RestSCADA

Release 1

Defossé, Nahuel - Pecile, Lautaro - Zappellini, Bruno Damián - Wahler Sebastián Pablo

September 02, 2009

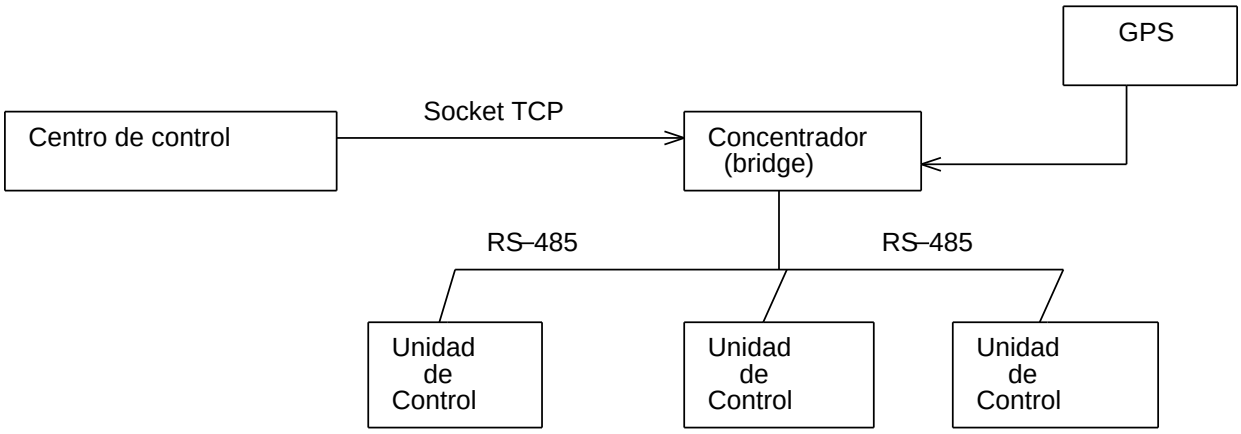
Índice general

Contents:

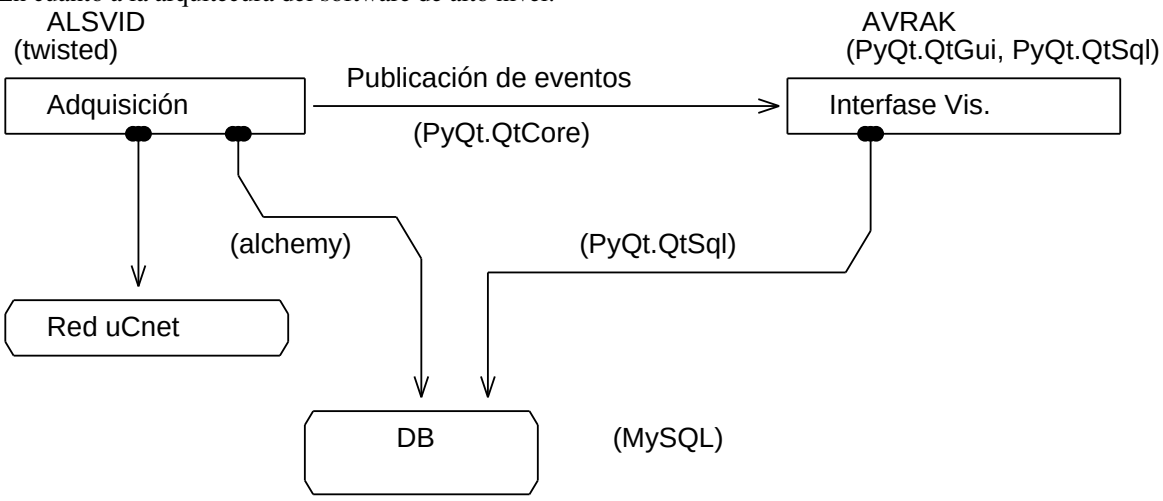
Introducción

1.1 Motivación

Durante el años 2006 al 2009 se llevó a cabo el proyecto de investigación denominado *Microcontroladores e Internet* en el cual se presentó un protocolo de comunicaciones y una arquitectura para sistemas de control.



En cuanto a la arquitecura del software de alto nivel:



El software de adquisición de estados y eventos se bautizó Alsvid, y la interfase visual montada sobre este motor (ejecutada en el mismo proceso), se denominó Avrak. Se utilizaron varias librerías de software:

- PyQt

  PyQt es un binding para la librería de C++ Qt de Nokia. Permite manipular la librería de C++ desde Python.

  - QtCore, propagación transversal de eventos
  - QtSql, conexión con bases de datos con facilidades de CRUD en GUI
  - **QtGui, elementos gráficos.** ○ Graphics Framework

- **sqlalchemy** Toolkit de acceso a base de datos, permite realizar el mapeo de tablas sobre objetos de Python y envolver las consultas.
- **Twisted** Framework de red orientado a eventos.

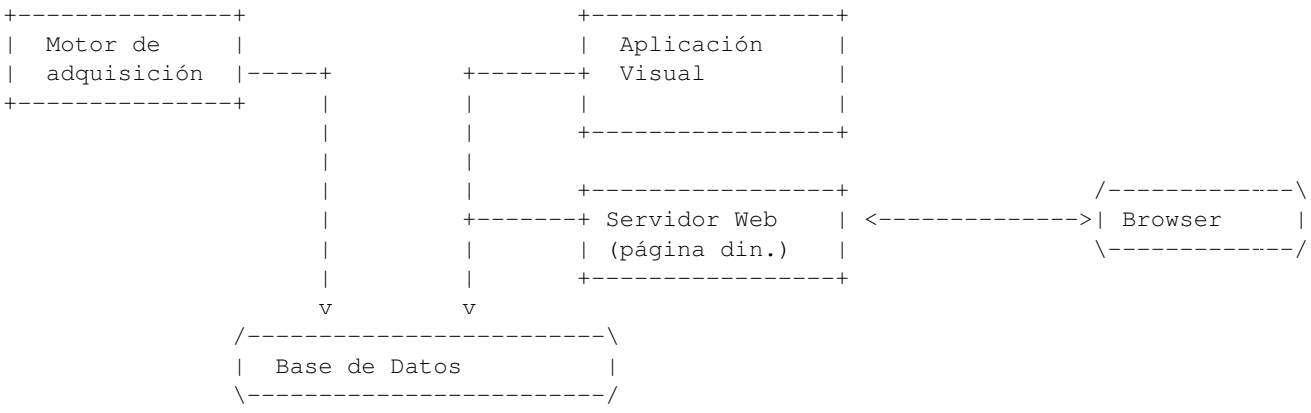
Alsvid comenzó siendo una aplicación CLI [1] con configuración offline, mientras que Avrak no solo se trata de la interfase visual, además intercepta la recepción de paquetes válidos y emite una *señal*. Los elementos gráficos conectados a esa señal ('packageReceived(pkg)') actualizaban la interfase, siendo solo necesario recurrir a la base de datos para un listado acutalizado de eventos.

Algunas de las falencias detectadas en el Avrak son las siguientes:

- Sistema de permisos débil y codificado en la GUI
- Dos mecanimos de acceso a la base de datos diferentes para un mismo fin, sqlalchemy en el motor y PyQt.QtSql en la GUI.
- Sistema de propagación de eventos basado en una librería grande (PyQt).
- El esquema de la base de datos de adquisición se modificaba conforme se extendía el dominio de la aplicación.
- La web esta ausente en el estudio
- A pesar de los standars sobre los lenguajes SQL, las bases de datos tienen sus características particulares que las hacen inaprpiadas como un mecanismo "estandard" de comunicación. Manejo del valor NULL, manejo del tipo Datetime, etc.

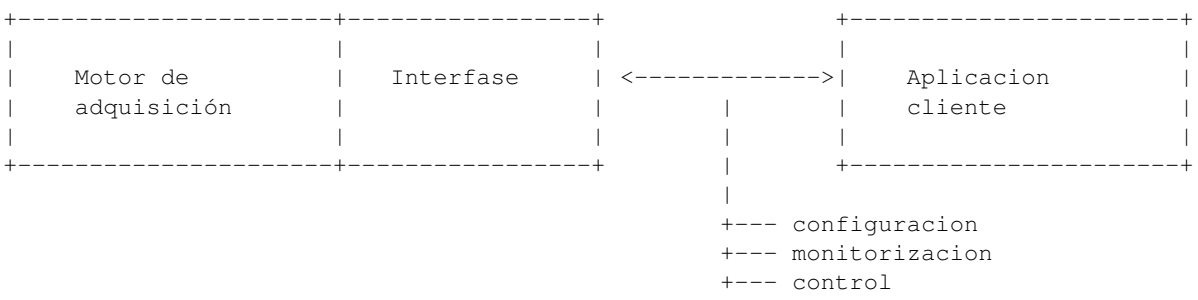
Nota: Donde digo, *se extendía el dominio de la aplicación* hablo de la modificación de los requrimientos o revisiones de la GUI.

Se require por lo tanto un mecanismo que reemplaze el esquema



[1] http://es.wikipedia.org/wiki/L%C3%ADnea_de_comandos

Se propone genrer una interfase de acceso uniforme, universal y basada en estándares al motor que permita **configuracion** [2] , **monitorizacion** [3] y **control** [4] del sistema.



[2] Crear entidades en la base de datos
[3] Recepción asincrónica de eventos en la GUI
[4] Envio de comandos a las unidades de control

CAPÍTULO 2

REST

2.1 ¿Qué es?

Es un estilo arquitectural. Un estilo arquitectural es un conjunto coordinado y nombrado de restricciones arquitecturales. En palabras de su creador:

> Una arquitectura de software está definida por una configuración de elementos-componentes arquitecturales, cuyos datos están restringidos en sus relaciones, a fin de alcanzar el conjunto deseado de propiedades arquitecturales. **Roy Fielding**

REST describe un sistema en red en términos de:

- datos (recursos, identificadores de recursos, representaciones)
- conectores (clientes, servidores, caché, resolvedores, túneles)
- componentes (servidor origen, *gateway*, *proxy*, *user agent*)

Los aspectos fundamentales de REST son:

- Los datos se acceden por una interface estandarizada, la URI.
- Los componentes se comunican transfiriendo representaciones de recursos a través de ésta interface, en vez de operar en el recurso mismo.
- Los conectores presentan una interface abstracta para la comunicación con componentes, escondiendo los detalles de implementación de los mecanismos de comunicación.
- Los componentes usan conectores para acceder, proveyendo o mediando acceso a los recursos.
- Todos los requerimientos realizados a los conectores, deben contener toda la información necesaria para la comprensión del requerimiento, independientemente de cualquier requerimiento previo.

2.2 Principios fundamentales

De la manera en que está diseñado REST, si uno se adhiere a éstos principios, obtendrá un sistema que explota la arquitectura web para beneficio propio. Los principios son:

- **Darle a todo una identificación** Todo lo que podria ser identificable, debería llevar una clave unívoca que lo determine. En la Web, esto se representa mediante la URI. Las URIs definen espacios de nombres, y permiten determinar un recurso a nivel global. Una ventaja de ésto es que el esquema es estándar, está

probado y está siendo utilizando ámpliamente. Se puede identificar tanto recursos, como conjuntos de recursos, objetos virtuales y resultados de cómputos. En fin, cualquier información susceptible de ser requerida específicamente por alguien.

- **Enlazar las cosas entre ellas** Usar enlaces a recursos e información relacionada siempre que sea posible. Esto aprovecha las capacidades de la *hypermedia*.
- **Usar métodos estándar** Para lograr una correcta interacción de los clientes con la aplicación se deberían usar adecuadamente los métodos provistos por el protocolo HTTP, `GET`, `PUT`, `POST`, `DELETE`. Lograr esto permite que una mayor cantidad de aplicaciones ya existentes puedan dialogar sin problemas con nuestro sistema, de una manera estándar y previsible.
- **Recursos con múltiples representaciones** Se deberían proveer múltiples representaciones de cada uno de los recursos, de acuerdo a las necesidades del cliente.
- **Comunicarse sin estado** REST establece que el estado de la aplicación debería ser encapsulado en un recurso, o en el cliente. En otras palabras, el servidor no debería retener algún tipo de información sobre el estado de la comunicación para ninguno de los clientes. De otra manera la aplicación no podría ser escalable ya que la carga aumentaría mientras más clientes se encuentren comunicándose con el servidor. Además, esto separa al cliente de los cambios que puedan suceder en el servidor, entre dos requerimientos consecutivos.

2.3 Convención de URLs adoptada en el sistema

Una URL modelo tendrá la siguiente forma

`http://<dominio>/<entidad>[s]/{id_entidad}/<sub_entidad>[s]/{id_sub_entidad}/.../<entidad>`

Lo que está entre corchetes es opcional. Lo que está entre llaves denota una clave de identificación. Lo que está entre corchetes rectos es un nombre de alguna entidad o dominio.

En el ejemplo del sistema que estamos desarrollando ahora, esto se podría ver de la siguiente manera:

- `http://<dominio>/cos` —> Listado de de todos los concentradores
- `http://<dominio>/co/1/ucs` —> Listado de todas las unidades de control del concentrador 1
- `http://<dominio>/co/1/uc/1/` —> Estado de la UC 1 del concentrador 1
- `http://<dominio>/co/1/uc/1/di/1` —> Estado de la entrada digital 1 de la UC 1 en el contentrador 1
- `http://<dominio>/co/1/uc/1/json/` —> Estado de la UC 1 del concentrador 1 en formato JSON (por defecto es HTML)

En nuestra aplicación las entidades posibles serán

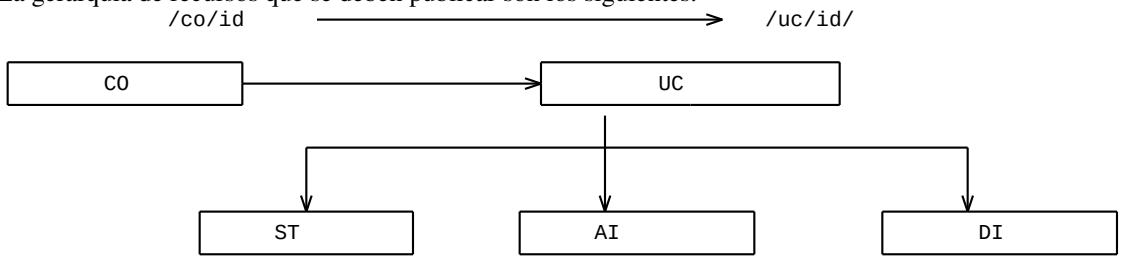
- *uc* unidad de control
- *co* concentrador
- *ev* evento
- *di* entrada digital
- *ai* entrada analógica

El formato en el cual se esperan los datos será alguno de los siguientes:

- json
- html

2.3.1 Nombrado de recursos en Restscada

La gerarquía de recursos que se deben publicar son los siguientes.



2.3.2 Semántica de los métodos *HTTP*

Recursos	GET	PUT	POST	DELETE
Colecciones de recursos como la **URI** /cosas/	**Listar** todo los los elementos de la colección	**Reemplazar** la colección completa de elementos	**Crear** un nuevo elemento en la coleccion cuyo ID sera asignado automaticamente	**Borrar** la colección completa de elementos
Miembros de una coleccion como /cosa/{id}	**Recuperar** la representacion apropiada del recurso en el tipo MIME	**Actualizar** el elemento identificado por el ID o actualizarlo	??	**Eliminar** e elemento identificado por el **ID**

2.3.3 Recursos del sistema

Entidades de la base de datos publicadas en mediante Representational State Transfer

- /cos/

  Todos los concentradores Métodos soportados: GET, POST, DELETE.

- /co/{id}

  Un concentrador dado su id Métodos soportados: GET, PUT, DELETE.

- /co/{id}/ucs

  Todas las Unidades de Control de un Concentrador Métodos soportados: GET, POST, DELETE.

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

  Unidad de control Métodos soportados: GET, PUT, DELETE.

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

  Todas las entradas analógicas de una unidad de control Métodos soportados: GET

- /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

  Nota: 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}

  Variable de estado de una unidad de control

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

  Eventos de un concentrador, no tiene sentido la url */co/{id}/uc/{id}/ev/1*. Los eventos deberían discrimirnarse por la query http, especificando los rangos de fechas en formato ISO8601 [1].

  /co/{id}/uc/{id}/evs?from_date=2008-11-05T13:15:30Z&to_date=2009-11-05T13:15:30Z

Fuentes

http://bitworking.org/news/How_to_create_a_REST_Protocol http://www.infoq.com/articles/rest-introduction http://rest.blueoxen.net/

Peticón asincrónica de datos mediante HTTP (HTTP PUSH)

[1] http://www.w3.org/TR/NOTE-datetime