Sistema de monitorització de mesures elèctriques

TOMEU CAPÓ
Universitat de les Illes Balears

Donat la gran necessitat d'estalvi energètic degut en l'encariment del preu del KW, com també intentar de conscienciar del consum intel·ligent d'energia a les empreses, en el cas de l'illa les empreses hoteleres. Es cerquen sistemes de monitorització preventiva de consum elèctric, de manera que així es pot fer una actuació més correcte i fer un ús més òptim de l'electricitat. Per això aquest projecte és centra en el desenvolupament d'un sistema que permeti adquirir mesures elèctriques, i així es pugui fer una estimació del cost equivalent tenint en compte les tarifes pertinents per calcular el seu cost. En el mercat existeixen moltes solucions propietàries (no obertes) que impliquen un cost addicional en llicències elevat, que ha cops no es suportable per l'empresa, per tant pretenem donar un sistema obert el qual es puguin integrar diferents solucions de hardware en un sol sistema per a la monitorització i anàlisi de mesures que sigui totalment flexible alhora de implementar integracions amb serveis web que es pugin emprar des de webs, aplicacions mòbils en telèfons i *tablets*. Així com també esser flexible alhora d'integrar-se amb altres solucions hardware per a l'adquisició de mesures elèctriques, i d'aquesta manera oferir una solució integral i oberta.

1. INTRODUCCIÓ

L'àmbit del projecte no és centra en la part del hardware utilitzat per mesurar variables referents al consum elèctric, si no que, vol omplir un petit buit que pensem que actualment existeix en el programari privatiu actual. Per tant, el projecte es centraria en el disseny i desenvolupament d'un sistema obert de monitorització, emmagatzematge i anàlisi de mesures elèctriques. Les solucions existents de pagament estan centrades en un tipus de hardware específic, per tant no permeten interactuar amb diferents solucions de hardware i consolidar-ho tot en un paquet de software, aleshores un dels objectius principals és poder fer flexible l'ús de diferents solucions hardware per l'adquisició de mesures i fer una solució integral de monitoratge, emmagatzematge i anàlisi del consum elèctric a distància, des de: la web, dispositius mòbils (*smartphone o tablet*).

1.1 Centrals de mesura

Primerament per poder parlar de captures de mesures elèctriques, cal fer una introducció sobre quins instruments existeixen al mercat per fer anàlisi de xarxes elèctriques, coneguts com analitzadors d'energia o centrals de mesura. Les centrals de mesura són equips (principalment d'ús industrial) que s'instal·len damunt la xarxa elèctrica a mesurar, connectat-se físicament a ella mitjançant transformadors d'intensitat els quals permeten fer lectures dels paràmetres importants de la xarxa elèctrica com pugin esser: intensitats per fase, potència activa KW i reactiva KVAR, COS φ, harmònics, ... Aquesta darrera variable és important per que permet poder detectar i preveure harmònics generats per equips electrònics que perjudiquin a elements existents dins una instal·lació com a bateries de condensadors[1], aquestes són emprades en indústries per fer correcció del factor de potència o COS φ i així evitar recàrrecs en concepte de reactiva en la factura mensual. Molts de cops, moltes d'empreses tenen instal·lades aquestes centrals de mesura sense donar-lis un ús, sense tenir-ho integrat amb el seu sistema de IT, ja sigui per desconeixement o bé per falta de pressupost per adquirir una solució software que ho integri.

Les centrals de mesura es comuniquen mitjançant comunicacions sèrie via RS-232 només per un dispositiu o bé via RS-485 que permet encadenar en un bus una sèrie de dispositius de mesura. Existeixen altres centrals de mesura que funcionen a traves de *Ethernet* o *WiFi* però en el nostre cas no les emprarem. Ens centrarem en dos models diferents de dos fabricants diferents: Un és la central de mesura *PowerLogic* PM710 de Schneider-Electric que ens hi comunicarem emprant RS-485 amb un protocol estàndard per aquests tipus de dispositius conegut *ModBUS*, i l'altre és l'analitzador d'energia CVMk de Circutor que ens hi comunicarem via RS-232 i empra un protocol propietari en mode ASCII.

1.2 El sistema

Mirant en el mercat, no hi ha molts de sistemes oberts que integrin diferents solucions hardware i pugui consolidar les dades en un sol sistema de gestió i monitorització. Amb aquest sistema he intentat aplicar la regla de que no estigui lligat a un sol hardware, i fer-lo lo més flexible possible. Per tant, un dels primers punts a tenir en compte és el llenguatge triat pel seu desenvolupament: Python, s'ha triat aquest ja que simplifica el tractament de la informació així com l'ús de diversos paradigmes de programació: Funcional, imperatiu i OO.

El sistema està pensat per que pugui esser distribuït, amb això diem que podem tenir una sèrie de nodes i cada node amb una sèrie d'analitzadors d'energia connectats a ell. A afectes pràctics, podria passar que una empresa, per exemple, una cadena hotelera te diferents hotels i es vol monitoritzar el consum energètic de cada hotel i centralitzar les mesures de consum a un servidor central, per tant tenim un node per hotel. Definim com a node a un ordinador amb el sistema servidor de mesures el qual fa tota la tasca de planificar cada quin temps t_{sample} es van agafant mostres (lectures) i guardar les a disc cada t_{store} on normalment $t_{sample} \leq t_{store}$. El node també podrà enviar dades a un *webservice* i que aquest guardi les lectures dins una base de dades central per després fer estadístiques i poder fer estimacions de costs en base del consum elèctric.
Com que volem flexibilitzar l'ús de qualsevol central de mesura, el sistema també ha de permetre configurar el hardware sense tocar la funcionalitat de la totalitat del sistema, d'aquesta manera, definirem una capa que serà intercanviable que seran els *drivers*. Els *drivers* ens

[1] Les bateries de condensadors són equips formats per diversos condensadors disposats en paral·lel, que permeten reduir de manera considerable la demanda d'energia reactiva d'una instal·lació, millorant d'aquesta manera la qualitat del subministrament i optimitzant el seu rendiment, a la vegada que s'obté un important estalvi en la factura de consum elèctric.

permetran definir una capa d'abstracció del hardware en qüestió, de manera que en base al un fitxer de configuració puguem instanciar depenent del hardware que vulguem manejar.

2. PROBLEMÀTICA

Inicialment aquest projecte començar per una necessitat de poder capturar mesures de consum energètic a partir d'un sol analitzador d'energia (Circutor CVMk), i degut a que el software proporcionat pel fabricant estava ja obsolet i que només funcionava per MS-DOS. Es volia integrar en una web aquestes mesures i representar en una sèrie de gràfiques, es va decidir començar la implementació d'una petita aplicació feta amb *Python* que fes les captures cada cert temps, mesures de consum i les guardés dins un fitxer de text. A partir començaren a sorgir problemes implícits a les diferents necessitats i requeriments.

2.1 Comunicacions

Una de les primers necessitats va esser implementar el mòdul de comunicacions per comunicar-mos amb l'analitzador i així poder extreure dades, per això va esser necessari: Primer buscar una llibreria per *Python* per manejar el port sèrie. Segon agafar l'especificació del protocol del analitzador d'energia CVMk per implementar els mètodes de enviar i rebre trames. En aquest punt, un dels problemes que sorgiren foren els temps d'espera que existeixen entre les transmissions i recepcions de trames i la necessitat de buidar els buffers de entrada/sortida del port sèrie per no tenir problemes de solapament de trames, i que si falla la comunicació una vegada que tornem a llegir el port sèrie per un altre trama totalment diferent no trobem a la recepció restes de la trama anterior provocant així un error d'interpretació de les respostes. Per tant, buidem el buffer de recepció una vegada s'ha rebut total o parcial (degut a errades) la resposta del dispositiu.

2.2 Concurrència

Per poder realitzar diferents tasques i organitzar les diferents parts de l'aplicació emprem diferents fils d'execució. Hem de tenir en compte que nosaltres empra'm la llibreria estàndard de *Threading* de *Python* que realment crea fils de sistema operatiu però depenent de si s'empra una implementació o un altre de la VM de *Python* podem tenir un resultat o un altre a nivell de rendiment. En *CPython* i degut al *GIL*[2] un fil només pot executar el seu codi alhora, ara bé quan es volen executar diverses tasques de E/S alhora solen esser apropiats l'ús de *mutex*. Nosaltres una de les coses que fem es tenir un fil per cada analitzador que volem consultar.

Quan hi ha un sol dispositiu a controlar es bastant ideal, i no necessitem controlar concurrència d'us d'un mateix recurs per consultar diferents dispositius. Com hem dit abans, pot passar que vulguem comunicar nos amb diferents centrals de mesures des de un sol node i molt concretament des de un recurs compartit com un sol port sèrie, aleshores, existeixen diferents maneres de resoldre-ho: Fer consultes serialitzades (una darrera un altre), primer consultar tot un dispositiu i després un altre i fer-ho amb el mateix codi, això pot augmentar el temps de resposta del sistema. O bé definir un *thread* per cada central de mesura i fer que cada operació lectura/escriptura estigui dins una secció crítica englobada per mutex que sigui el propi driver. Inicialment es va fer que la secció crítica estigues dins tot una operació completa de consulta del dispositiu:

```
mutex.acquire();
analitzador.consulta();
mutex.release();
```

Tenint en compte que el mètode de consulta engloba moltes operacions de lectura/escriptura d'un dispositiu, per tal d'atomitzar les operacions hem baixat el bloqueig a nivell de lectura/escriptura de cada una de les peticions de consulta per separat.

```
mutex.acquire();

Si no dispositiu.enviar(peticio) llavors
    mutex.release();
    continuar;
fi si;
resposta = dispositiu.rebre();

mutex.release();
```

Els mètodes enviar i rebre empren operacions de lectura/escriptura del dispositiu amb controls de *timeout* per evitar esperes actives.

3. REQUERIMENTS

Es necessita poder capturar les mesures de consum energètic i emmagatzemar les a disc en fitxers de format de text que es pugin importar des de un full de càlcul, així com també es pugi publicar via web les gràfiques de consum energètic diari i totes les altres variables que s'enregistrin i que l'usuari vulgui visualitzar. Aquestes mesures les enviarà cada servidor emprant un servei web tipus REST, i tindrà la capacitat de emmagatzemar les lectures que no pugui enviar de manera que si hi ha una errada puntal de comunicacions no es perdran les lectures fetes. El sistema servidor, s'ha de poder executar damunt qualsevol plataforma.

4. ARQUITECTURA GENERAL

El sistema està compost per diferents parts o mòduls, els quals tenen una missió especifica. Com hem dit a la introducció, cada node te el sistema de lectures en el qual hi ha connectat un o uns analitzadors d'energia. El servidor de lectures, és com s'anomena el mòdul principal i és el que s'encarrega d'instanciar i gestionar tot el procés entre els diferents mòduls secundaris. Cada mòdul te la seva funció especifica:

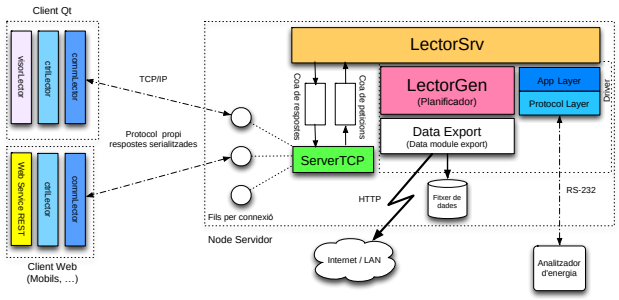


Figura 1 Diagrama de blocs del sistema

- **LectorSrv**: Servidor de lectures, s'encarrega d'instanciar els *drivers*, planificador i servidor TCP. També gestiona els missatges que ens ve per els clients que es connecten via TCP amb el ServerTCP.

[2] ***General Interpreter Lock**: The mechanism used by the CPython interpreter to assure that only one thread executes Python bytecode at a time. This simplifies the CPython implementation by making the object model (including critical built-in types such as **dict**) implicitly safe against concurrent access. Locking the entire interpreter makes it easier for the interpreter to be multi-threaded, at the expense of much of the parallelism afforded by multi-processor machines.*

- **Drivers**: Està dividit amb dues parts ben clares, i que explicarem més endavant: **App Layer** i **Protocol Layer**. Aquestes dues capes són modulars, o sigui que, si es necessita comunicar amb un altre dispositiu simplement s'implementa els dos drivers i es carrega com si fos un plugin.

- **LectorGen**: Es la part que gestiona la manera en cada quan es fan les lectures i cada quan es graven a disc. Actua com a planificador. Instancia'm un mòdul d'aquests per analitzador connectat a un node. Aquest mòdul es de tipus *Thread*, o sigui que podem tenir instanciat 3 LectorGen's i poden actuar de manera independent un de l'altre.

- **Data Export:** Exporta les dades obtingudes de l'analitzador o analitzadors. Instancia'm un **DataExport** dins el mòdul **LectorGen**, això significa que tenim un *thread* d'exportació de dades per analitzador. Aquest mòdul permet configurar diferents submòduls per poder gravar les dades a un fitxer local al mateix node o a un node central a través d'un client *webservice* de tipus REST. Es configurable des de el fitxer de configuració general de l'aplicació

- **ServerTCP**: Petit servidor de *socket* TCP que permet gestionar tot el servidor de lectures de manera remota, així com permet donar lectures instantànies de les variables capturades pels *LectorGen* de cada analitzador connectat a un node.

4.1 Drivers

Un dels objectius d'aquest projecte també és de que no depengui de cap fabricant en concret pel que fa al hardware. Per això s'ha emprat un concepte que en *Python* es sol conèixer com a *plugin*, això ens permet a nivell de codi poder fer una càrrega d'un mòdul *Python* arbitrari en temps d'execució i sense necessitat de tocar el codi base del programa, simplement li especificarem via fitxer de configuració el mòdul que volem carregar. Cada *driver* s'ha decidit que tingui dues parts: la de Protocol Layer y la de App Layer.

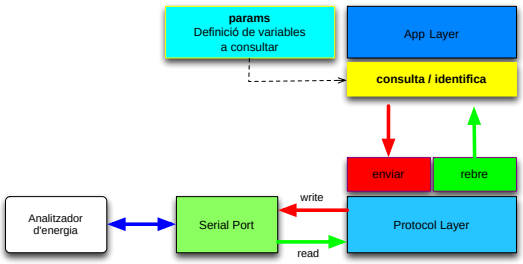


Figura 2 Estructura dels drivers

- **Protocol Layer:** és una classe que implementa dos mètodes bàsics: **enviar** i **rebre**, els quals defineixen les trames i poden interpretar les trames en el format específic del hardware. Implementen el protocol de comunicacions del dispositiu a monitoritzar/controlar.
  Encara que la capa de protocol podria esser emprant protocols de molt alt nivell com TCP/IP, en el nostre cas implementarem els que necessitem com: ModBUS RTU i ASCII (Circutor). El protocol ModBUS[3] RTU s'empra en comunicacions sèrie, es un protocol tipus mestre-esclau, només hi pot haver un màster connectat al bus. Els nodes esclaus del bus no es comuniquen entre si, i és el màster el que inicia la comunicació sempre.
  El node màster pot efectuar una petició als nodes esclaus de dues formes: **Mode unicast**, el màster empra la adreça individual del esclau amb el qual es vol comunicar i l'esclau una vegada rebuda la petició respon al màster. **Mode broadcast**, el màster envia una trama a tots els esclaus, però aquests no tornen resposta, normalment s'empra l'adreça 0 per aquest mode.

  Les trames són en format binari, està composta per: una operació, unes dades i un codi de comprovació CRC de 16 bits. Tant les trames de enviament o bé de resposta tenen la mateixa forma.

  Primer ve **l'adreça** del dispositiu esclau a consultar o controlar, pot anar de 1 a 247.
  El **camp de funció** especifica el tipus d'operació que volem realitzar damunt el dispositiu esclau:

Funció	Descripció
3	**Llegeix** n paraules de l'àrea de lectura/escriptura
4	**Llegeix** n paraules de l'àrea de lectura
16	**Escriu** n paraules de l'àrea de lectura/escriptura

  El camp de dades depèn del codi de funció, aquesta part pot tenir un màxim de 60 bytes. Un mínim de 3 bytes per les trames de resposta, o un mínim de 4 bytes per les trames de consulta. Per exemple si volem consultar una variable del dispositiu la àrea de dades està compost per la adreça del registre a llegir i la quantitat de paraules que es vol llegir:

  L'altre protocol que hem implementat és el dels analitzadors del fabricant **Circutor** i que estan bastats en caràcters purament ASCII. És molt senzill, consta de: un inici de trama, la adreça del dispositiu, la comanda i un *checksum* que es calcula mitjançant una suma de tots els caràcters anteriors mòdul 256,

[3] ***Modbus*** *is a serial communications protocol published by Modicon in 1979 for use with its programmable logic controllers (PLCs). Simple and robust, it has since become a de facto standard communication protocol, and it is now amongst the most commonly available means of connecting industrial electronic devices.*

com que es transmet en ASCII s'envia el *checksum* passat a un seqüència alfanumèrica, ja que es transmet i el dispositiu el llegeix en hexadecimal.

STX	@ Slave	Command	Checksum	ETX
1 byte	2 bytes	3 bytes	2 bytes	1 byte

Una exemple de transmissió i recepció d'una comanda, seria la de consultar la tensió simple de la xarxa elèctrica:

TX: $00RVI75

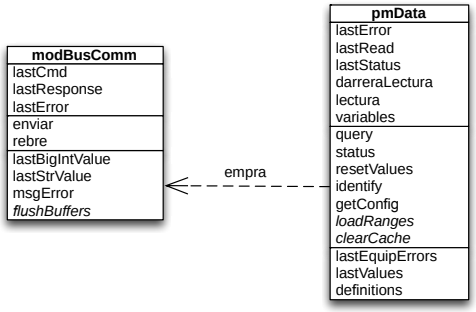
RX: $00000000234000000233000000232000000233 64

Tant la trama de petició com la de resposta comencen i acaben de la mateixa forma, en el nostre cas la adreça del dispositiu és la 0 i la comanda es la RVI: Tensió simple, ens torna totes les tensions en un sistema trifàsic: $V_R = 234v, V_S = 233v, V_T = 232v$. I al final el *checksum* en hexadecimal, seguit d'un caràcter ETX que és el CHR(10).

- **App Layer**: és la que emprarem a nivell de **LectorGen** i que també te una sèrie de mètodes ben definits i sempre es diuen el mateix per cada *driver*: **query**, **identify, getConfig, status, lastValues**. El mètode de **query** permet llançar una bateria de peticions per llegir un conjunt de variables (Potències, Tensions i Intensitats, ...) i guardar les temporalment a una estructura de tipus diccionari de llistes que cada clau del diccionari és el nom de la variable en qüestió. Per determinar quines variables suporta cada *driver* existeix un diccionari de definicions anomenat *params*, a on tenim definit cada nom de variable i la seves característiques tant per el registre físic on te emmagatzemat la dada a dins el dispositiu, el factor d'escala, el valor màxim:

```
"PAF":{"registre"  : 4036,
       "numRegs"   : 3,
       "descripcio": "Pot. Activa/fase",
       "regEscala" : "FEP",
       "valEscala" : 0,
       "valMax"    : 32767,
       "compost"   : False}
```

La capa de aplicació (App Layer) necessita de la de protocol per poder-se comunicar amb el dispositiu, i la de aplicació es la encarregada de preparar i deixar les dades en el tipus correcte. Una funcionalitat que es pot implementar a la capa d'aplicació és la de tenir una petita persistència que guarda la configuració llegida amb el mètode **getConfig** la primera vegada en el cas del *driver* ***pmData*** per l'analitzador *PowerLogic PM710* guardam els rangs de les variables que ve donat pel mateix equip, la cadena identificadora del producte i el darrer estat vàlid del equip. Aquest mètode només es crida un cop, quan arrenca'm l'aplicació i el *LectorGen* corresponent a aquest analitzador.



4.2 LectorSrv

El Lector Server és el fil principal d'execució que es llança del programa principal, el qual: Llegeix el fitxer de configuració, obri ports de comunicacions, instancia els *drivers* pels analitzadors definits dins el fitxer de configuració (config.xml), com hem dit al apartat 2.2 si dos analitzadors comparteixen el mateix bus de comunicacions o sigui el mateix port de comunicacions, necessitarem un *mutex* per port, aquest es crea al mateix moment que s'obri el port de comunicacions això es fa al nivell del LectorSrv. Una vegada instanciats els *drivers* i els dispositius de comunicacions el que fem es per cada analitzador definit dins la configuració instanciar un *thread* de tipus *LectorGen*. Així com també instancia el servidor de comunicacions TCP ServerTCP, els dos *threads* es comuniquen mitjançant dues coes: La de peticions que venen del ServerTCP i la coa de respostes.

Per poder

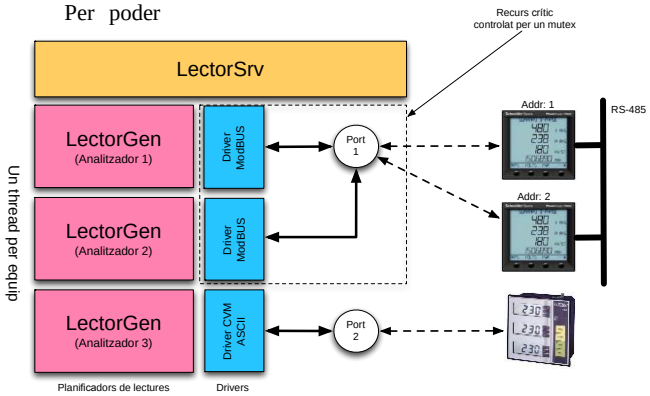


Figura 3 Exemple de instanciat múltiple compartint un recurs únic.

despatxar de manera, més eficient els missatges i evitar contingència definim un "pool de *workers*" el qual cada worker agafa el missatge existent a la coa de peticions, aquest missatge és una *tupla* formada per: La operació, el argument, la referència al *socket* del part del client per tornar la resposta, de manera que el que s'encarrega de les comunicacions és únicament el *ServerTCP*. El argument generalment és l'identificador del analitzador que emprarà l'operació, només hi ha, de moment, una operació en la qual el seu argument no és el *id* de analitzador, que és la de l'obtenció de la configuració del servidor que permet dir-li si ho volem en format XML o en format d'objecte natiu Python serialitzat.

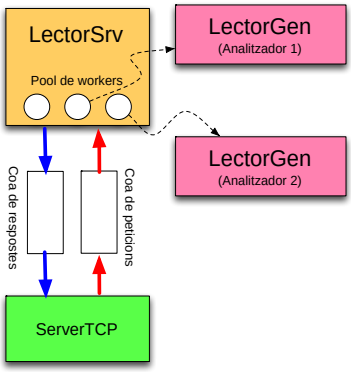


Figura 4 Comunicació interna entre fils (LectorSrv i ServerTCP) mitjançant coes.

El propi *LectorSrv* determina quin mètode cridar en base a la operació rebuda mitjançant un diccionari de *Python* que relaciona les operacions amb els mètodes a cridar, si no esta definida la operació demanada també o comprova i torna un error de comanda no valida o no implementada.

Comanda	**Descripció**
GETID	Ens torna l'identificador d'un analitzador.
GETVARS	Ens torna el diccionari que conté les variables que es consulten cada t segons pel LectorGen.
GETCONF	Ens torna la configuració del sistema o amb XML o amb forma d'objecte serialitzat.
GETDEFS	Ens torna la llista de paràmetres que te el driver de l'analitzador que consultem, amb la descripció.
GETSTATUS	Ens torna l'estat de l'analitzador.
IDENT	Força d'identificació de l'analitzador.
FORCEREQ	Força una petició d'una lectura completa de les variables d'un analitzador.
SHUTDOWN	Atura completament l'aplicació.
START	Arranca el LectorGen que especificam
STARTALL	Arranca tots els LectorGen definits en la configuració.
PAUSE	Posa en pausa un LectorGen
RESUME	Continua l'execució d'un LectorGen
RELOAD	Recarrega el fitxer de configuració en calent.
STATUS	Determina l'estat d'un LectorGen
QUIT	Surt de la sessió amb el LectorSrv

Es pot dir que tenim dos tipus de comandes: les de consulta i les administratives per controlar el servidor. Les de consulta permeten com es pot veure a la taula consultar l'estat, darreres lectures i identificació dels analitzadors connectats.

El LectorSrv es configura mitjançant un fitxer XML (config.xml) que el llegim al principi i conté: La configuració dels ports de comunicacions amb els que el sistema treballarà, i el conjunt d'analitzadors amb la seva configuració pertinent:

```
<devices>
   <device id="1" type="serial" enabled="true">
      <port>/dev/tty.usbserial-00004006</port>
      <baud>19200</baud>
       <bits>8</bits>
       <timeout>1</timeout>
   </device>
</devices>
```

La configuració de l'analitzador conté el dispositiu el qual emprem per connectar-nos a ell, o sigui el *id* del *device* creat al principi com hem dit, els *drivers* que emprarem per dialogar amb ell, els exportadors de dades com per exemple: guardar el fitxers en local i el client REST. A la secció del propi analitzador, es posa el conjunt de peticions que el LectorGen llançarà cada vegada que consulti el dispositiu, aquesta bateria de peticions es defineix amb una llista que conté el conjunt de variables a llegir en format abreviat, que ve especificat pel driver emprat. Per tenir una referència, són abreviatures: **TPF** (Tensió Per Fase), **IPF** (Intensitat Per Fase), **FRE** (Freqüència), **PAF** (Potència Activa Per Fase).

```
<analitzador id="1" device="1" model="PM710"
driver="modBusComm" dataDriver="pmData">

  <params>
       <fabricant>Schneider</fabricant>
       <tempsgravacio>5</tempsgravacio>
       <tempslectura>60</tempslectura>
  </params>

  <dataexports>
    <dataexport type="localFile" target="dades" />
    <dataexport type="clientRest"
target="http://servidor/central/ws" />
  </dataexports>

  <peticions>
       <consulta>TPF</consulta>
       <consulta>IPF</consulta>
       <consulta>FRE</consulta>
       <consulta>PAF</consulta>
  </peticions>

</analitzador>
```

Tenim un objecte anomenat *configurador* el qual és el que llegeix el fitxer i comprova que tot estigui correcte tant sintàcticament com semànticament, això vol dir que si definim un analitzador i fem referència a un dispositiu no definit saltarà una excepció de que falta un dispositiu per aquest analitzador, podem tenir *n* dispositius de comunicacions així com *m* analitzadors cada configuració.

4.3 LectorGen

El Lector Genèric és un planificador de lectures. L'avantatge d'aquest mòdul es que és independent del hardware, quan l'instanciem li passem el *driver* i com que tots els *drivers* de la capa d'aplicació tenen els mateixos mètodes, es transparent el canvi de *drivers* pel mateix codi, simplement instanciem un LectorGen amb un nou *driver* i funciona igual. Aquest mòdul una vegada arranca el que fa es identificar l'analitzador guardant-se així la cadena que identifica el hardware. A partir d'aquest punt el que fem es instanciar els exportadors de dades *DataExport* que veurem mes endavant el qual li passam dades mitjançant una coa.

El LectorGen, com a mostrejador s'inicialitza amb dos temps que

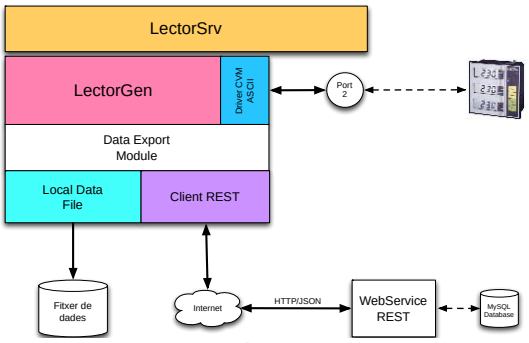


Figura 5 Estructura dels mòduls que carrega el LectorGen quan s'instancia.

hem comentat a la introducció: t_{sample} que és el temps de mostreig, és cada quan s'executa el mètode **query** del *driver* i que fa una bateria de peticions de cap a l'analitzador d'energia, aquestes peticions es configuren dins el fitxer de configuració (config.xml).

4.4 Data Export

Aquest mòdul és la que s'encarrega de poder exportar les dades capturades pel LectorGen i emmagatzemares com necessitem. Això significa que es flexible ja que permet definir submòduls per cada

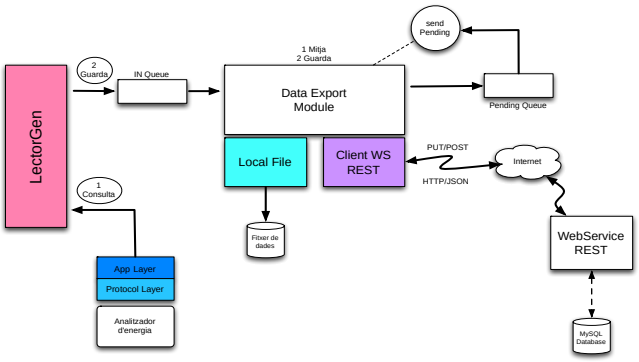


Figura 6 Diagrama de blocs i flux de dades entre el *LectorGen* i el *Data Export*.

tipus d'exportador de dades. Tenim definits dos tipus de submòduls: Local File i Client REST, un grava les lectures en local dins un directori especificat per fitxer de configuració, i el client REST es un petit client que exporta les dades de cap el servidor central que te l'aplicació central de mesures i estadístiques. Com es pot veure a la **¡Error! No se encuentra el origen de la referencia.**, tenim una coa per poder passar les lectures fetes en el **LectorGen** i gravar-les de manera asíncrona al data export de manera que no es demori el temps de gravació ni per temps de xarxa ni per temps de gravació en disc, simplement deixem la darrera lectura dins la coa i el **Data Export** les recoeix ja que s'està executant dins un altre thread ell sempre espera tenir una tasca a la coa de lectures pendents de gravar, l'agafa i per tots els submòduls definits per aquell LectorGen crida al mètode **save** del submòdul, en el cas del **Local File** grava a un fitxer de text i en cas del client REST envia de cap al webservice central.

En aquest punt poden sorgir problemes de comunicacions en el cas de que emprem el mòdul de Client WS REST, ja que el servidor o no hi podem arribar o bé està fora de servei, el sistema ha d'esser totalment autònom i poder emmagatzemar les lectures en memòria durant un temps llarg de manera que si en un interval de temps el servidor no estar disponible passar aquestes dades del moment a un altre coa de tasques pendents d'enviar i amb un *thread* a part anar mirant cada minut si es possible enviar les tasques pendents dins aquesta coa, si aquesta coa de pendents s'omple, el que fem es passar totes les dades i passar-les a una estructura en memòria tipus llista per a guardar-les a una petita base de dades local tipus SQLite per no perdre lectures, de manera que es pugin recuperar després quan hi torni haver comunicació.

4.5 Server TCP

Mòdul que gestiona les comunicacions TCP/IP del LectorSrv. Està basat amb un patró de disseny tipus reactor, es comunica amb el LectorSrv el qual s'encarrega d'executar les diferents ordres que li arriben a través de la coa de peticions i torna els resultats per la coa de respostes per que el ServerTCP les pugi enviar als clients. La missió principal d'aquest mòdul és la de escoltar les connexions TCP/IP, llegir la petició, interpretar-la i enviar-la de cap el LectorSrv però també controla el tema d'errors com les comandes invalides o comandes amb el format incorrecte i desconnectar als clients que sobre passen el nombre de comandes errònies per evitar segons quins tipus d'atacs al servei, així com també limita el nombre de clients concurrents al sistema amb el nombre de peticions encolades.

Per iniciar un diàleg amb el servidor és necessari iniciar un *handshake* inicial, mitjançant dos *tokens*: El client envia #$ i el servidor li respon $# en aquell punt el servidor està amb disposició de

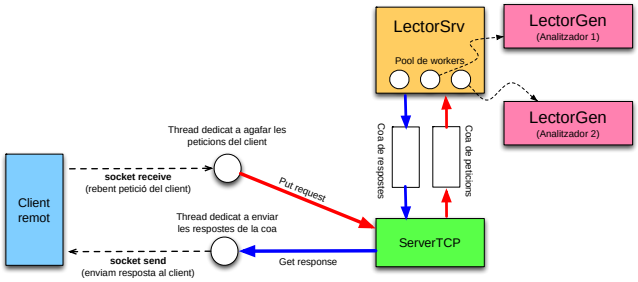


Figura 8 Flux de missatges entre el client i el servidor.

rebre comandes les quals comencen amb un $ i acaben amb un ; seguit d'un retorn de carro '\r'. Si el client ha iniciat sessió i sobrepassa el temps de *timeout* es desconnectat per inactivitat. Aquí tenim un petit exemple de comunicació amb el servidor:

TX: #$
RX: $#
TX: $GETID.1#\r
RX: PM710 Power Meter\r
TX: $QUIT#\r
RX: DISCONNECTED!\r

Per donar sopor a aquest protocol, s'ha implementat un API de desenvolupament molt bàsic tant per clients en PHP com en Python el qual defineix dues classes bàsiques, una de comunicació i l'altre que implementa totes les funcions específiques del servidor, un mètode per comanda. D'aqueixa manera existeixen dos clients, un de línia de comandes el clientLector.py, que permet enviar comandes al servidor i veure el resultat per text. L'altre client és el visor.py el qual és un client d'escriptori que esta basat Qt i que funciona tant per OSX, Windows i GNU/Linux.

4.6 Frontal web i web service

Per poder integrar tot aquest sistema de manera que es pugui comunicar amb l'exterior i proporcionar tant una interfície per permetre les consultes, generació de gràfiques històriques de consum energètic i altre variables que anem adquirint amb el sistema proporcionarem una interfície web i un webservice per la integració amb tecnologies mòbils com tablets i smartphones o integració amb sistemes de datawarehouse.

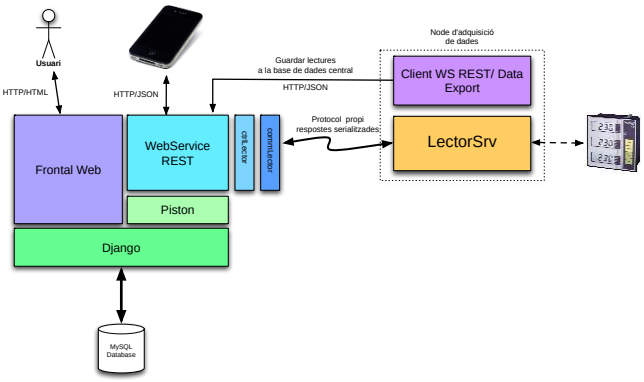


Figura 7 Arquitectura general entre el LectorSrv i els frontals web

Aquesta part ha estat desenvolupada amb el *framework* per *Python*, anomenat *Django* i un mini-framework per *Django* per implementar el *webservice* tipus REST anomenat Django-Piston. El *webservice*, te varies missions: Permetre la comunicació en temps real entre qualsevol tipus de client REST i un node ***LectorSrv*** per veure lectures en temps real, ja que hem definit diversos recursos REST al *webservice* un per a cada funció, tenim dos *handlers* principals el **lector_handler** i el **history_handler**. El lector_handler permet establir una comunicació emprant l'API de comunicacions que hem comentat abans format per dues classes la de **commLector** i la de **ctrlLector** la de que empra el *webservice* es la de **ctrlLector**:

```
cliLector = ctrlLector((host,50007))
retval = cliLector.getId(analitzadorId)
cliLector.desconnecta()
```

D'aqueixa manera tant amb un client per un smartphone com desde la mateixa web frontal amb AJAX podem consultar lectures en temps real emprant aquest recurs del webservice REST.
De manera que si ens connectem al webservice sol·licitant la adreça http://tesla.botilla.net/central/ws/rt/get/Vars/3 ens tornarà una estructura en JSON de les lectures instantànies que ha llegit el node 3:

```
{
    "values": {
        "TPF":  [229.20, 232.5, 232.60],
        "PAT":  [45.2],
        "CEA":  [3734937.19],
        "FRE":  [50.01],
        "FPT":  [0.972],
        "PRT":  [10.9],
        "PAF":  [18.2, 14.9, 12.20],
        "IPF":  [81.5, 65.2, 54.6, 35.6],
        "THDV": [2.30, 1.8, 1.70]
    },
    "lastRead": "21/10/2012 20:01:23"
}
```

Després el segon servei que ofereix el webservice és la gestió d'històric de lectures de tots els nodes que emmagatzemen emprant el data export+client REST, aquest servei te la capacitat de guardar (save) i carregar donat una data inici, data final, variable i ID del node, tornat així un conjunt de registres de les lectures compreses dins aquelles dues dates.

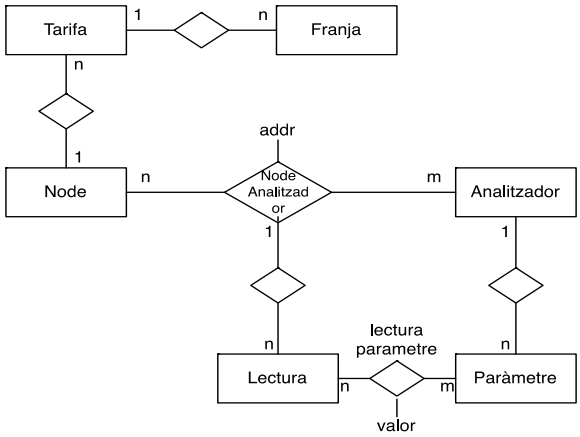
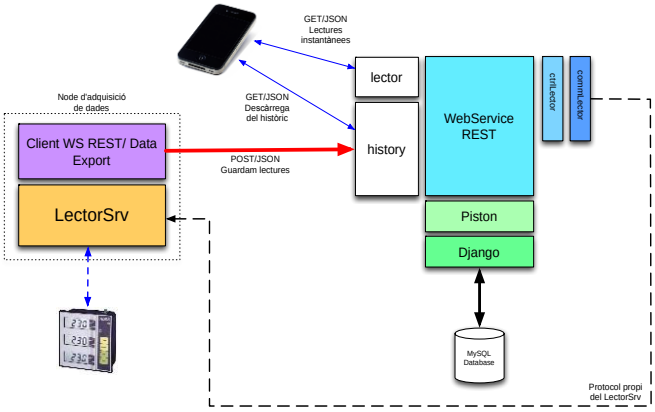


Figura 9 Model entitat-relació del la base de dades

La base de dades central per l'anàlisi de dades està creada emprant els models que incorpora Django seguint el model entitat-relació que podem veure a la Figura 9. Necessitarem parametritzar tots els nodes dins la base de dades així com tots els tipus d'analitzadors d'energia amb els seus paràmetres que suporta cada un d'ells. Les lectures s'emmagatzemen dins la taula ***LecturaParametre*** la qual conté el valor de la lectura, tan pugui esser un vector de valors com un valor sol, això dependrà de la variable que es llegeixi per exemple la **PAF** (Potència Activa per fase) és un vector de valors, aquesta la guardarem en format JSON dins una cadena de caràcters per no complicar el model hem decidit fer-ho d'aquesta manera i si és una variable d'un sol valor, per exemple **FRE** (Freqüència) aquest anirà dins un camp tipus *double*.



El frontal web consta de dues parts: El frontal de consultes en sí, i el backend o frontal de configuració que està fet amb el mateix frontal d'administració que ofereix el Django, allà és on es configura totes les taules bàsiques i es parametritzen i associen les tarifes que tenim a cada node o centre. El sistema permet fer un càlcul d'estimació de costs en base a la configuració de tarifes i franges horàries, la configuració de tarifes i franges horàries ve donada pel tipus de contracte que tinguem, per exemple podríem tenir diferents preus de kW/h a diferents hores. Amb aquesta configuració podem demanar al sistema que ens faci un càlcul de cost per tarifa que tinguem i així poder determinar el cost aproximat de la pròxima factura, de moment el sistema només calcula el cost del kW/h de la potència activa i estima un cost per dia, poden consultar des de el frontal des de una data a un altre.

5. RESULTATS

El sistema s'ha desplegat sense problemes a tres centres hotelers diferents i s'ha instal·lat un servidor central que conté la base de dades central de mesures que va emmagatzemant les dades que li envien els nodes a través del *webservice*, i s'ha configurat a un dels centres totes les tarifes per poder fer el càlcul de les estimacions de costs per kW/h, concretament s'han definit una tarifa i tres franges:

Tesla Analyzer

Iniciar la sessió | Registra't | Consulta de lectures | Estimació de costs | Visors temps real

Node: CIFP Joan Taix
Analitzador: PM710 (Addr = 1)
Desde: 10-10-2012 Fins: 22-10-2012
Consultar

	Tarifa P1		Tarifa P2		Tarifa P3		Totals	
Data	kW	Import	kW	Import	kW	Import	Consum (kW)	Import
10-10-2012	850.00	152.73	5468.68	756.75	1198.87	103.87	7517.55	1013.35
11-10-2012	804.84	144.61	5080.28	703.00	1195.09	103.54	7080.20	951.16
12-10-2012	265.76	47.75	1647.94	228.04	1070.09	92.71	2983.79	368.50
13-10-2012	234.75	42.18	1473.01	203.83	1007.67	87.30	2715.44	333.32
14-10-2012	234.14	42.07	1481.40	204.99	991.23	85.88	2706.77	332.94
15-10-2012	729.35	131.05	7455.34	1031.66	1119.44	96.99	9304.13	1259.70
16-10-2012	827.69	148.72	6698.51	926.93	1138.90	98.67	8665.10	1174.33
17-10-2012	806.43	144.90	6204.63	858.59	1171.75	101.52	8182.81	1105.01
18-10-2012	819.76	147.30	5140.40	711.32	684.90	59.34	6645.06	917.96
19-10-2012	728.19	130.84	6867.19	950.28	1224.67	106.10	8820.05	1187.22
20-10-2012	239.86	43.10	1449.71	200.61	998.86	86.54	2688.42	330.25
21-10-2012	236.22	42.44	1444.66	199.91	984.78	85.32	2665.66	327.68
22-10-2012					87.78	7.61	87.78	7.61

P1, P2 i P3 horàries amb tres costs per kW/h, hem de tenir en compte que per calcular els cost el que fem es fer la mitja de lectures en un període $T_i =< H_i, C_i >$ on $H_i = [t_1, t_2]$, el seu cost C_i. Com

$$P_i = \frac{\sum_{i=1}^{n} p_i}{n} \tilde{T}(H_i)$$

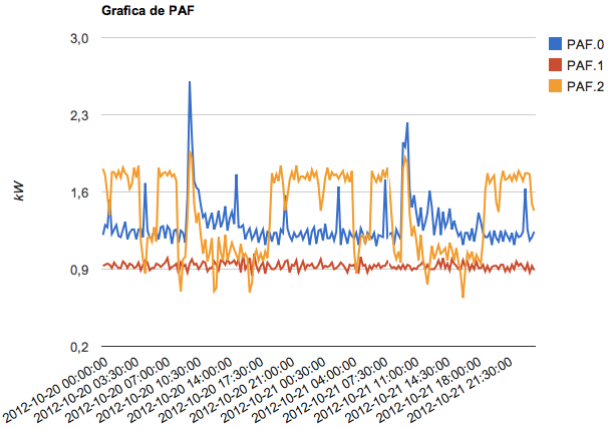
que cada període ens dona un rang de temps T_i necessitarem una

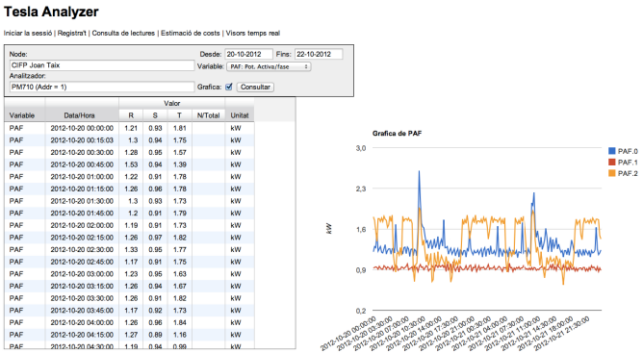
$$\tilde{C}_i = P_i C_i$$

operació que ens determini la diferència de temps per saber el nombre d'hores que compren un període que el podem definir com $\tilde{T}(H_i) = t_2 \ominus t_1$ per poder calcular la potència consumida P_i en un període T_i en un període donat com:

p_i és la suma de les potències de la tarifa T_i, aleshores la potència total del el seu cost serà en la tarifa T_i serà de, sabent que C_i és el cost en euros de la tarifa T_i:
El sistema aleshores permet donat un node i un analitzador, poder fer el càlcul d'estimació de costs per tarifa per dia, donant així el total en kW i en euros, com es pot veure a la **¡Error! No se encuentra el origen de la referencia.**.

A part també es pot fer un extracte de lectures per node i analitzador donat: data inici, data final i la variable que es vol consultar el seu històric i podem generar gràfiques damunt aquest històric:





A més es pot fer consultes en temps real (Figura 11 Visor de lectures instantànies en "temps real"), com havíem dit abans, d'un analitzador

des de el propi frontal web emprant internament el webservice el qual es comunica directament amb el node seleccionat, emprant el API de

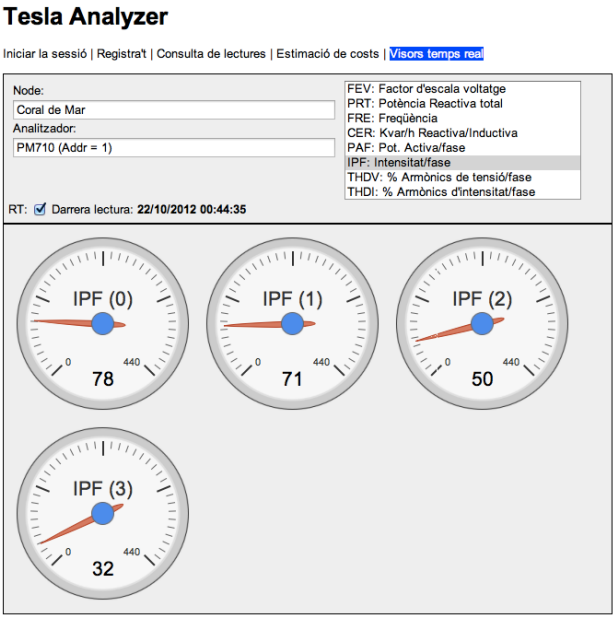


Figura 10 Visor de lectures instantànies en "temps real"

comunicacions que també és emprant des de l'aplicació mòbil per iPhone, anomenada iAnalitzador.

6. CONCLUSIONS

Podem concloure que, gràcies que de cada vegada hi ha protocols oberts i que es poden emprar lliurament i a la obertura de les especificacions en protocols de comunicacions emprats en els analitzadors d'energia fa que tercers pugin desenvolupar solucions obertes sobre adquisició, anàlisi de dades sobre el consum energètic. Ja que avui en dia es tan necessari tenir una consciència ben clara sobre l'estalvi energètic fa que sorgeixin solucions d'aquest tipus de cada vegada més, d'aquestes i d'altres que s'integrin amb les dades de de productivitat de l'empresa, com pugin esser de les estàncies ocupades d'una cadena hotelera i contrastar les amb el seu consum energètic i l'estimació de costs tant de consum energètic com el guanyat en base de les estàncies ocupades.

REFERÈNCIES

MARTELLI, A. 2003. Python in a nutshell. *O'Reilly*, 245-252.

RICHARDSON, L. RUBY, S. 2007. RESTful Web Services. *O'Reilly,* 23-47, 54.

LUTZ, M. 2006. Programming Python (Third Edition). *O'Reilly,* 175, 709-720.

BRESHEARS, C. 2009. The Art of Concurrency. *O'Reilly*.

HALSALL, F. 1989. Data Communications, Computer Networks and OSI (2nd Edition). *Addison-Wesley*. 98-104

CIRCUTOR. 1995. Supply Network Analyzer CVMk Series Instruction Manual. *Circutor S.A*. 12-13.

SCHNEIDER ELECTRIC. 2004. Power Meter PM700, Central de Medida. *Schneider Electric*. 113-120.

MODBUS.ORG. 2006. ModBUS over serial line specification and implementation guide V1.02. *ModBUS Organization*.

MODBUS.ORG. 2006. ModBUS Application protocol specification v1.1b. *ModBUS Organization*.

http://docs.python.org/library/threading.html