Informe Final Capa de Alto Nivel del Poyecto Redes y Microncontroladores

Autor: Nahuel Defossé, Lautaro Pecile

Fecha: Marzo 2010

Introducción

El objetivo de la capa de alto nivel del proyecto "Microcontroladores e Internet" es la adquisición y persistencia de datos, además de proveer un mecanismo de representación de la información adquirida permitiendo el envío de ordenes (comandos) como lazo de realimentación humana sobre el sistema. Este tipo de software se conoce como sistema SCADA (*Supervisory Control and Data Acquisition*). Un sistema SCADA generalmente se acompaña de algun tipo de interfase que permite la interacción con el usuario, denominado HMI, (*Human Machine Interfce*).

Se estableció también como objetivo la utilziación de *Software Libre*, para el desarrollo integro del software, con el objeto de garantizar la independecia del fabricante y bajo coste de implantación del la tecnología.

El proyecto inicialmente se orientó en el desarrollo de una terminal de consulta y envío de comandos similar a ArquiCom utilizando el lenguaje Python y el toolkit gráfico Qt. Se comprobó durante esta etapa que el lenguaje Python a pesar de ser de muy alto nivel, está preparado para trabajar en comunicaiciones donde se deben realizar operaciones a nivel de bit con bastate facilidad. Una vez superadas las pruebas iniciales, se plantaron los requerminetos genrales como visualización de eventos, ubicación de las terminales (RTU) en mapa o esquema, y alarmas.

Orientación a Eventos

Los lenguajes de muy alto nivel orientados a objetos son ejecutados sobre máquinas virtuales. Generalmente estas proveen el mecanismo de hilos (*Threads*) para logar concurrencia, pero están limitados en varios aspectos [StepehDoyle2009].

Los hilos son la herramienta más común a la hora de realizar concurrencia y consisten en la capacidad de poder ejecutar concurrentemente varios fragmentos de código por parte de un programa. Un ejemplo puede ser la atención de varios clientes, o en el caso de el presente proyecto, la consulta a varios concentradores.

Un hilo consiste en un proceso o método (en un lenguaje OO), el cual es ejecutado por un programador o scheduler apropiativo provisto por el sistema operativo. No se puede saber a ciencia cierta en que momento se va a ejecutar cada hilo y se suelen usar primitavas de sincronización como semáforos o monitores para garantizar la integridad de los datos (debido a que todos los hilos comparten su estado/datos).

En el caso de una aplicación orientada a eventos, la concurrencia se logra a través de la subdivición voluntaria en el momento de la programación en pequeños fragmentos de código (funciones o métodos) que son ejecutados por un scheduler, generalmente apodado reactor en honor a un patrón de diseño de concurrencia [WikiReactor]. En este enfoque, no se requieren primitivas de sincronización debdo a que se trata de multitarea cooperativa.

En general las arquitecutras orientadas a eventos en entornos con alta carga de entrada salida son más beneficiosas que los hilos. Se puede tomar como ejemplo de este hecho, el proyecto Apache [MINA].

El framework de red orientado a eventos [Twisted] fue elejido para la implementación de la capa de comunicaciones con las unidades remotas, debido a que provee un enfoque simple y orientado a objetos de la multitarea colaborativa.

Software de Adquisición de Datos

Una vez seleccionado el framework de comunicaciones asincrónicas, se decidió el método de acceso a la base de datos. En vez de seleccionar una base de datos particular, se optó por utilizar un mapeador objeto-realcional, que consiste en una capa de software que presenta la base de datos como objetos.

Si bien el lenguaje de acceso a las bases de datos, conocido como SQL está normado por ISO/ANSI, cada fabricante incorpora sutiles diferencias que hacen que se requieran pequeños arreglos para poder manejar una base de datos u otra. Los mapeadores objeto relacionales relizan una abstracción por sobre estas diferencias.

Se seleccionó como mecanismo de acceso a la base de datos el mapeador objeto relacional [SQLAlchemy] debido a que pemite un nivel de detalle muy alto a la hora de realizar los mapeos, posee varios subproyectos de interés como Elixir y tiene una extensa comunidad.

Las tablas definidas para la adquisición de datos fueron las siguientes:

- CO - Concentrador

  Posee un número y la dirección IP

- UC - Unidad de Control

  Posee un número de UC, una relación hacia el CO al que pertenece

- SV - Variable de Sistema

  Representa una variable de sistema, posee como atributo el número (posición), y está relacionada con la UC a la que pertenece.

- DI - Variable de Sistema

  Representa una variable digital de un bit, posee como atributo el número (posición), y está relacionada con la UC a la que pertenece.

- AI - Variable de Sistema

  Representa una variable analógica, posee como atributo el número (posición), y está relacionada con la UC a la que pertenece.

- EV - Evento

  Representa un evento en el sistema, posee la impronta de tiempo impuesta par la unidad de control, la relación con la unidad de control, el tipo de evento y el valor (repsesentan pasajes de 0 a 1 o de 1 a 0).

Ciertas vairables de configuración fueron almacenadas en archivos con formato legilbe para el humano, provista por el paquete [config]:

```
picnet :
{
  tcp_port : 9761
  id_rs485 : 1
}
scada :
{
  verbosity : 1
  type : 'sqlite3'
  user : ''
  password : ''
  host : ''
  dbname : 'dsem.db'
  options : ''
```

```
  save_di : False
  save_ai : False
  save_ev : True
  save_sv : False
}
```

A este software se lo bautizó como Alsvid en honor a un corcel de la mitología escandinava, su nombre significa *"muy velóz"*.

Integración de SCADA/HMI: Sistema Avrak

Se desarrolló como actividad de extensión una implementación sobre el motor de adquisición de datos una interfase hombre máquina (HMI) para un sistema de semaforización.

Por motivios de practicidad se decidió embeber el motor SCADA en la aplicación HMI, obteniéndose un único ejecutable que solo depende de el motor de base de datos. En este caso se seleccionó MySQL.

Este sistema presenta sobre un mapa de la ciuidad el conjunto de puntos que representan el estado de cada esquina. Para lograr este objetivo se agregaron nuevas tablas a la base de datos:

- Calle

  Almacena el nombre de la calle y un identificador entero

- Esquina

  Almacena las relaciones con esquina calle y la posición en el mapa

- Esquina_Calles

  Almacena el nombre de las calles, el sentido y el ángulo

- Semáforo

  Almacena la relación con la UC, el tipo de movimiento y el subitpo de movimiento permitiendo definir un semáforo vehicular, semáforo de giro y semáforo peatonal.

La interfase visual consistió en 4 pestañas:

- Mapa

  Permite la visualización de las diferentes esquinas de la ciudad

- Eventos

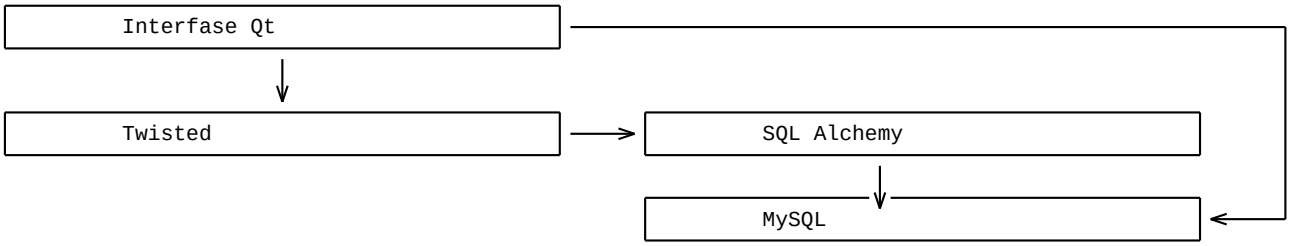
  Listado de eventos, permitiendo filtrado en función de su fecha y su tipo, además de relizar impresiones.

- Configuración

  Permite la definición de Unidades de Control y Concentradores.

En este proyecto se realizó un desarrollo de una interfase de configuración gráfica mediante Qt. Se puede ver un ejemplo de la funcionalidad en el siguiente video [YouTubeAvrak].

El siguiente digrama expone la arquitecutra del sistema:



Este proyecto se bautizó Avrak, otro corcel de la mitología escandinava.

Definición de SCADA basado en Web y HTML5

Avrak constituyó una prueba de que una arquitecutra de mutitarea colaborativa puede ser usada en conjunto con otros compoenentes de software para lograr un sistema de monitoreo y control, sin embargo cuenta con la desvetaja de que el sistema depende en buena parte de código específico para la plataforma como la biblioteca gráfica Qt y la base de datos.

Se planteó como mejora la utilización del protcolo HTTP como mecanismo de acceso y control a los datos y la separación de el manejo de el protcolo de comunicaciones (picnet) de la aplicación específica de control (semáforos en el caso de Avrak). El nuevo objetivo será acceder a los datos tanto con un browser (navegador) como con aplicaciones que puedan generar y consumir HTTP (aplicaciones de escritorio, celulares, demonios de tele-control, etc.).

Para lograr exponer los datos mediante HTTP se planteó la definición de una API, para la cual se eligió REST.

Qué es REST

Representational State Transfer (REST) es una arquitectura de software, introducida por Roy Fielding en el año 2000. Define un estilo arquitectural, basado en el diseño del protocolo HTTP.

Las aplicaciones REST poseen un estilo arquitectural en la manera que realizan las comunicaciones cliente-servidor. La idea es que, en vez de usar mecanismos como CORBA, RPC o SOAP, se utilize el protocolo HTTP para proveer todas las facilidades necesarias para las comunicaciones. Se puede ver a la World Wide Web como el ejemplo de Web Service REST más generalizado.

Se utiliza el concepto de *Recurso*, el cual es unívocamente direccionado a través de una URL. Un recurso en REST es un conjunto de datos, de la cual por ejemplo, una página web puede ser una representación del mismo. Es decir que la misma información puede tener diferentes representaciones dependiendo del URI con el que es accedido. Es el cliente el que tiene la libertad de elegir la representación más adecuada en función de la aplicación. Los Recursos son el elemento clave en las arquitecturas REST, en contraste a los "métodos" o "servicios" utilizados en RPC o SOAP. El recurso engloba tanto el estado como la funcionalidad.

Utilizando los métodos existentes en HTTP (POST, PUT, GET, DELETE), se puede realizar una interfase CRUD (Create, Read, Update, Delete) en el servidor que permita a los clientes trabajar sobre recursos accedidos mediante URLs.

No existe estado en la conexión, es decir que cada interacción entre cliente y servidor contiene toda la información necesaria por si misma, y no depende de interacciones anteriores. REST permite definir la validez de un recurso y la "cacheabilidad" del mismo. Esto es, que el protocolo permite definir explícitamente qué recursos pueden ser cacheados y por cuánto tiempo. Para este fin se utilizan las cabeceras de control de caché de HTTP. De la misma manera, es posible utilizar proxys HTTP para escalar y mejorar la performance de los servicios disponibles.

REST permite utilizar todas las ventajas provistas por HTTP en la comunicación, tales como encriptación de flujo de datos, autenticación, sesiones, compresión de datos, etc. Esto simplifica el sistema y ahorra esfuerzo en el desarrollo de las aplicaciones.

Diferencias con otras arquitecturas de Web Services

Existen otras posibilidades para la generación de Web Services, tales como SOAP, RPC y CORBA. Tales arquitecturas gozan de herramientas más maduras, lo cual es su principal ventaja frente a REST. Sin embargo esto puede cambiar con el tiempo. Otra ventaja es la seguridad de tipos ofrecida en los request XML, utilizados por ejemplo en XML-RPC o SOAP. Esto también puede ser modificado en REST por el programador.

La principal ventaja del enfoque REST es la facilidad de implementación, la agilidad de diseño y el enfoque de peso liviano. Otras ventajas tienen que ver con la performance: soporte de cache, peticiones y respuestas de pequeño tamaño y parseo simple de las peticiones. Esto se traduce en clientes y servidores simples y menor utilización de la red.

REST es un sistema que descansa en un protocolo abierto y bien conocido. En cambio con SOAP/RPC cada diseñador debe definir un nuevo conjunto de sustantivos/verbos para la aplicación, y puede o no utilizar HTTP como protocolo de comunicaciones. Muchas veces este nuevo conjunto es propietario o no está correctamente documentado, por lo que la compatibilidad con otros sistemas se ve perjudicada.

Utilización en el Proyecto

Se utiliza REST mediante los verbos HTTP para implementar semántica CRUD y acceder a los recursos almacenados en la base de datos.

POST

Crear un recurso dentro de una colección dada

GET

Obtener un recurso

PUT

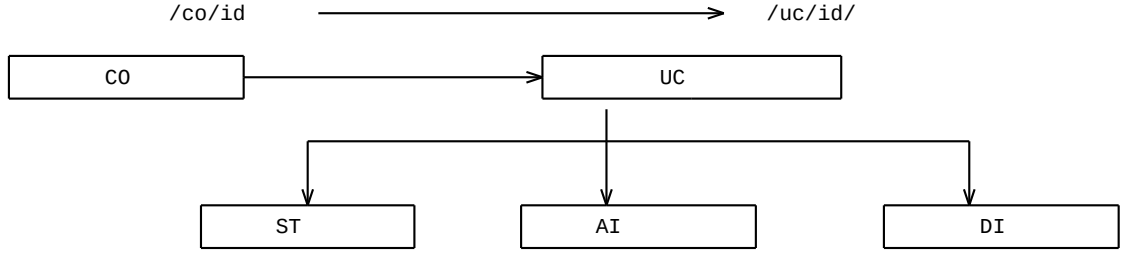
Actualizar un recurso

DELETE

Eliminar un recurso

Para aquellos clientes (navegadores) que no soporten los métodos PUT y DELETE, es posible emularlos proveyendo un parámetro extra que indique el verbo a ejecutar.

La principal unidad de operación es la "colección" la cual típicamente corresponde a los datos agrupados dentro de una tabla. La siguiente figura indica la estructura de "sustantivos" de nuestro dominio:



En consecuencia, algunas de las rutas de operación serían

- /cos/

  Todos los concentradores

- /co/{id}

  Un concentrador dado su id

- /co/{id}/ucs

  Todas las Unidades de Control de un Concentrador

- /co/{id}/uc/{id}

  Unidad de control

- /co/{id}/uc/{id}/ais

  Todas las entradas analógicas de una unidad de control

- /co/{id}/uc/{id}/ai/{n}

  Entrada analógica de una unidad de control

- /co/{id}/uc/{id}/dis

  Todas las entradas digitales de una unidad de control

- /co/{id}/uc/{id}/di/{n}

  Entrada digital de un una unidad de control

- /co/{id}/uc/{id}/evs

  Eventos de una unidad de control

  > ***Note***
  >
  > Hay que ver porque el resultado puede ser bastante grande

- /co/{id}/uc/{id}/sts

  Variables de estado de una unidad de control

- /co/{id}/uc/{id}/st{n}
- /co/{id}/uc/{id}/ev

A continuación se verán unos casos de ejemplo hipotéticos:

Obtener la lista de todos los concentradores

GET dominio/cos/

Obtener la información de un concentrador dada su clave

GET dominio/co/1234

Crear una nueva unidad de control

POST dominio/co/1234/ucs/

Actualizar la información de una Unidad de control dada

PUT dominio/co/1234/uc/123

Representación Gráfica

Los navegadores actuales son capaces de generar gráficos en tiempo real mediante la etiqueta CANVAS. Además, posen soporte para SVG, un formato de gráficos que utiliza XML para definir figuras escalables.

Objetivos Futuros

Los objetivos futuros del proyecto consisten en replicar Avrak en una estructura cliente servidor, utlizando REST como mecanismo de acceso a los recursos y HTML para la representación gráfica.

Además se añaden piezas de software al sistema como sistema asincrónicos de eventos mediante alguna técnica de [HTTP_Push].

Bibliografía sobre REST

http://en.wikipedia.org/wiki/Representational_State_Transfer

http://en.wikipedia.org/wiki/Http

http://microformats.org/wiki/rest/urls

http://en.wikipedia.org/wiki/Create,_read,_update_and_delete

http://www.cs.virginia.edu/~cs650/assignments/papers/p407-fielding.pdf

---

WikiReactor	Patrón de diseño reactor http://en.wikipedia.org/wiki/Reactor_pattern
StepehDoyle2009	http://softwareramblings.com/2008/07/multi-thread-scaling-issues-with-python-and-ruby.html
MINA	http://mina.apache.org/performance-test-reports.html
Twisted	http://twistedmatrix.com/trac/
config	http://pypi.python.org/pypi/config/0.3.7
SQLAlchemy	http://www.sqlalchemy.org/
YouTubeAvrak	http://www.youtube.com/watch?v=GS1JYdZlrK4
HTTP_Push	http://en.wikipedia.org/wiki/Push_technology