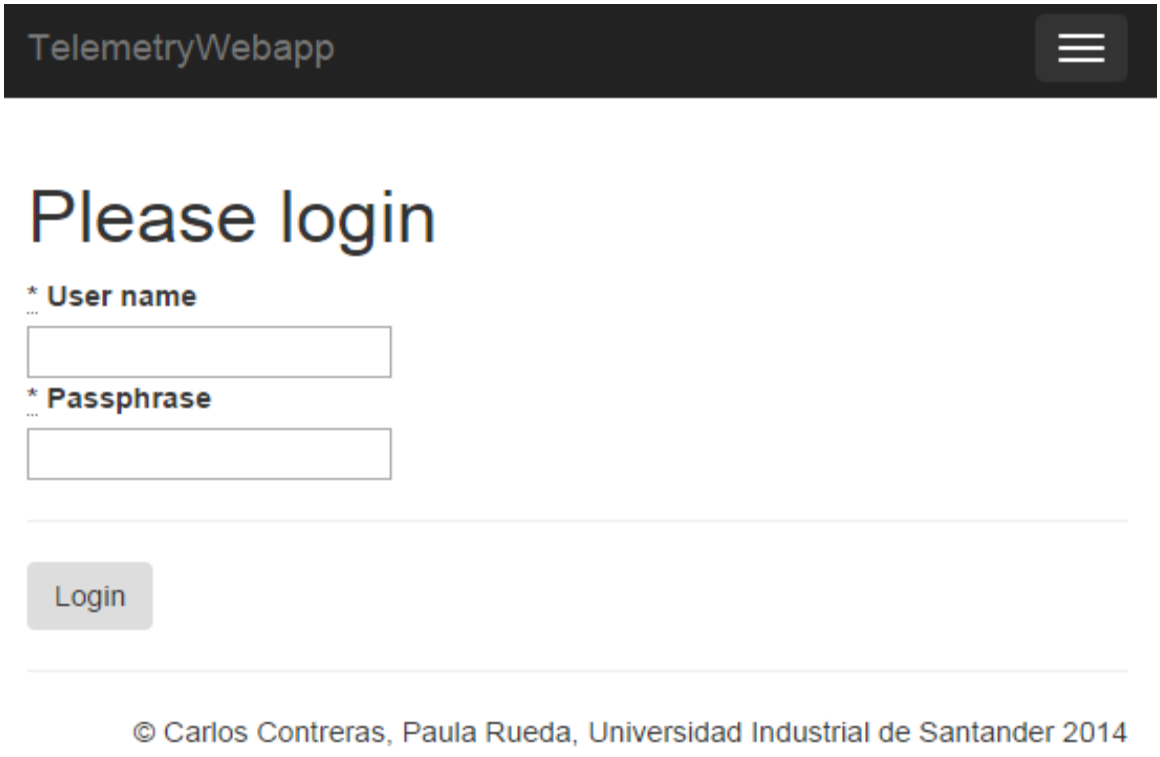
5.4.6 Construir el componente de visualización.

Las RTU y los Usuarios del sistema están relacionados en dos niveles, el primer nivel consta de una relación de “propiedad”, en la cual cada RTU es propiedad de un único dueño (Usuario), y la segunda es mediante la relación de “contacto”, en esta a cada RTU se le asigna una persona como persona de contacto (Usuario), estas relaciones son importantes dado que el servidor controla que solo los usuarios autorizados interaccionen con las RTU correspondientes, los usuarios pueden cuentan un rango de posibilidades tanto para observar el comportamiento de las RTU como para enviar órdenes a las mismas, esta interacción se da mediante una interfaz web, es decir, como complemento del servidor, también se proporciona una aplicación web sencilla que sirve como interfaz de acceso a las funcionalidades del sistema, esta interfaz representa una manera cómoda de realizar la visualización de información, así como la ejecución de determinadas acciones.

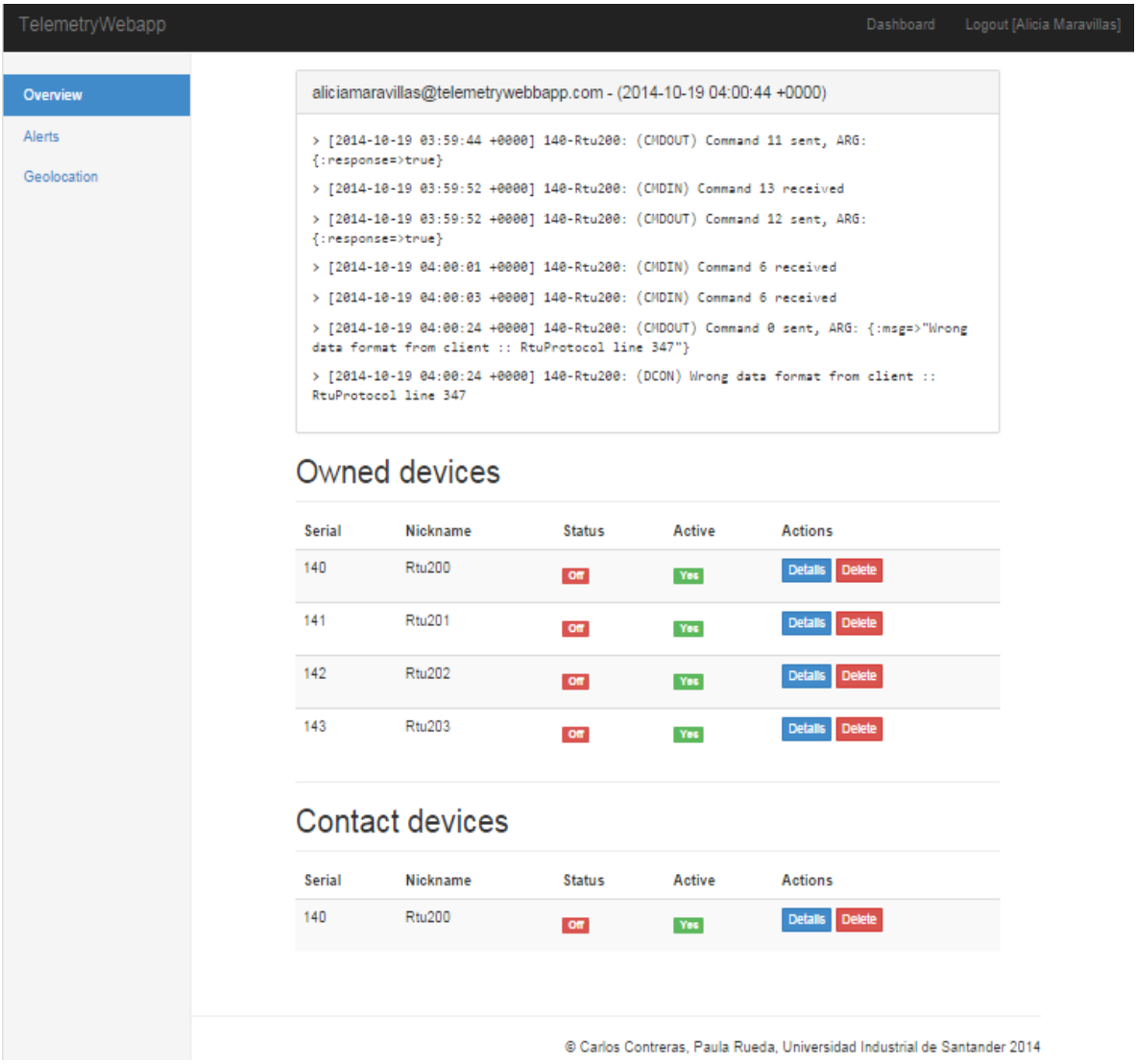
El servidor web utilizado fue WebBrick, herramienta de servidor web integrada por defecto en el marco de trabajo utilizado Rails y como motor de base de datos se utilizó SQLite, sin embargo, gracias a la herramienta Active Records (descrita anteriormente) el funcionamiento se da independiente al motor de base de datos que se use para la ejecución del prototipo propuesto en este proyecto.

La página de inicio o “homepage” del usuario cuenta con un mensaje inicial de bienvenida y la opción de iniciar sesión, así:

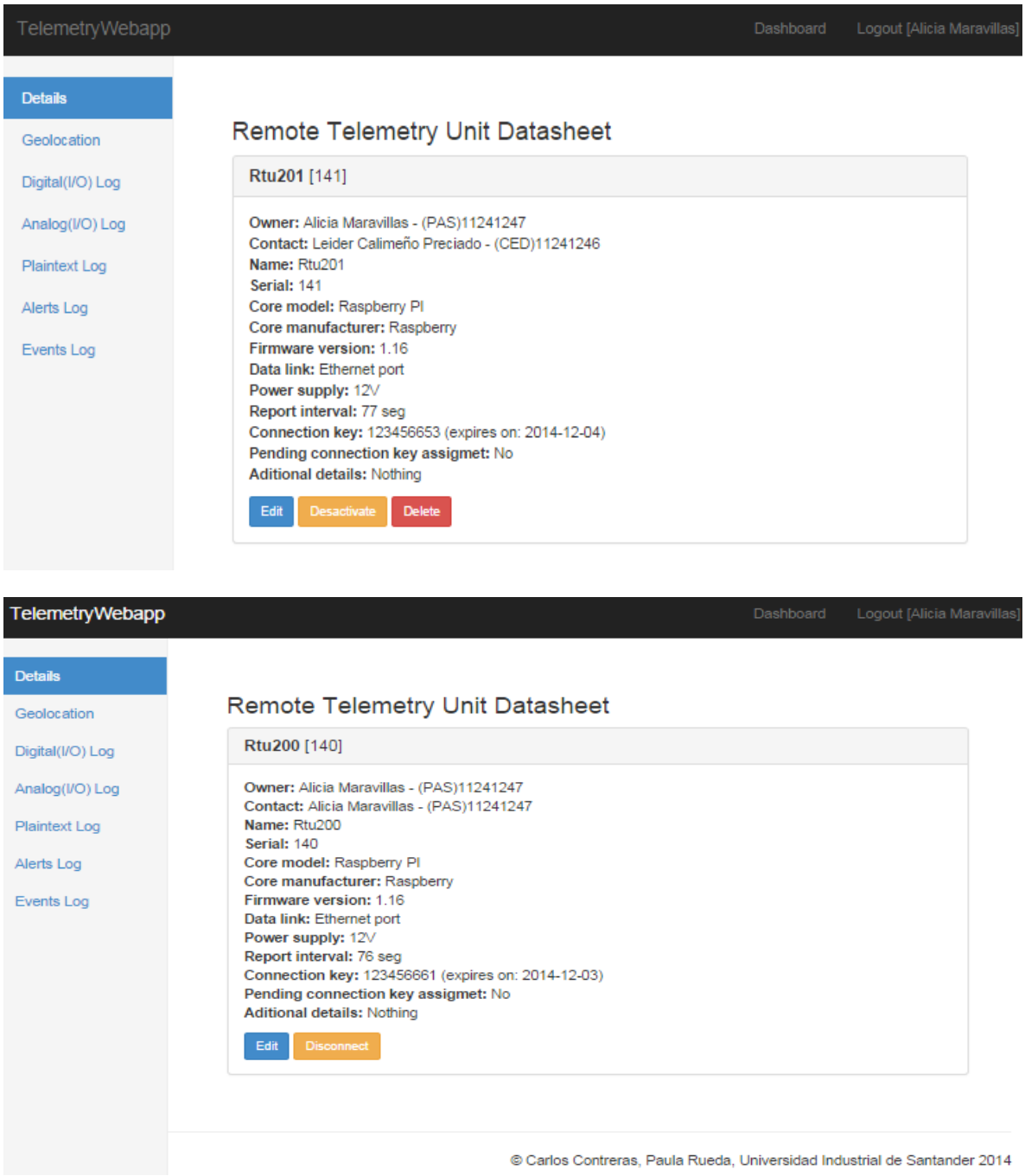




Luego de insertados los datos de ingreso, la aplicación muestra diversas opciones a cada usuario dependiendo del tipo. Si el rol es propietario o contacto, se visualizarán tres opciones en el menú izquierdo: Overview que en primer lugar muestra los últimos eventos relacionados con la o las RTU correspondientes a ese usuario, luego muestra la relación de las RTU de las cuales es propietario y contacto, cada una representada con colores verde y rojo su estado de conexión (conectada o desconectada), de actividad (activa o inactiva) y por último dos acciones que puede tomar: ver detalles y eliminar la RTU, así:

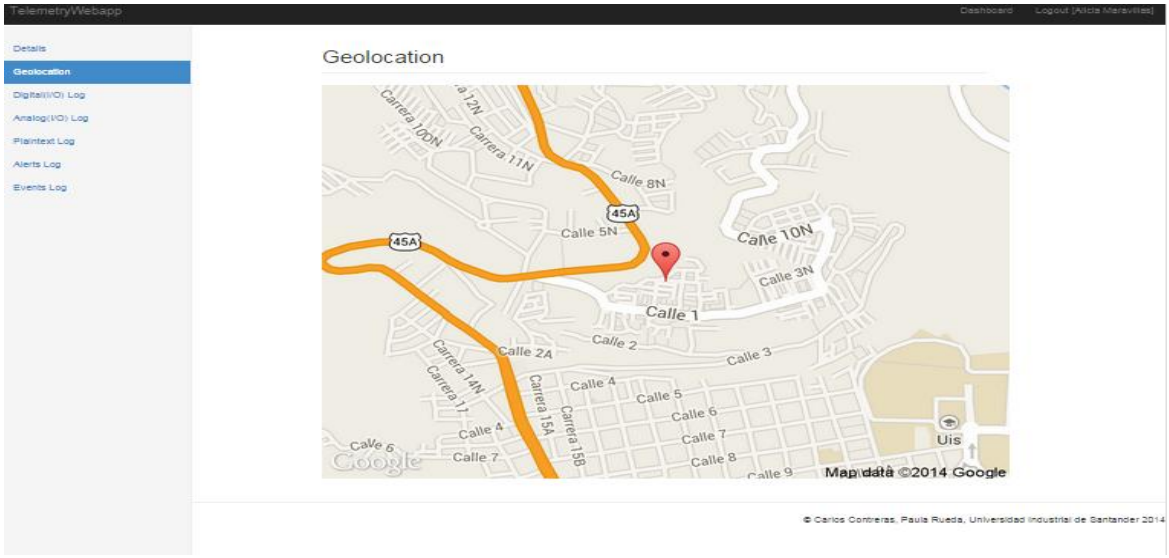


En la acción detalles se puede visualizar la ficha técnica de la RTU y tres opciones de acciones a realizar: editar información, desactivar y eliminar la RTU. Es importante recalcar que para RTU inactivas la segunda opción será activar, y para RTU conectadas solo estarán las opciones editar y desconectar; además cada vez que se opte por la opción eliminar, la aplicación realizará una pregunta de confirmación debido a que esta acción implica eliminar el registro de la RTU impidiendo a partir de ese momento cualquier intento de conexión. En el menú superior izquierdo el usuario podrá encontrar las opciones de cerrar sesión y dirigirse nuevamente al tablero de control o “DashBoard” como se muestra a continuación:

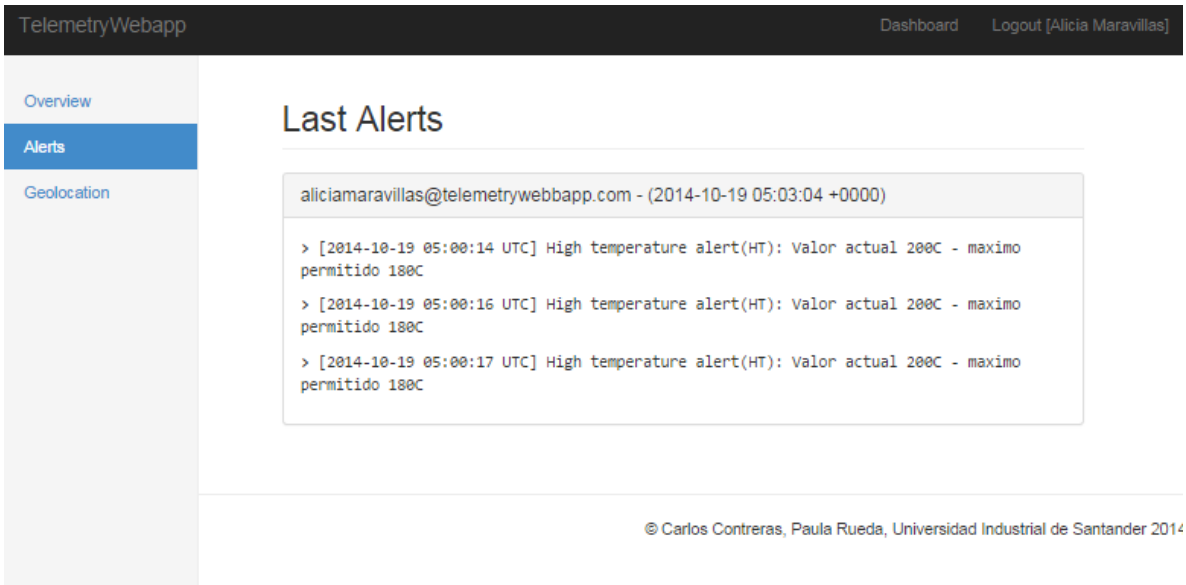


Además de la visualización de la información, la acción “Details” despliega 6 menús adicionales, para un total de 7 en el panel izquierdo, en su mayoría(menús 3 al 7) hacen referencia al “Log” o registros individuales de la RTU como la distribución de entradas salidas análogas y digitales, texto plano y alarmas recibidas, nuevamente los últimos eventos ocurridos en la interacción con esta RTU (esto debido a que los usuarios que tienen más de una RTU, en la página

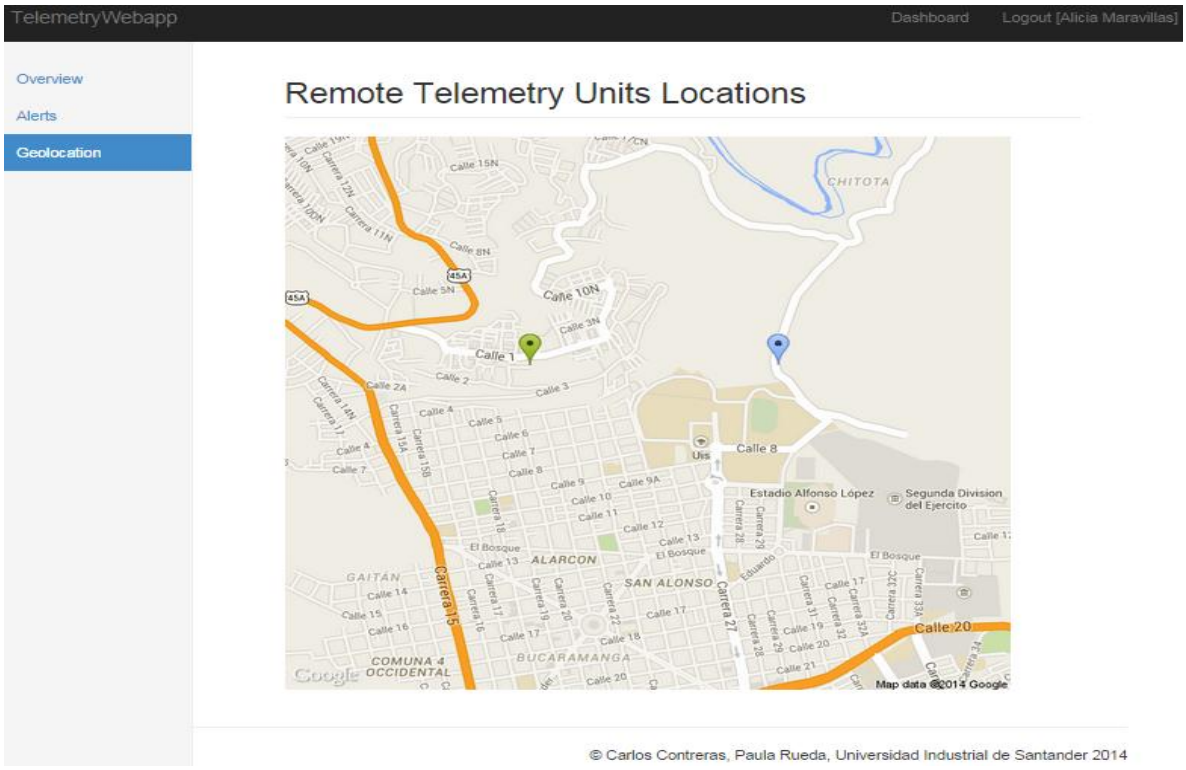
inicial el log general mezcla la interacción ocurrida con todas, por lo tanto en este menú podría visualizarse independiente la de cada una) y en el menú número 2 el usuario puede encontrar la opción de "Geolocation" en donde de manera gráfica mediante un mapa se muestra la localización geográfica de la RTU, así:



Continuando con las opciones generales para los usuarios tipo propietario o contacto, la segunda es "Alerts", hace referencia a las alertas enviadas por la(s) RTU:



En tercer y último lugar se encuentra la opción de “Geolocation” que muestra mediante un mapa la localización geográfica de todas las cajas relacionadas a ese usuario, así:



5.4.7 Realizar el proceso de validación del prototipo.

Esta característica se desarrolla con detalle en el siguiente capítulo denominado ‘Pruebas’.

6. PRUEBAS

6.1 Introducción (Descripción del software al que se le va hacer pruebas)
RTU-Server es un prototipo de software servidor que se ejecuta en la capa de aplicación del modelo del conjunto de protocolos TCP/IP, su implementación se ajusta a la especificación del protocolo “RTUP”, por tanto las funcionalidades de este prototipo software pueden dividirse en dos categorías principales, la primera de ellas es la atención a unidades remotas de telemetría o “Remote Telemetry Units (RTU)” y la segunda es la atención a usuarios del sistema.

En cuanto a la atención a las RTU, el servidor es capaz de autenticar la identidad de estas previamente a la asignación de recursos para la atención completa, esta función se lleva a cabo de acuerdo la implementación del protocolo “RTUP” el cual define 4 pasos ordenados para lograr autenticar una RTU, una vez este proceso se lleva a cabo de manera exitosa, las RTU poden comunicar una variedad de comandos al servidor de los cuales la mayoría son comandos de atención automatizada.

6.2 Supuestos para aplicación de las pruebas
Para las pruebas se contó con una base de datos con registros completos en el sentido de que se asemejó a una base de datos debidamente poblada en un ambiente de producción, para dicho propósito se utilizó el motor de bases de datos “SQLite”, las pruebas se realizaron en una máquina personal y ambiente local con las siguientes características de hardware:

- Computador portátil, Acer Aspire E1-472-6496
- Procesador: Intel core i5 4200U 1.6Ghz – 2.6Ghz (Turbo Boost Tecnology)
- Tarjeta de gráficos: Intel HD Graphics 4400
- Memoria RAM 4GB DDR3
- Sistema Operativo GNU/Linux: Distribución Fedora 20 Codename: Heisenburg.
- Nombre de dominio: localhost
- Puerto de pruebas: '1234' para el prototipo, '3000' para la interfaz web
- Versión de las herramientas: Ruby 2.1.2p95, Rails 4.1.4, Sqlite 3.8.6.

Así mismo se asume que las pruebas unitarias fueron realizadas previamente como parte del proceso de desarrollo, este plan define la etapa de pruebas generales del sistema, este plan se enfoca en corroborar el funcionamiento en conjunto de todas las partes que conforman el sistema, tiene un enfoque funcional

ligado a la metodología de desarrollo que se siguió en el proceso de desarrollo (FDD) o desarrollo orientado por características.

6.3 Listado de herramientas aprobadas para realizar las pruebas funcionales.
Las herramientas que se utilizaron para realizar estas pruebas fueron seleccionadas por la facilidad, comodidad o rapidez que aportan al proceso de realización de las pruebas, a continuación el listado de herramientas.

1) **Telnet**: como herramienta principal para probar el componente de red, telnet abre túneles de comunicación TCP/IP a una determinada dirección IP y puerto y permite enviar texto plano hacia el otro extremo.
2) **Irb**: "Interactive Ruby Shell" o Consola interactiva del lenguaje de programación Ruby, se trata de una herramienta que permite ejecutar líneas o bloques de código directamente desde la consola, y muestra los resultados inmediatamente en consola.
3) **Rails Console**: herramienta proporcionada por el Framework de desarrollo para la web denominado "Ruby on Rails", esta herramienta presenta un acceso fácil y rápido a la base de datos mediante código Ruby, es muy similar a "irb" pero lo rebasa en potencia, ya que cuenta con muchas mas librerías pre-cargadas que aportan mayor poder al momento de su utilización.
4) **Ruby**: Como lenguaje de programación que se empleado para generar cualquier fragmento de código que se requiera para la realización de las pruebas, principalmente en la automatización de pruebas.

6.4 Pruebas funcionales
En condiciones controladas y siguiendo casos predefinidos, al inicio de estas pruebas se esperaba un resultado y se considerarían exitosas solo si este resultado fuera alcanzado, de lo contrario la funcionalidad debería ser corregida para lograr que el resultado esperado. Dado que el enfoque de estas pruebas fue global, las pruebas funcionales se dividieron en 2 grandes categorías, la primera tuvo como objetivo corroborar el adecuado funcionamiento de la capa de red implementada para el prototipo, la segunda fase tuvo por objetivo corroborar el funcionamiento del sistema de cara a las RTU, es decir comprobar si la lógica y la persistencia funcionan de acuerdo a lo esperado.

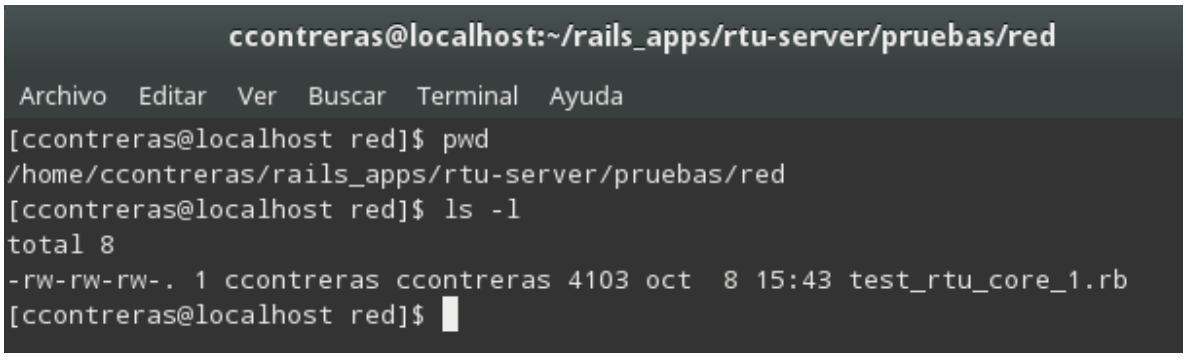
6.4.1 Fase 1- Capa de comunicación en red
El componente encargado de solucionar la comunicación en red utiliza la librería denominada 'eventmachine' para lenguaje Ruby, esta librería implementa "The

reactor pathern” o el patrón de reactor, lo que brinda un medio para captar eventos del sistema, en este caso se enfoca el uso del reactor a captar los eventos del sistema relacionados con la tarjeta de red, para este fin se definió el puerto de pruebas “1234” y la dirección IP “localhost” o “0.0.0.0”, de tal manera que los eventos de Red que involucran dicho puerto y dirección IP serán captados y manejados por el componente de red, a su vez el componente de red se encargara de generar objetos en tiempo de ejecución que representen a los clientes que se conectan a el sistema, estos objetos serán instancias de la clase “RtuCore”.

Procedimiento: Se cargó la clase “RtuCore” por separado, ya que según el diseño del componente se sabe que esta clase hace uso de dos módulos extra los cuales implementan ayudas explicitas en la lógica y persistencia del sistema, por tal motivo al separar el componente de red de estos dos módulos se obtuvo el componente de red puro, una vez aislado este componente, se ejecutó inmediante “irb” el reactor que ofrece la librería 'eventmachine', para observar en tiempo real el funcionamiento de este componente primero se creó un directorio de pruebas y un sub-directorio con el nombre “red”, dentro de esta carpeta se ejecutó la herramienta “irb” cargando así la clase “RtuCore” y se ejecutó dentro del 'irb' la librería 'eventmachine' para corroborar el funcionamiento del componente según cada caso.

→Conexión de un cliente:

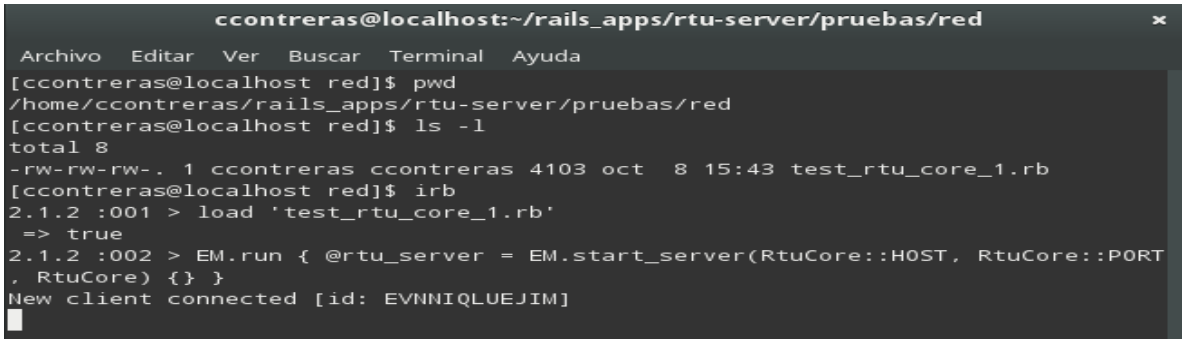
- Valor esperado: Que se cree un objeto de la Clase “RtuCore” una vez se detecta una conexión de un cliente a la dirección 'localhost' y puerto de prueba '1234', asigne un id aleatorio e imprima esta acción en la consola.
- Desarrollo de la prueba: Primero se estableció la ubicación en la carpeta objetivo y se posicionó allí la clase objetivo de la prueba.

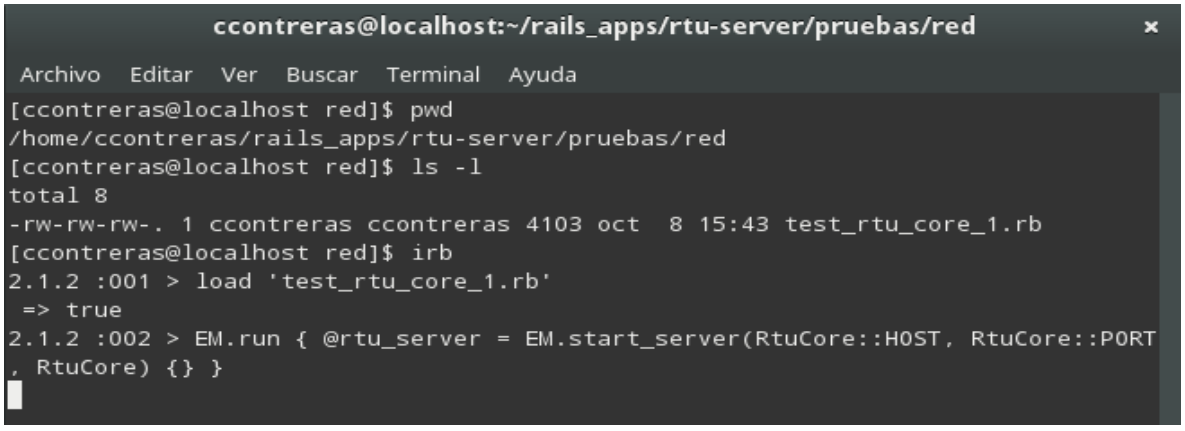


Después se ejecutó 'irb' y se cargó esta clase, luego se procedió a ejecutar 'eventmachine' y se procedió a realizar una conexión mediante la herramienta 'telnet'.

Posteriormente se realizó una conexión a este servidor recién implementado usando 'Telnet'.

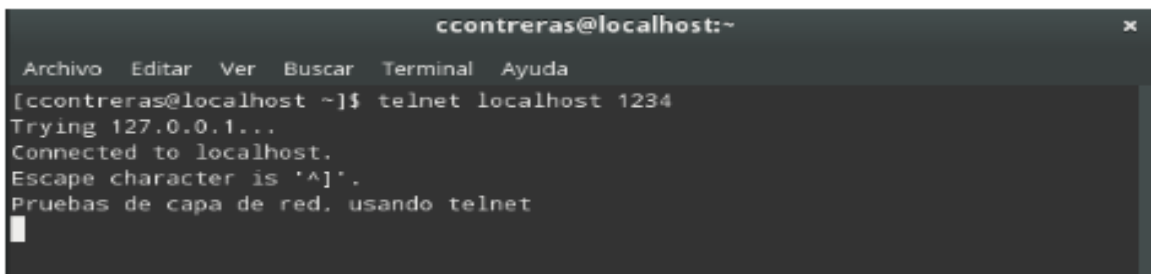
- Resultado: a conexión fue exitosa, y el resultado fue el esperado tal como lo confirma el siguiente comportamiento.





→**Recepción de los datos de un cliente**:

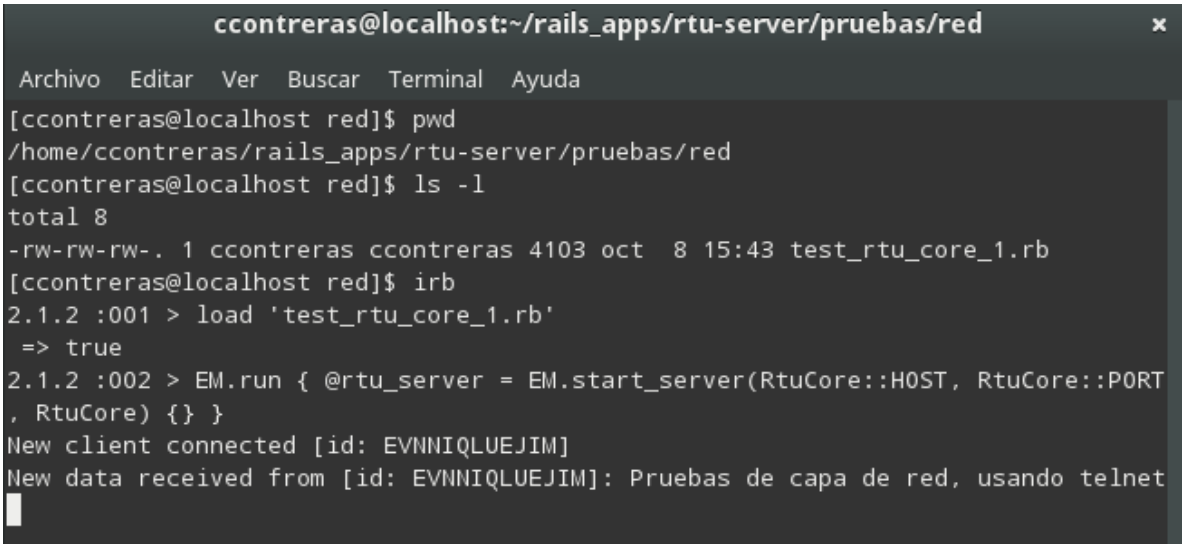
- Valor esperado: Se esperaba que el objeto de la clase “RtuCore” que se creó en la prueba anterior recibiera el paquete de texto que se le envió desde el cliente 'telnet' y que imprimiera en pantalla la acción así como el texto que recibió.
- Desarrollo de la prueba: Continuando en la situación anterior, se procedió a enviar texto mediante 'telnet' a espera del resultado por parte del objeto creado anteriormente.

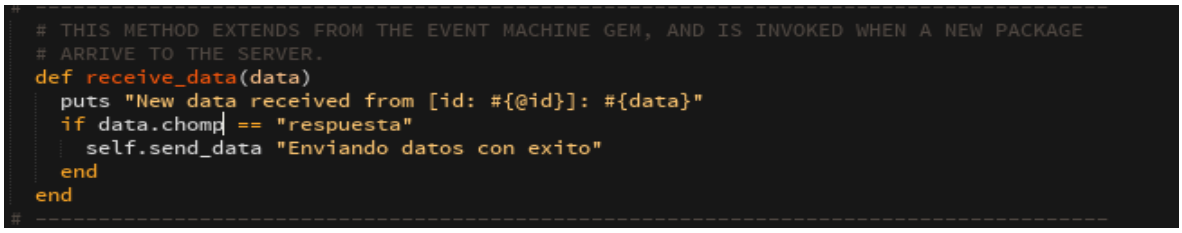


- Resultado: el resultado es exitoso, el objeto imprime en la consola lo esperado, de tal manera que se consigue el comportamiento propuesto.

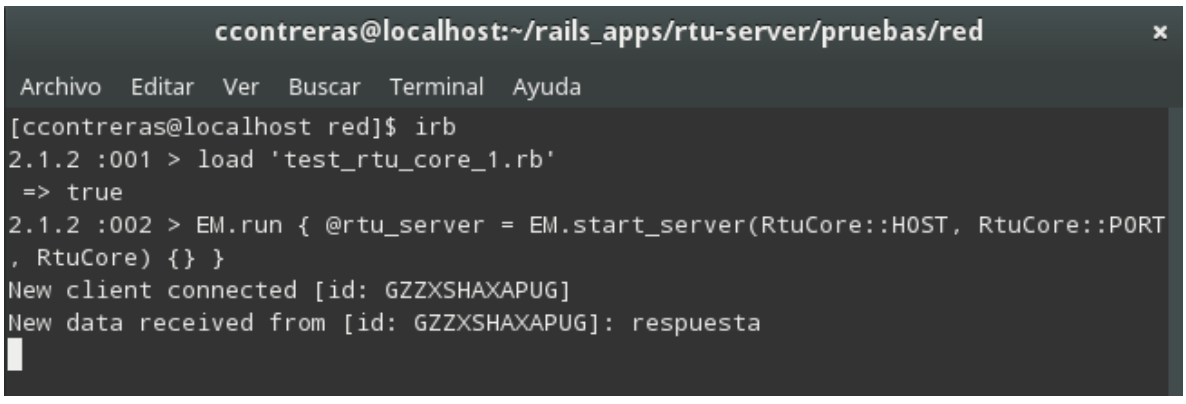
→**Envío de datos**:

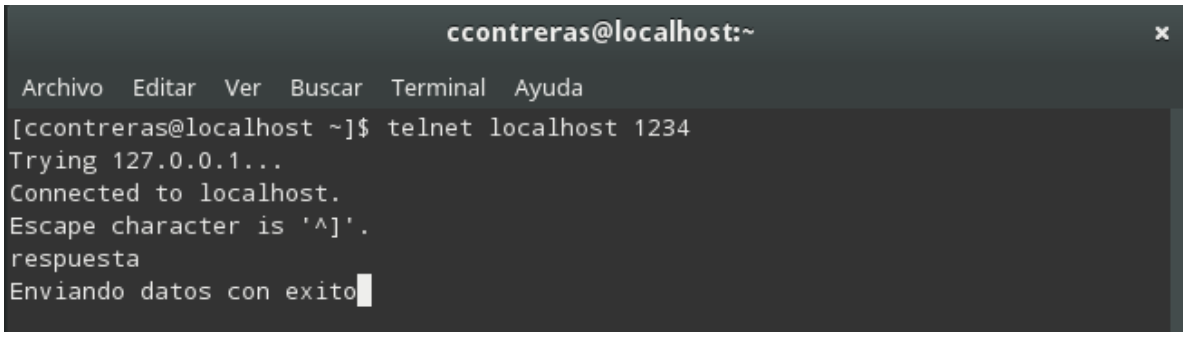
- Valor esperado: en este caso se esperaba que en el cliente 'telnet' se recibiera la cadena de texto “Enviando datos con éxito”, tras el cliente enviar la cadena de testo “respuesta”, este caso se implementó de la siguiente manera:





- Desarrollo de la prueba: las siguientes capturas de pantalla se observa en comportamiento de la prueba, la primera es la consola donde corre el servidor y la segunda es la consola donde corre el cliente telnet:

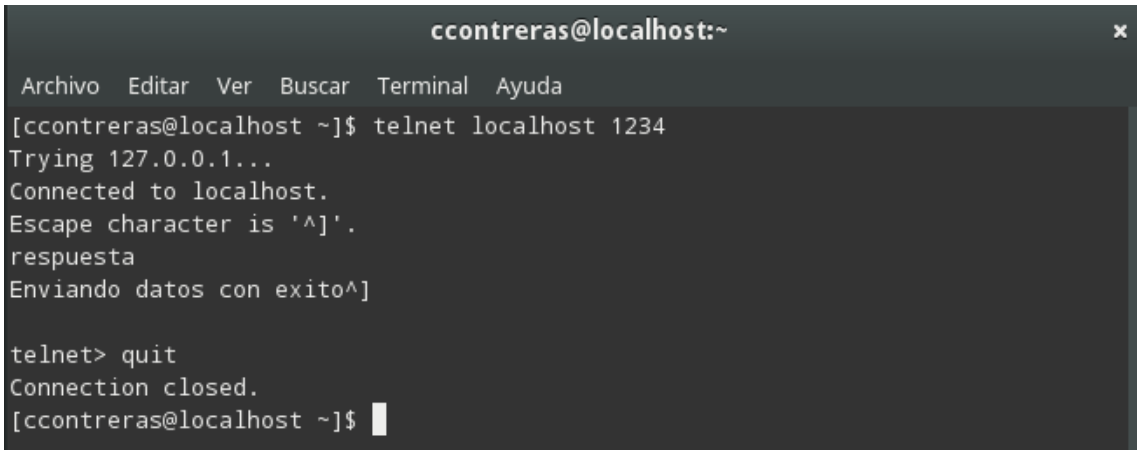


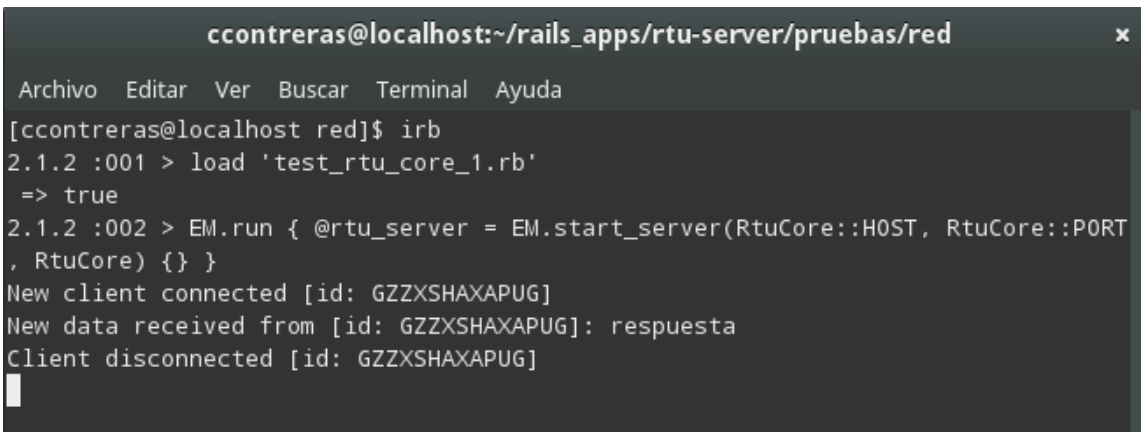


- Resultado: El resultado de la prueba fue exitoso, se detectó el comportamiento esperado.

→Desconexión del cliente.

- Valor esperado: Cuando un cliente que se encuentre conectado se desconecte cualquiera que sea la razón, el componente de red debe detectar la desconexión e imprimir en pantalla este suceso, también se espera que el servidor continué arriba para nuevas conexiones.
- Desarrollo de la prueba: Para tal efecto partiendo de la situación final de la prueba anterior se procedió a desconectar el cliente 'telnet', las siguientes capturas de pantalla muestran lo ocurrido.





- Resultado: El resultado de la prueba fue exitoso, el servidor continuaba arriba a pesar del cierre del cliente, y fue capaz de detectar esta desconexión e imprimir en la consola el suceso. En conclusión, tras las pruebas se corroboró el funcionamiento correcto del componente de red, adicionalmente se corroboró su funcionamiento de manera concurrente atendiendo hasta a 10 clientes al mismo tiempo.

6.4.2 Fase 2 - Pruebas de lógica y persistencia.

Estas pruebas corroboraron el funcionamiento general del prototipo, pretendía analizar las capas de lógica y persistencia en conjunto con el componente de red, así se observaría una perspectiva general de las funcionalidades implementadas.

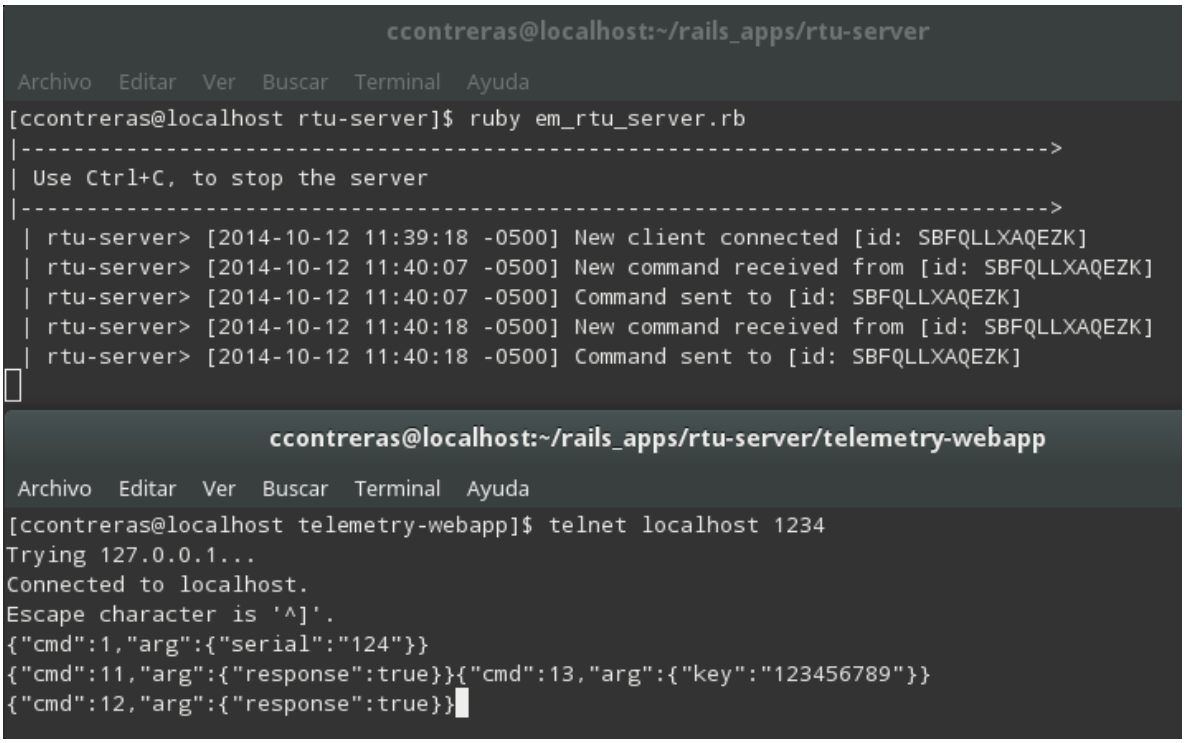
Procedimiento: para estas pruebas se ejecutó el prototipo como tal partiendo de la estructura original del proyecto, y se utilizó “Telnet” a manera de emulación de un cliente, se enviaron comandos prefabricados con intención conocida y resultado es esperado, las pruebas se realizaron en orden de comandos y casos particulares de situaciones propias de cada comando, en algunos casos también se utilizó la herramienta “Rails console” para corroborar los cambios que debieron ser efectuados en la capa de persistencia.

→Pruebas de fase de autentificación y comandos iniciales.

Esta prueba consiste en corroborar que el funcionamiento de la autentificación es el esperado, consta de los siguientes casos:

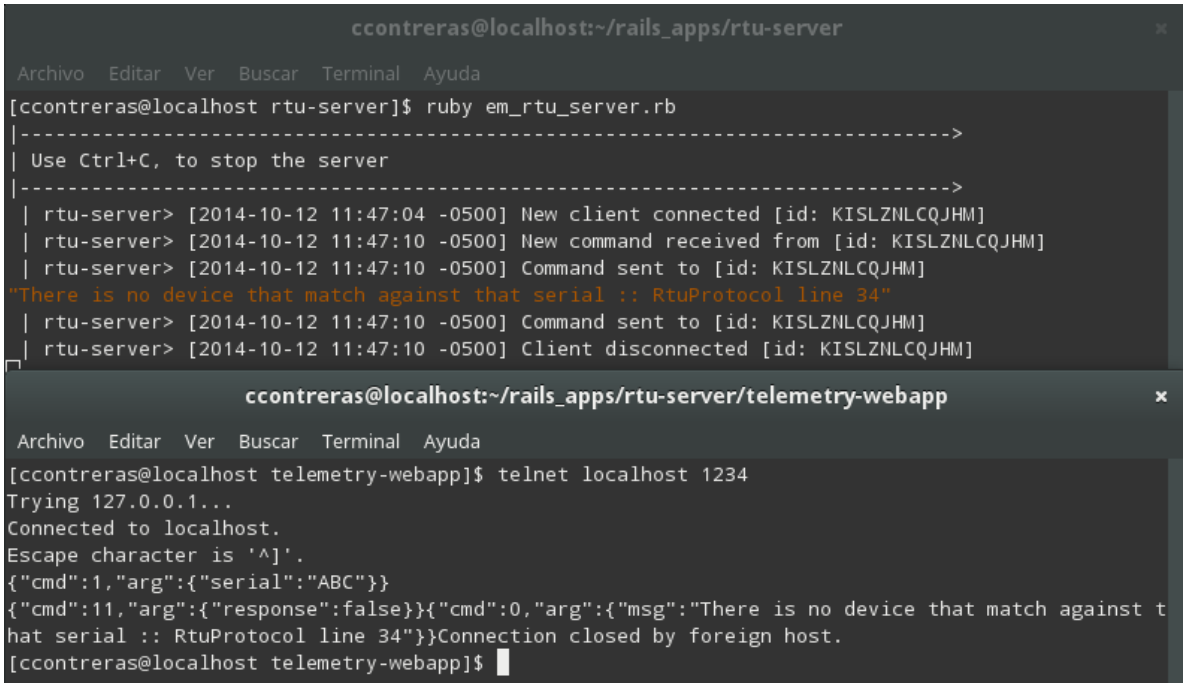
Caso 1: Autentificación exitosa:

- Valor esperado: Se esperaba que el servidor comunique los comandos 11 y 12 con una respuesta satisfactoria dado que los comandos serían enviados en orden 1 y 13, con datos válidos presentes en la base de datos.
- Realización de la prueba: para esta prueba se enviaron los siguientes comandos desde el cliente “Telnet”, estos comandos fueron construidos en base a información presente en la base de datos de prueba.
  - {"cmd":1,"arg":{"serial":"124"}}
  - {"cmd":13,"arg":{"key":"123456789"}}
- Resultado: Se corroboró el comportamiento de forma exitosa, y se incluyen las siguientes capturas de pantalla en las cuales se observa la interacción cliente-servidor.



Caso 2: Autenticación fallida por serial inexistente.

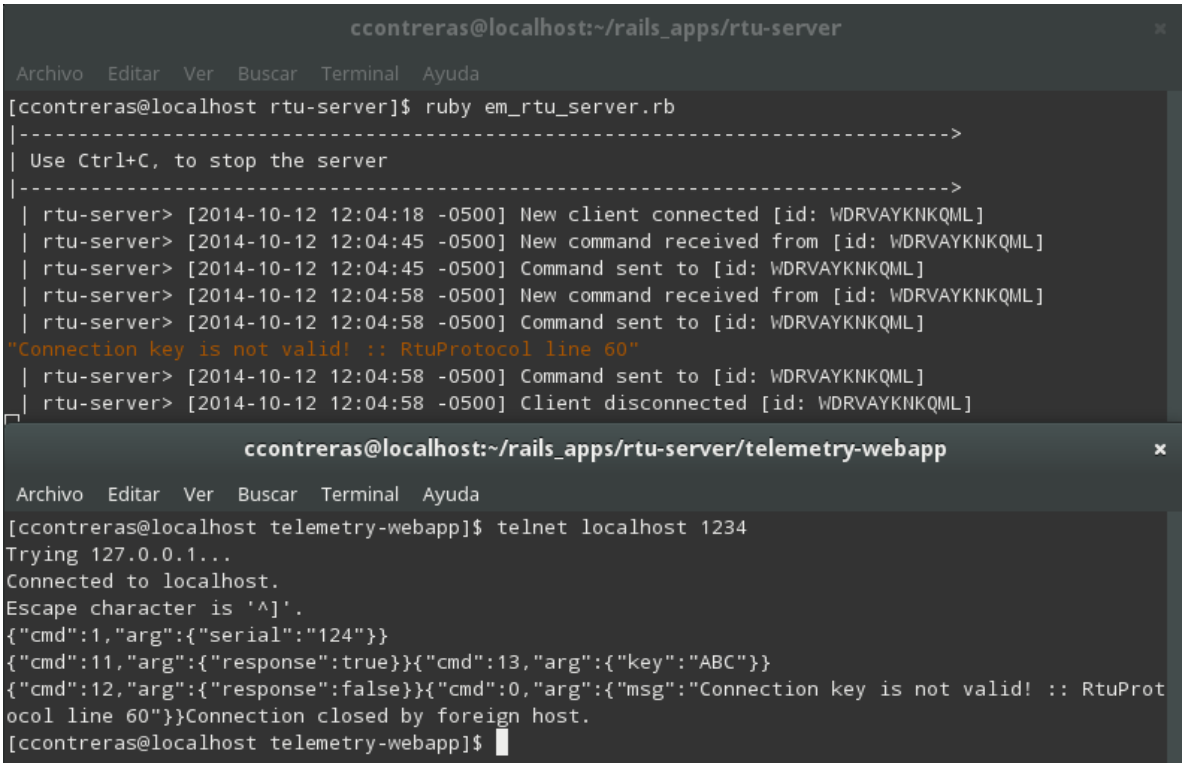
- Valor esperado: Se esperaba que el servidor comunicara el comando 11 con una respuesta negativa, seguido del comando 0 con un mensaje significativo que explicara el porqué de la desconexión.
- Realización de la prueba: para esta prueba se envió el siguiente comando desde el cliente “Telnet”, este comando se caracterizó por llevar en sus argumentos un serial inexistente en la base de datos.
  - {"cmd":1,"arg":{"serial":"ABC"}}
- Resultado: Se corroboró el comportamiento de forma exitosa, a continuación las capturas de pantalla para observar nuevamente la interacción cliente - servidor.



Caso 3: Autenticación fallida por una llave de conexión inexistente

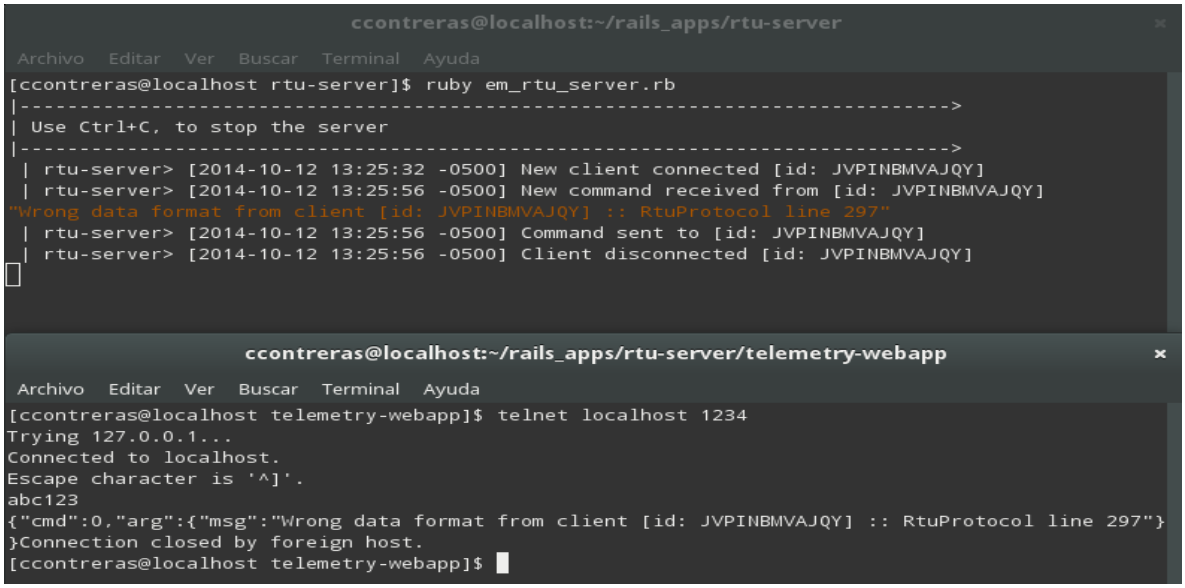
- Valor esperado: Se esperaba que el servidor comunicara el comando 11 con una respuesta negativa, seguido del comando 12 con una respuesta fallida dado que la llave de conexión es inválida, seguido del comando 0 con un mensaje significativo que explicara el porqué de la desconexión.
- Realización de la prueba: para esta prueba se enviaron los siguientes comandos desde el cliente “Telnet”, el primero fue un comando 1 válido para ser aceptado en conexión, el segundo se caracterizó por llevar en sus argumentos una llave de conexión inexistente en la base de datos:
  - {"cmd":1,"arg":{"serial":"124"}}
  - {"cmd":13,"arg":{"key":"ABC"}}

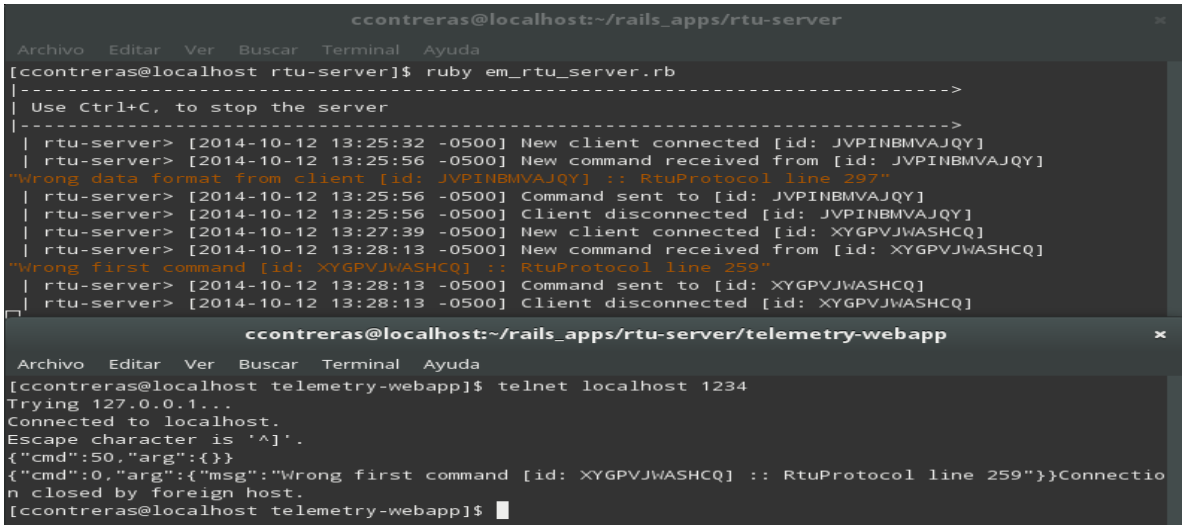
- Resultado: Se corroboró el comportamiento exitoso, a continuación las capturas de pantalla.

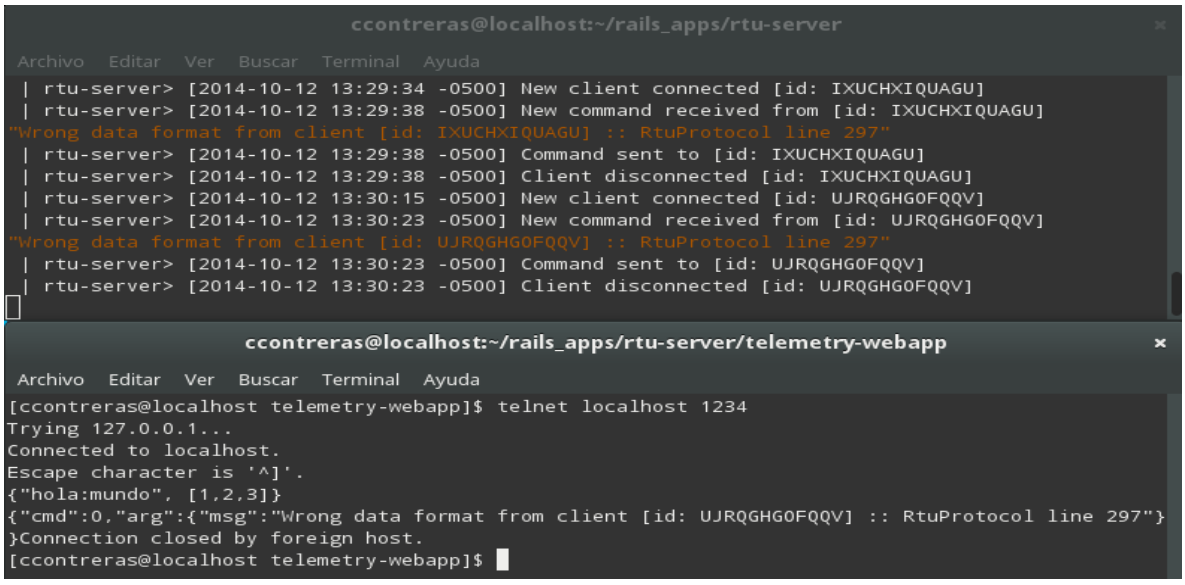


Caso 4: Captura de error por falla en la estructura del comando

- Valor esperado: Se esperaba que el servidor comunicara el comando 0 con un mensaje adecuado para cada sub-caso.
- Realización de la prueba: para esta prueba se enviaron los siguientes comandos desde el cliente "Telnet", el primero es una cadena de texto convencional, el segundo es un comando con un número que no existe, el tercero es una estructura json que no sigue la convención para el protocolo:
  - abc123
  - {"cmd":50,"arg":{}}
  - {"hola:mundo",[1,2,3]}
- Resultado: Se corroboró el comportamiento de forma exitosa, para cada sub-caso, el servidor correctamente detectó los errores en los paquetes y los desplegó en la pantalla, envió el comando 0 con un mensaje significativo y cerró el canal de comunicación para el cliente en cada caso. Se visualizan a continuación las capturas correspondientes:


ccontreras@localhost:~/rails_apps/rtu-server
Archivo Editar Ver Buscar Terminal Ayuda
[ccontreras@localhost rtu-server]$ ruby em_rtu_server.rb
|-->
| Use Ctrl+C, to stop the server
|-->
| rtu-server> [2014-10-12 13:25:32 -0500] New client connected [id: JVPINBMVAJQY]
| rtu-server> [2014-10-12 13:25:56 -0500] New command received from [id: JVPINBMVAJQY]
"Wrong data format from client [id: JVPINBMVAJQY] :: RtuProtocol line 297"
| rtu-server> [2014-10-12 13:25:56 -0500] Command sent to [id: JVPINBMVAJQY]
| rtu-server> [2014-10-12 13:25:56 -0500] Client disconnected [id: JVPINBMVAJQY]
ccontreras@localhost:~/rails_apps/rtu-server/telemetry-webapp
Archivo Editar Ver Buscar Terminal Ayuda
[ccontreras@localhost telemetry-webapp]$ telnet localhost 1234
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
abc123
{"cmd":0,"arg":{"msg":"Wrong data format from client [id: JVPINBMVAJQY] :: RtuProtocol line 297"}
}Connection closed by foreign host.
[ccontreras@localhost telemetry-webapp]$

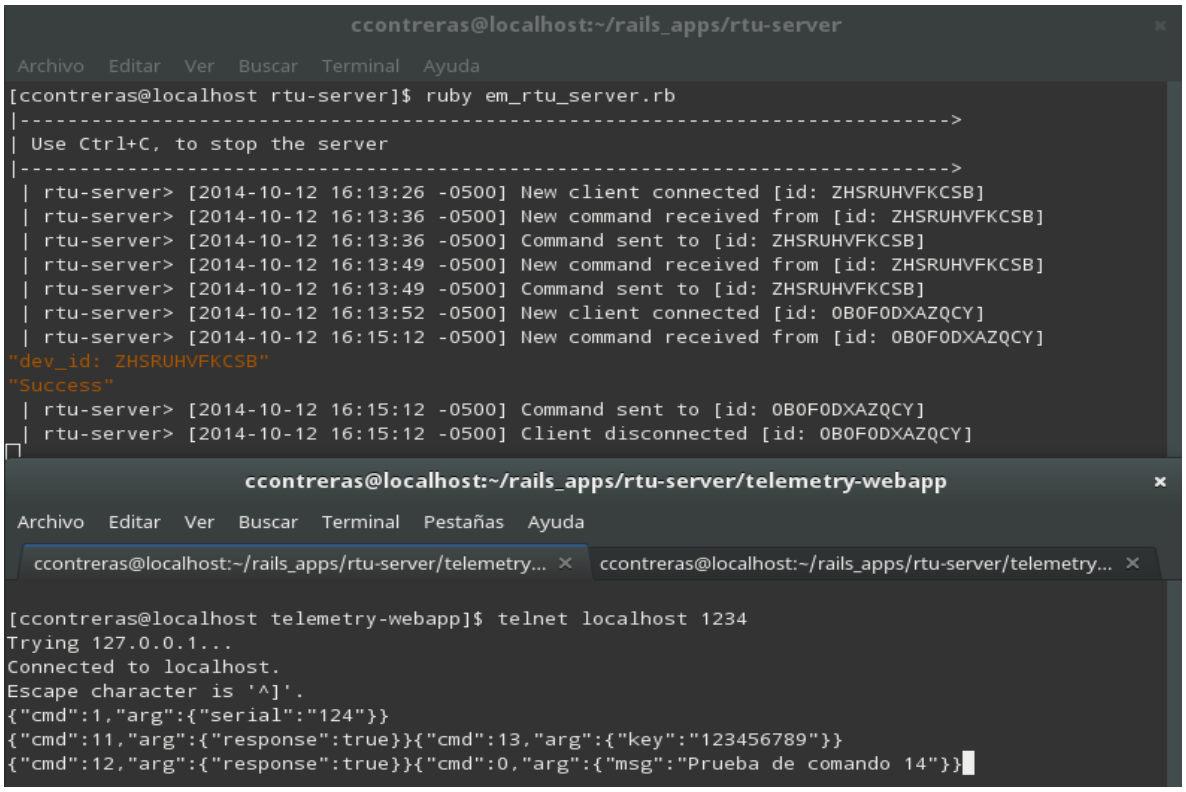

ccontreras@localhost:~/rails_apps/rtu-server
Archivo Editar Ver Buscar Terminal Ayuda
[ccontreras@localhost rtu-server]$ ruby em_rtu_server.rb
|-->
| Use Ctrl+C, to stop the server
|-->
| rtu-server> [2014-10-12 13:25:32 -0500] New client connected [id: JVPINBMVAJQY]
| rtu-server> [2014-10-12 13:25:56 -0500] New command received from [id: JVPINBMVAJQY]
"Wrong data format from client [id: JVPINBMVAJQY] :: RtuProtocol line 297"
| rtu-server> [2014-10-12 13:25:56 -0500] Command sent to [id: JVPINBMVAJQY]
| rtu-server> [2014-10-12 13:25:56 -0500] Client disconnected [id: JVPINBMVAJQY]
| rtu-server> [2014-10-12 13:27:39 -0500] New client connected [id: XYGPVJWASHCQ]
| rtu-server> [2014-10-12 13:28:13 -0500] New command received from [id: XYGPVJWASHCQ]
"Wrong first command [id: XYGPVJWASHCQ] :: RtuProtocol line 259"
| rtu-server> [2014-10-12 13:28:13 -0500] Command sent to [id: XYGPVJWASHCQ]
| rtu-server> [2014-10-12 13:28:13 -0500] Client disconnected [id: XYGPVJWASHCQ]
ccontreras@localhost:~/rails_apps/rtu-server/telemetry-webapp
Archivo Editar Ver Buscar Terminal Ayuda
[ccontreras@localhost telemetry-webapp]$ telnet localhost 1234
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
{"cmd":50,"arg":{}}
{"cmd":0,"arg":{"msg":"Wrong first command [id: XYGPVJWASHCQ] :: RtuProtocol line 259"}}Connectio
n closed by foreign host.
[ccontreras@localhost telemetry-webapp]$


ccontreras@localhost:~/rails_apps/rtu-server
Archivo Editar Ver Buscar Terminal Ayuda
| rtu-server> [2014-10-12 13:29:34 -0500] New client connected [id: IXUCHXIQUAGU]
| rtu-server> [2014-10-12 13:29:38 -0500] New command received from [id: IXUCHXIQUAGU]
"Wrong data format from client [id: IXUCHXIQUAGU] :: RtuProtocol line 297"
| rtu-server> [2014-10-12 13:29:38 -0500] Command sent to [id: IXUCHXIQUAGU]
| rtu-server> [2014-10-12 13:29:38 -0500] Client disconnected [id: IXUCHXIQUAGU]
| rtu-server> [2014-10-12 13:30:15 -0500] New client connected [id: UJRQGHGOFQQV]
| rtu-server> [2014-10-12 13:30:23 -0500] New command received from [id: UJRQGHGOFQQV]
"Wrong data format from client [id: UJRQGHGOFQQV] :: RtuProtocol line 297"
| rtu-server> [2014-10-12 13:30:23 -0500] Command sent to [id: UJRQGHGOFQQV]
| rtu-server> [2014-10-12 13:30:23 -0500] Client disconnected [id: UJRQGHGOFQQV]
ccontreras@localhost:~/rails_apps/rtu-server/telemetry-webapp
Archivo Editar Ver Buscar Terminal Ayuda
[ccontreras@localhost telemetry-webapp]$ telnet localhost 1234
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
{"hola:mundo", [1,2,3]}
{"cmd":0,"arg":{"msg":"Wrong data format from client [id: UJRQGHGOFQQV] :: RtuProtocol line 297"}
}Connection closed by foreign host.
[ccontreras@localhost telemetry-webapp]$

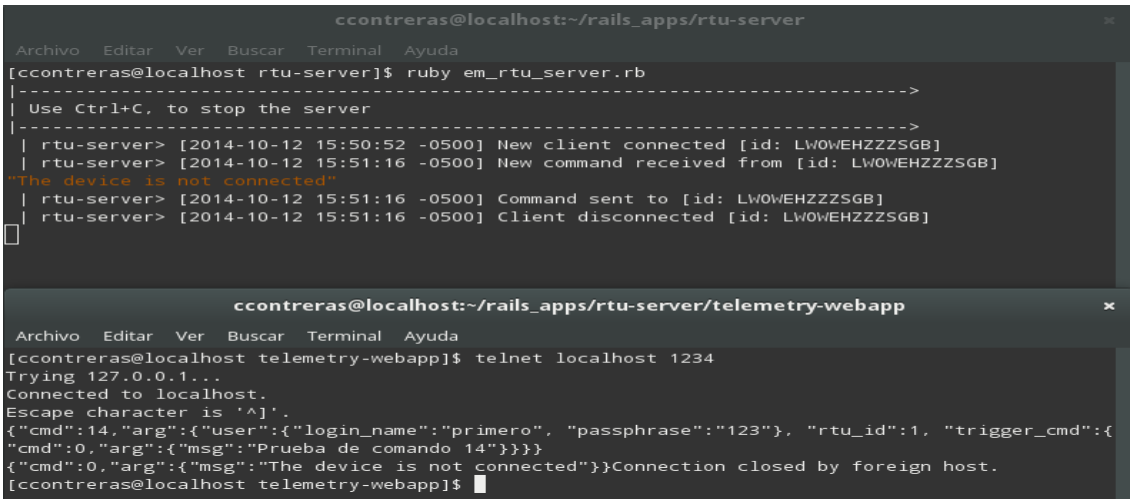
Caso 5: Reconocimiento exitoso del comando 14.

- Valor esperado: Se esperaba que el servidor respondiera enviando el comando que se le pasó mediante el comando 14 hacia la caja negra que estaba siendo emulada por un cliente "telnet".
- Realización de la prueba: para esta prueba se enviaron los siguientes comandos desde el cliente "Telnet":
  - {"cmd":14,"arg":{"user":{"login_name":"primero", "passphrase":"123"}, "rtu_id":1, "trigger_cmd":"{"cmd":0,"arg":{"msg":"Prueba de comando 14"}}"}}
  - {"cmd":14,"arg":{"user":{"login_name":"quinto", "passphrase":"123"}, "rtu_id":1, "trigger_cmd":{"cmd":0,"arg":{"msg":"Prueba de comando 14"}}}}

Al ejecutar la prueba en primera instancia el servidor detectó un error dado que para que el comando sea exitoso el servidor exige que la RTU indicada esté conectada, y no era el caso, así que se procedió a abrir un cliente "Telnet" que emulara la conexión de la RTU indicada.



Una vez que se ha conectado un cliente "Telnet" emulando ser la RTU a la cual se desea enviar el comando, se observó que el servidor acertadamente envía el comando que se deseaba enviar.

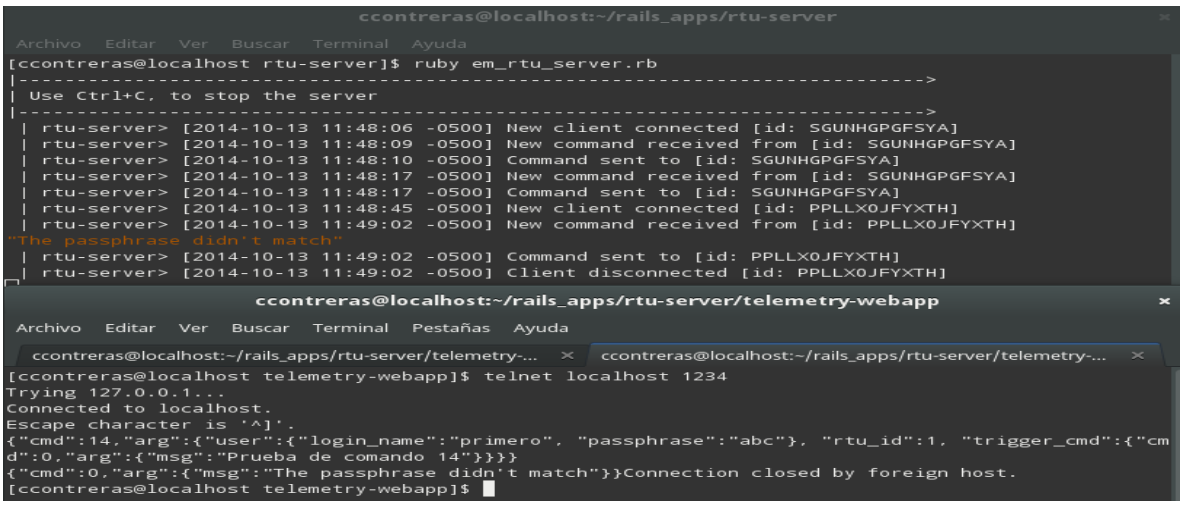


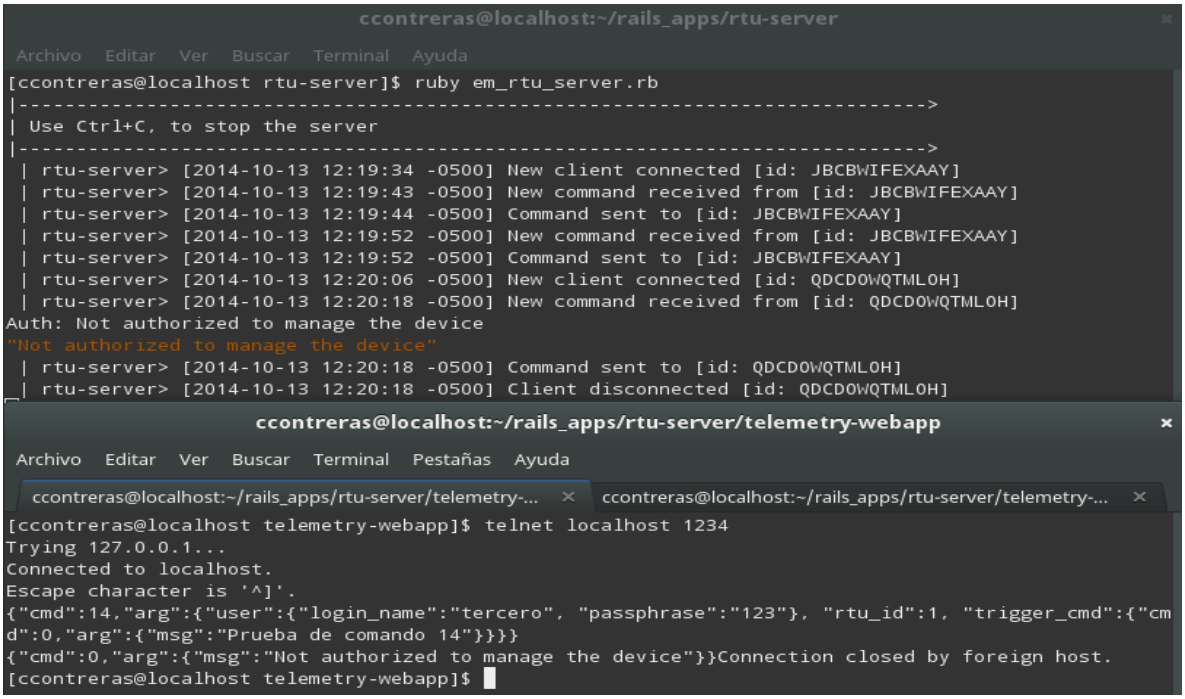
- Resultado: Se corroboró el comportamiento de forma exitosa en ambos sub-caso, ya que se envió el comando adecuado tanto para el usuario relacionado con rol de propietario como para el usuario relacionado con rol de contacto y los mensajes desplegados fueron los esperados.

Caso 6: Prueba de autentificación fallida para usuarios en comando 14

- Valor esperado: Se esperaba que en ambos casos el servidor negara el recurso dado que en primera instancia se pasaría una contraseña no válida y en el segundo un usuario no relacionado a la RTU.
- Realización de la prueba: para esta prueba se enviaron los siguientes comandos desde el cliente “Telnet”:
  - {"cmd":14,"arg":{"user":{"login_name":"primero", "passphrase":"abc"}, "rtu_id":1, "trigger_cmd":"{"cmd":0,"arg":{"msg":"Prueba de comando 14"}}"}}
  - {"cmd":14,"arg":{"user":{"login_name":"tercero", "passphrase":"123"}, "rtu_id":1, "trigger_cmd":{"cmd":0,"arg":{"msg":"Prueba de comando 14"}}}}

Como era de esperarse el servidor correctamente detectó las inconsistencias y comunicó mensajes significativos en consola y al cliente, tal como se aprecia en las siguientes capturas de pantalla:



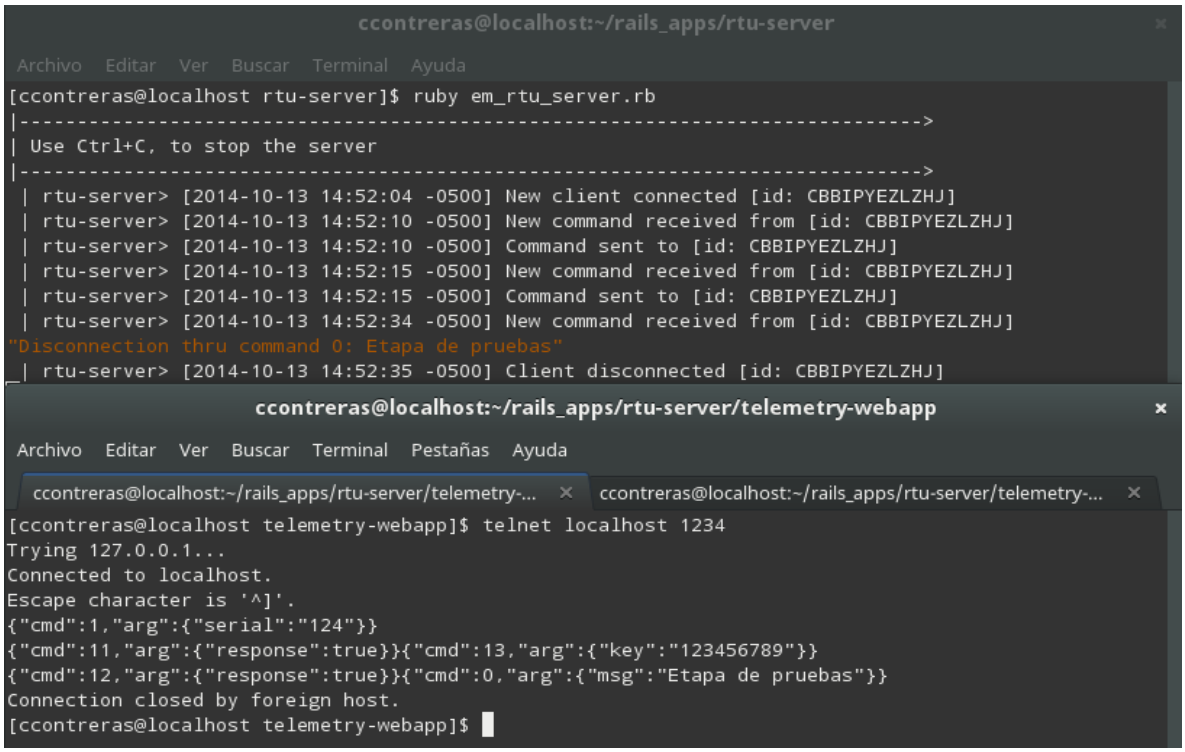


→Pruebas para comandos de atención.

Los comandos de atención son aquellos que una vez las RTU superaron la etapa de autentificación, se emplean para atender de manera automática los paquetes que se reciben, provenientes de las cajas negras, estas pruebas contemplan los comandos del 2 al 9 y el 0, el escenario en todos los casos se referirá a la misma RTU.

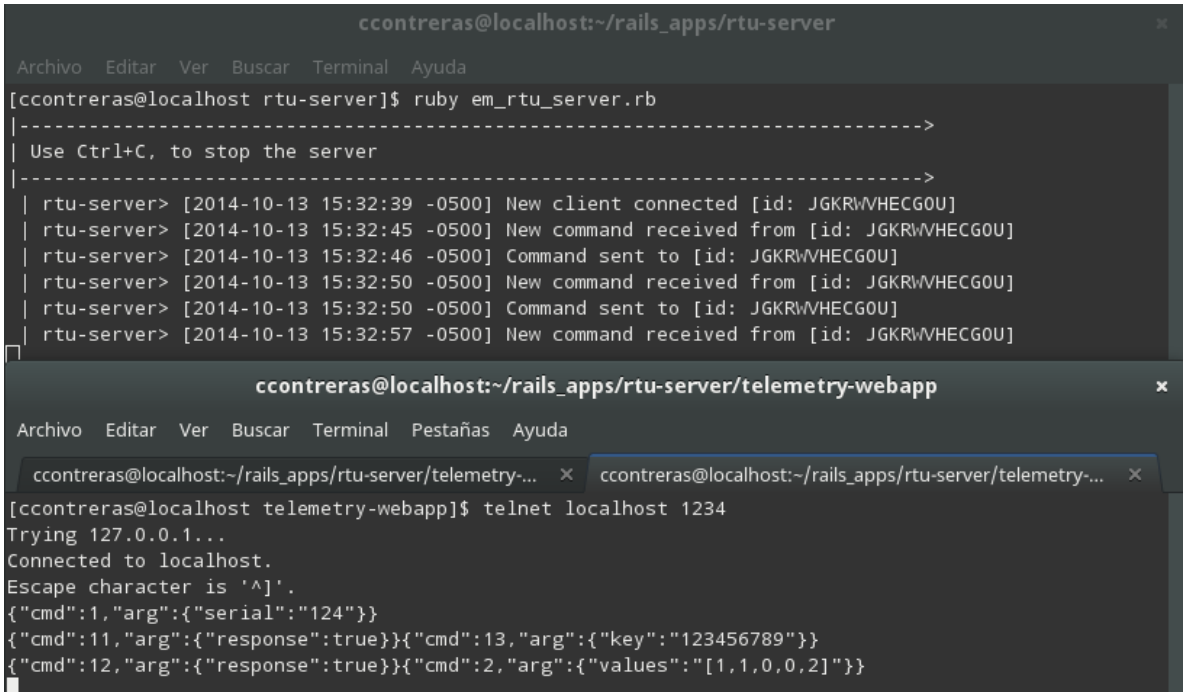
Caso 1: Funcionamiento del comando 0.

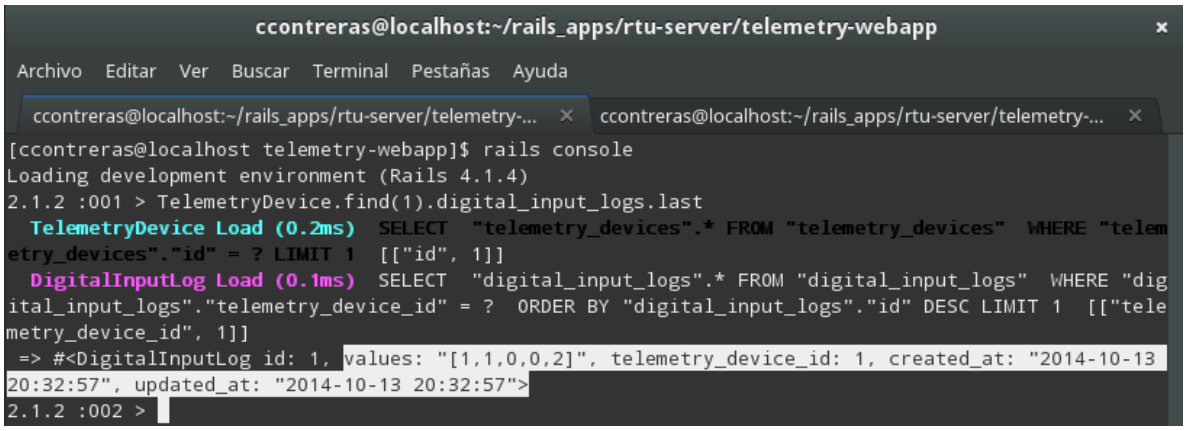
- Valor esperado: Se esperaba que el servidor procesara la secuencia de desconexión para la RTU inmediatamente después de recibir un comando 0 de una RTU que esté debidamente en etapa de atención.
- Realización de la prueba: para esta prueba se enviaron los siguientes comandos desde el cliente "Telnet", el primero fue un comando 1 válido para ser aceptado en conexión, el segundo comando 13 válido para entablar la etapa de atención, finalmente se envía el comando 0:
  - {"cmd":1,"arg":{"serial":"124"}}
  - {"cmd":13,"arg":{"key":"123456789"}}
  - {"cmd":0", arg":{"msg":"Etapa de pruebas"}}
- Resultado: Se corroboró el comportamiento de forma exitosa y se incluyen las siguientes capturas de pantalla en las cuales se observa el resultado:



Caso 2: Funcionamiento del comando 2.

- Valor esperado: Se esperaba que el servidor almacenara los valores para las entradas digitales reportadas por la RTU en la base de datos.
- Realización de la prueba: para esta prueba se enviaron los siguientes comandos desde el cliente "Telnet", el primero fue un comando 1 válido para ser aceptado en conexión, el segundo comando 13 válido para entablar la etapa de atención, finalmente se envió el comando 2 con los valores correspondientes, sabiendo que la distribución de pines de las salidas digitales para esta RTU era la siguiente: [10,11,12,13,14]
  - {"cmd":1,"arg":{"serial":"124"}}
  - {"cmd":13,"arg":{"key":"123456789"}}
  - {"cmd":2", arg":{"values":"[1,1,0,0,2]"}}
- Resultado: Una vez se observó que el comando se ejecutó correctamente, se utilizó la herramienta "Rails Console" para corroborar que se creó el registro en la base de datos, y efectivamente se observó que así fue. Así se corroboró el comportamiento de forma exitosa, y se incluyen las siguientes captura de pantalla en las cuales se observa este resultado:



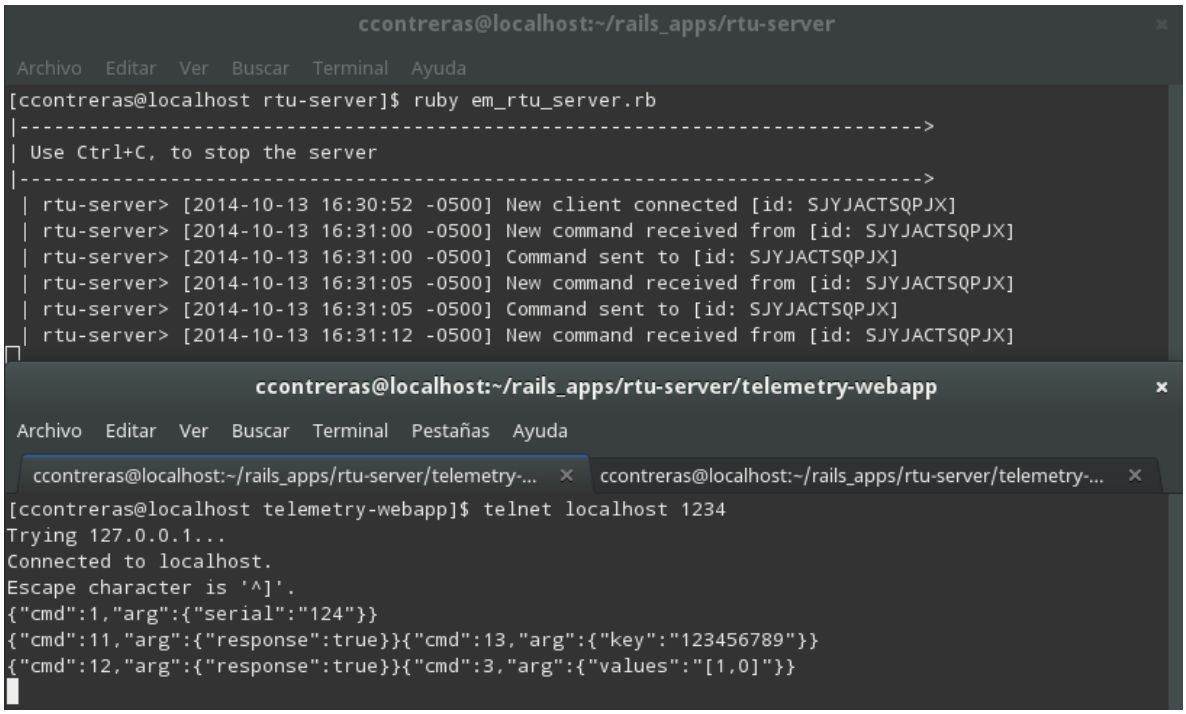


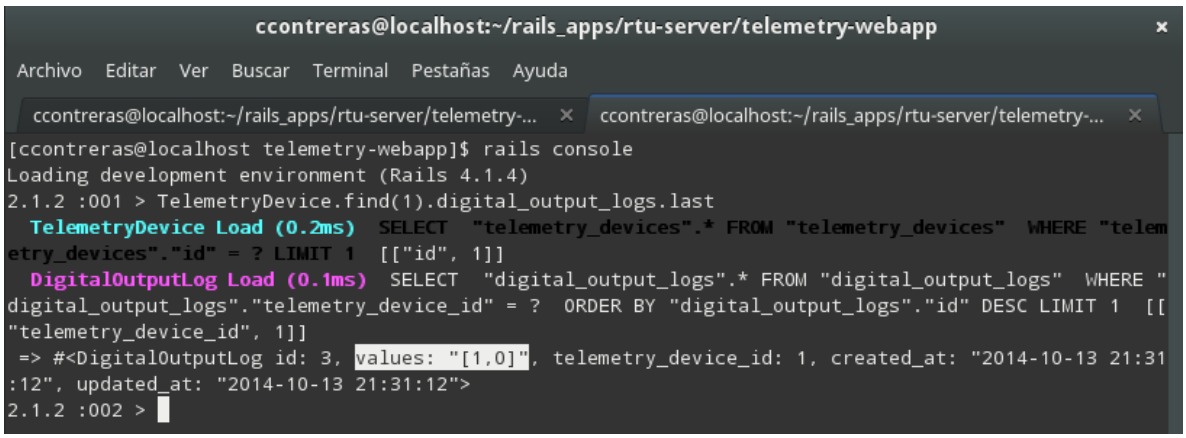
Caso 3: Funcionamiento del comando 3.

- Valor esperado: Se esperaba que el servidor almacenara los valores para las salidas digitales reportadas para el comando 2 en la base de datos.
- Realización de la prueba: para esta prueba se enviaron los siguientes comandos desde el cliente "Telnet", el primero fue un comando 1 válido para ser aceptado en conexión, el segundo comando 13 válido para entablar la etapa de atención, finalmente se envió el comando 3 con los valores correspondientes, sabiendo que la distribución de pines de las salidas digitales para esta RTU era la siguiente: [15,16]

- {"cmd":1,"arg":{"serial":"124"}}
- {"cmd":13,"arg":{"key":"123456789"}}
- {"cmd":3", arg":{"values":"[1,0]"}}

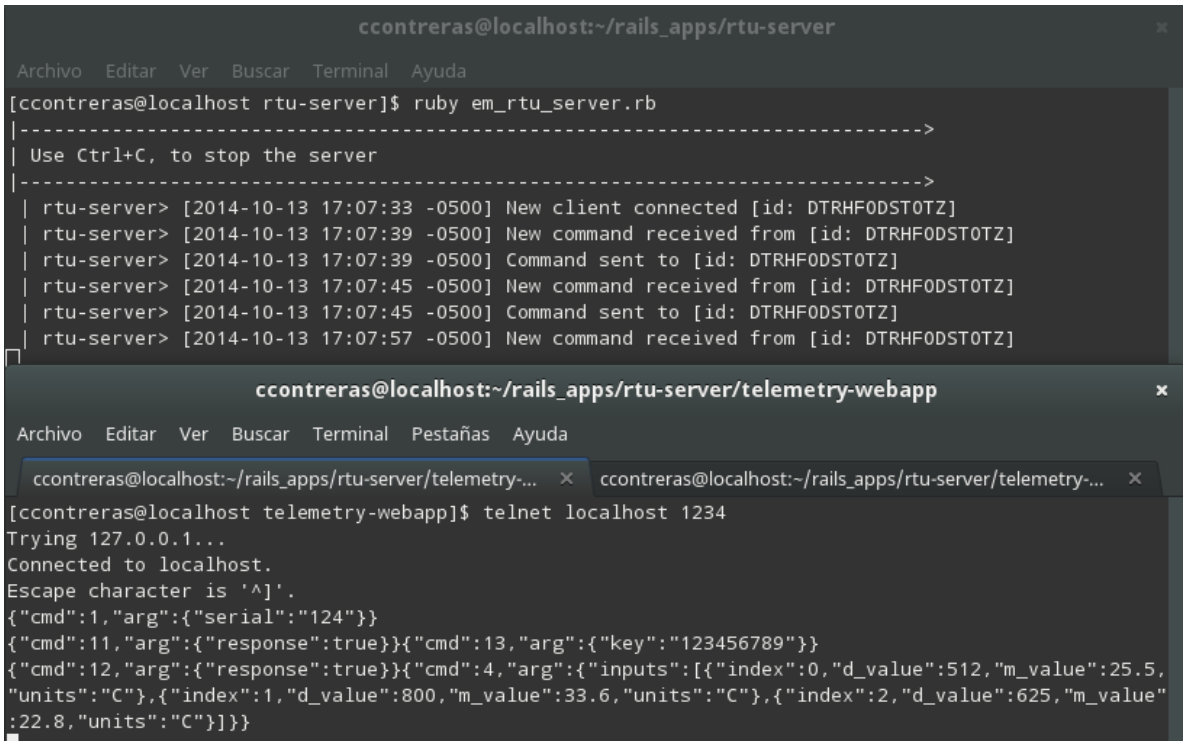
- Resultado: Una vez se observó que el comando se ejecutó correctamente, se utilizó la herramienta “Rails Console” para corroborar que se creó el registro en la base de datos, y efectivamente se observó que así fue. Así se corroboró el comportamiento de forma exitosa, y se pueden observar a continuación las siguientes capturas de pantalla:

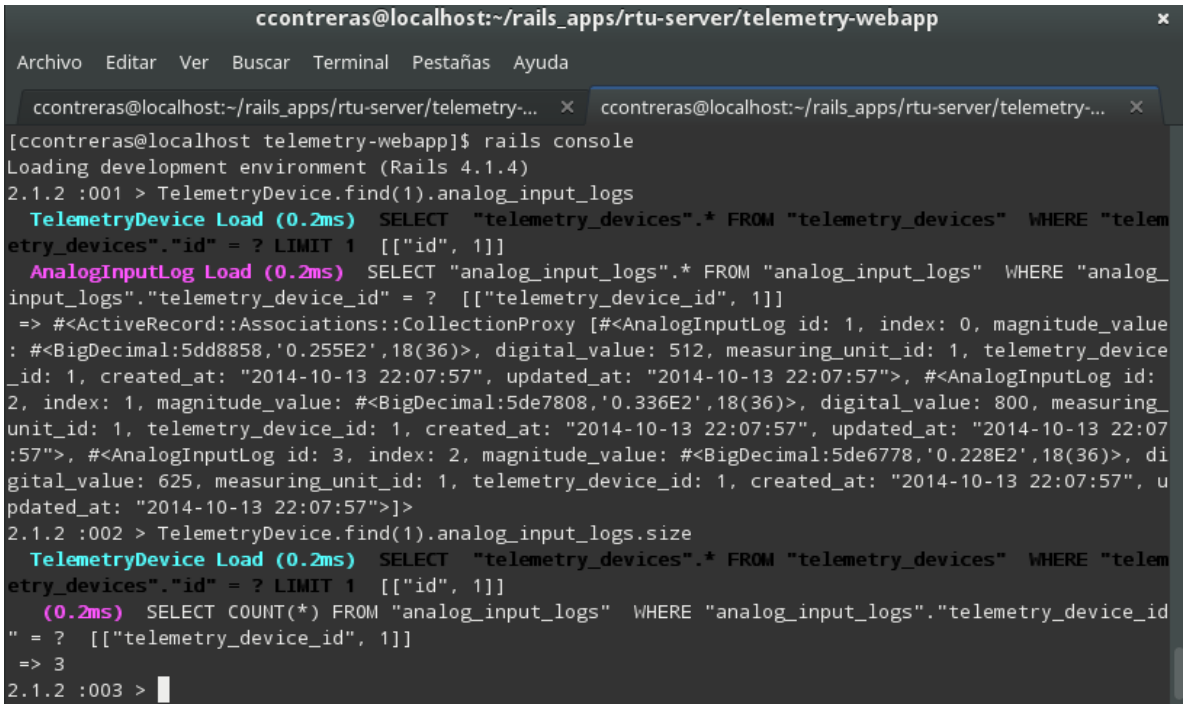




Caso 4: Funcionamiento del comando 4.

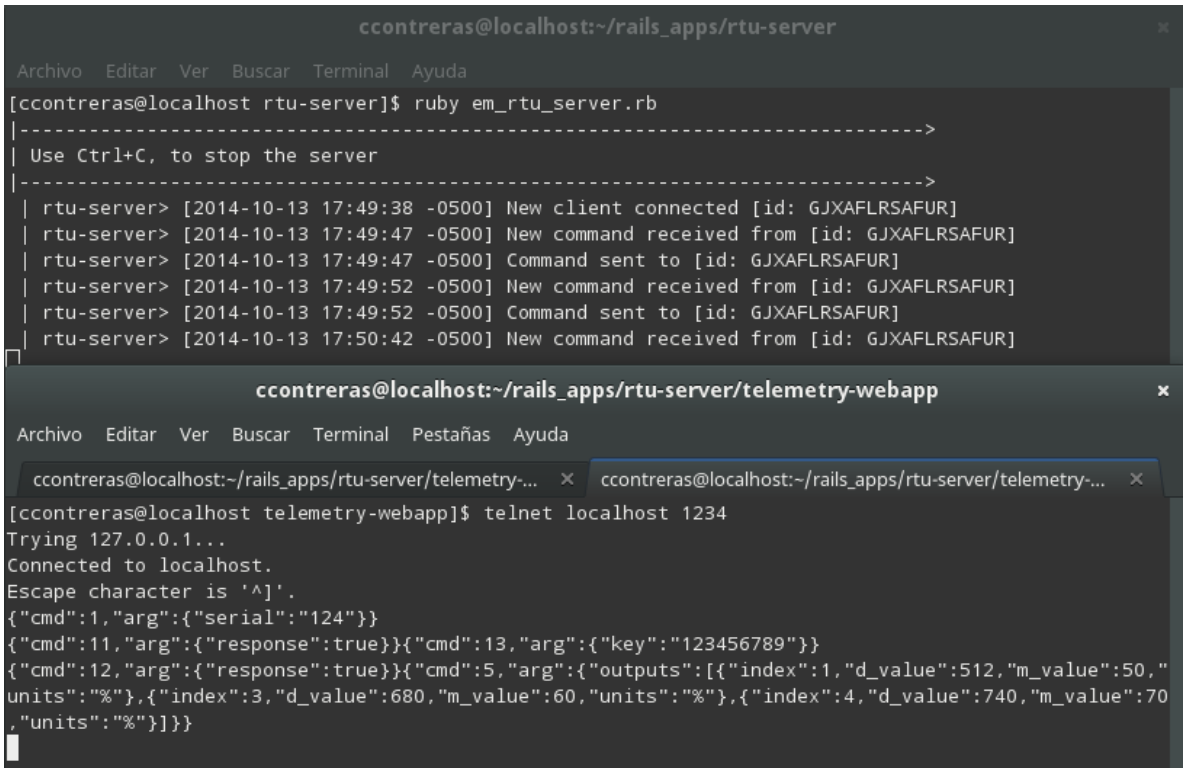
- Valor esperado: Se esperaba que el servidor almacenara los valores para las entradas análogas reportadas por el comando 4 en la base de datos.
- Realización de la prueba: para esta prueba se enviaron los siguientes comandos desde el cliente "Telnet", el primero fue un comando 1 válido para ser aceptado en conexión, el segundo comando 13 válido para entablar la etapa de atención, finalmente se envió el comando 4 con los valores correspondientes, sabiendo que la distribución de pines de las entradas análogas para esta RTU era la siguiente: [1,2,3,4]
  - {"cmd":1,"arg":{"serial":"124"}}
  - {"cmd":13,"arg":{"key":"123456789"}}
  - {"cmd":4", arg":{"inputs":
    [{"index":0,"d_value":512,"m_value":25.5,"units":"C"},
    {"index":1,"d_value":800,"m_value":33.6,"units":"C"},
    {"index":2,"d_value":625,"m_value":22.8,"units":"C"}]}}
- Resultado: Una vez se observó que el comando se ejecutó correctamente, se utilizó la herramienta "Rails Console" para corroborar que se creó el registro en la base de datos, y efectivamente se observó que así fue.
  En esta parte se ve que en los tres objetos de tipo AnalogInput Log fueron creados y se visualizan sus valores. Así se corroboró el comportamiento de forma exitosa mostradas a continuación:

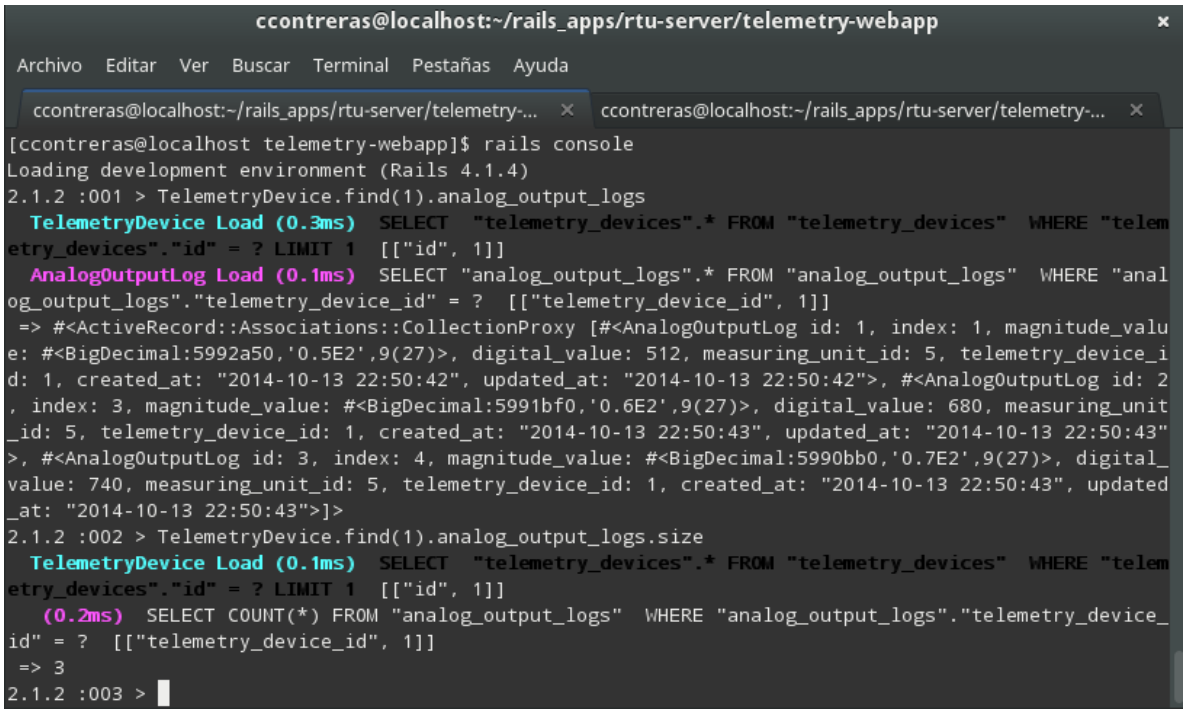




Caso 5: Funcionamiento del comando 5.

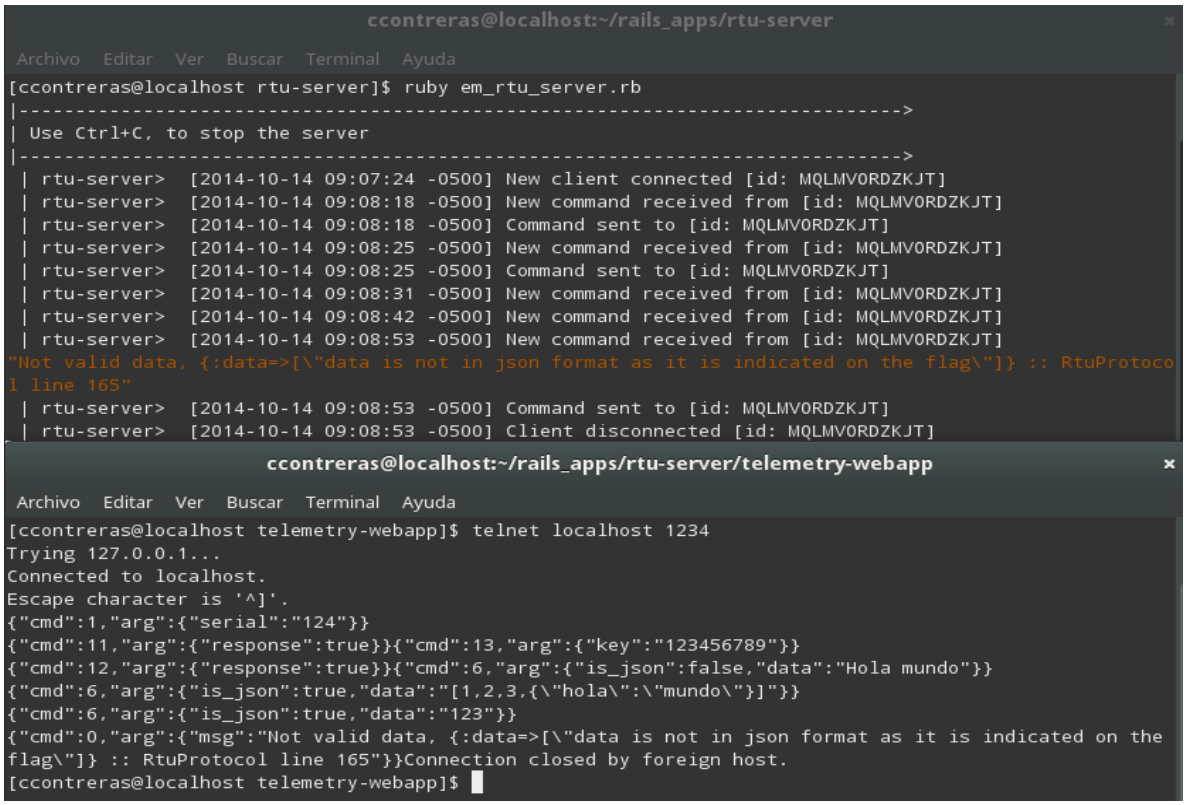
- Valor esperado: Se esperaba que el servidor almacenara los valores para las salidas análogas reportadas por el comando 5 en la base de datos.
- Realización de la prueba: para esta prueba se enviaron los siguientes comandos desde el cliente "Telnet", el primero fue un comando 1 válido para ser aceptado en conexión, el segundo comando 13 válido para entablar la etapa de atención, finalmente se envió el comando 5 con los valores correspondientes, sabiendo que la distribución de pines de las entradas análogas para esta RTU era la siguiente: [5,6,7,8,9]
  - {"cmd":1,"arg":{"serial":"124"}}
  - {"cmd":13,"arg":{"key":"123456789"}}
  - {"cmd":5", arg":{"inputs":
    [{"index":1,"d_value":512,"m_value":50,"units":"%"},
    {"index":3,"d_value":680,"m_value":60,"units":"%"},
    {"index":4,"d_value":740,"m_value":70,"units":"%"}]}}
- Resultado: Una vez se observó que el comando se ejecutó correctamente, se utilizó la herramienta "Rails Console" para corroborar que se creó el registro en la base de datos, y efectivamente se observó que así fue. En esta parte se ve que en los tres objetos de tipo AnalogOutputLog fueron creados y se visualizan sus valores. Así se corroboró el comportamiento de forma exitosa y se adjuntan los resultados obtenidos:

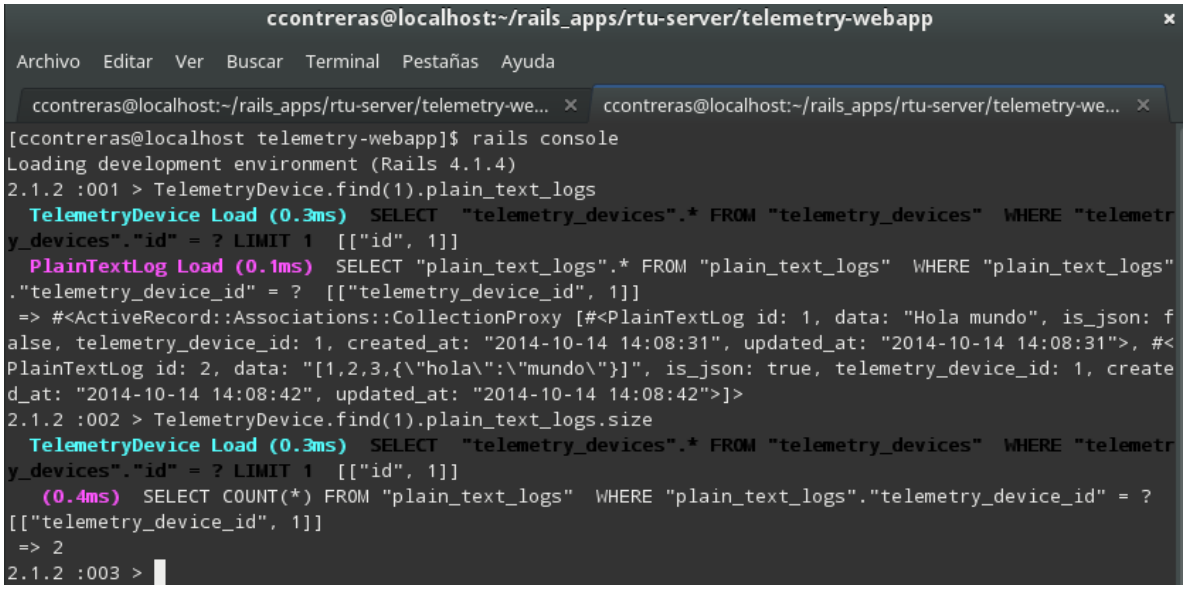

ccontreras@localhost:~/rails_apps/rtu-server
Archivo Editar Ver Buscar Terminal Ayuda
[ccontreras@localhost rtu-server]$ ruby em_rtu_server.rb
|-->
| Use Ctrl+C, to stop the server
|-->
| rtu-server> [2014-10-13 17:49:38 -0500] New client connected [id: GJXAFLRSAFUR]
| rtu-server> [2014-10-13 17:49:47 -0500] New command received from [id: GJXAFLRSAFUR]
| rtu-server> [2014-10-13 17:49:47 -0500] Command sent to [id: GJXAFLRSAFUR]
| rtu-server> [2014-10-13 17:49:52 -0500] New command received from [id: GJXAFLRSAFUR]
| rtu-server> [2014-10-13 17:49:52 -0500] Command sent to [id: GJXAFLRSAFUR]
| rtu-server> [2014-10-13 17:50:42 -0500] New command received from [id: GJXAFLRSAFUR]
ccontreras@localhost:~/rails_apps/rtu-server/telemetry-webapp
Archivo Editar Ver Buscar Terminal Pestañas Ayuda
ccontreras@localhost:~/rails_apps/rtu-server/telemetry-...
ccontreras@localhost:~/rails_apps/rtu-server/telemetry-...
[ccontreras@localhost telemetry-webapp]$ telnet localhost 1234
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
{"cmd":1,"arg":{"serial":"124"}}
{"cmd":11,"arg":{"response":true}}{"cmd":13,"arg":{"key":"123456789"}}
{"cmd":12,"arg":{"response":true}}{"cmd":5,"arg":{"outputs":[{"index":1,"d_value":512,"m_value":50,"
units":"%"},{"index":3,"d_value":680,"m_value":60,"units":"%"},{"index":4,"d_value":740,"m_value":70
,"units":"%"}]}}


ccontreras@localhost:~/rails_apps/rtu-server/telemetry-webapp
Archivo Editar Ver Buscar Terminal Pestañas Ayuda
ccontreras@localhost:~/rails_apps/rtu-server/telemetry-...
ccontreras@localhost:~/rails_apps/rtu-server/telemetry-...
[ccontreras@localhost telemetry-webapp]$ rails console
Loading development environment (Rails 4.1.4)
2.1.2 :001 > TelemetryDevice.find(1).analog_output_logs
TelemetryDevice Load (0.3ms) SELECT "telemetry_devices".* FROM "telemetry_devices" WHERE "telem
etry_devices"."id" = ? LIMIT 1 [["id", 1]]
AnalogOutputLog Load (0.1ms) SELECT "analog_output_logs".* FROM "analog_output_logs" WHERE "anal
og_output_logs"."telemetry_device_id" = ? [["telemetry_device_id", 1]]
=> #<ActiveRecord::Associations::CollectionProxy [#<AnalogOutputLog id: 1, index: 1, magnitude_valu
e: #<BigDecimal:5992a50,'0.5E2',9(27)>, digital_value: 512, measuring_unit_id: 5, telemetry_device_i
d: 1, created_at: "2014-10-13 22:50:42", updated_at: "2014-10-13 22:50:42">, #<AnalogOutputLog id: 2
, index: 3, magnitude_value: #<BigDecimal:5991bf0,'0.6E2',9(27)>, digital_value: 680, measuring_unit
_id: 5, telemetry_device_id: 1, created_at: "2014-10-13 22:50:43", updated_at: "2014-10-13 22:50:43"
>, #<AnalogOutputLog id: 3, index: 4, magnitude_value: #<BigDecimal:5990bb0,'0.7E2',9(27)>, digital_
value: 740, measuring_unit_id: 5, telemetry_device_id: 1, created_at: "2014-10-13 22:50:43", updated
_at: "2014-10-13 22:50:43">]>
2.1.2 :002 > TelemetryDevice.find(1).analog_output_logs.size
TelemetryDevice Load (0.1ms) SELECT "telemetry_devices".* FROM "telemetry_devices" WHERE "telem
etry_devices"."id" = ? LIMIT 1 [["id", 1]]
(0.2ms) SELECT COUNT(*) FROM "analog_output_logs" WHERE "analog_output_logs"."telemetry_device_
id" = ? [["telemetry_device_id", 1]]
=> 3
2.1.2 :003 >

Caso 6: Funcionamiento del comando 6.

- Valor esperado: Se esperaba que el servidor almacenara los valores de texto plano para los dos primeros sub-casos y que para el último dado que la flag "is_json" estaba establecida como verdadera y el valor de "data" no estaba bajo el formato json, un error debía ser detectado.
- Realización de la prueba: para esta prueba se enviaron los siguientes comandos desde el cliente "Telnet", el primero fue un comando 1 válido para ser aceptado en conexión, el segundo comando 13 válido para entablar la etapa de atención, después se enviaron dos comandos 6 con los valores correspondientes:
  - {"cmd":1,"arg":{"serial":"124"}}
  - {"cmd":13,"arg":{"key":"123456789"}}
  - {"cmd":6,"arg":{"is_json":false,"data":"Hola mundo"}}
  - {"cmd":6,"arg":{"is_json":true,"data":"[1,2,3,{\"hola\":\"mundo\"}]"}}
  - {"cmd":6,"arg":{"is_json":true,"data":"123"}}
- Resultado: Una vez se observó que el comando se ejecutó correctamente, se utilizó la herramienta "Rails Console" para corroborar que se creó el registro en la base de datos, y efectivamente se observó que fue así. En esta parte se puede visualizar que los dos objetos de tipo PlanTextLog fueron creados y se visualizan sus valores, así se demuestra el éxito en los tres casos:

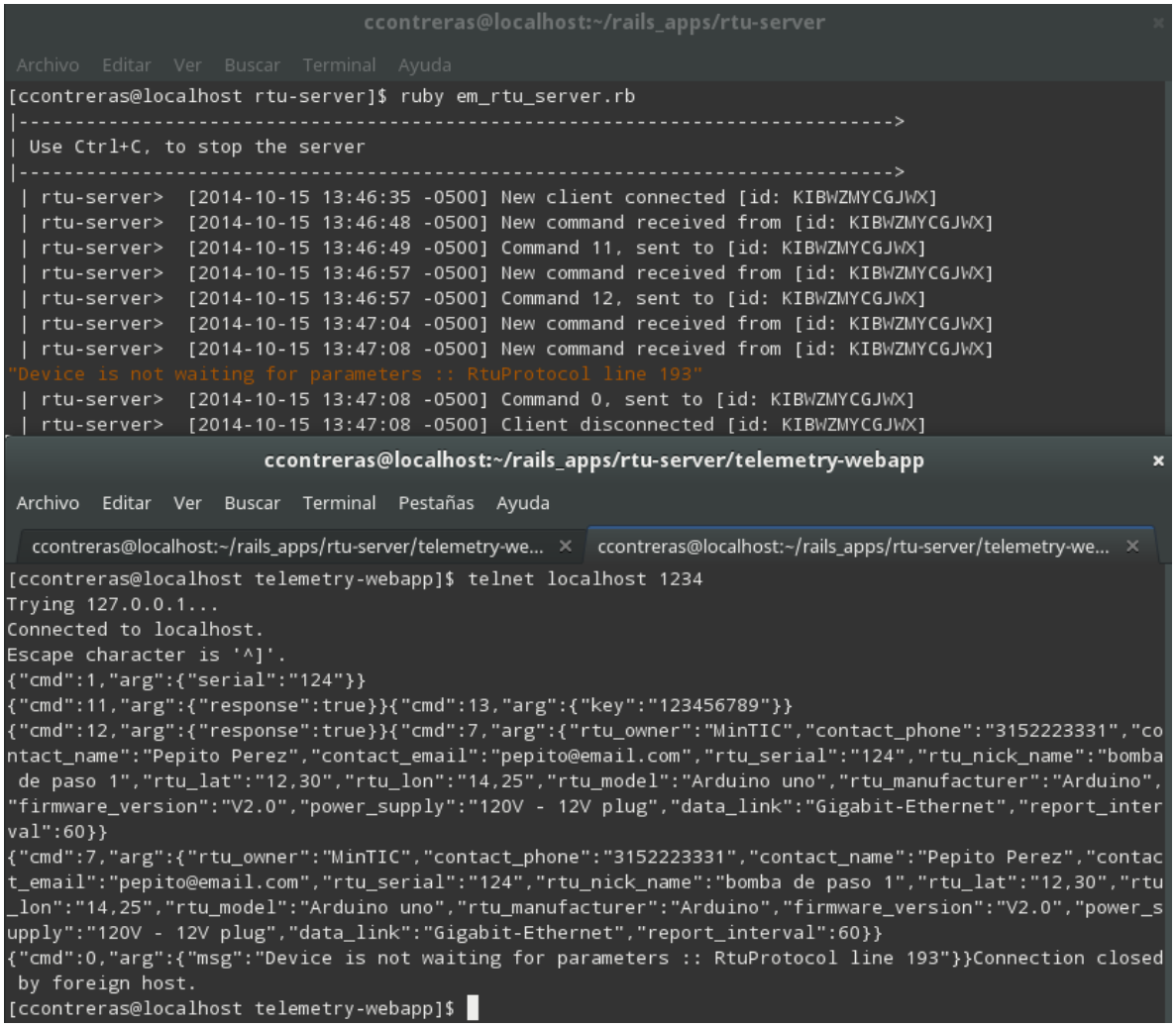




Caso 7: Funcionamiento del comando 7.

- Valor esperado: Se esperaban dos resultados, dado que se ejecutaron dos sub-casos de prueba, para el primero se esperaba que el envío de variables fuera exitoso y estas quedaran actualizadas en la RTU, en el segundo caso, se esperaba que el comando fallara teniendo en cuenta que para esta RTU la bandera “waiting_parameters” estaba establecida en falso.
- Realización de la prueba: para esta prueba se enviaron los siguientes comandos desde el cliente “Telnet”, primero fue un comando 1 válido para ser aceptado en conexión, luego el comando 13 válido para entablar la etapa de atención y finalmente se enviaron dos comandos 7 exactamente iguales con la diferencia que el primero la bandera “waiting_parameters” de la caja negra estaba en valor “falso” y el segundo no, por lo que el primero debía ser negado.
  - {"cmd":1,"arg":{"serial":"124"}}
  - {"cmd":13,"arg":{"key":"123456789"}}
  - {"cmd":7,"arg":
    {"rtu_owner":"MinTIC","contact_phone":"3152223331","contact_name":"Pepito Perez","contact_email":"pepito@email.com","rtu_serial":"124","rtu_nick_name":"bomba de paso 1","rtu_lat":"12,30","rtu_lon":"14,25","rtu_model":"Arduino uno","rtu_manufacturer":"Arduino","firmware_version":"V2.0","power_supply":"120V - 12V plug","data_link":"Gigabit-Ethernet","report_interval":60}}

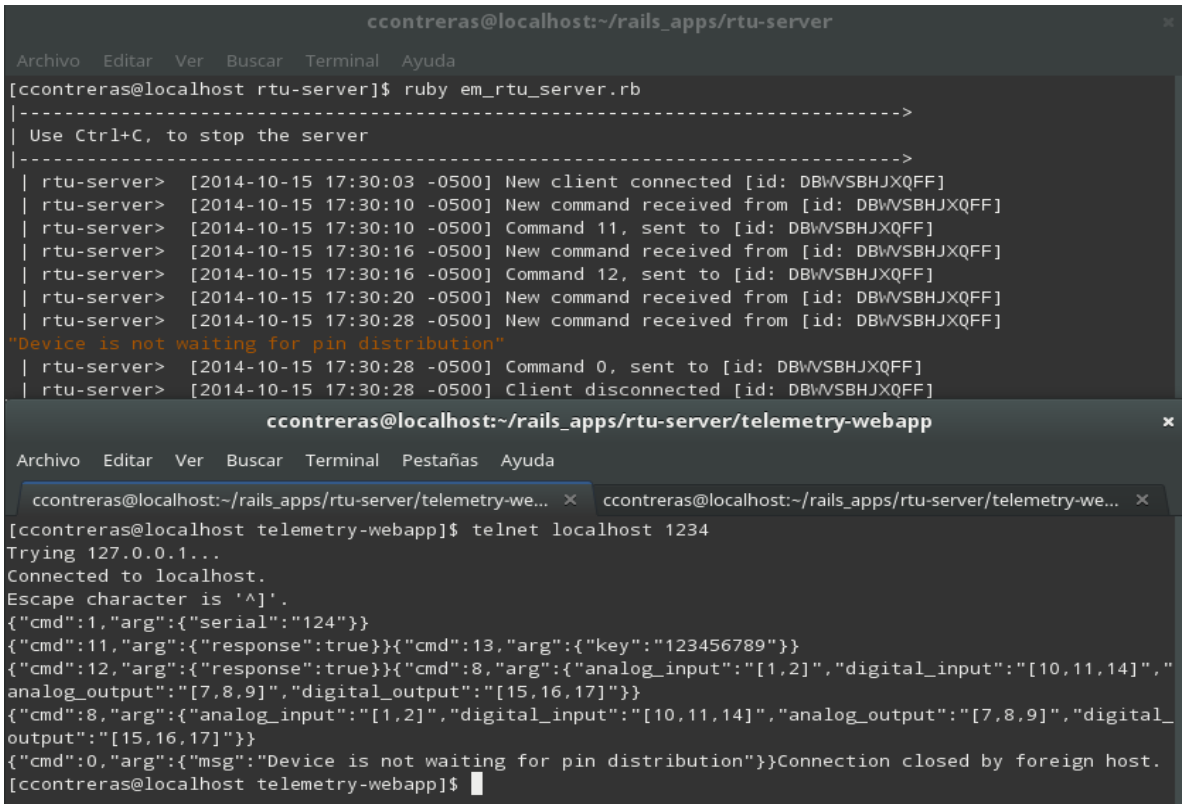
- Resultado: Una vez se observó que el comando se ejecutó correctamente, se utilizó la herramienta “Rails Console” para corroborar que se creó el registro en la base de datos, y efectivamente se observó que fue así. A continuación se adjuntan las capturas correspondientes:

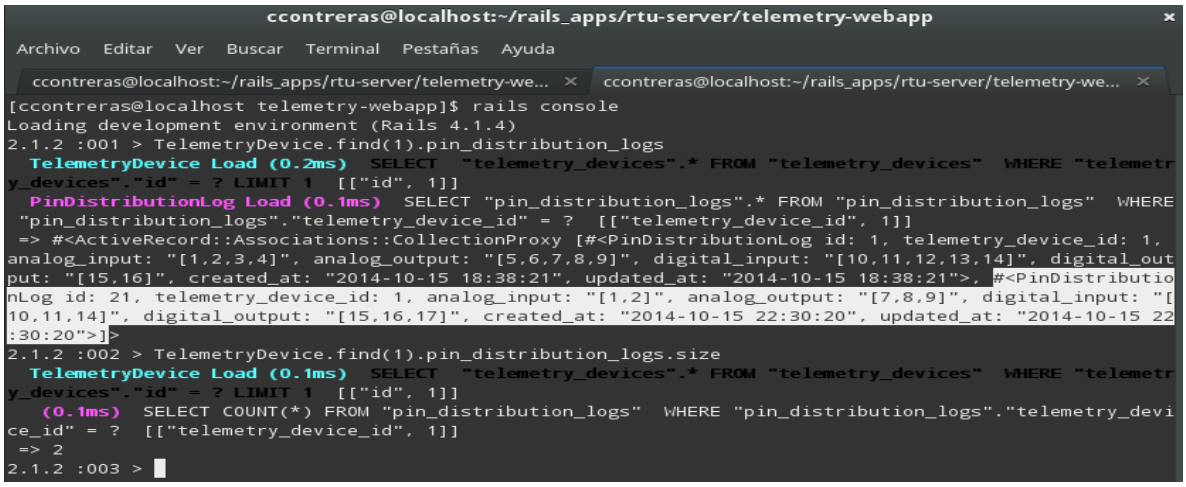


Caso 8: Funcionamiento del comando 8.

- Valor esperado: Se esperaba que el servidor registrara la nueva distribución de pines reportada por la RTU, no obstante debido a que la bandera "waiting_pin_distribution" estaba establecida con valor "falso", el comando debía ser ignorado y se debía interrumpir la comunicación inmediatamente.
- Realización de la prueba: para esta prueba se enviaron los siguientes comandos desde el cliente "Telnet", primero fue un comando 1 válido para ser aceptado en conexión, luego el comando 13 válido para entablar la etapa de atención y finalmente se envió el comando 8 con los valores correspondientes, en el primer intento se configuró la bandera "waiting_pin_distribution" con valor verdadero:

  - {"cmd":1,"arg":{"serial":"124"}}
  - {"cmd":13,"arg":{"key":"123456789"}}
  - {"cmd":8,"arg":{"analog_input":"[1,2]","digital_input":"[10,11,14]","analog_output":"[7,8,9]","digital_output":"[15,16,17]"}}
  - {"cmd":8,"arg":{"analog_input":"[1,2]","digital_input":"[10,11,14]","analog_output":"[7,8,9]","digital_output":"[15,16,17]"}}





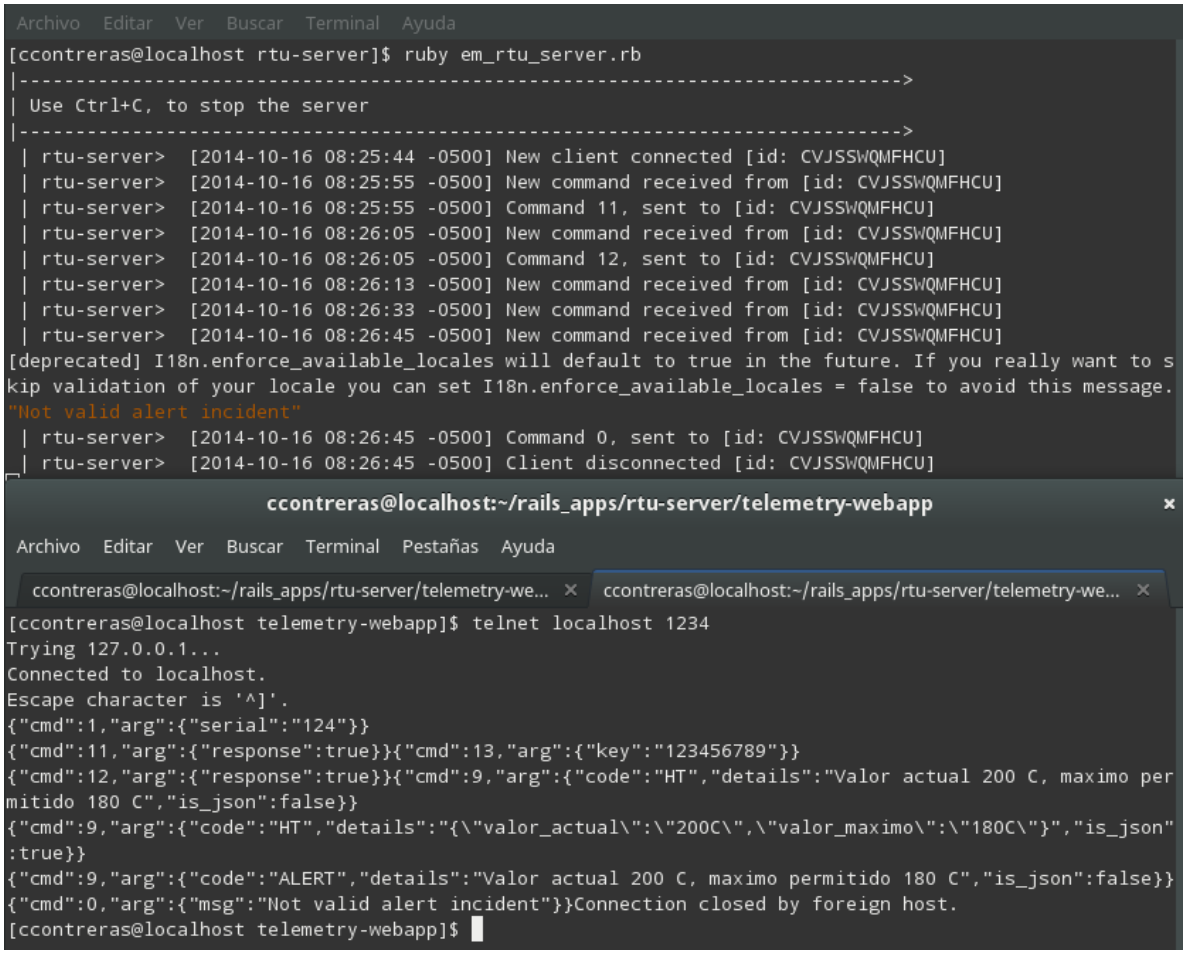
- Resultado: Una vez se observó que el comando se ejecutó correctamente, se utilizó la herramienta “Rails Console” para corroborar que se creó el registro en la base de datos, y se observó que efectivamente fue así. Como se visualiza en las capturas anteriores se corroboró el comportamiento de forma exitosa tanto en el éxito esperado como en la captura de un error de lógica.

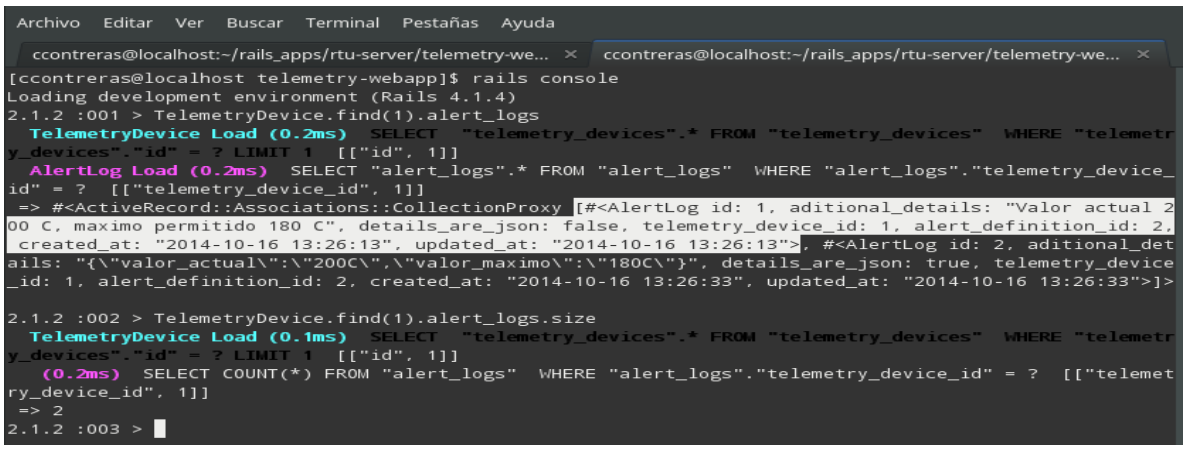
Caso 9: Funcionamiento del comando 9.

- Valor esperado: Se esperaba de ese comando que recibiera y almacenara los dos primeros y que no recibiera los dos últimos, elprimero por código inexistente y el segundo por incoherencia de flag y formato.
- Realización de la prueba: para esta prueba se enviaron los siguientes comandos desde el cliente “Telnet”, primero fue un comando 1 válido para ser aceptado en conexión, luego el comando 13 válido para entablar la etapa de atención y finalmente se envió el comando 9 para comprobar que el comportamiento fuera el esperado:

  - {"cmd":1,"arg":{"serial":"124"}}
  - {"cmd":13,"arg":{"key":"123456789"}}
  - {"cmd":9,"arg": {"code":"HT","details":"Valor actual 200 C, maximo permitido 180 C","is_json":false}}
  - {"cmd":9,"arg":{"code":"HT","details":"{\"valor_actual\":\"200C\",\"valor_maximo\":\"180C\"}","is_json":true}}
  - {"cmd":9,"arg":{"code":"ALERT","details":"Valor actual 200 C, maximo permitido 180 C","is_json":false}}
  - {"cmd":9,"arg":{"code":"HT","details":"Nada"}","is_json":true}}

- Resultado: Una vez se observó que el comando se ejecutó correctamente, se utilizó la herramienta “Rails Console” para corroborar que se creó el registro en la base de datos, y se comprobó así el comportamiento exitoso, las capturas adjuntas a continuación:





6.5 Pruebas de rendimiento

Las pruebas se realizaron con el servidor y los clientes siendo ejecutados en una misma máquina, en ambiente local, la cual cuenta con 4GB de memoria RAM, procesador Intel core i5 4200u con 2 núcleos de procesamiento a una velocidad de 2.6 Ghz y tecnología Turbo-boost que otorga 4 núcleos virtuales.

Para efectuar estas pruebas en primer lugar se construyó una base de datos de prueba que contiene datos para 1000 unidades remotas de telemetría, después se creó un emulador capaz de crear varios clientes al mismo tiempo, en este caso el emulador crea hasta 1000 clientes y los conecta al servidor, esto con el fin de determinar el comportamiento en la respuesta del servidor de cara a una cantidad considerable de clientes interaccionando de manera concurrente por unidad de tiempo, pruebas en las que la idea fue determinar la cantidad de clientes que el servidor es capaz de atender cada segundo, para este fin se emularon diversas cantidades de clientes (entre 1 a 1000 clientes) conectándose al mismo tiempo, y se tomó la lectura del tiempo que tarda el servidor en poner esa cantidad de clientes en el estado de comunicación transparente, es decir de atender dos peticiones en línea(mensajes 1 y 13), los resultados de dichas pruebas se muestran en la siguiente tabla:

Además de las observaciones anteriores se realizaron un segundo tipo de pruebas en las que la idea fue determinar la cantidad de clientes que el servidor es capaz de atender cada segundo, para este fin se emularon diversas cantidades de clientes entre 1 a 1000 conectándose al mismo tiempo, y se tomó la lectura del tiempo que tarda el servidor en poner esa cantidad de clientes en el estado de comunicación transparente, es decir de atender dos peticiones en línea, los resultados de dichas pruebas se muestran en la siguiente tabla:

Tabla 9. Resultados pruebas rendimiento

Medida	Tiempo (seg)	Clientes (unidades)
48	24	100
99	49,5	200
151	75,5	300
209	104,5	400
250	125	500
286	143	600
301	150,5	700
329	164,5	800
340	170	900
356	178	1000

Recomendación

Se plantea realizar un análisis detallado en materia de escalabilidad en número de clientes atendidos al mismo tiempo, inicialmente se sabe que dos elementos importantes que influyen en este rendimiento, el primero de ellos es el motor de bases de datos SQLite el cual no se encuentra optimizado para el manejo de concurrencia, segundo el incremento en el tiempo de procesamiento debido al lenguaje de programación dinámico Ruby, el cual genera una carga extra debido a la intervención del intérprete del mismo, además de estos factores también podría considerarse realizar mejoras al código propuesto en el prototipo, de tal manera que se optimice el rendimiento en atención a unidades de clientes por segundo, y finalmente realizar pruebas en una infraestructura con mayor poder de cómputo para determinar el comportamiento esperado en un entorno de producción más exigente pero a la vez más robusto.

7. CONCLUSIONES

- Mediante el uso de la metodología Feature Driven Development se pudo realizar un desarrollo organizado, si bien está diseñada para un numeroso grupo de implementación, utilizar la idea de desarrollo por características aún en un grupo limitado, permite tener una visualización general de lo que se quiere desarrollar, alcanzar objetivos paso a paso y refinar cuando es necesario lo que ya está hecho.

- La especificación del protocolo RTUP evidencia posibilidades de escalabilidad, dado que se realizó siguiendo una sintaxis estándar predefinida para el intercambio de información (JSON), lo cual contribuye a una fácil lectura tanto para el hombre, es decir, es fácil de entender, como para la máquina, es decir, fácil de implementar debido a la existencia de las librerías para el manejo de este estándar.

- Implementar componentes que permitan el manejo de la infraestructura red de datos de un computador es una tarea colosal, especialmente si se desea que este componente se comunique con otros, se concluye entonces que el uso del patrón de reactor, es un elemento que brinda facilidad y reduce la complejidad en el código a la hora de implementar este tipo de componentes.

- Cuando se trata del almacenamiento de datos provenientes de las unidades remotas de telemetría, la intención de su recolección radica en que se desea brindar facilidades de administración y posibilidades de generar valor agregado, por este motivo que la técnica de mapeo relacional de objetos, en la cual una tabla en la base de datos, se convierte en una clase en tiempo de ejecución y una fila de esta tabla se convierte en un objeto en tiempo de ejecución, permite que el desarrollo se centre en la lógica de estas tareas y no en los temas técnicos de conexión y consultas hacia los motores de bases de datos.

- Se concluye que la construcción del prototipo propuesto constituye un paso exitoso hacia la masificación de la telemetría y los sistemas de control, y hace parte de la tendencia global conocida como el internet de las cosas (IoT), ya que mediante el prototipo se presenta una solución que se caracteriza por dar prioridad a la convención sobre la configuración, y que en términos técnicos es de fácil implementación, no requiere hardware especializado y es portable a varios sistemas operativos, lo cual expande las posibilidades de su uso en ambientes de producción.

8. RECOMENDACIONES

- Expandir las características de los comandos abiertos del RTUP como asignación de nuevas alarmas.

- Implementar en la interfaz web un módulo referente al análisis de los datos obtenidos de las cajas negras que puede contemplar gráficas de tendencias.

- Implementar encriptación para el envío de información entre el servidor y las unidades remotas de telemetría.

- Implementar funcionalidad para exportar datos desde la interfaz TelemetryWebApp.

- Expandir las características del protocolo, añadiendo nuevos comandos que conserven el estándar propuesto y solucionen nuevas necesidades que puedan presentar con variados diseños de unidades remotas de telemetría.

- Realizar pruebas en ambiente real con unidades remotas de telemetría.

BIBLIOGRAFIA

Apache Software Foundation. «Apache MINA». 2003-2012. [En línea] [Consultado 20/04/2014]. Disponible: http://mina.apache.org/

Arquitectura Cliente Servidor. Capítulo 5. Universidad de las Américas Puebla. 2008. [En línea] [Consultado 22/03/2014]. Disponible: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/marquez_a_bm/capitulo5.pdf

A. Tanenbaum. Redes de Computadoras. México D.F. Pearson, Prentice Hall. 2010. Cap 1. P.41.

Cisco. «Internet of Things». [En línea] [Consultado 31/05/2014]. Disponible: http://www.cisco.com/web/solutions/trends/iot/overview.html

«Event Machine». [En línea] [Consultado 20/04/2014]. Disponible: http://rubyeventmachine.com/

F. Palmer. «A Practical Guide to Feature-Driven Development». Prentice Hall 2002. [En línea] [Consultado 08/03/2014]. Disponible: http://www.nebulon.com/articles/fdd/download/fddprocessesA4.pdf

GALEANO, Gustavo. Programación de Sistemas Embebidos en C. Colombia. Alfaomega, 2009. P.2.

Git Hub Wikis. «Event Machine Introduction». 2014. [En línea] [Consultado 20/04/2014]. Disponible: https://github.com/eventmachine/eventmachine/wiki/General-Introduction

J. Postel. «RFC 760» DoD Standard Internet Protocol. Enero 1980. [En línea] [Consultado 22/03/2014]. Disponible: http://tools.ietf.org/html/rfc760

J. Postel. «RFC 1122» Comunication Layers. Octubre 1989. [En línea] [Consultado 22/03/2014]. Disponible: http://tools.ietf.org/html/rfc1122

J. Postel. «RFC 791» Internet Protocol. Septiembre 1981. [En línea] [Consultado 22/03/2014]. Disponible: http://tools.ietf.org/html/rfc791

J. Postel. «RFC 1180» A TCP/IP Tutorial. Septiembre 1991. [En línea] [Consultado 22/03/2014]. Disponible: http://tools.ietf.org/html/rfc1180

Ruby EventMachine. «Reactor Pattern». [En línea] [Consultado 24/04/2014]. Disponible: https://www.igvita.com/2008/05/27/ruby-eventmachine-the-speed-demon/

Ruby on Rails Guides. «Active Record Basics». [En línea] [Consultado 21/06/2014]. Disponible: http://guides.rubyonrails.org/active_record_basics.html#what-is-active-record-questionmark

Ruby on Rails Guides. «CRUD – Reading and Writing Data». [En línea] [Consultado 22/06/2014]. Disponible: http://guides.rubyonrails.org/active_record_basics.html#crud-reading-and-writing-data

Ruby on Rails Guides. «Object Relational Mapping». [En línea] [Consultado 31/05/2014]. Disponible: http://guides.rubyonrails.org/active_record_basics.html#object-relational-mapping

S. Goyal. «Major Seminar on Feature Driven Development» Technical University Munich. Agosto 2007. [En línea] [Consultado 08/03/2014]. Disponible: http://csis.pace.edu/~marchese/CS616/Agile/FDD/fdd.pdf

«Socket». [En línea] [Consultado 20/04/2014]. Disponible: http://xsocket.org/

The Netty Project. «Netty». 2014. [En línea] [Consultado 20/04/2014]. Disponible: http://netty.io/

Twisted Matrix Labs. «Twisted». 2014. [En línea] [Consultado 20/04/2014]. Disponible: https://twistedmatrix.com/trac/

W. Bolton. Mecatrónica. Sistemas de control electrónico en la Ingeniería Mecánica y eléctrica. Quinta edición. México D.F. Alfaomega, 2013. Cap 17. P.360 – 374.