DISEÑO DE UN SISTEMA EMBEBIDO QUE PERMITE HACER SEGUIMIENTO Y CONTROL A VEHÍCULOS DE TRANSPORTE PÚBLICO BASADO EN BRT

WILSON RAMIRO PRADA CAMACHO

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2017

DISEÑO DE UN SISTEMA EMBEBIDO QUE PERMITE HACER SEGUIMIENTO Y CONTROL A VEHÍCULOS DE TRANSPORTE PÚBLICO BASADO EN BRT

WILSON RAMIRO PRADA CAMACHO

Trabajo de grado para optar al título de
Ingeniero de Sistemas

DIRECTOR:
GABRIEL RODRIGO PEDRAZA FERREIRA
Ph.D. Ciencias de la computación

CO-DIRECTOR:
DIEGO FERNANDO ACOSTA ORTIZ
Ingeniero de Sistemas

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2017

DEDICATORIA

A mis padres Ramiro Prada Anaya y Rosalina Camacho Leal quienes han sido un gran ejemplo durante toda mi vida, me han mostrado el camino correcto y me han confortado en mis momentos de debilidad.

A mis hermanos Gabriel, Carlos y Juan quienes han sido de gran apoyo durante mi periodo estudiantil y carrera universitaria.

A todos mis amigos y compañeros especialmente a todos aquellos con los que he compartido alegrías y enfrentado juntos desafíos saliendo victorioso con grandes lecciones de vida.

AGRADECIMIENTOS

A mis padres que me apoyaron durante mi infancia, adolescencia y ahora en mi madurez, estando presentes en cada una de mis etapas y en cada ocasión de mi vida académica respaldándome.

A mis hermanos y familiares que ayudaron en mi formación como persona, de otra persona no sería el ser humano que soy actualmente.

A los docentes durante toda mi vida académica, especialmente durante mi vida universitaria, con un énfasis especial al profesor Gabriel Rodrigo Pedraza quien decidió compartir conmigo y mis compañeros su experiencia y guía en este trabajo de grado.

A aquellos compañeros con quienes trabaje hombro a hombro en sacar adelante este macroproyecto del cual hago parte; Andrés Pereira, Giovanni Flórez, Rafael Sánchez, Antonio Cortés y Ángel Valbuena, gracias por este último año de trabajo constante.

A mi codirector Diego Fernando Acosta quien compartió su vasta experiencia en el área de sistemas embebidos y por apoyarme en mi capacitación en el área, también a los compañeros de CURSOL por compartir y hacer más ameno el espacio de aprendizaje.

A la Universidad Industrial de Santander por la inmensa oportunidad que me dio de forjarme como un profesional y como ser humano de bien.

Y en general a todos mis amigos y conocidos que han compartido tantas experiencias y momentos de mi vida. Gracias a todos

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

RESUMEN

TÍTULO: DISEÑO DE UN SISTEMA EMBEBIDO QUE PERMITE HACER SEGUIMIENTO Y CONTROL A VEHÍCULOS DE TRANSPORTE PÚBLICO BASADO EN BRT[*]

AUTOR: PRADA CAMACHO, WILSON RAMIRO[**]

PALABRAS CLAVE: GNSS, SISTEMAS EMBEBIDOS, BRT, INTEL GALILEO, MONITOREO.

DESCRIPCIÓN:

Como alternativa de transporte masivo de pasajeros en áreas urbanas de gran concentración de Latinoamérica y el mundo, se ha implementado un modelo de transporte conocido como BRT (Bus Rapid Transit), o buses de tránsito rápido. De las muchas problemáticas con las que cuenta este modelo de transporte se quiere atacar la falta de información, se pretende que al atacar a la problemática de la información mejore la toma de decisiones tales como distribuir los autobuses de maneras más eficientes en los recorridos, también se puede tener una mejor idea de la distribución de pasajeros, mediante el estudio de la prestación de servicios.

Se propone un macroproyecto con un grupo de estudiantes de último nivel de la Universidad Industrial de Santander y el director de proyecto, dicha propuesta es un conjunto de trabajos de grado que apuntan a una solución integrada con miras a solucionar la problemática de falta de información oportuna. Cada uno de los trabajos de grado trabaja un aspecto diferente, entre ellos la recolección de datos de los vehículos, una plataforma back-end que presta servicios de almacenamiento y tratamiento de datos, y una aplicación para los operarios del sistema de buses. El enfoque de este trabajo de grado es la recolección de datos de los buses que forman parte de un BRT, para ello se realiza el diseño de un prototipo de sistema embebido y se implementa dicho prototipo montado sobre una tarjeta Intel Galileo. La finalidad de este prototipo es recibir datos de GNSS y darles un tratamiento básico para luego transmitirlo a la plataforma back-end. El aporte más importante de este trabajo de grado es el diseño de una arquitectura flexible y extensible es decir permite reemplazar funcionalidades debido al diseño desacoplado y facilita la agregación de nuevos sensores.

[*] Trabajo de grado

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería de Sistemas. Director: Gabriel Rodrigo Pedraza Ferreira. Co-Director: Diego Fernando Acosta Ortiz

ABSTRACT

TITLE: DESIGN OF AN EMBEDDED SYSTEM THAT ALLOWS TO MAKE MONITORING AND CONTROL OF PUBLIC TRANSPORT VEHICLES BASED ON BRT[*]

AUTHOR: PRADA CAMACHO, WILSON RAMIRO[**]

KEYWORDS: GNSS, EMBEDDED SYSTEM, BRT, INTEL GALILEO, MONITORING.

DESCRIPTION:

As an alternative to mass transit of passengers in urban areas of high concentration in Latin America and the world, was implemented a model of transport known as BRT (Bus Rapid Transit). Of the many problems with which this model of transport is concerned, we want to attack the lack of information, it is intended that when attacking the information problem improves decision making such as distributing buses in more efficient ways in the routes, you can also have a better idea of the distribution of passengers, through the study of the provision of services.

It is proposed in a macroproject with a group of students of last level of the Industrial University of Santander and the project director, the proposal is a set of degree projects that point to an integrated solution with a view to solving the problem of the lack of timely information. Each of the degree projects works a different aspect, including data collection of vehicles, a back-end platform that provides data storage and processing services, and an application for bus system operators. The focus of this degree project is the collection of data of the buses that form part of a BRT, for this is realized the design of a prototype of embedded system and this prototype is implemented mounted on an Intel Galileo card. The purpose of this prototype is to receive GNSS data and give them a basic treatment for then transmit it to the back-end platform. The most important contribution of this degree project is the design of a flexible and extensible architecture, That is to say, it allows to replace functionalities due to the decoupled design and facilitates the aggregation of new sensors.

[*] Degree work

[**] Faculty of Physico-Mechanical Engineering. School of Systems Engineering. Director: Gabriel Rodrigo Pedraza Ferreira. Co-Director: Diego Fernando Acosta Ortiz

INTRODUCCIÓN

En la actualidad uno de los ejes de desarrollo urbano en el cual se está invirtiendo mucho empeño es el transporte, específicamente el transporte masivo de pasajeros en áreas urbanas. Como una solución a esta problemática de trasladar un flujo masivo de usuarios en zonas urbanas surgió el modelo de transporte BRT o buses de tránsito rápido, un sistema con muchas fortalezas pero aun cuenta con varias problemáticas a mejorar, entre ellas la falta de información para administrar mejor los recorridos de los buses y atender la demanda de los usuarios de manera más oportuna.

Con el fin de proponer una solución a la problemática a la falta de información con la que cuenta un sistema de transporte basado en buses de tránsito rápido, en un esfuerzo conjunto varios estudiantes de la Universidad Industrial de Santander queremos proponer toda una infraestructura bien organizada que ofrece la posibilidad de monitoreo de los buses, brindando información en tiempo real. Bajo ese marco de macroproyecto se cuenta con la recolección de información de los vehículos (el cual es el enfoque específico de este trabajo de grado), una plataforma Backend que recibe dichos datos y los tiene disponibles ya procesados para ciertos usuarios como servicios, y así mismo un aplicativo para operarios del sistema de transporte.

Siendo cada uno de estos un trabajo de grado separado pero que se planea integrar finalmente. La recolección de información de los vehículos se realizará mediante el uso de una plataforma embebida en cada vehículo perteneciente al sistema de buses, la finalidad del proyecto es la proposición de un prototipo de software que funcionara en la plataforma embebida para recolectar y transmitir los datos, en un primer prototipo funcional se cuenta con capturar y transmitir datos de posición geográfica (capturados con un sensor de GNSS). Para el diseño del prototipo del software de recolección y transmisión de datos se pensó en que fuera un software flexible en sentido que se puedan reemplazar funcionalidades con facilidad y extensible para en un futuro añadir nuevos sensores adicionales aparte del sensor de GNSS.

1. PLANTEAMIENTO Y JUSTIFICACIÓN DEL PROBLEMA

El modelo de transporte basado en buses de tránsito rápido (BRT, del inglés Bus Rapid Transit) [1] en la actualidad es un modelo a gran escala que más y más ciudades alrededor de Colombia y el mundo han ido adoptando para responder a la gran demanda de transporte de pasajeros en áreas urbanas con alta concentración. Las ventajas que ofrece adoptar este sistema de transporte frente a otros como transporte sobre rieles o cableado por ejemplo son la velocidad de implementación, ya que el tiempo de planeamiento y puesta en marcha es mucho más corto, el costo tanto de construcción, mantenimiento y de operación es considerablemente más bajo y la conectividad de las redes o rutas es más sencilla ya sea que se vaya a construir o simplemente no haya un bus que cubre la ruta, en muchos casos usar buses alimentadores soluciona estos problemas, a diferencia del sistema de transporte sobre rieles que en algunos casos la infraestructura requiere la excavación para la construcción de túneles la cual es costosa en tiempo y dinero [2].

Frente a todas esas ventajas hay aún muchas falencias que opacan y obstaculizan el progreso de estos sistemas en la mayoría de las ciudades donde se ha implementado. Dichas falencias surgen algunas por falta de infraestructura, falta de vehículos y otras que son de orden administrativo, es decir que pese a contar con los vehículos más sofisticados, las estaciones, rutas y paradas adecuadas, que podrían cumplir teóricamente de manera aceptable las necesidades requeridas, no lo están haciendo y generan uno de los focos de inconformismo de este modelo de transporte a nivel global.

Esta última razón es producto de un problema de falta de información, con la información adecuada se podría mejorar la toma de decisiones tales como distribuir los autobuses de maneras más eficientes, también se puede tener una mejor idea de la distribución de pasajeros, mediante el estudio de la prestación de servicios. Este trabajo de grado está orientado a solucionar la problemática de información clave para la toma de decisiones y la propuesta inicial es hacer

seguimiento de la posición geográfica en tiempo real de los autobuses que hacen parte del sistema.

Sea válida la aclaración que se está proponiendo toda una infraestructura dispuesta a dar soporte a la problemática de la falta de información en los sistemas de transporte basados en BRT, En un esfuerzo conjunto bajo el marco de un macroproyecto, siendo mi contribución al macroproyecto la parte de recolección de datos a través de un sistema embebido, datos que serían algunos de los principales insumos a otras partes del macroproyecto, como aquella cuyo trabajo que se enfoca en brindar un servicio de base de datos y consecuente a esta hay otro proyecto para crear una aplicación destinada a los operadores del sistema de transporte. Finalmente hay una última parte que puede considerarse perteneciente al macroproyecto, dicho Trabajo de grado estudia el problema de integración continua, usando este macroproyecto como caso de estudio.

La recolección, tratamiento y transmisión de datos de los sensores de los buses es el enfoque de este trabajo de grado, para ello se planteara un diseño de software flexible en base a desacoplar las funcionalidades, es decir que permita modificar o reemplazar funcionalidades sin afectar las demás, adicionalmente el diseño tendrá la propiedad de ser extensible, permitiendo agregar nuevos sensores a la arquitectura.

2. OBJETIVOS

2.1. OBJETIVO GENERAL:

- Desarrollar un modelo software de un middleware embarcado en vehículos, que permita la recolección, tratamiento y comunicación de datos para el monitoreo del servicio basado en BRT.

2.2. OBJETIVOS ESPECÍFICOS:

- Identificar las necesidades de datos claves a recolectar en la prestación del servicio de transporte basado en BRT que sirvan de insumo para mejorar de la calidad del servicio.

- Proponer una arquitectura software para una plataforma hardware embebida que permita la recolección, tratamiento y transmisión de los datos adquiridos a una plataforma tecnológica que presta servicios de información.

- Implementar un prototipo software de la arquitectura propuesta anteriormente, el cual se enfocara en datos de posición geográfica.

- Realizar las validaciones pertinentes usando el prototipo que incluya recolección, tratamiento y envío de los datos (simulaciones con datos, pruebas de campo).

3. MARCO DE REFERENCIA

3.1. SISTEMAS DE NAVEGACIÓN GLOBAL POR SATÉLITE

Sistemas de navegación global por satélite o GNSS [3] (del inglés, Global Navigation Satellite System) es el término estándar genérico para referirse a sistemas que proveen posicionamiento tridimensional con cobertura global. Actualmente existen múltiples GNSS en el mundo como el GPS (Estados Unidos) [4], GLONASS (Rusia) [5], GALILEO (Europa) [6] entre otros, cada uno con su propia constelación de satélites y su propia banda para alojar frecuencias.

3.1.1. Determinación de la posición

Es necesario al menos recibir la señal de cuatro satélites para poder resolver la posición tridimensional (latitud, longitud, altitud) en tiempo real del receptor en un momento dado. Con dichas señales se miden de pseudorangos o pseudodistancias (distancia medida desde algún satélite hasta el receptor hallada calculando el tiempo que le toma viajar a la señal desde el satélite a la antena receptora) tres de estas pseudodistancias se usan para la posición y una adicional para sincronizar el reloj del receptor, para la posición bidimensional basta con tres pseudodistancias. Cabe destacar que entre más señales reciba el receptor más precisa va a ser la determinación de la posición, incluso hoy en día existen receptores que ya cuentan con la capacidad de captar 15 o más señales [7].

3.1.2. Formatos estándar GNSS

Para poder interactuar e intercambiar datos captados por receptores nacen ciertos estándares, entre los más usados hoy en día encontramos el formato RINEX [8] (Receiver Independent

Exchange Format), el NGS-SP3, el RTCM SC-104 y el NMEA 0183. A continuación, una breve descripción del formato NMEA 0183 [9] el cual se decidió usar en este trabajo de grado.

- **Formato NMEA 0183**

NMEA es el acrónimo para National Marine Electronics Association la cual es una organización para establecer estándares para comunicación de dispositivos electrónicos marítimos, después de la NMEA 0180 y NMEA 0182 como reemplazo surgió el estándar NMEA 0183 la cual ha tenido varias versiones la última conocida es la versión 4.10 en el 2012. El estándar NMEA 0183 son flujos de datos en formato ASCII, transmitidos a una tasa de 4800 bps de un hablante (emisor, p. ej. Un receptor GPS) a un oyente (receptor, p. ej. Un laptop).

El flujo de datos transmitido contiene una serie de sentencias en las cuales esta expresada la posición, la hora, datos de satélites, cifras de corrección, entre otros. Toda sentencia del estándar inicia con un "$" y cierra con "<CR><LF>", algunas de las sentencias están dedicadas al GPS o GLONASS, mientras que el resto soportan otros dispositivos como sonares, giroscopios, etc.

Para posicionamiento la sentencia de interés es aquella con el identificador "GGA" la cual contiene datos del arreglo de sistema de posicionamiento (Global Positioning System Fix Data). A continuación se encuentra la estructura general de una secuencia GGA con su respectiva explicación [10].

```
$GPGGA,hhmmss.ss,llll.ll,a,yyyyy.yy,a,x,xx,x.x,x.x,M,x.x,M,x.x,xxxx*hh<CR><LF>
$GPGGA,115417.00,4338.123456,N,07938.123456,W,1,10,01.1,095.095,M,,M,999,0000
```

Figura 1: Sentencia GGA del estándar NMEA 0183. Tomado de Introduction to GPS: The Global Positioning System *(El-Rabbany, 2002)*[10]

Tabla 1: Descripción de los campos de la sentencia GGA

$	Delimitador de inicio
GP	Identificador del hablador
GGA	Identificador de sentencia (en este caso Global Position System Fix Data)
hhmmss.ss	Hora de la posición en sistema UTC, en horas-minutos-segundos-centésimas
llll.ll	Latitud, en grados-minutos-cifras decimales que contienen segundos y centésimas
a	Norte o sur(N/S)
yyyy.yy	Longitud, en grados-minutos-cifras decimales que contienen segundos y centésimas
a	Este u oeste (E/W)
x	Identificador de calidad GPS
xx	Número de satélites usados para producir la solución
x.x	HDOP
x.x	Altura ortométrica
M	Unidad de medida de la altura
x.x	Altura geoidal
M	Unidad de medida de la altura de geoide
xxxx	ID de la estación de referencia
*****	Delimitador de cifra Checksum
hh	Checksum
<CR><LF>	Finalizador de sentencia

3.2. SISTEMAS EMBEBIDOS

Sistemas embebidos pueden ser encontrados en una amplia gama de actividades y entornos, tales como en la industria manufacturera o en la comodidad del hogar en electrodomésticos. Es característico de estos sistemas que el software reaccione a eventos generados por el hardware y emite a menudo señales de control a tales eventos, como resultado a esas señales se generan acciones específicas como iniciar llamadas telefónicas, cerrar válvulas, un despliegue del estado del sistema, por mencionar algunas acciones posibles.

Es característico en estos sistemas que por lo general la memoria es solo de lectura o en algunos casos solo almacena datos temporalmente mientras se realiza toma de decisiones, además que la gran mayoría de estos sistemas responden en tiempo real, es decir que el sistema tiene un tiempo límite para responder a un estímulo del hardware, de no cumplirse un resultado satisfactorio en ese tiempo, significa que el sistema no funciona adecuadamente. En sistemas embebidos que trabajan en tiempo real, el tiempo de respuesta es un factor crítico, ya que un fallo puede traer desde pérdidas económicas graves hasta pérdidas humanas. En la actualidad los sistemas embebidos constituyen parte de la vida cotidiana y ya existen en mayor proporción que otros tipos de sistemas, es posible encontrar en una vivienda a la mucho unas cuatro computadoras personales y unos veinte o treinta sistemas embebidos, ejemplos concretos en el hogar son los teléfonos, hornos microondas, lavadoras, etc.

Diferencias importantes entre los sistemas embebidos y otros sistemas de software son:

- Los sistemas embebidos suelen operar continuamente, es decir operan desde la activación del hardware y deben ejecutarse hasta que el hardware se desactive. Por este motivo la ingeniería del software tiene un enfoque de funcionamiento que sea fiable, otra implicación de la operación continua de un sistema embebido es el hecho que de ser necesaria una actualización debe hacerse mientras el sistema se encuentra en servicio en muchas ocasiones.
- Los sistemas embebidos que funcionan en tiempo real reaccionan a un entorno que muchas veces puede ser incontrolable e impredecible, estos sucesos inesperados conducen a que se

diseñen sistemas embebidos basados en concurrencia, con algunos procesos que se ejecutan en paralelo.

- A la hora del diseño de un software embebido hay que tener en cuenta muchas limitaciones que se convierten en requerimientos del sistema tales como limitaciones de energía (batería), espacio físico que ocupa (volumen), peso, restricciones en la cantidad de componentes (chips), entre otros.
- En ocasiones los sistemas embebidos interactúan con otros dispositivos de hardware los cuales no cuentan con controladores de dispositivo por separado.
- La protección y fiabilidad dominan el diseño de los sistemas embebidos, ya que muchos de ellos controlan dispositivos en los que un fallo puede desencadenar pérdidas humanas y económicas. Ya que la confiabilidad es un factor crítico, con frecuencia esto conduce a diseños conservadores, es decir usan técnicas que ya se han probado y demuestran ser eficientes, en vez de incursionar en técnicas de la vanguardia que pueden introducir nuevos modos de fallos.[11]

3.3. RECEPTORES DE GNSS

Un receptor de GNSS es un dispositivo que consta de la unión de hardware y software, este permite determinar la posición, velocidad y tiempo preciso (PVT, del ingle Position, Velocity and Time) en base a las señales difundidas por los satélites, debido al movimiento continuo de los satélites respecto al receptor, este tiene que estar en constante adquisición y seguimiento de dichas señales de los satélites que estén a su vista. Un diferencial de tiempo es tomado entre el punto de partida de la señal hasta llegar al receptor, esta diferencia de tiempo se transforma en una gama falso, el "pseudorango", multiplicándose por la velocidad de la luz en el vacío, $R = c \times dt$. La pseudodistancia podría ser vista como una estimación muy aproximada de la verdadera [12].

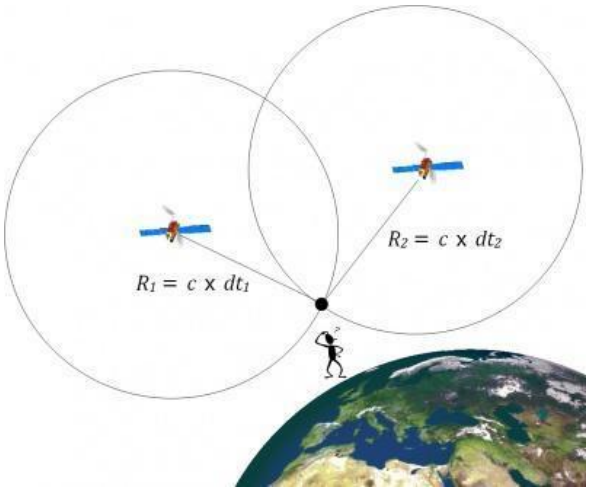

Figura 2: El Receptor determina su solución PVT mediante el procesamiento de las señales transmitidas por un conjunto de satélites a la vista. Tomado de Navipedia [12]

Un receptor convencional está conformado por una antena que está enlazada a una serie de circuitos, Trabajando juntos hardware y software. Un ejemplo de configuración común para un receptor consta de una antena, un chip RF front-end los cuales alimentan el resto de chips de correlaciones digitales. Subsecuente a ello hay una serie de pasos de procesamiento controlados a través de interfaces desde el ASIC hasta el procesador central. Como resultado de este proceso obtenemos una estimación de posición, velocidad y tiempo preciso (PVT). A continuación un diagrama que representa la configuración descrita [13].

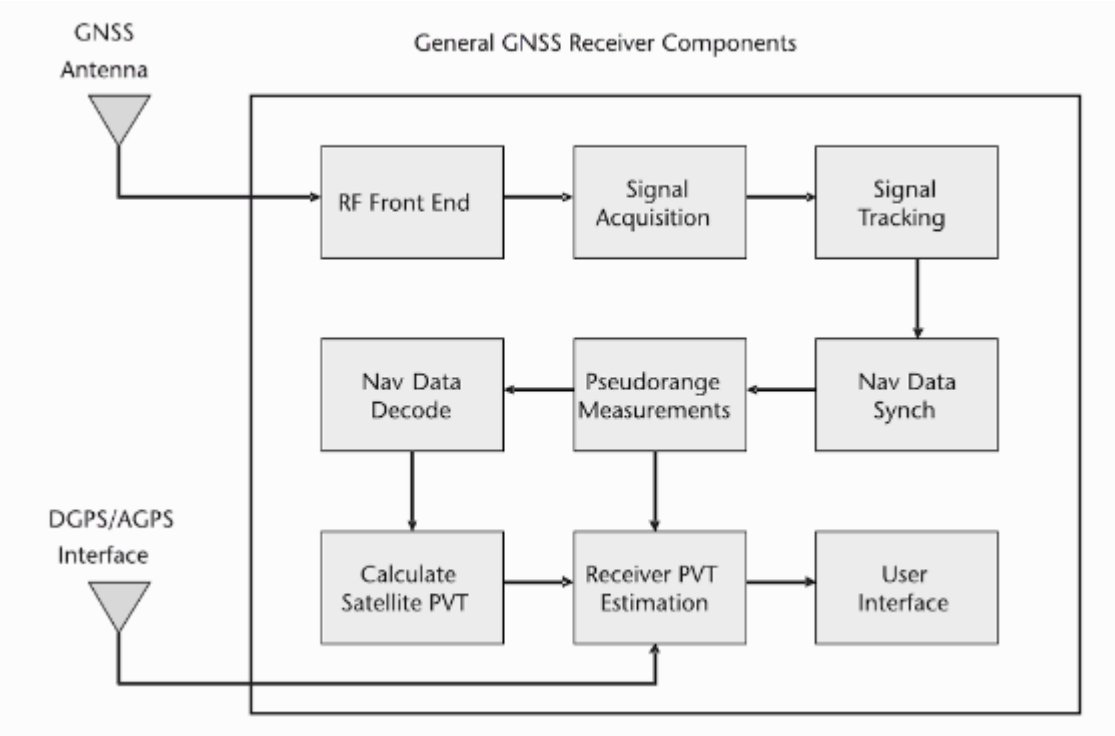

Figura 3: diagrama en bloque de un receptor GNSS convencional. Tomado de GNSS Applications and Methods *(Gleason & Gebre-Egziabher, 2009)*[13]

3.4. ARQUITECTURA DEL SOFTWARE

La arquitectura del software es disciplina que estudia cómo se estructura un sistema de software, y como la estructura influye directamente en sus propiedades tales como rendimiento, seguridad, disponibilidad, mantenibilidad, etc. El objetivo de la arquitectura de software es entender cómo debe organizarse un sistema y como se debe diseñar la estructura global del mismo. Las entradas de este proceso de diseño son los requerimientos del sistema, ya que hay que identificar los componentes del sistema y cómo se relacionan entre ellos, la salida del proceso es un modelo de la organización del sistema, un conjunto de componentes y sus mecanismos de comunicación. Hay arquitecturas que pueden describir individualmente una aplicación, sus componentes y cómo se comunican esos componentes o incluso arquitecturas tan complejas que describen otros sistemas,

aplicaciones y componentes, las relaciones de estos y además cómo se distribuye la carga de cómputo en diferentes ordenadores que pueden estar incluso en otras empresas.

Generalmente se afirma que los componentes individuales en los que se divide un programa o un sistema representan requerimientos funcionales en su mayoría, los requerimientos no funcionales dependen de la arquitectura del sistema, es decir de cómo los componentes se organizan y se comunican entre sí, en algunos sistemas solucionar un requerimiento no funcional se puede realizar también con componentes individuales. Algunas de las ventajas del diseño y documentación del software de maneras explícitas son:

- Comunicación con los participantes: dado que la arquitectura del software es una representación de alto nivel del sistema, facilita discusiones en grupos con amplio número de participantes de manera muy enfocada y objetiva.
- Análisis del sistema: con el diseño de la arquitectura del sistema se puede analizar desde etapas tempranas si el sistema puede cubrir con algunos requerimientos críticos tales como rendimiento, fiabilidad y mantenibilidad.
- Reutilización a gran escala: Por lo general arquitecturas de sistemas con requerimientos similares presentan la oportunidad de reutilizar modelos que han sido probados exitosamente. haciendo posible que arquitecturas probadas previamente se hagan un posible patrón que brinde una solución a una problemática.

En cuanto a este trabajo de grado uno de los objetivos es la proposición de una arquitectura de software embebido, para la cual también se han planteado algunos patrones de diseño de arquitecturas de sistemas en tiempo real, entre ellos destacan:

- Observar y reaccionar, en este patrón hay un conjunto de sensores desplegados y en espera de uno o varios eventos específicos, de tal manera que al cumplirse las condiciones necesarias el sistema inicia un proceso en respuesta del evento.
- Control ambiental, este patrón cuenta con sensores que proporcionan información del entorno al sistema y un conjunto de actuadores que de ser necesario reciben señales del sistema para realizar cambios en el entorno.

- Segmentación de proceso, o procesos pipeline el cual convierte un flujo de datos de entrada a otro tipo de representación que sea necesaria según el objetivo del software para hacerlos más fácil de analizar y procesar, por medio de un conjunto de transformaciones consecutivas, esos nuevos datos generados podrían ser almacenados o también podrían terminar como una instrucción para un actuador.

Respecto a las tareas que está destinada la arquitectura propuesta en este trabajo de grado es obtener flujos de datos de uno o varios sensores, convertir dichos datos y seguido enviarlos o almacenarlos según sea la necesidad, algo similar a lo que describe la segmentación de procesos , añadiendo a eso un valor agregado por las siguientes características:

- Flexibilidad: dado que la arquitectura propuesta está ideada para que cada componente esté tan desacoplado como sea posible de los demás, facilitando así que a futuro este componente sea removido, reemplazado o mejorado futuramente, por ejemplo si fuera necesario cambiar el componente de envío de mensajes a la plataforma cloud, basta con cambiar una clase Java.
- Mantenibilidad: En tanto la arquitectura propuesta es bastante flexible gracias al diseño de componentes con débil acoplamiento también es fácil de realizar el mantenimiento
- Extensibilidad: Nuevas funcionalidades pueden ser fácilmente añadidas, por ejemplo, el diseño tolera con facilidad la agregación de nuevos tipos de sensores, o que se realice tareas diferentes con los mensajes armados [14].

3.5. SOFTWARE PARA DISPOSITIVOS EMBEBIDOS

El software para dispositivos embebidos es diferente al software que se hace para los ordenadores corrientes, quizás la principal característica que en qué difiere al software convencional es que no hay o tiene muy pocas funciones controladas a través de interfaz humana, sino a través de interfaces con otras máquinas. Otro desafío que enfrenta el software destinado a estos

dispositivos es que con recursos limitados debe poder realizar tantas tareas como sea posible, en cuanto a interfaz de usuario de ser necesaria suele ser la consola.

3.5.1. Sistemas operativos dirigidos a dispositivos embebidos

Dispositivos Embebidos tales como la placa Intel Galileo que es la arquitectura objetivo, tienen unos requerimientos tales como el poco espacio de almacenamiento, la reducción del consumo de memoria como sea posible, además que no es necesario una interfaz gráfica de usuario (GUI). Un sistema operativo con alta compatibilidad con la tarjeta es Proyecto Yocto.

- **Proyecto Yocto**

Es un proyecto de colaboración de código abierto que proporciona plantillas, herramientas y métodos para ayudar a crear sistemas personalizados basados en Linux para productos embebidos. Fue fundada en 2010 como una colaboración entre muchos fabricantes de hardware, proveedores de sistemas operativos de código abierto y las compañías de electrónica, con el fin de poner orden al caos de desarrollo de Linux embebido [15].

3.5.2. Librerías y aplicaciones para software embebido

En general todo software que tiene como objetivo una arquitectura embebida, debe ser compilado de una manera especial, diferente del software convencional para ordenadores. El proceso de compilado para una arquitectura diferente al equipo host (el equipo que posee el compilador), se le llama compilación cruzada, Este procedimiento garantiza los binarios ejecutables en una arquitectura target. También es posibles encontrar en internet los binarios para ciertas aplicaciones.

Este trabajo de grado requiere una aplicación que permita la adquisición y comunicación de datos originados de un receptor de GNSS, la aplicación que se seleccionó fue RTKLIB [16].

- **RTKLIB**

RTKLIB es una librería de código abierto que tiene una serie de aplicaciones para posicionamiento estándar y preciso usando GNSS. RTKLIB está compuesto por un conjunto de aplicaciones de adquisición, análisis, post-procesamiento y comunicación de datos capturados por un sensor de GNSS. Las características de RTKLIB son:

- Soporte a algoritmos para posicionamiento estándar y preciso con GPS, GLONASS, Galileo, QZSS, BeiDou y SBAS.
- Soporte para procesado en tiempo real y post procesado de señales GNSS.
- Soporte a varios formatos, estándares y protocolos usados en GNSS, por mencionar algunos RINEX, RTCM, NMEA.
- Soporte a varios mensajes propietarios de receptores GNSS.
- Soporte a comunicación externa vía serial, TCP (IP, NTRIP, local log file y FTP/HTTP.

Las siguientes son las aplicaciones que ofrece RTKLIB junto a su debida función, además una separación entre aquellos que tienen interfaz gráfica (GUI) y aquellas que tienen interfaces para consola (CUI)

Tabla 2: Aplicaciones pertenecientes a la librería RTKLIB. [16]

Función	**GUI AP**	**CUI AP**
Lanzador de aplicaciones	RTKLAUNCH	-
Posicionamiento en tiempo real	RTKNAVI	RTKRCV
Comunicación	STRSVR	STR2STR
Post-procesamiento	RTKPOST	RNX2RTKP

convertidor a formato RINEX	RTKCONV	CONVBIN
observación y graficado de datos	RTKPLOT	-
descargador de datos GNSS	RTKGET	-
Navegador NTRIP	SRCTBLBROWS	-

Para este trabajo de grado la aplicación de RTKLIB que fue de gran utilidad para comunicar datos fue STR2STR dada la arquitectura, se necesitaba software sin interfaz gráfica, además que STR2STR ofrece varios servicios, entre ellos recibir datos de un receptor GNSS y transmitirlos a cierta frecuencia, ya sea guardándolos en un archivo de texto, o bien se puede simular un cliente TCP que envíe estos datos a un determinado puerto que fue lo que se hizo en este proyecto.

4. ESTADO DEL ARTE

Actualmente las aplicaciones que usan el geoposicionamiento van en aumento y esto se ha facilitado gracias a la gran cantidad de satélites que transmiten datos de para sistemas de navegación por satélite (GNSS). Particularmente conozco un proyecto de grado que hace pocos semestres se sustentó, por el estudiante en ese entonces Diego Acosta quien es mi actual codirector de proyecto. En su proyecto se trabajó acerca de la construcción de un sistema embebido como una propuesta económica para adquisición de datos para GNSS, se trabajó con datos en bruto (RAW data) o datos Binarios convirtiéndolos a un formato bien conocido para datos GNSS, el formato RINEX, para luego estudiar esos datos. En particular la experiencia de ese trabajo de grado fue inspiración para este proyecto.

En el mercado existen múltiples aplicaciones y compañías que ofrecen sus portafolios de servicios de monitoreo para vehículos particulares y de trabajo, Un ejemplo que podría citar es el caso de iTRAK, una compañía que desarrolla e implementa sistema de rastreos que emplean GPS para flotas de trabajo ofreciendo monitoreo 24 horas en tiempo real, con planes de cobro diarios. Esta compañía ofrece servicios hospedados en sus servidores en los cuales, y de dichos datos de los vehículos de una flota se pueden usar para monitoreo ya sea a través de informes o en un minimapa de Microsoft MapPoint.NET [17].

Existen múltiples compañías y aplicaciones que prestan servicios similares al caso de iTRAK, lo que se resalta en este trabajo de grado, es que se ofrece la posibilidad de un software con características de extensibilidad, flexibilidad y mantenibilidad, muy convenientes en un caso tan cambiante como lo es el transporte de pasajeros basado en BRT, pues el transporte basado en BRT aún tiene mucho que evolucionar y esos cambios pueden suponer cambios en las necesidades del software.

5. METODOLOGÍA

Para este trabajo de grado se hacía necesaria en la fase de desarrollo de un prototipo funcional pruebas en la plataforma embebida, la cual contaba con propiedades diferentes al equipo donde se realizaba el trabajo de programar un prototipo de la arquitectura propuesta, por eso para el proyecto se optó por usar una adaptación de la metodología de modelo de prototipos, este modelo de desarrollo permite ir construyendo prototipos con las características propuestas inicialmente, testeando, y en base a las pruebas ir evolucionando los prototipos y el mismo diseño [18].

Los pasos a seguir en el modelo de prototipos son:

- Definir tan detallado como sea posible los requerimientos del sistema.
- Hacer un diseño preliminar del sistema.
- En base al diseño propuesto hacer un primer prototipo con una aproximación a las características del producto final.
- Hacer análisis de ese prototipo con un usuario, señalando las fortalezas y falencias del prototipo, en base a ese análisis se realizan los ajustes pertinentes y se actualiza el diseño propuesto.
- Se crea un nuevo prototipo reforzando las falencias del anterior.
- Nuevamente se evalúa el desempeño del actual prototipo.
- Se repite este proceso de prototipado y evaluación hasta que se encuentre la satisfacción del usuario.
- El sistema final es el último prototipo el cual ya pasó por una serie de pruebas y tiene la aprobación del usuario, después de eso es posible que luego de la puesta en marcha haya que hacer mantenimiento en base a nuevas funcionalidades pedidas por el usuario.

Dicho estas aclaraciones sobre la metodología de modelo de prototipos, lo siguiente es explicar la adaptación que se usa en el transcurso de trabajo de grado. Se identificaron cuatro fases en el ciclo de vida del proyecto, explicadas individualmente así:

- **FASE 1: Adecuación de la infraestructura embebida**

Durante esta primera fase se propondrán e instalarán el sistema operativo y herramientas necesarias para el funcionamiento de la plataforma embebida. Además se estudiarán a fondo las especificaciones técnicas de la infraestructura embebida a utilizar para proponer las herramientas que se usarán en la labor de desarrollo.

- **FASE 2: Identificación de las necesidades de datos clave**

En esta fase se identificarán necesidades de datos que tiene el sistema, además el formato, la frecuencia con la que es pertinente captarlos y los tratamientos que se pueden realizar en la plataforma embebida antes de transmitirlos.

- **FASE 3: Diseño de la arquitectura**

Teniendo en cuenta los requerimientos que se han identificado y las especificaciones de la plataforma embebida proponer una arquitectura software que satisfaga dichas exigencias.

- **FASE 4: Implementación y validación**

En esta fase se pone hace diseño de componentes a baja escala, la codificación y pruebas para validar el prototipo y realimentar el proceso.

A Partir de la fase 3 se pone en marcha la metodología de modelo de prototipos, repitiendo las fase 3 y 4, retroalimentando el diseño de componentes especialmente y priorizando que la arquitectura nos provea Flexibilidad, mantenibilidad y extensibilidad que fue uno de los enfoques prácticos que se le debió dar como valor agregado a la aplicación. También es a considerar que respecto a la metodología propone reuniones con el usuario, en este caso particular como no será un usuario quien lo use, sino es una aplicación que funciona de manera constante (en tiempo real), siendo el caso la opinión del “usuario” es dada por compañeros que desarrollaron la

plataforma cloud, plataforma a la cual están destinados los mensajes que transmite la plataforma embebida.

Adicionalmente se hacía sesiones de orientación con el director y el codirector de proyecto para temas concernientes a la arquitectura y la infraestructura embebida, sesiones que se llevaban a cabo semanalmente con una duración de entre 1 a 2 horas. En el siguiente diagrama se puede apreciar la metodología adaptada según se describió por fases

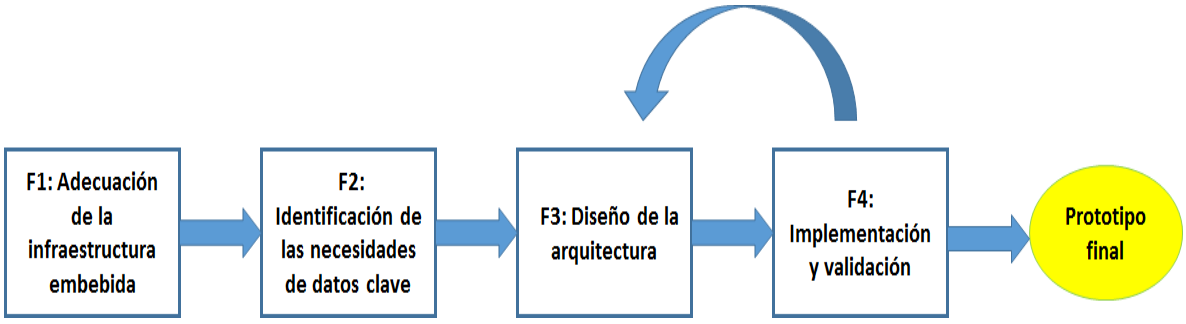

Figura 4: Etapas de la metodología adaptada basada en el modelo de prototipos

6. DESARROLLO

En el diagrama de la metodología se plantearon cuatro fases que se ejecutaron durante el ciclo de vida del proyecto, en base a ellas se describirá el procedimiento que dio fruto a este trabajo de grado.

6.1. ADECUACIÓN DE LA INFRAESTRUCTURA EMBEBIDA

La infraestructura embebida utilizada para realizar las pruebas del software de monitoreo a realizarse se propuso desde el inicio, siempre se contó con la intención que dicho dispositivo fuera una tarjeta Intel® Galileo, en cuanto a que teóricamente era suficiente para las tareas a realizarse, argumento al cual se llegó a través de una experiencia anterior un previa a este trabajo de grado, un proyecto similar llevado a cabo por el actual codirector de este proyecto, el ingeniero Diego Acosta y principalmente por la disponibilidad, ya que había la posibilidad de trabajar con ella desde etapas tempranas, dado que dicho dispositivo pertenecía a un profesor de la Universidad, quien generosamente decidió prestarlo durante la duración de este trabajo de investigación.

Después de haberse tomado la decisión de la infraestructura objetivo, el siguiente paso a seguir fue estudiar las cualidades de dicha tarjeta, la Intel® Galileo Gen 1 o simplemente Intel® Galileo.

6.1.1. Estudio de la infraestructura objetivo

- **Intel® Galileo**

La tarjeta la Intel® Galileo es una tarjeta microcontroladora basada en procesador Intel® Quark SoC X1000, un procesador de 32 bits SoC (System on Chip), es decir posee en un solo chip los componentes básicos (procesador, memorias, microcontrolador) [19].

Figura 5: vista frontal y posterior de la tarjeta microcontroladora Intel® Galileo

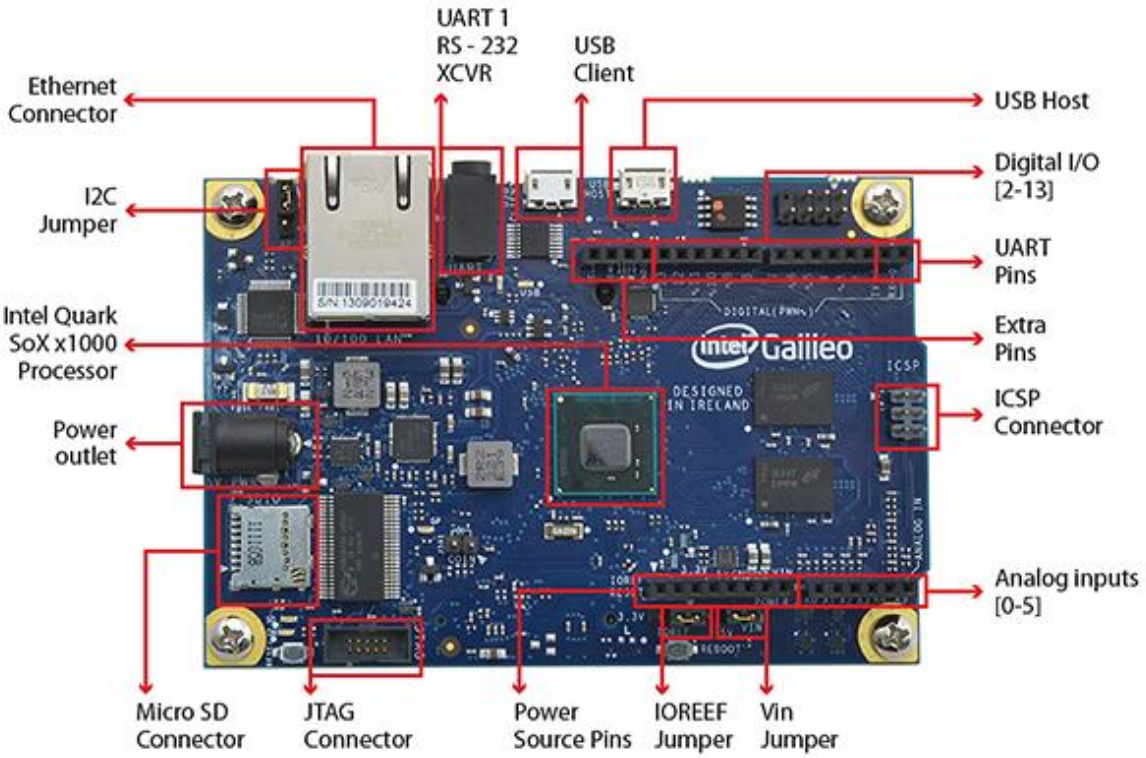

Figura 6: componentes de la tarjeta microcontroladora Intel® Galileo [20]

Las principales características de esta placa microcontroladora son:

Tabla 3: Características principales de la placa Intel® Galileo [19]

SoC	Intel® Quark X1000 - Single Core
Arquitectura	i586
Clock	400 MHz
RAM	256 MB
Almacenamiento	Interno: 8MB; Externo: MicroSD (De hasta 32GB)
Ethernet	10/100 M

puertos microUSB	puerto 2.0 host puerto 2.0 client
Alimentación	3.3V -5V, por defecto el adaptador convierte a 5V

- **Receptor** GNSS **(NV08C-CSM)**

Al igual que con la placa microcontroladora Intel Galileo una de las razones por las que se terminó escogiendo este receptor fue por la disponibilidad, otro argumento principal fue que el receptor NV08C-CSM es multi-constelación (Recibe señales de múltiples sistemas GNSS), en el caso de este sensor capta señales de GPS, GLONASS y GALILEO. No menos importante a los dos anteriores motivos y quizás el argumento con mayor peso, es que las salidas del dispositivo se presentan a través de dos interfaces de conexión UART (interfaz A y B), uno de los cuales proporciona datos RAW en un protocolo de fabricante conocido como BINR (interfaz B), con posibilidad a convertirse a formatos estándar tales como Rinex y la interfaz de salida que es de interés para este trabajo de grado, la cual proporciona datos en el protocolo NMEA 0183 (interfaz A) [21].

Figura 7: vista frontal del receptor de GNSS NV08C-CSM v4.1, interfaz UART A para salida en formato NMEA 0183

Figura 8: Receptor NV08C-CSM v4.1, interfaz UART B (BINR).

6.1.2. Instalación del sistema operativo y aplicaciones.

Una vez analizadas las características principales de la tarjeta Intel® Galileo, lo siguiente es la adecuación de la placa para que pueda ser accedida desde un computador vía ethernet, ya que la placa carece de periféricos de salida y posteriormente le pueda conectar el receptor GNSS, lo cual nos lleva en primera medida a la instalación de un sistema operativo, en este caso Yocto Linux para Intel Galileo

Previo a instalar un sistema operativo a la medida hubo una serie de procesos que tuvieron lugar antes de llegar al actual sistema operativo, los cuales fueron:

- **Probar el correcto funcionamiento de la tarjeta Intel Galileo**

- Se puede confirmar el buen funcionamiento del hardware usando la IDE de arduino, software que puede ser obtenido de la página de Intel [22].
- Se descarga la versión para el sistema operativo host, windows en mi caso, se extrae y se instala.
- Hay que verificar que el firmware de la placa galileo esté actualizado, se puede seguir el manual de referencia de actualización de firmware, seguido se realiza la conexión serial [23].
- Se prueban varios sketches de Arduino, como por ejemplo el sketch “BLINK” que hace parpadear el LED 13, es el sketch que se usa de manera genérica para confirmar la interacción entre una host y la placa galilea, es casi el equivalente a un “holaMundo”

- **Instalación de una imagen de Yocto Linux para galileo, obtenida de la página de Intel**

- Para esta primera experiencia instalando sistemas operativos en software embebidos, se usa una imagen genérica descargada de la página de Intel, siguiendo las instrucciones del "Empieza galileo" (get starting Galileo) [24].
- Siguiendo dichas instrucciones se instala la imagen obtenida en una memoria microSD, de esa manera tendremos una microSD booteable.
- Se conecta la microSD en el módulo SD de la Intel Galileo y se conecta la fuente de alimentación, seguido a la conexión de la fuente de alimentación se autoarranca el sistema operativo cargado en la microSD.

- **Establecer una conexión ethernet entre el equipo host y la Intel Galileo**

- Se realiza para tener una conexión más robusta entre los dispositivos, además que otros tipos de conexiones requieren conseguir conectores adicionales que no están incluidos en el hardware
- Requiere una conexión con un cable ethernet a la terminal ethernet 10/100 de la tarjeta galileo.
- El equipo host requiere servidor DHCP para asignar una dirección IP a la tarjeta Galileo.
- Hay varias maneras de obtener la dirección IP, que le fue asignada a la tarjeta, una de ellas es usando software para inspeccionar los mensajes del protocolo de red (wireShark por ejemplo), y se realiza filtrado por la MAC de la tarjeta, dicha MAC está siempre impresa en el puerto Ethernet de la Galileo. Otro método de obtener la IP es escaneando la tabla de direcciones ARP y comparando con la MAC.
- Una vez conocida la dirección IP, se puede tener acceso a través de SSH.

Figura 9: montaje de la placa Intel Galileo conectada al equipo host con cable ethernet y al sensor de GNSS por el USB host.

- **Verificación del funcionamiento del receptor GNSS.**

- Se realiza la conexión entre el sensor y la placa galileo, se conecta el sensor al puerto USB host.
- como ya se tiene acceso a través de SSH, a través de esta conexión podemos interactuar con el sistema operativo Yocto que está instalado en la microSD.
- Se revisa el directorio /dev en él se debe hallar una nueva conexión de nombre ttyUSB0 o un nombre similar, si esta aparece significa que está reconociendo la conexión con el sensor.
- Utilizando la aplicación "Screen" podemos verificar el flujo de datos para más seguridad, con la siguiente instrucción: Screen /dev/ttyUSB0 115200, de donde 115200 es la velocidad de transmisión
- Dependiendo de la interfaz UART que se conectó al receptor deberíamos ver un diferente flujo de datos, o bien un flujo de datos binarios o datos en protocolo NMEA 0183.

En este punto se pudo verificar el correcto funcionamiento de la placa Intel Galileo, ahora el sistema operativo que se usó en estas pruebas es un sistema genérico, con ciertas aplicaciones básicas en distribuciones embebidas de Yocto Linux, además el sistema de ficheros tiene un tamaño fijo, el cual no es demasiado grande, descontando a ese espacio lo que ocupa el sistema operativo no restan sino unos cuantos MBs los cuales son insuficientes para instalar nuevas aplicaciones, por ello es necesario construir una imagen personalizada de Yocto Usando compilación cruzada.

Esta tarea de construcción de la imagen personalizada requiere un entorno de creación para la nueva imagen, se requiere un equipo en el cual previamente se hayan instalado una serie de dependencias y librerías. Es completamente recomendable antes de la construcción de la imagen personalizada documentarse acerca de Yocto Project, OpenEmbedded y Bitbake. Más acerca de la creación de imágenes personalizadas de Yocto puede ser encontrado en la página de Intel. El conjunto de tareas a seguir para construir una imagen personalizada de Yocto está consignado a continuación.

- **Creación de imágenes personalizadas de Yocto Linux**

- Hay una serie de dependencias, aplicaciones y librerías que hay que instalar previamente como se indica en las tutoriales de Intel, entre ellas: build-essential, gcc-multilib, vim-common, uuid-dev, gawk, wget, git-core, subversión, diffstat, chrpath, texinfo, iasl, autoconf y libtool
- Se instaló bitbake, las capas de software “poky” correspondientes al proyecto Yocto Linux y las capas de software (metadatos) para Galileo.
- Se ejecutó Bitbake llamando a la receta de construcción de la imagen Yocto Linux.
- La construcción de un entorno personalizado puede demorar horas, pero la ventaja de hacerlo es que se le pueden agregar librerías y aplicaciones que son necesarias, en particular este proyecto de grado requiere java JDK 8, RTKLIB, entre otras. Además la construcción personalizada da la posibilidad de aumentarle el tamaño del sistema de ficheros el cual se fijó en aproximadamente 4GB.

Después de obtener una imagen personalizada, al igual que con Yocto obtenido de la página, se instala en una microSD, se revisa su correcto funcionamiento, se asegura la correcta instalación de las aplicaciones de la versión personalizada, se establecen las conexiones, y finalmente se verifica el correcto funcionamiento del receptor GNSS.

En caso faltara alguna librería, aplicación o dependencia por incluir, se puede buscar los binarios precompilados en internet o de ser necesario se realiza compilación cruzada para obtener los binarios dirigidos a la arquitectura del galileo (i586)

6.2. IDENTIFICACIÓN DE LAS NECESIDADES DE DATOS CLAVE A TRANSMITIR

Se realizó una visita al Centro de Control del Sistema Metrolínea (Ubicado en la estación de Provenza occidental), con el fin conocer el funcionamiento del centro de control, además de entender los recursos, las funciones del personal y las herramientas que actualmente usan. Ingenieros a cargo comentan respecto al software que fue desarrollado con algunas tecnologías algo anticuadas. Además por medio de una entrevista nos enteramos que los recorridos se programan cada cierto tiempo (semanal o quincenalmente) y se establecen discutiendo las necesidades evidenciadas. En lugar de disponer un estudio de la prestación de servicio. Respecto a la información que se puede recargar de los vehículos, comentan los ingenieros que los vehículos cuentan con varios sensores como lo son un sensor de GPS, un sensor de peso, un sensor de cerrado de puertas, etc. De los sensores mencionados solo los datos del sensor de GPS son notificados al centro de control.

En base a esas observaciones de la visita al centro de control de Metro línea, se puede concluir que la prioridad en cuestión de monitoreo de los vehículos es la información referente a posición geográfica, además la información de los demás sensores no se está usando por lo tanto se puede

proponer que en un futuro se diseñe una plataforma embebida que envié también datos de otros sensores y así mismo requiere una plataforma Backend que haga uso de estos nuevos datos.

Respecto a las necesidades que se han identificado se pueden considerar en función de entradas y salidas del sistema

6.2.1. Salidas del sistema

En medida que el proyecto fue madurando se determinó que la salida de datos de la plataforma embebida debía ser a través del protocolo Http, enviando mensajes Json, un formato de texto ligero para intercambio de datos. La manera de realizar este intercambio de información fue consumiendo un servicio que se montó en la plataforma Cloud que pertenece al trabajo de grado de dos compañeros más, quienes trabajan en la infraestructura Back-end que presta los servicios de base de datos, dicho servicio pertenece a una arquitectura Restful y es consumido por un cliente restful en la plataforma embebida.

Ese mensaje Json debe contener la siguiente información:

- un dato que identifique el vehículo, optamos por usar la placa para este propósito.
- La fecha y la hora del momento en el que el mensaje es enviado desde la plataforma.
- información de la próxima parada del recorrido del bus, que después de una concesión con los demás compañeros del macroproyecto de decidió representar cada parada en un recorrido con un entero.
- información de la posición geográfica del vehículo en un momento dado. envuelto en una clase llamada Coordenada, con los atributos latitud y longitud.
- Información del dispositivo que envía el mensaje, pensando en que a futuro no será solo un dispositivo embebido quien transmite a la plataforma cloud, sino podrían ser cientos de ellos.
- Adicionalmente a los ya mencionados, es posible que nuevas funcionalidades o sensores sean añadidas a la plataforma embebida, por lo tanto se simuló un sensor en este caso un sensor de temperatura que solo envía datos constantes.

Conforme se discutieron las necesidades el mensaje Json requerido obtuvo la siguiente estructura:

```
{
  "Placa": "ZOE101",
   "Tde":"2016/10/16 13:13:00",
   "ProximaParada":1,
   "Coordenada":{
  "Latitud": "9.113633",
  "Longitud":"-72.114842"
"CodigoDispo":"B001",
"Temperatura":"32 grados”}
}
```

En cuanto a la frecuencia de envió dicho mensaje debe ser enviado con una frecuencia entre 10 a 30 segundos, en últimas etapas del proyecto se decidió que enviar datos de la próxima parada reduce la carga de cómputo de la plataforma cloud, a cambio cuando un mensaje es entregado a la plataforma cloud, está en respuesta retorna la próxima parada, de allí que la plataforma embebida conozca ese dato.

6.2.2. Entradas del sistema

Respecto a entradas al sistema solo se cuenta solo se está contando con la entrada de datos en Formato NMEA 0183, se está captando un mensaje en este formato cada segundo, y de él se pretende extraer la coordenada solamente.

Ya conocidas las entradas y las salidas, el siguiente paso a seguir es proponer una arquitectura para un software que realice la transformación de datos y que garantice el envío, además que se pensó para que fuera posible agregarle nuevas funcionalidades o responder fácilmente a modificaciones futuras

6.3. DISEÑO DE LA ARQUITECTURA

En base a las necesidades que se identificaron, además de los requerimientos no funcionales se plantea un diseño que puede ser representado mediante el siguiente diagrama.

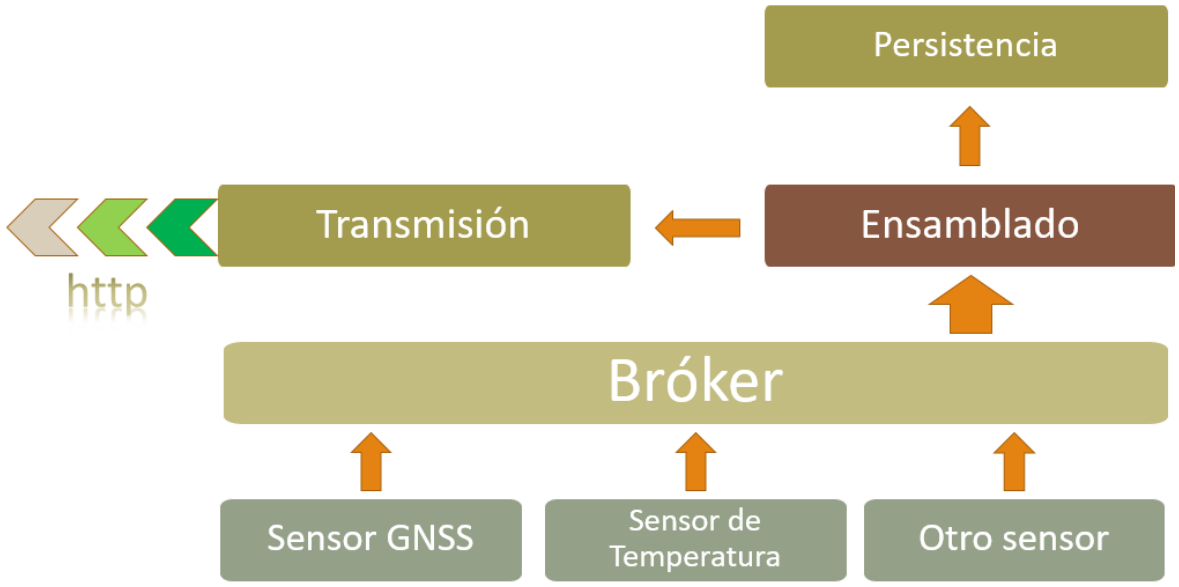

Figura 10: Representación de la arquitectura completa, en ella se puede evidenciar como fluye la información entre módulos

En el diagrama que representa el diseño se puede observar como una serie de sensores envían información a un elemento intermediario (bróker del patrón de comunicación publish/suscribe) y este a su vez le envía a un componente que ensambla los mensajes según sean los requerimientos, una vez armado un mensaje este se transmite a una plataforma back-end ubicada en un servidor remoto y para terminar se almacena el mensaje en la plataforma embebida (persistencia).

6.3.1. Comunicación entre sensores y la aplicación

Para darle flexibilidad a la arquitectura, especialmente en las circunstancias de que nuevos sensores quieran ser incluidos en el prototipo, se decidió en lugar de simplemente tomar datos a incluir, armar el mensaje y enviarlo, utilizar un patrón de comunicación en el que se desacoplara tanto como fuera posible la funcionalidad de ensamblar mensajes, de aquellas funcionalidades que generan los datos para enviar, que en el modelo vienen siendo los módulos que recogen la información de los sensores. Se ha empleado el patrón de comunicación publicador/suscriptor (pub/sub o publish/subscribe) el cual consta de uno o varios elementos que envían mensajes (publicadores), un elemento intermedio que recoge esos mensajes (bróker) y una serie de elementos que están interesados en mensajes con algunas características o tópicos de interés (suscriptores), según sea el interés de los suscriptores el bróker se encarga de hacerle llegar los mensajes que pertenezcan al tópico del suscriptor [25].

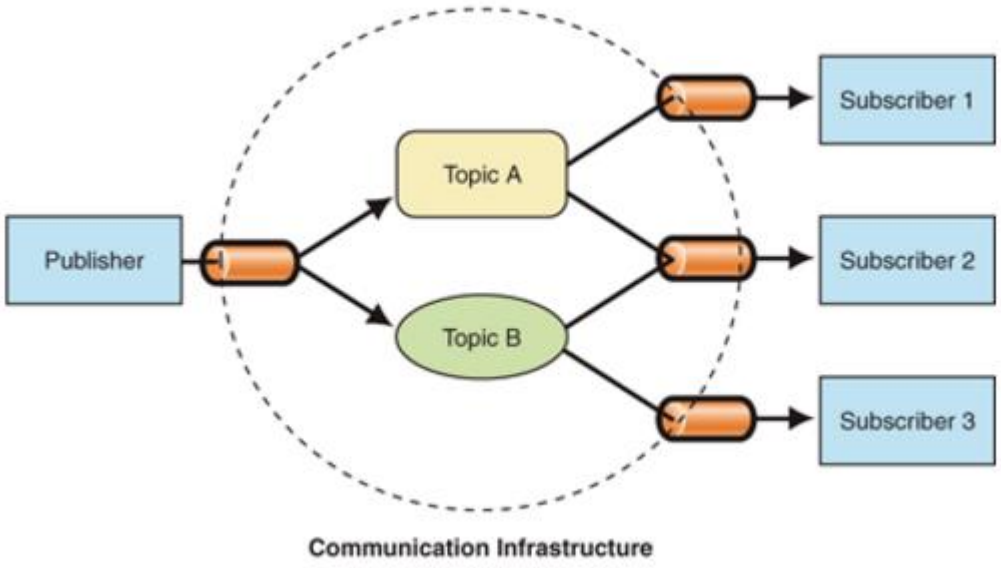

Figura 11: Patrón de comunicación Publish/Suscribe. [25]

Están disponibles algunas implementaciones del patrón pub/sus entre ellas la empleada en el proyecto “EventBus”, el cual fue desarrollado por Google y se encuentra incluido en Guava, una

librería de código abierto. Lo que se hizo en la arquitectura fue construir paquetes que representan los diferentes sensores, por ahora solo funciona el sensor de GNSS y se está simulando el funcionamiento de un sensor de temperatura, dichos sensores envían sus datos adquiridos y previamente procesados a un único suscriptor que es el elemento que ensambla los mensajes.

6.3.2. Ensamblado de mensaje

Se requirió que se armaran mensajes en formato Json (JavaScript Object Notation), un formato que representa objetos de java como cadenas de caracteres usado para el intercambio de datos, estos mensajes contienen la información de los sensores y el dispositivo que los envía, El módulo de ensamblado adquiere los datos de los sensores a través del "eventBus" que es la implementación del patrón publish/suscribe que se emplea en la comunicación con los módulos de los sensores y los añade a el mensaje Json a ensamblar. Lo interesante de usar este diseño para el módulo de ensamblado en específico es que permite añadir nuevos sensores en un futuro, sin necesidad de hacer grandes modificaciones al módulo de ensamblado para añadir los nuevos datos al mensaje Json. Otra ventaja que otorga el diseño del módulo de ensamblado es que, al estar desacoplado del resto de la arquitectura, de ser necesario en un futuro cambiar el formato de transmisión de los datos, por ejemplo, que usáramos XML en lugar de Json u otros cambios que implicara modificar dicho modulo, esto no afectaría los demás módulos de la aplicación.

6.3.3. Transmisión de mensajes a la plataforma Back-end

Seguido a armar los mensajes está la decisión de cómo comunicarse con la plataforma cloud, esto se logra enviando mensajes en formato Json. Para el envío de mensajes se crea un cliente Restful y se consume un servicio alojado en una plataforma Back-end para recibir estos mensajes con un formato de datos previamente acordado. De ser necesario cambiar el modelo de transmisión de los mensajes, sea un posible escenario que se use SOAP en lugar de REST, basta con reemplazar el módulo de transmisión.

6.3.4. Persistencia de mensajes en la plataforma embebida

Un componente adicional fue construido para almacenar los mensajes, en caso que se haga necesario tener un histórico de los recorridos del bus. Dicha persistencia se está realizando con cierta frecuencia en el dispositivo y hace uso de archivos de texto plano para registrar los mensajes que han sido construidos. Diariamente se crea un nuevo archivo de texto en el que se consignan los mensajes producidos de la actividad del dispositivo, los archivos persisten durante cierto tiempo y luego son borrados para no llenar la memoria en el dispositivo.

6.3.5. Módulo de GNSS

Los mensajes que son ensamblados y posteriormente transmitidos por la aplicación, tienen como insumo los datos de los sensores para cada sensor se ha hecho un módulo propio e independiente el cual implementa la recepción y tratamiento a los datos según sea conveniente, para un primer prototipo de la aplicación se ha incluido un sensor de GNSS, el modulo que recibe y trata sus datos debería recibir datos en formato NMEA 0183 que recibía de STR2STR quien a su vez recibe del sensor de GNSS, luego se debe establecer un mecanismo de comunicación entre la aplicación a realizarse y la aplicación STR2STR, se decidió que la comunicación se realizará estableciendo un servidor TCP en la aplicación para recibir datos NMEA que enviará STR2STR, una de las ventajas de STR2STR es que puede enviar los datos a un puerto específico como si se tratara de un cliente TCP.

La siguiente imagen representa la comunicación entre la aplicación STR2STR y la aplicación de envío de datos, el punto de contacto con STR2STR se da a través del módulo GNSS, quien recibe y da transforma los datos recibidos en formato NMEA 0183 a una representación menos ambigua como lo es una coordenada (latitud-longitud).

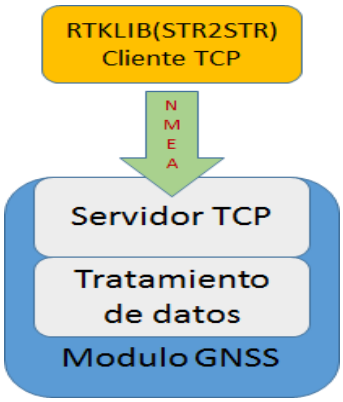

Figura 12: Comunicación entre STR2STR y el módulo GNSS de la aplicación de envío de datos.

6.4. IMPLEMENTACIÓN Y VALIDACIÓN

Como se ha estado comentando ocasiones previas, el diseño del software de tratamiento y transmisión de datos estuvo ideado con la premisa de ser tolerante a fallos, flexible y extensible, para hacer dicha aplicación se usó Java ya que es un lenguaje multiplataforma y con buena documentación.

6.4.1. Implementación del diseño propuesto

En base al diseño propuesto de la arquitectura software para envío de mensajes a la plataforma Back-end, se construye un proyecto Java, cuyos paquetes coinciden con las funcionalidades descritas en el diseño, adicionalmente los paquetes común y clases del sistema que contienen clases necesarias para el funcionamiento, además del paquete lanzador que ejecuta la aplicación.

El proyecto contiene una serie de dependencias incluidas con maven tales como guava (eventBus) para la implementación del patrón de comunicación Publish/Subscribe y javax.Json para la creación de mensajes en dicho formato.

A Continuación, una vista de los paquetes de la aplicación “EmbebidoBRT”, nombre actual que se le dio al proyecto de envío de datos a la plataforma back-end.

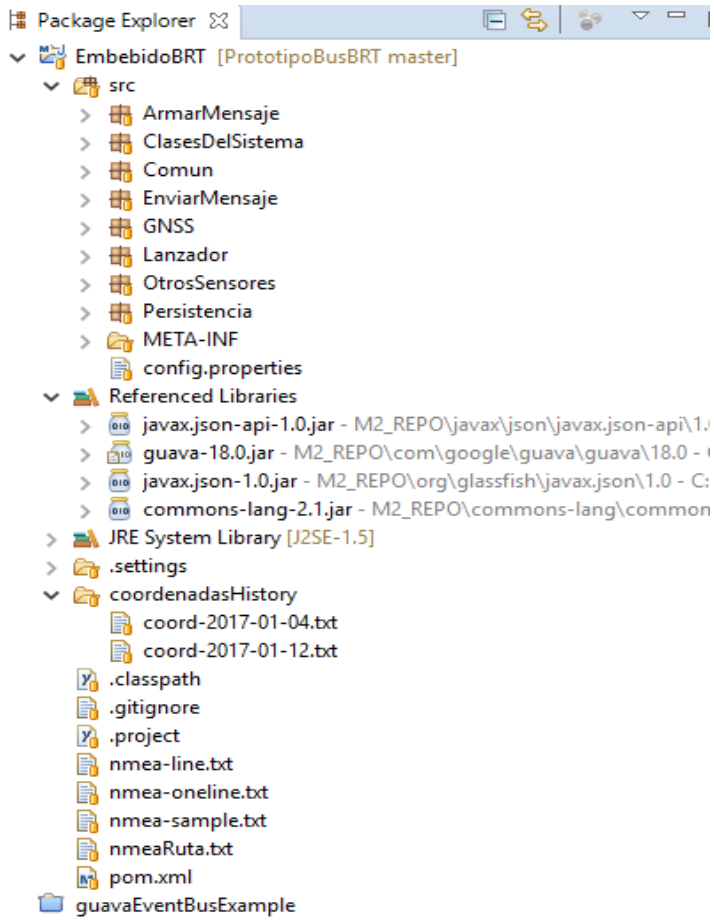

Figura 13: Vista de paquetes de la aplicación EmbebidoBRT

6.4.2. Pruebas de validación de funcionamiento

Para la validación del funcionamiento de manera correcta en el dispositivo, se realizaron dos comprobaciones. La primera para asegurarse que se está capturando el flujo de datos del sensor de manera correcta y la segunda para verificar que el software de transmisión funciona de manera correcta en el dispositivo target (tarjeta microcontroladora Intel Galileo).

- **Validación de recolección de datos GNSS en el sistema embebido**

En la recolección de datos se ha hecho un script en Linux que toma un flujo de datos Nmea 0183 y lo envía como un cliente a un puerto específico, el siguiente es un ejemplo de los datos recogidos por el sensor haciendo un énfasis en la sentencia que le es de mayor interés para el presente trabajo, véase resaltada en la sentencia los campos que representan la posición

```
root@quark:/gnss/datatools# cat sensor.txt
$GPGGA,131013.11, 3554.928,N,07602.498,W,0,,,-18.0,M,18.0,M,,*5F
$GPRMC,131013.11,V,3554.928,N,07602.498,W,00.00,000.0,300117,,,N*43
$GPGSV,1,1,01,29,00,000,19*4B
$GLGSV,1,1,00*65
$GPGSA,A,1,,,,,,,,,,,,,,*1E
$PORZD,V,999.9*2B
```

El siguiente es el script en Linux que arranca la aplicación STR2STR, tomando los datos del puerto USB2.0 host y enviándolos al puerto 9091 del servidor local.

```
#!/bin/bash
INPUT=serial://ttyUSB0:115200:8:n:1:off
OUTPUT=tcpcli://127.0.0.1:9091
exec ./bin/str2str -in "$INPUT" -out "$OUTPUT" -f 0 -t 0
```

La siguiente imagen muestra la aplicación rtkcolector.sh en ejecución, dicho script es el responsable de la captura de datos del sensor y el envió al servidor TCP de la aplicación java. Lo que se imprime en consola es la salida estándar de la aplicación STR2STR, enviado a un puerto de un servidor TCP local.

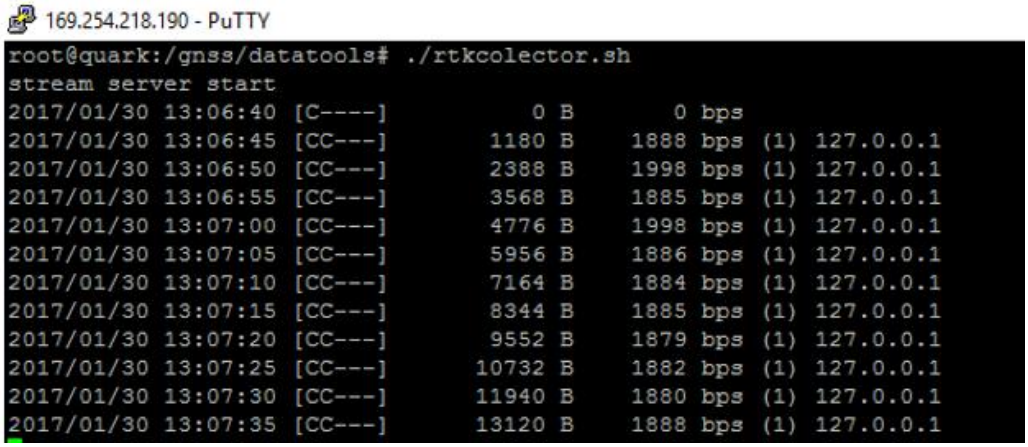

Figura 14: Salida estándar de la aplicación STR2STR, el flujo de datos se dirige hacia un puerto TCP específico

- **Validación de la transmisión de datos del sistema embebido a una plataforma back-end.**

Teniéndose la confirmación del sensor está en funcionamiento y enviando datos, y asegurándose que se tiene una conexión establecida con la plataforma Back-end que recibe los datos a transmitir, se puede ejecutar la aplicación EmbebidoBRT-1.0.java, nombre que se le dio al ejecutable de la aplicación construida basada en la arquitectura propuesta.

Para ejecutar dicha aplicación se hace con el comando, adicionalmente se le pide que envié la salida estándar a un archivo de texto y que corra en segundo plano

```
java -jar  EmbebidoBRT-1.0.java  >output.txt  &
```

Como resultado a ejecutar esa línea se puede verificar lo siguiente:

- La aplicación está funcionando bien en la arquitectura target
- No hay problemas iniciando el servidor TCP
- Hay flujo de datos desde el sensor
- Se pueden armar de manera correcta los mensajes en formato Json
- Se pueden enviar directamente a la plataforma back-end
- Se está almacenando el mensaje en la persistencia de la tarjeta galileo

A continuación, se muestra la salida estándar de la aplicación EmbebidoBRT-1.0.java.

```
root@quark:/gnss/datatools# cat output.txt

iniciando el server ...

Servidor iniciado

{"Iniciado":true}

Guardando mensaje en persistencia

coordenadas tcpserver(7.00274,-73.05468)

----- Intentando enviar el siguiente mensaje ----

{"Placa":"XDB725","Tde":"2017/01/30
08:20:55","ProximaParada":0,"Coordenada":{"Latitud":"7.00274","Long
itud":"-73.05468","CodigoDispo":"B001","Temperatura":"19"}}

-------------------------------------------------

{ "ProximaParada" : 0 , "Terminado" : false}


coordenadas tcpserver(7.00297,-73.05478)

coordenadas tcpserver(7.00383,-73.05516)

coordenadas tcpserver(7.00496,-73.05565)

----- Intentando enviar el siguiente mensaje ----

{"Placa":"XDB725","Tde":"2017/01/30
08:21:00","ProximaParada":0,"Coordenada":{"Latitud":"7.00496","Long
itud":"-73.05565","CodigoDispo":"B001","Temperatura":"30"}}

-------------------------------------------------

{ "ProximaParada" : 0 , "Terminado" : false}


Guardando mensaje en persistencia

coordenadas tcpserver(7.00509,-73.0557)

coordenadas tcpserver(7.0052,-73.05573)

----- Intentando enviar el siguiente mensaje ----

{"Placa":"XDB725","Tde":"2017/01/30
08:21:05","ProximaParada":0,"Coordenada":{"Latitud":"7.0052","Longi
tud":"-73.05573","CodigoDispo":"B001","Temperatura":"30"}}
```

```
------------------------------------------------

{ "ProximaParada" : 0 , "Terminado" : false}


coordenadas tcpserver(7.00536,-73.05577)

coordenadas tcpserver(7.00556,-73.0558)

coordenadas tcpserver(7.00584,-73.05582)

----- Intentando enviar el siguiente mensaje ----

{"Placa":"XDB725","Tde":"2017/01/30
08:21:10","ProximaParada":0,"Coordenada":{"Latitud":"7.00584","Long
itud":"-73.05582","CodigoDispo":"B001","Temperatura":"21"}}

------------------------------------------------

{ "ProximaParada" : 0 , "Terminado" : false}


Guardando mensaje en persistencia

coordenadas tcpserver(7.00647,-73.05585)

coordenadas tcpserver(7.00681,-73.05586)

----- Intentando enviar el siguiente mensaje ----

{"Placa":"XDB725","Tde":"2017/01/30
08:21:15","ProximaParada":0,"Coordenada":{"Latitud":"7.00681","Long
itud":"-73.05586","CodigoDispo":"B001","Temperatura":"21"}}

------------------------------------------------

{ "ProximaParada" : 0 , "Terminado" : false}


coordenadas tcpserver(7.00727,-73.05589)

coordenadas tcpserver(7.00784,-73.0559)

coordenadas tcpserver(7.0081,-73.05589)

----- Intentando enviar el siguiente mensaje ----

{"Placa":"XDB725","Tde":"2017/01/30
08:21:20","ProximaParada":0,"Coordenada":{"Latitud":"7.0081","Longi
tud":"-73.05589","CodigoDispo":"B001","Temperatura":"31"}}

------------------------------------------------
```

6.4.3. Pruebas de rendimiento

Otra comprobación que debía realizarse era evaluar el rendimiento continuo de la aplicación durante un periodo prolongado y examinar el comportamiento del consumo de CPU y RAM, se realizaron simultáneamente sobre la aplicación brtLauncher dichas pruebas de rendimiento, para ello se usó la herramienta sysstat, específicamente la aplicación pidstat que me permite realizar monitoreo de una aplicación conociendo el id del proceso.

Para estas mediciones se puso en funcionamiento la aplicación y se examinó durante 6 horas o 21600 segundos.

El siguiente grafico muestran los resultados obtenidos durante ese periodo de ejecución para consumo de CPU.

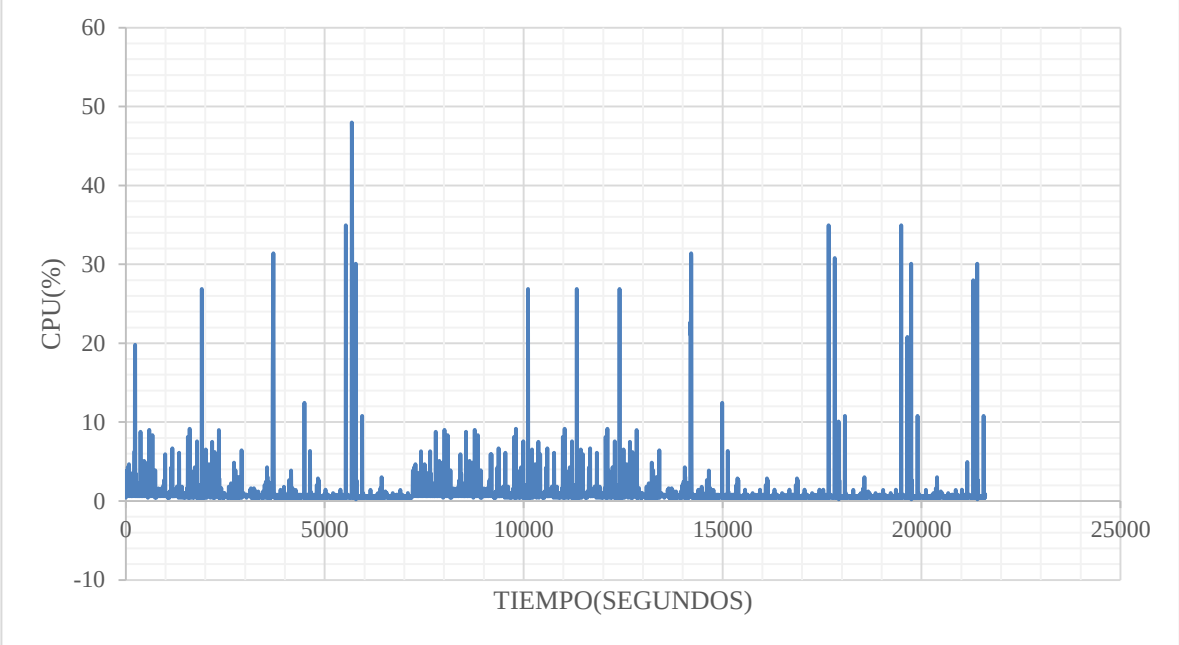

Figura 15: Consumo de CPU de la aplicación brtLauncher

En cuanto a consumo de RAM se observa lo siguiente:

Figura 16: Consumo de RAM de la aplicación brtLauncher

7. CONCLUSIONES

- Se presentó una arquitectura software que asegura dos propiedades importantes para la plataforma embebida como son:
    - Extensibilidad: la arquitectura software permite agregar nuevos tipos de sensores sin necesidad de modificar la aplicación, solo se agrega un componente nuevo de tipo sensor que utilice el mismo sistema de comunicación (un bus de datos) y el resto de la aplicación funcionará de la misma forma.
    - Flexibilidad: como los componentes están bien definidos la arquitectura permitirá modificarlos según diferentes necesidades, por ejemplo el componente encargado de transmitir el mensaje es quien conoce el protocolo utilizado (HTTP) y puede ser cambiado en un contexto que utilice otro protocolo IoT como COAP (Constrained Application Protocol).

- Este trabajo de grado fue de gran utilidad para ampliar conocimientos de sistemas embebidos, GNSS, comunicación entre plataformas, IoT entre otras áreas que serán de utilidad para futuros trabajos en estas áreas.

- El software embebido requiere una compilación cruzada, un proceso que a veces resulta complejo, pero se puede generar binarios para una arquitectura específica, para luego producir en masa réplicas del modelo de software funcional para dispositivos con las mismas características.

- El prototipo desarrollado muestra un uso adecuado de los recursos computacionales de la plataforma embebida en la cual se experimentó. Esto garantiza que la plataforma hardware tiene los suficientes recursos para ejecutar el prototipo y que la plataforma software puede ejecutarse en una plataforma hardware genérico, de bajo costo y con un uso adecuado de energía.

- Dentro de las perspectivas de trabajo futuro se puede indicar:

    - Diseñar una familia de plataformas donde los componentes se ensamblen dependiendo de las necesidades de la plataforma específica a construir (por ejemplo protocolos de comunicación y/o sensores a ensamblar.
    - Utilizar una tecnología de inyección de dependencias permitiría construir de forma automática la plataforma específica a ensamblar.
    - Estudiar la posibilidad de tener una plataforma dinámica que permita agregar o quitar componentes en caliente para casos de uso donde la plataforma embebida deba estar funcionando todo el tiempo.
    - Proyectar escenarios donde la plataforma embebida no solo interactúe con sensores, sino que también tenga la posibilidad de accionar actuadores en base a decisiones de la plataforma tomadas en función del ambiente.

CITAS

[1] *What is Bus Rapid Transit?* (s.f.). Obtenido de Regional Transit AuthorityRegional Transit Authority: http://www.rtamichigan.org/what-is-bus-rapid-transit/

[2] Winstock, A., Hook, W., Peplogle, M., & Cruz, R. (Mayo de 2011). *Recapturing Global Leadership in Bus Rapid Transit. A Survey of Select U.S. Cities.* Obtenido de ITDP: https://www.itdp.org/wp-content/uploads/2014/07/20110526ITDP_USBRT_Report-LR.pdf

[3] Rao, G. S. (2010). *Global Navigation Satellite System with essentials of Satelite Comunications.* New Delhi: Tata McGraw Hill.

[4] *Official U.S. government information about the Global Positioning System (GPS) and related topics*. (4 de octubre de 2016). Obtenido de GPS.gov: http://www.gps.gov/systems/gps/space/

[5] *GLONASS*. (2017). Obtenido de https://www.glonass-iac.ru/en/GLONASS/

[6] *what is Galileo?* (12 de diciembre de 2016). Obtenido de Galileo navigation - Eesa: http://www.esa.int/Our_Activities/Navigation/Galileo/What_is_Galileo

[7] Hofmann Wellenhof, B., Lichtenegger, H., & Wasle, E. (2008). *GNSS – Global Navigation Satellite Systems: GPS, GLONASS, Galileo, and more. (1 ed.).* Viena: Springer-Verlag Wien.

[8] Gurtner, W. (26 de junio). *RINEX: The Receiver Independent Exchange Format Version 2.11*. Obtenido de 2012: https://igscb.jpl.nasa.gov/igscb/data/format/rinex211.txt

[9] *NMEA 0183 Standard*. (2016). Obtenido de National Marine Electronic Assosiation: https://www.nmea.org/content/nmea_standards/nmea_0183_v_410.asp

[10] El-Rabbany, A. (2002). *Introduction to GPS: The Global Positioning System.* Boston: ARTECH HOUSE, Inc.

[11] Somerville, I. (2011). *Software Ingeniering-9 edition* (págs. 537- 563). Mexico: Pearson Education.

[12] GMV. (2015). *GNSS Receivers General Introduction.* Obtenido de Navipedia: http://www.navipedia.net/index.php/GNSS_Receivers_General_Introduction

[13] Gleason , S., & Gebre-Egziabher, D. (2009). *GNSS Applications and Methods.* Boston: Artech House.

[14] Somerville, I. (2011). *Software Ingeniering-9 edition* (págs. 147 - 163). Mexico: Pearson Education.

[15] *About Yocto Project*. (s.f.). Obtenido de Yocto Project: https://www.yoctoproject.org/about

[16] Takasu, T. (2013). *RTKLIB: An Open Source Program Package for GNSS Positioning*. Obtenido de RTKLIB: http://www.rtklib.com/

[17] *iTRAK Corporation*. (2007). Obtenido de http://www.itrak.com/index_spanish.html

[18] *Prototyping model*. (agosto de 2005). Obtenido de Techtartget: http://searchcio.techtarget.com/definition/Prototyping-Model

[19] *Intel Galileo Datasheet.* (7 de octubre de 2007). Obtenido de Intel: http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/galileo-g1-datasheet.pdf

[20] Libelium. (s.f.). *Using Intel Galileo with the Arduino and Raspberry Pi Shields*. Obtenido de Cooking Hacks: https://www.cooking-hacks.com/documentation/tutorials/intel-galileo-tutorial-using-arduino-and-raspberry-pi-shields-modules-boards/

[21] *NV08C-CSM v.3.x, v.4.x*. (febrero de 2015). Obtenido de nvs-gnss.com: http://nvs-gnss.com/support/documentation/item/download/79.html

[22] *Installing the Arduino* IDE*. (s.f.). Obtenido de Intel: https://software.intel.com/en-us/get-started-arduino-install

[23] *Intel galileo FirmwareUpdaterGuide*. (marzo de 2015). Obtenido de Intel: https://downloadmirror.intel.com/24748/eng/IntelGalileoFirmwareUpdaterUserGuide-1.0.4.pdf

[24] Getting *Started with Intel Galileo Board*. (s.f.). Obtenido de Intel: https://software.intel.com/en-us/get-started-galileo-windows-step1

[25] *Publish/Subscribe pattern*. (2017). Obtenido de Microsoft: https://msdn.microsoft.com/en-us/library/ff649664.aspx

BIBLIOGRAFÍA

El-Rabbany, A. (2002). *Introduction to GPS: The Global Positioning System.* Boston: ARTECH HOUSE, Inc.

Gleason , S., & Gebre-Egziabher, D. (2009). *GNSS Applications and Methods.* Boston: Artech House.

GMV. (2015). *GNSS Receivers General Introduction.* Obtenido de Navipedia: http://www.navipedia.net/index.php/GNSS_Receivers_General_Introduction

Hofmann Wellenhof, B., Lichtenegger, H., & Wasle, E. (2008). *GNSS – Global Navigation Satellite Systems: GPS, GLONASS, Galileo, and more. (1 ed.).* Viena: Springer-Verlag Wien.

Rao, G. S. (2010). *Global Navigation Satellite System with essentials of Satelite Comunications.* New Delhi: Tata McGraw Hill.

Somerville, I. (2011). *Software Ingeniering-9 edition.* Mexico: Pearson Education.

ANEXOS

Script para autoarranque de la aplicación EmbebidoBRT-1.0.java y correrla en segundo plano

```
#!/bin/bash
# Script de Prueba
### BEGIN INIT INFO
# Provides:          ejecutar
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: inicia y administra el proceso brtlauncher.
# Description:       arranca un proceso en segundo plano, lo permite cerrar.
### END INIT INFO
start() {
  if [ -f /var/run/stLauncher.pid ] && kill -0 $(cat /var/run/stLauncher.pid); then
    echo "Servicio BRT laucher en ejecucion"
    return 1
  fi
  echo "Iniciando serivicio BRT launcher..."
  cd /gnss/datatools
  java -jar EmbebidoBRT-1.0.jar >output.text 2>errores.txt &
  echo $! > /var/run/stLauncher.pid
  echo "Servicio BRT launcher inicializado"
}
```

```
stop() {
  echo "Deteniendo el sercivio BRT launcher... "
  PID=`cat /var/run/stLauncher.pid`
  kill -9 $PID
  rm -f /var/run/stLauncher.pid
  echo "Servicio BRT launcher detenido"
}
status() {
  systemctl sttatus brtLauncher.service
}
case "$1" in
  start)
    start
    ;;
  stop)
    stop
    ;;
  status)
    status
    ;;
  *)
    echo "Usage: $0 {start|stop|status}"
esac
```

Script para autoarranque de la aplicación rtkcolector.sh y correrla en segundo plano

```
#!/bin/bash
# Script de Prueba
### BEGIN INIT INFO
# Provides:          ejecutar
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: inicia y administra el proceso str2str de  rtklib.
# Description:       arranca un proceso en segundo plano, lo permite cerrar.
### END INIT INFO
start() {
  if [ -f /var/run/strtklib.pid ] && kill -0 $(cat /var/run/strtklib.pid); then
    echo "Service already running"
    return 1
  fi
  echo "Starting ejecutar service..."
  cd /gnss/datatools
  ./rtkcolector.sh >/dev/null 2>/dev/null &
  echo $! > /var/run/strtklib.pid
  echo "Service rtklib started"
}
stop() {
  echo "Stopping rtklib service... "
  PID=`cat /var/run/strtklib.pid`
```

```
  kill -9 $PID
  rm -f /var/run/strtklib.pid
  echo "Service strtklib stopped"
}
status() {
  systemctl status rtkcole.service
}
case "$1" in
  start)
    start
    ;;
  stop)
    stop
    ;;
  status)
    status
    ;;
  *)
    echo "Usage: $0 {start|stop|status}"
esac
```