< SMART SENSOR >

WWW.PARADIGMA.CL
INFO@PARADIGMA.CL

DICIEMBRE 13, 2018

Contenido

1. INTRODUCCIÓN

Detallaremos la interfaz para nuestra red de sensores que denominamos *GlobalConnectSmartSensor*, y definiremos sus funcionalidades.

Por funcionalidades se comprende un conjunto de elementos (TYPES, METHODS, EXCEPTIONS y EVENTS) de la interfaz.

La interface *GlobalConnectSmartSensor* tiene actualmente 39 items de funcionalidad: 13 propiedades, 26 métodos, 0 eventos, y 0 excepciones.

Ya que el diseño de la interfaz puede ser compleja, se usarán algunas definiciones de interfaz DOF existentes (ver página *https://interface.opendof.org/browse?4*), reutilizando las que sean apropiadas para nuestros intereses:

Las interfaces a reutilizar son:

1) Thermomer
2) Hygrometer
3) GPS Location
4) Schedule
    a. SchduleAt
    b. Schedule Between
    c. RescheduleAt
5) Lifetime Energy Meter
6) Status
7) Device Manager

Algunas de las interfaces están basadas sólo en el tipo de dato, en ese caso se programará el comportamiento (métodos, eventos y excepciones).

2. El Enfoque OOP en IoT

El Internet de las cosas (IoT) es uno de los sectores de más rápido crecimiento de la industria tecnológica. Sin embargo, la forma en que IoT está evolucionando plantea serias preocupaciones, hay demasiadas complejidades, partes móviles, diversidades, tendencias y tecnologías en competencia que deben gestionarse al desarrollar soluciones de IoT.

Muchas de estas preocupaciones tienden a pasarse por alto a medida que los fabricantes se apresuran en enviar nuevos productos al mercado y, por lo tanto, se está utilizando demasiado código propietario en los productos de IoT.

Como resultado, los productos tienen graves agujeros de seguridad y no pueden adaptarse a los cambios que superarán su entorno y ecosistema en los próximos meses y años.

Un enfoque práctico que podría ayudar a lidiar con las complejidades de la IoT sería el uso del concepto de "separación de preocupaciones" y "abstracción" para crear soluciones que puedan lidiar con los problemas de seguridad y las diversidades en diferentes niveles, además de ser flexible. Ante los constantes cambios.

La ingeniería de software, nos explica cómo abordar las complejidades en el desarrollo de software mediante el uso de técnicas como la descomposición (dividir los problemas complejos en partes más pequeñas) y abstracción (ignorando los detalles no esenciales de las cosas y tratando con la interfaz generalizada del modelo). Estos conceptos han estado en el corazón de la programación exitosa en general durante años, y también pueden convertirse en la base para crear soluciones de IoT exitosas y escalables.

La llegada de la programación orientada a objetos (OOP) minimizó la "brecha de representación", permitiendo a los programadores crear componentes de software que se asignaron a objetos y conceptos reales en el dominio del problema. Esta es una idea que se implementa fácilmente en el dominio del software, pero se complica cuando se trata de sistemas IoT distribuidos en redes de dispositivos que varían en hardware y software.

El marco abierto de objetos distribuidos (OpenDOF) es una adaptación de OOP para sistemas distribuidos. Los ingenieros se centran en desarrollar soluciones de IoT a un nivel de abstracción que represente dispositivos, mientras que el marco maneja la mecánica de las comunicaciones y la seguridad. Al separar la conectividad y la seguridad de IoT de su lógica y funcionalidad, OpenDOF permite que los dos aspectos evolucionen y cambien sin romperse entre sí.

"Un programador de aplicaciones no debería necesitar saber o preocuparse por dónde se proporciona realmente la funcionalidad".

"Una buena capa de abstracción, que proporcione de forma segura la separación de preocupaciones, es fundamental para cualquier API de IoT".

En su esencia, OpenDOF es un conjunto de bibliotecas que permite a los desarrolladores crear módulos de interfaz y objetos que representan dispositivos reales, registrar instancias de esos dispositivos y permitir el acceso y descubrimiento controlados mediante el uso de servidores de autenticación. Los objetos pueden existir de manera independiente e interactuar entre sí sin verse afectados por los detalles de la implementación y los cambios que se producen con el tiempo.

Las abstracciones también abordan los problemas de seguridad al restringir las comunicaciones del dispositivo a un conjunto finito de contactos públicos, y evitar que los dispositivos "toquen las partes privadas de los demás", como se dice en la jerga OOP.

La flexibilidad de OpenDOF lo hace desplegable en una amplia gama de dispositivos IoT, lenguajes de programación y transportes. Una implementación mínima segura del marco "puede ejecutarse sin sistema operativo, sin administración de memoria y en menos de 64 KB de código". También se puede adaptar dinámicamente a diferentes configuraciones de red, incluyendo "punto a punto así como pasarelas locales y despliegues en la nube, todo integrado en la aplicación".

Los sistemas de IoT son intensivos en comunicación. Cada segundo que pasa, miles y millones de mensajes se intercambian entre dispositivos y se envían a servidores con fines de almacenamiento, análisis e informes. Estos mensajes pasan por una multitud de transportes y protocolos antes de llegar a su destino, y no hay estándares reales con los que trabajar, lo que hace que el entorno de desarrollo sea mucho más difícil.

Además, los desarrolladores de IoT generalmente provienen de un fondo de programación de sistemas embebidos con poca o ninguna experiencia en el manejo de sistemas conectados y grandes bases de datos, por lo que deben crear soluciones ad hoc que sean difíciles de desarrollar, no puedan adaptarse a los cambios que se producen en sus entornos y conllevan a graves problemas de seguridad.

Este es uno de los enfoques más holísticos para enfrentar los desafíos de desarrollo de IoT, en el que las comunicaciones, la seguridad y el almacenamiento se resumen en componentes flexibles que pueden evolucionar y cambiar sin afectar la lógica central del software en ejecución. Tener una plataforma confiable y unificada que reúna las piezas del rompecabezas de IoT permitirá a los desarrolladores centrarse en la lógica y la funcionalidad.

La abstracción y la separación de las preocupaciones han demostrado su valía una y otra vez al tratar y eliminar las complejidades e inconsistencias en sistemas muy grandes y distribuidos. Estos son conceptos que tienen casos de uso distintos e importantes en el panorama volátil y en constante cambio de la industria de IoT, y su aplicación puede ayudarlo a pasar sin problemas por sus etapas de crecimiento.

3. INFORMACIÓN

Para trabajar con la interfaz DOF se entregarán tres tipos de información:

1. Una descripción de los elementos de la interfaz
2. Los identificadores de elementos para todos los elementos de la interfaz.
3. Código para la interfaz.

Types	Methods
Active	setTime
Time	setActive
3gTransmissionCapacity	getGPS
SatelliteTransmissionCapacity	getTemperature
WISUNTransmissionCapacity	getHumidity
ListOfTemperatureReceived	getGasControl
ListOfHumidityReceived	get3GTransmissionCapacity
ListOfGPSReceived	getSatelliteTransmissionCapacity
ListOfGasControlReveived	getWISUNTransmissionCapacity
ListOfPartnerWithProxysCapacity	checkIfProxyCapacity
ListOfPartnerWithRoutersCapacity	checkRouterCapacity
RouterCapacity	sendToNeighborProxyCapacity
ProxyCapacity	sendToNeighborRouterCapacity
	sendThroughtTMS
	sendThougthNeighborTemperature
	sendThougthNeighborHumidity
	sendThougthNeighborGasControl
	sendThougthNeighborGPS
	sendTemperature
	sendHumidity
	sendGPS
	sendGasControl
	receiveTemperatureFromNeighbor
	receiveHumidityFromNeighbor
	receiveGPSFromNeighbor
	receiveGasControlFromNeighbor

3.1 Descripción de los Types y Propiedades de la Interface

1. **Active**
   Indica si en nodo esta activado o no, puede contener True o False.
   Type: isActive, boolean
   Propiedad: *Active*
   Read: true
   Write: true
   Type: 1 - isActive

2. **Time**
   Type:
   timestamp, datetime
   frecuencia, uint32
   Propiedad: *StartTime*
   Read: true
   Write: true
   Type: 1 - timestamp
   Propiedad: *StopTime*
   Read: true
   Write: true
   Type: 2 - timestamp
   Propiedad: *Frecuencia*
   Read: true
   Write: true
   Type: 3 - frecuencia
   Propiedad: *Verifica*
   Read: true
   Write: true
   Type: 4 - frecuencia

3. **ProxyCapacity**
   Indica si en nodo tiene capacidad de Proxy, puede contener True o False.
   Type: isProxy, boolean
   Propiedad: *ProxyCapacity*
   Read: true
   Write: true
   Type: 1 - isProxy

4. **3GTransmissionCapacity**
   Indica si en nodo tiene capacidad de transmitir vía 3G, puede contener True o False.
   Type: is3G, boolean
   Propiedad: *3GTransmissionCapacity*
   Read: true
   Write: true
   Type: 1 – is3G

5. **SatelliteTransmissionCapacity**
   Indica si en nodo tiene capacidad de transmitir vía Satellite, puede contener True o False.
   Type: isSatellite, boolean
   Propiedad: *SatelliteTransmissionCapacity*
   Read: true
   Write: true
   Type: 1 – isSatellite

6. **WISUNTransmissionCapacity**
   Indica si en nodo tiene capacidad de transmitir vía WISUN, puede contener True o False.
   Type: isWISUN, boolean
   Propiedad: *WISUNTransmissionCapacity*
   Read: true
   Write: true
   Type: 1 – isWISUN

7. **RouterCapacity**
   Indica si en nodo tiene capacidad de Router, puede contener True o False.
   Type: isRouter, boolean
   Propiedad: *RouterCapacity*
   Read: true
   Write: true
   Type: 1 – isRouter

8. **ListOf PartnerWithProxyCapacity**
   Arreglo con ID de los nodos vecinos que tienen capacidad de proxy.
   Type: partnerWithProxy, array, min:0, max:65535

   Propiedad: *ListOfPartnerWithProxyCapacvity*
   Read: true
   Write: true
   Type: 1 – partnerWithProxy

9. **ListOf PartnerWithRouterCapacity**
   Arreglo con ID de los nodos vecinos que tienen capacidad de router.
   Type: partnerWithRouter, array, min:0, max:65535

   Propiedad: *ListOfPartnerWithRouterCapacvity*
   Read: true
   Write: true
   Type: 1 – partnerWithRouter

10. **ListOfGPSReceived**
    Arreglo con la trama de dato GPS recibido desde el sensor o de otro nodo.
    Type: gpsReveived, array, min:0, max:65535

    Propiedad: *ListOfGPSReceived*
    Read: true
    Write: true
    Type: 1 – gpsReveived

11. **ListOfTemperatureReceived**
    Arreglo con la trama de dato temperatura recibida desde el sensor o de otro nodo.
    Type: temperatureReveived, array, min:0, max:65535

    Propiedad: *ListOfTemperatureReceived*
    Read: true
    Write: true
    Type: 1 – temperatureReveived

12. **ListOfHumidityReceived**
    Arreglo con la trama de dato humedad recibida desde el sensor o de otro nodo.
    Type: humidityReveived, array, min:0, max:65535

    Propiedad: *ListOfHumidityReceived*
    Read: true
    Write: true
    Type: 1 – humidityReveived

13. **ListOfGasControlReceived**
    Arreglo con la trama de dato control de gases recibida desde el sensor o de otro nodo.
    Type: gasesReveived, array, min:0, max:65535

    Propiedad: *ListOfGasControlReceived*
    Read: true
    Write: true
    Type: 1 – gasesReveived

3.2 Descripción de los Métodos de la Interface

1. **setActive**
   Mediante este método se almacena en la propiedad ***Active*** **(**propiedad de escritura) un valor booleano. Un valor true significa que el nodo está activado. *Nota: Un nodo puede contener uno o más sensores.*

2. **setTime**
   Este es un método que permite configurar la hora, minuto y segundo del evento en que el nodo solicita a cada sensor entregar el valor. El método debe escribir en la propiedad ***Time*** la fecha, hora, minuto y segundo inicial, la frecuencia en la que los sensores entregarán los datos, y una fecha, hora, minuto y segundo de término (si no hay entonces se considera que transmitirá siempre), y también la frecuencia en que los nodos verificarán sus capacidades de comunicación.

3. **getGPS**
   Si se ha cumplido con el momento indicado en la propiedad ***Time.Frecuencia*** y además el nodo está con ***Active*** igual a True, entonces el nodo solicita al GPS entregar la posición actual (latitud, longitud). Este debe ser el primer dato solicitado a los sensores.

4. **getTemperature**
   Si se ha cumplido con el momento indicado en la propiedad ***Time.Frecuencia*** y además el nodo está con ***Active*** igual a True, entonces el nodo solicita al sensor de temperatura entregar el valor de la temperatura. Este debe ser el segundo dato solicitado a los sensores.

5. **getHumidity**
   Si se ha cumplido con el momento indicado en la propiedad ***Time.Frecuencia*** y además el nodo está con ***Active*** igual a True, entonces el nodo solicita al sensor de humedad entregar el valor de la humadad. Este debe ser el tercer dato solicitado a los sensores.

6. **getGasControl**
   Si se ha cumplido con el momento indicado en la propiedad ***Time.Frecuencia*** y además el nodo está con ***Active*** igual a True, entonces el nodo solicita al sensor de humedad entregar el valor de la humadad. Este debe ser el cuarto dato solicitado a los sensores.

7. **checkIfProxyCapacity**
   Este método permite verificar, cada cierto tiempo indicado por la propiedad ***Time.Verifica***, si el nodo tiene la capacidad de enviar mediante métodos **get3GTransmissionCapacity** o **getSatelliteTransmissionCapacity**. Si tiene la capacidad entonces almacena un valor True en la propiedad ***ProxyCapacity***.

8. **get3GTransmissionCapacity**
   Este es un método que permite verificar, cada cierto tiempo indicado por la propiedad ***Time.Verifica***, si el nodo tiene la capacidad de transmitir vía 3G/4G. Si tiene la capacidad entonces almacena un valor True en la propiedad ***3GTransmissionCapacity***. Si no tiene debe al siguiente método.

9. **getSatelliteTransmissionCapacity**
   Este es un método que permite verificar, cada cierto tiempo indicado por la propiedad ***Time.Verifica***, si el nodo tiene la capacidad de transmitir vía Satélite. Si tiene la capacidad entonces almacena un valor True en la propiedad ***SatelliteTransmissionCapacity***. Si no tiene debe al siguiente método.

10. **checkIfRouterCapacity**
    Para que el nodo pueda ser considerado router, primero no debe ser proxy (***ProxyCapacity*** igual False). Mediante método getWISUNTransmissionCapacity verifica, cada cierto tiempo indicado por la propiedad ***Time.Verifica***, si es router. Si tiene la capacidad entonces almacena un valor True en la propiedad ***RouterCapacity***.

11. **getWISUNTransmissionCapacity**
    Este es un método que permite verificar, cada cierto tiempo indicado por la propiedad ***Time.Verifica***, si el nodo tiene la capacidad de transmitir vía Red Mesh WISUN. Si tiene la capacidad entonces almacena un valor True en la propiedad ***WISUNTransmissionCapacity***.

12. **sendToNeighborProxyCapacity**
    Si es proxy (***ProxyCapacity*** igual True), comunica a todos los nodos vecinos. Los nodos vecinos actualizarán su lista de proxys.

13. **sendToNeighborRouterCapacity**
    Si es router (***RouterCapacity*** igual True), comunica a los todos nodos vecinos. Los nodos vecinos actualizarán su lista de routers.

14. **sendThroughtTMS**
    Este método prepara los datos y les da el formato establecido por el TMS. Si el nodo es proxy (***ProxyCapacity*** igual True), los deja en la lista de salida para ser enviados. Si es router, los envía a nodos vecinos, indicandoles que son datos que serán enviados a TMS.

15. **sendThroughtNeighborGPS**
    Si no es proxy (***ProxyCapacity*** igual False) y es router (***RouterCapacity*** igual True), entonces le transmite a nodo vecino el dato del GPS.

16. **sendThroughtNeighborTemperature**
    Si no es proxy (***ProxyCapacity*** igual False) y es router (***RouterCapacity*** igual True), entonces le transmite a nodo vecino el dato del temperatura.

17. **sendThroughtNeighborHumidity**
    Si no es proxy (***ProxyCapacity*** igual False) y es router (***RouterCapacity*** igual True), entonces le transmite a nodo vecino el dato del humedad.

18. **sendThroughtNeighborGasControl**
    Si no es proxy (***ProxyCapacity*** igual False) y es router (***RouterCapacity*** igual True), entonces le transmite a nodo vecino el dato del control de gases.

19. **sendGPS**
    Si es proxy (***ProxyCapacity*** igual True), entonces le transmite el dato del GPS (guardado en ***ListOfGPSReceived***) por medio de ***3GTransmissionCapacity*** si es True, o sino por medio de ***SatelliteTransmissionCapacity*** si es True. En caso de que no se pueda enviar por ninguno de los dos, actualizará su condición a Router, enviará la condición a los nodos vecinos para que actualicen sus listas y enviará el dato del GPS a un nodo vecino que cumpla que sea router o proxy.

20. **sendTemperature**
    Si es proxy (***ProxyCapacity*** igual True), entonces le transmite el dato de la temperatura (guardado en ***ListOfTemperatureReceived***) por medio de ***3GTransmissionCapacity*** si es True, o sino por medio de ***SatelliteTransmissionCapacity*** si es True. En caso de que no se pueda enviar por ninguno de los dos, actualizará su condición a Router, enviará la condición a los nodos vecinos para que actualicen sus listas y enviará el dato del GPS a un nodo vecino que cumpla que sea router o proxy.

21. **sendHumidity**
    Si es proxy (***ProxyCapacity*** igual True), entonces le transmite el dato de la humedad (guardado en ***ListOfHumidityReceived***) por medio de ***3GTransmissionCapacity*** si es True, o sino por medio de ***SatelliteTransmissionCapacity*** si es True. En caso de que no se pueda enviar por ninguno de los dos, actualizará su condición a Router, enviará la condición a los nodos vecinos para que actualicen sus listas y enviará el dato del GPS a un nodo vecino que cumpla que sea router o proxy.

22. **sendGasControl**
    Si es proxy (***ProxyCapacity*** igual True), entonces le transmite el dato del control de gases (guardado en ***ListOfGasControlReceived***) por medio de ***3GTransmissionCapacity*** si es True, o sino por medio de ***SatelliteTransmissionCapacity*** si es True. En caso de que no se pueda enviar por ninguno de los dos, actualizará su condición a Router, enviará la condición a los nodos vecinos para que actualicen sus listas y enviará el dato del GPS a un nodo vecino que cumpla que sea router o proxy.

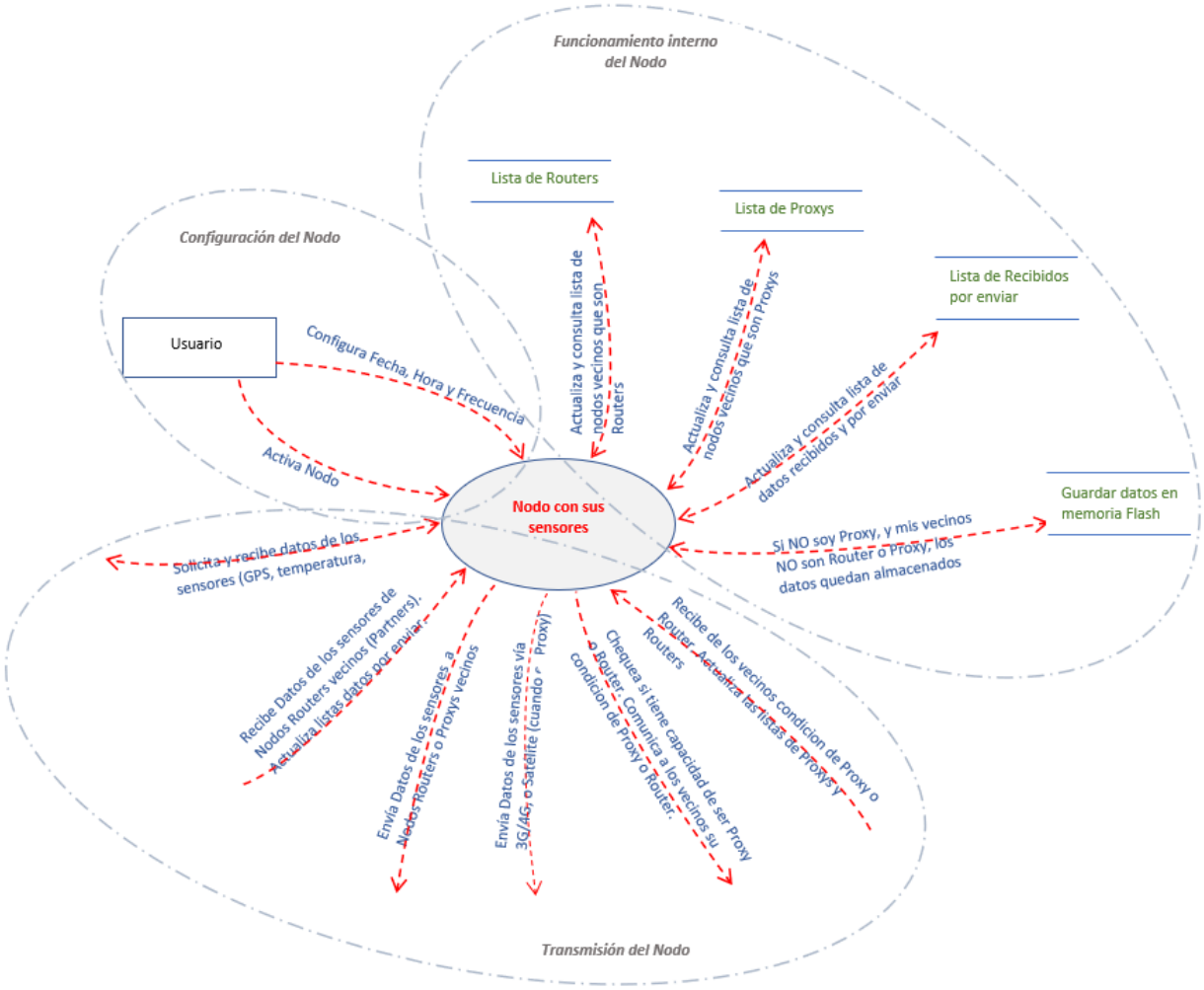
23. **receiveGPSFromNeighbor**
    Este método permite recibir dato de GPS de un nodo vecino y almacenarlo en ***ListOfGPSReceived***.

24. **receiveTemperatureFromNeighbor**
    Este método permite recibir dato de temperatura de un nodo vecino y almacenarlo en ***ListOfTemperatureReceived***.

25. **receiveHumidityFromNeighbor**
    Este método permite recibir dato de humedad de un nodo vecino y almacenarlo en ***ListOfHumidityReceived***.

26. **receiveGasControlFromNeighbor**
    Este método permite recibir dato de gases de un nodo vecino y almacenarlo en ***ListOfGasControlReceived***.

3.3 Diagrama de funcionalidades del Dispositivo



4. SEUDOCODIGO

4.1 setTime

4. GLOSSARY OF TÉRMS

OOP. Object-oriented programming
DOF. Distributed Object Frame

5. BIBLIOGRAPHY

[1] https://interface.opendof.org/browse?4