Implementación de Sistemas SCADA Utilizando Lenguajes de Alto Nivel

Nahuel Defossé, Diego Van Haaster, Lautaro Pecile, Fernando G. Tinetti[1], Ricardo A. López

Departamento de Informática Sede Trelew, Facultad de Ingeniería - UNPSJB

[1]III-LIDI, Facultad de Informática – UNLP

[1]Comisión de Investigaciones Científicas Prov. de Bs. As.

nahuel.defosse@gmail.com, diegomvh@gmail.com, lautaro.pecile@gmail.com, fernando@info.unlp.edu.ar, lopez.ricardoa@gmail.com

Resumen

Los Sistemas de Adquisición de Datos y Control (SCADA: Supervisory Control And Data Adquisition), toman del campo variables de estado provenientes de diferentes dispositivos. A fin de presentar al usuario una idea cabal del estado del proceso supervisado, el sistema debe ser capaz de trabajar con la totalidad de los datos obtenidos de los dispositivos y posteriormente ofrecer una representación amigable al humano.
En este documento, se describe un proyecto para la implementación del Núcleo de Adquisición de manera acorde a los requerimientos específicos que se plantean.

Palabras Claves: Scada, DSL (Domain Specific Language).

Contexto

Estas líneas de investigación y desarrollo forman parte del Proyecto de Investigación relacionado con el estudio de Protocolos orientados a aplicaciones eléctricas y relacionadas con aplicaciones WEB.
Asimismo, se enmarca en un análisis que se avoca al estudio de innovaciones en el campo del SCADA para sistemas eléctricos en particular y otros sistemas de adquisición de datos en lo general.

Introducción

Muchos sistemas SCADA pueden esquematizarse como lo muestra la Fig. 1. Los concentradores (que llamaremos CO de aquí en más) distribuidos en el campo son los encargados de encuestar a los "Dispositivos Electrónicos Inteligentes", o IEDs (por su sigla en inglés), y almacenar en su memoria interna una cantidad significativa de datos proveniente de los mismos.
Generalmente los dispositivos de adquisición se encuentran a una distancia significativa del lugar donde los datos serán almacenados definitivamente. Es por ello que se configuran dentro de anillos locales de alta velocidad, que dan respuesta automática ante determinadas alarmas. Idealmente, dentro de cada anillo debe existir un CO que dispare respuestas automáticas y al mismo tiempo concentre todos los datos relevantes al estado del proceso, que posteriormente serán centralizados con los datos provenientes de otros concentradores.
Además de capacidad de almacenamiento, Los CO deben contar también con la posibilidad de vincularse a una red Ethernet por la cual comunicar los datos relevados. Estas actividades hacen del Concentrador un dispositivo de relativa complejidad que requiere de un microcontrolador en el cual programar la lógica necesaria para realizar varias tareas asociadas. A modo de ejemplo mencionaremos aquí dos tareas:

1. Encuestar a los IED's y concentrar temporalmente los datos. El almacenamiento interno de un CO está dedicado, prácticamente en su totalidad, a los datos relevados. El CO sólo incorpora a dichos datos las marcas de tiempo del momento en el que fueron obtenidos y mantiene algunas variables de control

útiles para su funcionamiento.

2. Establecer una conexión Ethernet y responder con los datos relevados. Un CO implementa un protocolo de comunicación que tiene la principal tarea de transportar los datos del campo al núcleo de adquisición. Este protocolo, llamado MARA, se encuentra bien definido en proyectos anteriores [2] [3].

Para que el SCADA presente un estado global del sistema, debe existir un componente software que interactúe con los diferentes COs, este componente al que llamaremos Núcleo de Adquisición (NA) debe ser mínimamente capaz de obtener, identificar y persistir los datos relevados por cada CO.

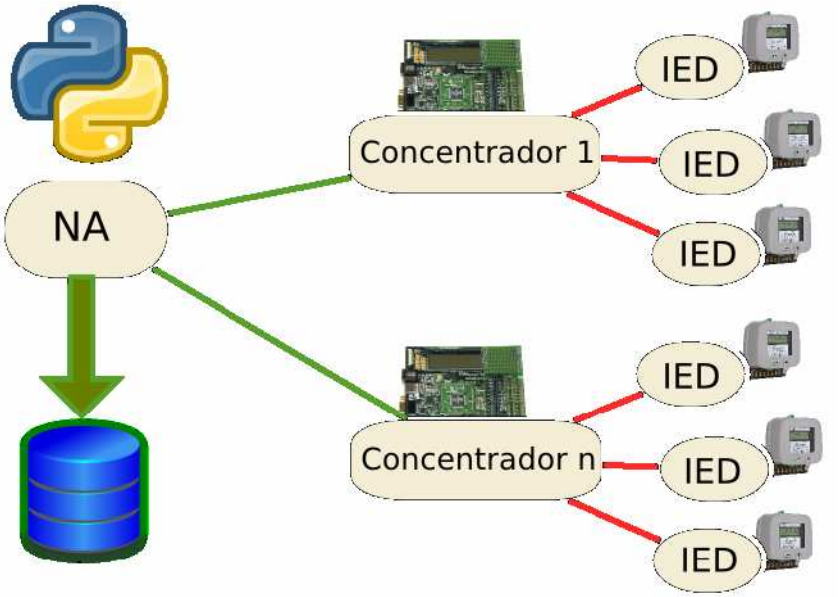

Fig 1.: Distribución de los componentes Hardware y Software.

El esquema de operación de este componente del sistema SCADA consiste básicamente en:

- abrir una conexión con un concentrador,
- negociar la obtención de los datos;
- posteriormente obtener los datos y almacenarlos;
- cerrar la conexión.

Asimismo, actuará en caso de pérdidas de la conexión y/o errores en la transmisión. La implementación del protocolo Mara por parte del NA está basado en el patrón de concurrencia Reactor (implementado por el framework Twisted) que permite separar el tratado del protocolo del mecanismo de comunicación subyacente, pudiendo utilizarse el mismo componente de protocolo bajo comunicación serial o con un mecanismo de sockets TCP/IP.

Arquitectura dirigida por eventos

La arquitectura dirigida por eventos o EDA por sus siglas en inglés, es un patrón de arquitectura de software centrado en el concepto de evento. Un evento puede ser definido como un cambio significativo en el estado del entorno. El entorno queda definido por el dominio del problema y el conjunto de fuentes de información que el software puede percibir. Desde esta perspectiva, cada evento se traduce para el sistema en un mensaje de notificación del evento que es producido en algún punto del entorno, propagado por el mismo, y detectado y consumido por el software de un modo asíncrono. Cabe destacar que el mensaje no es el evento en sí, sino su representación para el sistema software, pero de manera metonímica, es para el software el evento mismo.

Un sistema dirigido por eventos se

compone fundamentalmente por emisores de eventos (o también llamados "agentes") y consumidores de eventos (o "sumideros"). Los sumideros son los que reaccionan de alguna manera al evento, ya sea enviando otro evento al entorno (a otro sumidero) o bien consumiéndolo y transformándolo (por ejemplo, para ser presentado de alguna manera al usuario). La reacción puede ser llevada a cabo por uno o más sumideros que actúen de manera colaborativa.

La arquitectura dirigida por eventos requiere que el sistema sea capaz de responder concurrentemente a los eventos a medida que se van presentando en el tiempo. De manera nativa, Python ofrece un modo de concurrencia con hebras internas al intérprete, mediante un mecanismo denominado GIL ("Global Interpreter Lock"). Este mecanismo no escala adecuadamente en determinados casos y resulta ser un de los principales motivos para buscar una alternativa basada en eventos. Existen ya numerosas herramientas que deberán ser analizadas para definir la arquitectura basada en eventos, básicamente estas herramientas aportan un patrón de desarrollo (Callback, Generator, Tasklet, etc.), que orienta la construcción de software, mientras que en el bajo nivel se soportan sobre primitivas del sistema operativo ya conocidas.

Como ejemplo de tales herramientas podemos destacar Twisted [6], el cual es un framework para el desarrollo de aplicaciones de red dirigidas por eventos. Entre otras características, Twisted permite:

- Separación entre protocolo y transporte: Twisted está diseñado para separar el protocolo de comunicaciones (HTTP, POP3 o el que el desarrollador provea), del canal a través del cual se produce la comunicación (comunicaciones seriales, sockets, librerías SSL y otros).
- Manejo de eventos a través del patrón Reactor. El manejador de servicios demultiplexa las peticiones entrantes y entrega los mensajes correspondientes a los manejadores asociados. Esto permite desacoplar la lógica del manejo concurrente de eventos llevada a cabo por el reactor, de la lógica de los servicios ofrecidos provista por el programador.

Twisted define 3 entidades básicas: el protocolo (que administra el estado de una conexión, vinculado a una instancia de transporte), una entidad intermedia factory, (que genera los protocolos cuando una conexión es establecida) y el reactor encargado de administrar los eventos e interactuar con el sistema operativo.

Cuando el reactor se pone en funcionamiento cada constructor (objeto basado en el patrón "factory") crea una instancia del protocolo MARA y define un intervalo para enviar el comando de encuesta al concentrador. El factory mantiene estado entre desconexiones con el CO y permite la re-instanciación de este último en caso de pérdidas de conexión. Todos los eventos de red están mapeados en métodos de la instancia protocolo creada por el factory. De esta manera, a medida que se reciben datos, el protocolo puede discernir cuando se ha completado una trama y efectúa la acción asociada.

Una de las características deseables del NA es la capacidad de ser agnóstico del protocolo de comunicaciones utilizado para dialogar con cada concentrador. Este proyecto ha definido su propio protocolo de comunicaciones llamado MARA. En el esquema hasta aquí planteado esta situación no implicaría un inconveniente para otro tipo de concentrador o equipo de recolección, bastará con implementar una abstracción del protocolo necesario y dejar que el framework cree una instancia del mismo cuando los eventos en la capa de transporte ocurran.

Identificación de las tramas de datos

En lo que se refiere al trabajo que el NA realiza con las trama, tanto para consultar a los concentradores como para analizar la respuesta, se plantea la alternativa de construir un módulo que sea flexible a los

cambios en el protocolo de comunicación. Con este objeto se evalúa aquí la alternativa de utilizar un lenguaje de dominio específico para definir la estructura de las tramas y la utilización de una librería que simplifique la serialización y deserialización de los datos a ser transmitidos

Lenguajes de Dominio Específico

Los DSLs son lenguajes de programación de expresividad limitada, especializados a un dominio de aplicación particular. Martin Fowler [1] divide los DSLs en dos clases:

1. Los DSLs internos son una forma particular de utilizar un lenguaje de propósito general, utilizando un subconjunto de las características del lenguaje anfitrión para encarar un aspecto particular del sistema.
2. Los DSL externos son un lenguaje separado del lenguaje anfitrión en el cual está realizada la aplicación. Usualmente poseen una sintaxis propia y deben efectuarse un "parking" de alguna manera por el lenguaje anfitrión antes de utilizar algún script escrito en ese lenguaje.

Los DSL proveen varias ventajas con respecto a la manipulación "manual" de los datos:

- Validación a nivel de dominio
- Código auto-documentado
- La solución se expresa en el mismo nivel de abstracción del problema
- Facilitan la comunicación con los expertos en el dominio del problema
- Mejora la calidad, mantenibilidad y productividad del programa

Sin embargo existen algunas desventajas, entre ellas:

- El costo de diseñar, implementar y mantener el lenguaje.
- Mayor necesidad de potencia de procesamiento al tener que realizar un nivel más de cómputo.
- Necesidad de capacitación extra en los recursos humanos dedicados a mantener el sistema.
- Multiplicidad de lenguajes en la aplicación.

En nuestro proyecto, hemos utilizado la librería "Construct" [10] para generar un DSL interno que permita trabajar desde un alto nivel de abstracción con las tramas, sin perder por ello la capacidad de manipularlas a nivel de bit. Esta librería ofrece una rica API para "parking" y construcción de datos binarios estructurados, de manera declarativa. Posee facilidades para trabajar con construcciones atómicas (estructuras de bits simples, tales como enteros de diferentes tamaños, o datos binarios especializados), y para componer éstas en construcciones más complejas. Además posibilita transformar secuencias de bits en objetos y viceversa, facilitando la manipulación de los datos binarios de manera considerable.

De esta manera, trabajar con las tramas del protocolo de comunicación MARA consiste en describir las tramas con la API que Construct brinda, posteriormente delegar en la librería la serialización de las tramas binarias para encuestar a los CO y la deserialización de los datos obtenidos. Esta forma de operar con las tramas de datos en un alto nivel resulta homogénea con el resto de los objetos del sistema y permite una rápida respuesta al cambio del protocolo de comunicación, sin perder el detalle de operación que se consigue con lenguajes como C o ensamblador.

La utilización de esta biblioteca se realiza cuando el NA debe enviar una consulta al CO (generalmente asociada a un timer) y en el evento de recepción de datos. El proceso de comunicación y procesamiento de los datos adquiridos por el NA se produce a grandes rasgos de la siguiente manera:

- En el método connectionMade de la clase que representa el protocolo se construye la trama petición de datos representada por el comando 0x10 en MARA y se escriben en la capa de transporte.
- En el método dataReceived, ya antes mencionado, se parsean los datos pasados como parámetro y se obtiene un objeto de alto nivel simple de ser

analizado.

De esta manera, el DSL creado encapsula de una forma sucinta y mantenible el proceso de comunicación entre el NA y los COs, permitiendo desarrollar más eficientemente el sistema.

Persistencia de los datos obtenidos en campo

Por otra parte el NA debe mantener conexiones con el Servidor de la Base de Datos (BDD), donde se efectuará el almacenamiento definitivo de los datos. Estos eventualmente serán consumidos por capas superiores según los requerimientos y formatos de la información.

Luego de obtenidos los datos, se inicia un proceso de centralización y persistencia de la información en una BDD. En el presente proyecto, el enfoque para el almacenamiento consiste en el uso de una BDD relacional, donde los datos serán guardados junto con la identificación del origen de los mismos. De este modo la base de datos reflejará la estructura de cada concentrador (variables digitales, variables analogicas y eventos).

Observando los de relevados de los concentradores es fácil ver una estructura jerárquica donde: el NA encuesta a muchos COs, los cuales relevan datos de muchos IEDs quienes a su vez están sensando variables digitales, analógicas y eventos. Esta estructura es simple de persistir en una BDD relacional utilizando lenguaje de consultas estándar; sin embargo el uso de herramientas que hagan el mapeo de los objetos obtenidos de las tramas, en tablas del modelo relacional, proveerán al proyecto de otro punto de desacople, resultando en un NA agnóstico del motor de base de datos subyacente.

Un requerimiento importante del proyecto es mantener la responsividad del sistema a los distintos eventos que se produzcan. Es por ello que el proceso de persistencia de datos no puede bloquear al sistema durante el tiempo que se esté realizando. A fin de lograr esto, se hace uso de un mecanismo provisto por Twisted denominado Deferred, que permite delegar la tarea de persistencia a una hebra aparte, la cual ejecutará de manera asincrónica el procedimiento bloqueante, permitiendo continuar con la ejecución del programa normalmente.

Líneas de Investigación y Desarrollo

Inicialmente, se debe deben relevar las características de los concentradores y del protocolo MARA utilizado para la comunicación sobre ethernet.

Se deberá efectuar un análisis basado en bibliografía y experimentación de las ventajas y desventajas de utilizar una arquitectura dirigida por eventos, como así también evaluar alternativas a la definición del motor de adquisición basado en Twisted.

Se deberá definir la estructura de las tramas MARA en el DSL y experimentar con la serialización y deserialización de datos binario, para luego integrar la definición del protocolo con el motor de adquisición, inicialmente con la implementación del comando de obtención de datos debería bastar para probar la integración.

Se deberá definir una estructura jerárquica de almacenamiento, para los datos obtenidos de los concentradores; esta deberá permitir identificar tanto al dato como a la fuente del mismo. Para esta línea se deberá también evaluar las alternativas de herramientas de mapeo a relacional o, eventualmente el uso de otro tipo de base de datos que se integre con el motor de adquisición.

Resultados y Objetivos

El software del NA fue íntegramente implementado en Python, utilizando las librerías de Twisted para el manejo de eventos, Construct para análisis y síntesis de tramas del protocolo MARA y herramientas ORM para la persistencia de datos. Si bien no es usual el empleo de un lenguaje de alto nivel para trabajo con datos a nivel de bit -como lo requiere la

comunicación de los datos mediante MARA- en un entorno concurrente en el que se requiere atender múltiples eventos y tareas simultáneos, el lenguaje y las librerías utilizadas han demostrados ser sobradamente eficaces para alcanzar los requerimientos del proyecto. Además, el lenguaje permite una mayor mantenibilidad y experimentación de características nuevas, sean estas cambios en el protocolo MARA o capacidades a agregar al sistema SCADA.

Formación de Recursos Humanos

En este aspecto se destaca el trabajo en conjunto de un equipo de investigación conformado, también nutriendo a alumnos de los años superiores en la conformación de sus tesis de grado y posgrado. Se estima la concreción de una tesis de posgrado y una tesis de grado.

También se orientará a brindar servicios concretos a distribuidoras eléctricas de nuestra región, donde entendemos existe un buen campo debido a la necesidad de adaptación de equipamiento y software de generaciones anteriores para cumplir con los requisitos de apertura y organización a los que propenden las nuevas normas.

Referencias

1. Fowler, Martin. Domain Specific Languages.Addison Wesley 2010.
2. XV CACIC 2009, Universidad Nal de Jujuy, Octubre 5 al 9 de 2009, ISBN 978 89724068-4-1. Sincronización de Microcontro-ladores en red, Implementación y Evaluación. Fernando G. Tinetti, Ricardo A. López, Marcelo E. Gómez, Sebastián Wahler.
3. López R. Protocolos en Redes de Microcontroladores. 1ra Edición – Septiembre 2010. Tesis de Magister: Redes de datos. Director:Fernando F. Tinetti. Publicó: Fac.Informática UN La Plata.
4. David Beazley, Understanding the Python GIL, http://www.dabeaz.com /GIL/, 20 de Febrero de 2010
5. Wikipedia contributors, "Event-driven architecture," Wikipedia, The Free Encyclopedia, http://en. wikipedia.org/w/index.php?title= Event-driven_architecture&oldid =598151439, 19 deMarzo de 2014.
6. Twisted Matrix Labs https: //twistedmatrix.com/trac/
7. Nicholas Piël, Asynchronous Servers in Python, http://nichol.as/ asynchronous-servers-in-python,22 de Diciembre de 2009.
8. Dan Kegel, The C10K problem http://www.kegel.com/c10k.html#strategies,5 de Febrero de 2014
9. Wikipedia contributors, "Modbus" Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/Modbus, 13de Febrero de 2014.
10. Construct, Powerful declarative parser and builder for binary data http://construct.readthedocs.org/en/latest/