TREBALL FINAL DE GRAU EN ENGINYERIA INFORMÀTICA

DISSENY D'UN SISTEMA DE CONTROL PER A L'ENTORN DE LA INDUSTRIA 4.0

ALUMNE: VICENT ALBEROLA CANET
TUTOR: MANUEL LLORCA ALCÓN
CURS 2016/17

Índex

Introducció

En aquest treball de fi de grau es tracta el paradigma de la Indústria 4.0. S'explica aquest concepte que molts ho denominen com la quarta revolució industrial, el qual pot arribar a canviar la forma de treballar de les industries i inclòs generar nous models de negoci.

Aquest concepte fou creat al 2011 però encara no està estés. El principal problema ha sigut el cost acompanyat del desconeixement, ja que qualsevol canvi d'aquestes magnituds sense saber cert si serà beneficiós o no per a l'empresa és un pas molt arriscat des del punt de vista de l'empresari. Aquestos canvis tenen repercussió a mitjà o llarg termini i mai a curt termini, mentre que la implantació pot durar anys fins que es completa a tots els nivells.

El desconeixement anirà desapareguent quan més empreses adapten el paradigma de Indústria 4.0. Per primera els sistemes aniran consolidant-se i els resultats de les empreses que ja han passat a 4.0 serà el que motive a la resta a adoptar aquest paradigma.

Per altra banda, les noves empreses deurien instaurar des d'un primer moment la Indústria 4.0.

Es mostra un cas real de funcionament d'una fàbrica 4.0 per a poder entendre-ho millor, poder visualitzar les millories que aporta aquest paradigma a la indústria i comprendre la complexitat que pot arribar a tindre aquest sistema.

Finalment s'adapta a un petit negoci per a demostrar tant la escalabilitat, adaptabilitat i beneficis del paradigma (o part d'aquest) com les eines i components necessaris par a poder implementar un sistema d'aquestes característiques.

Què és la Indústria 4.0

La Indústria 4.0 fa referència a una fabricació avançada i el seu objectiu és el posar en marxa les 'fàbriques intel·ligents" o dit d'una altra manera: fer la quarta revolució industrial.

Aquest concepte fou creat per el govern alemany al 2011 per a començar a digitalitzar les seves fàbriques. Aquesta digitalització es basa en la implantació del Big Data, realitat augmentada, internet de les coses i la cultura DIY.

Amb aquesta digitalització les fàbriques arriben a ser més autònomes, més eficients tant a la producció com en temes d'estalvi energètic i es capaciten per a produir productes personalitzats sense deixar la producció en massa.

Tenint la demanda del client, la fàbrica pot ajustar les produccions per a tenir el mínim estoc de producte acabat, demanar automàticament al proveïdor la matèria prima necessària. També es pot fer un control de les màquines que automàticament avisa quan es necessita fer un manteniment preventiu o planificat, desconnectar maquinari que no es necessiti, controlar el consum de llums, millorar la traçabilitat del producte final,...

Algunes de les tecnologies per a dur a terme aquestes característiques ja fa anys que existeixen. El que passa és que a dia de hui és més fàcil i més econòmic el poder implementar-les.

Els pilars de la Industria 4.0

www.itainnova.es

Big Data i anàlisis

Element clau per a guardar tota la informació generada i poder actuar segons la interpretació de les dades. Permetent optimitzar la producció, estalviar energia i prendre decisions en temps real.

Robots autònoms

Els robots ja s'utilitzen a la industria però ja van apareguent nous models que són autònoms i a més a més es poden relacionar amb altres robots o humans.

A la llarga aquestos robots seran més barats i més eficients, fins i tot podent aprendre per a perfeccionar la seva funció

Simulació

Les simulacions (que ja solen aplicar-se en departaments com el d'enginyeria) també es realitzen ja a altres departaments com producció. Permetent preveure les necessitats, optimitzar configuracions i preparar amb temps, si és necessari, aquestes configuracions o canvis per a no influir negativament a la producció.

Sistemes d'integració horitzontal i vertical

Es pretén comunicar tots els departaments de l'empresa i compartir tota la informació per a millorar l'empresa

Internet Industrial de les coses (I2OT)

És el internet de les coses aplicat a la industria.

Ciberseguretat

Amb la Industria 4.0 també arriba l'accés de les dades des de qualsevol lloc, arriba la connectivitat total de les dades de l'empresa. Per tant es obligatori aplicar mesures de seguretat per a que no puga accedir qualsevol intrús a les dades de l'empresa.

Computació al núvol

El núvol és el futur, van apareguent més serveis al núvol i millorant l'accés a aquest (més ràpid, més sensació d'aplicació local,...). Fins i tot les dades de control i producció seran enviades directament al núvol.

Fabricació additiva

La personalització de productes per a clients a xicoteta escala, amb ajuda de la fabricació d'elements amb impressores 3D ajuden a reduir temps de transport i el magatzem, mentre que obrin nou mercat sense perdre rendiment de producció

Realitat augmentada

La realitat augmentada ajudarà a reparacions de maquinari, selecció de peces a un magatzem, us de maquinari,... aportant informació addicional important que permetrà reduir notablement el temps de procés d'aquestes accions.

Cas pràctic: la fàbrica 4.0

Fàbrica embotelladora de refrescs de marca blanca, la qual té diversos clients i cada client té les seves exigències en quan a producte, presentació i format d'aquest, empaquetament, etiquetat del empaquetament, distribució,...

Aquesta fàbrica 4.0 té parametritzats totes les exigències dels clients, paràmetres que ha definit l'esser humà al sistema: dissenys del producte, composició de la beguda, previsió de venta al client, matèria prima necessària per a la producció del producte final,...

Amb la previsió de venta als clients la fàbrica 4.0 es planifica automàticament la producció de cada producte per a tenir-lo el mínim temps possible ocupant estoc a la fàbrica, realitza les comandes necessàries als proveïdors de la matèria prima al moment en que serà necessari i contracta el transport.

Al magatzem automatitzat també està tot digitalitzat. Gràcies a l'etiquetat (tant de la matèria prima com del producte acabat) es sap la ubicació de cada material i els carretons autònoms s'encarreguen de gestionar el magatzem introduint la matèria prima dels proveïdors, emmagatzemant el producte produït a les línies de producció i extraient el producte acabat per a transportar al client.

És molt important el manteniment de les màquines que s'ocupen de tot aquest procés. Qualsevol màquina necessita d'un manteniment planificat però també és necessari un manteniment preventiu per a evitar parades innecessàries per averia i per no tenir material de reemplaçament.

Aquestes màquines són monitoritzades controlant la temperatura, consum elèctric, hores en marxa, parades en producció, recompte de botelles, consum real de la matèria prima, i altres dades més específiques de cada màquina. Per exemple als carretons autònoms es detecta quan es necessari reemplaçar la bateria, així el carretó 'torna a casa' on es canvia la bateria per una altra i pot continuar la feina.

Altre exemple seria la monitorització de temperatura i consum d'un variador, quan el sistema detecta que té un consum elèctric una temperatura superior al rang que s'ha definit com 'normal' el sistema avisa a manteniment per a revisar aquesta peça i si és necessari es reemplaça.

I un altre exemple de manteniment planificat és el comptabilitzar les hores que ha hagut consum per part d'una màquina i quan arriba al màxim d'hores definit abans d'una revisió d'aquesta, el sistema avisa (a un proveïdor o al propi departament de manteniment) per a que es revise l'estat d'aquesta.

Finalment també està l'estalvi d'energia. Amb la monitorització del consum de la fàbrica 4.0 el sistema optimitza l'ús de la energia, habilitant o des habilitant les zones o maquinari segons el seu us, controlant la calefacció,...

Per suposat, tota la informació que el sistema té es transmesa de forma resumida (taula de dades, gràfics, informes,..) als directius de l'empresa, que s'encarreguen de controlar la fàbrica intel·ligent, revisar despeses, beneficis i els ajuda a invertir en millores.

El que més benefici dona a la Indústria 4.0 és el tenir tots els sistemes interconnectats, des dels quals es pot tenir una visió real del funcionament de tota la fàbrica. Aquesta visió real s'aconsegueix relacionant el ERP amb el Big data on es conté tota la informació recopilada per el I2OT (Indústria Internet Of Things).

Implantació de sistemes

Per a poder tenir una Industria 4.0 s'ha anomenat varis sistemes que estan interconnectats:

- ERP (SAP, Navision,...).
- Sistema d'etiquetat.
- Sistema de control energètic (SGE,...).
- Sistema de control industrial i bigdata (Wonderware,...).
- Sistema de planificació automàtic (JDA,...).
- Sistema d'extracció d'informació 'resumida' (Board,...).
- ...

Les industries solen tindre un ERP i un sistema d'etiquetat (pot ser comunicats per interfases o no) però els demés sistemes solen no estar implementats. Aleshores la implementació necessària de tots els sistemes suposa un cost elevat tant econòmic com de recursos. Es pot anar implementant sistemes per seccions per a no invertir tants recursos en un període curt de temps, la qual cosa fa que augmente el temps necessari per a convertir-se en una fàbrica 4.0

En aquest cas pràctic es va tindre que remodelar per complet els sistemes, el ERP i sistema de etiquetat ja eren antics, la fàbrica havia crescut massa per a aquestos sistemes i s'estava constantment realitzant modificacions per a adaptar el sistema a les necessitats.

Aquestes modificacions començaren a l'any 2009. Al 2016:

- El nou sistema ERP es pot dir que és estable.
- Ja s'ha passat per dues versions al sistema d'etiquetat.
- El sistema de planificació encara no està completament automatitzat, si que genera planificació però l'usuari encarregat del sistema és el que realment la fa.
- Encara queda molt per implantar al camp de monitorització de màquines.
- Encara queden proveïdors als quals no se'ls pot fer comandes automàticament.
- Al magatzem encara no s'han implantat els carretons autònoms.

Com es pot observar, la fàbrica no és 4.0 al 100%. En aquest cas s'ha optat per a allargar en temps la implementació de Industria 4.0 i poder invertir més en altres millores amb resultats a curt termini o que estan més lligats amb la producció.

Sistema de Control

Com ja s'ha comentat anteriorment, un sistema de control és un dels pilars fonamentals a la Industria 4.0. És important controlar el consum energètic, el consum de matèria prima, el consum del temps (temps invertit per a produir), també és important controlar el funcionament de les màquines, el producte acabat,...

Aquest control, a diferència del control de compres/vendes, despeses/guanys que es pot fer a nivell del ERP, es fa amb les dades que es recopilen en temps real per part dels sensors. Amb aquestes dades ja es pot prendre decisions i actuar de la forma més adequada a cada moment.

Si ens remuntem en la historia, al sistema de control sempre ha estat present l'esser humà i ha pres decisions pel que ha aprés durant l'us d'aquest sistema de control i amb les dades que obté en temps real.

Un exemple prehistòric (i que actualment encara està en marxa) és el control del foc. Des de la prehistòria ha sigut molt importat el foc i tindre un control sobre aquest per a mantenir-lo 'viu' i controlat ha sigut una necessitat.

Altre sistema de control històric és el que les grans ciutats han creat per a obtindre aigua, des de les grans ciutats d'Egipte amb els seus dipòsits per a emmagatzemar la poca aigua que hi havia i els corresponents conductes per a poder utilitzar-la intentant minimitzar les pèrdues, passant pel sistema de control que hi havia a l'antiga Roma fins als sistemes de control actuals de les grans ciutats.

Un sistema de control més actual és el que tenen les actuals empreses elèctriques per a suportar les necessitats i no sobre carregar la xarxa elèctrica, evitant així averies per excés o

talls de llum per deficiència. En tot moment estan mesurant l'us, l'estat de la xarxa, la producció, la demanda,... i actuen segons les dades que obtenen.

Altre exemple més actual és un sistema de control de siroperia per a la producció de refrescs. Des d'un SCADA es controla tot el sistema de dipòsits, conductes, mixers (màquines encarregades de fer les mescles de líquids). Aquestos sistemes solen estar compostos per un (o varis) PLC, panels tàctils HMI a les màquines per a poder interactuar 'in situ' i un SCADA des d'on l'operari té una visió general del sistema.

Al sistema ja estan configurades les receptes dels refrescs, indicant quin refresc i quina quantitat es vol produir el sistema indica quanta matèria prima es necessita, l'operari introdueix la matèria prima als contenidors corresponents i des de l'escada controla tot el procés. Aquestos sistemes estan preparats per a poder gestionar varies produccions a l'hora, depenent dels dipòsits dels que es disposen.

En aquestos sistemes ja van apareguent el registre de dades però d'una forma molt precària, amb base de dades que acaben ocupant molt d'espai pel volum de dades i amb ferramentes amb les que costa interpretar la informació.

Com és un sistema de control autònom, aquest no comptabilitza les despeses de matèria prima o els litres produïts al sistema, són accions que es fan manualment entre altres. Ni tampoc es registra el consum energètic, temperatura de les màquines, cabdal de líquid en diferents punts.... que verifique que el funcionament del sistema és correcte.

Als sistemes de control de la Industria 4.0, a més de les característiques que tenen els sistemes de control anteriors, apareix noves característiques: la 'historiarització' de totes les dades, representació gràfica d'aquestes amb altres dades de producció, deixa de ser un sistema de control aïllat per a connectar-se als demés sistemes de l'empresa aconseguint ser més autònoma aquesta, es fa un seguiment del funcionament del sistema per a previndre averies i reduir consums,...

Per a poder fer aquesta 'historiarització' de totes les dades i que siguen accessibles es formen varis nivells de registre de dades:

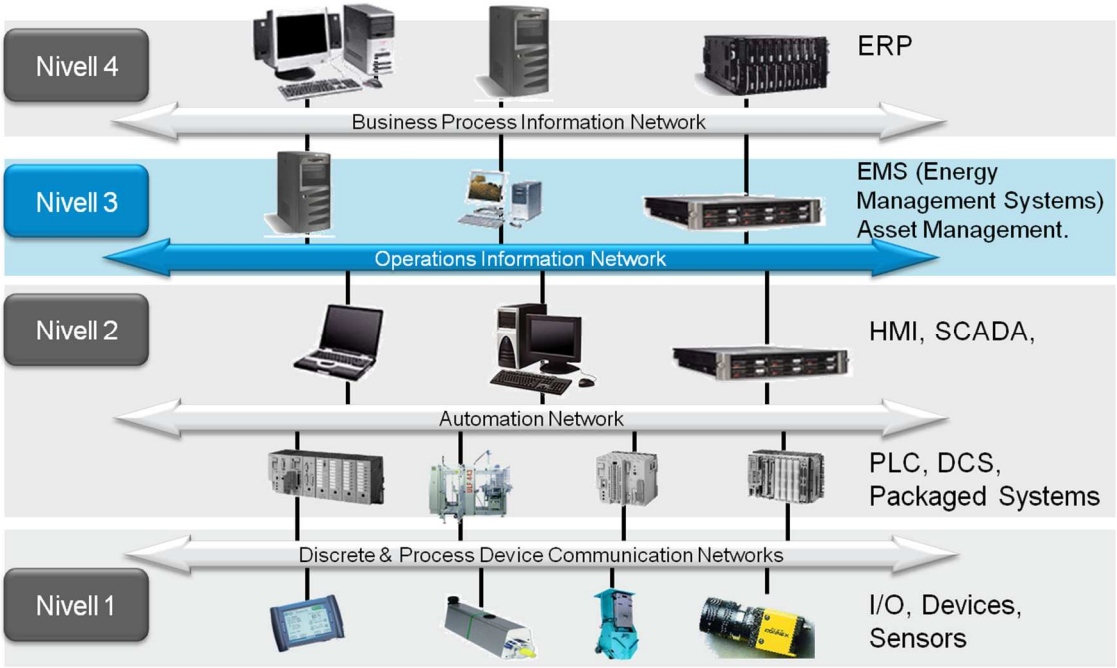
- En primer nivell està el registre de dades en temps real: Aquest servidor s'encarrega d'emmagatzemar totes les dades que es recullen amb el temps acordat (1 segon, 5 segons, 10 segons....). Amb aquestos intervals es necessitaria discs durs amb molta capacitat per a emmagatzemar totes les dades que es generen. A més a més tanta informació sols es necessita per a monitoritzar en temps real i no per a històrics (excepte casos puntuals). Per tant es sol definir una durabilitat de les dades (1 mes, per exemple), així esté una base de dades que es recicla i té un llímit definit. Una dada emmagatzemada cada 5 segons significa que en 24h es tenen 17.280 registres, i en un mes 518.400.
- Els següents nivells es definixen segons les necessitats, el que fan és una 'foto' de les dades del nivell anterior amb un interval més ampli. Per exemple un segon nivell seria un servidor que cada 24h es copie del primer nivell totes les dades del dia amb intervals de 10 minuts. Tinguent com a referència els 518.400 registres mensuals del nivell anterior, es podrien emmagatzemar aquestes dades durant 10 anys (utilitzant el mateix espai). Es podria fer un altre nivell que fera una foto del segon amb dades cada hora o cada dia,.... i així segons les necessitats.

Aquesta fragmentació de dades es fa seguint la relació d'espai de disc que es pot utilitzar, velocitat d'accés a les dades i temps que es voles guardar les dades. Amb una gran influència del cost que s'està disposat a assumir per a aquest sistema per part de l'empresa.

Sistema de control a la Industria 4.0

El sistema de control actual és la evolució dels sistemes PLC-HMI. Fins ara un sistema de control estava format per una part que interactuava amb els elements: el PLC, i una part que interactuava amb l'esser humà: HMI (human machine interface), el qual solia ser un SCADA.

Al SCADA està representat d'una forma lògica el sistema que controla amb dades en temps real i des del qual pot realitzar les accions necessàries per a que funcione correctament el sistema. Tant les dades que rep com les accions que s'envien son tractades pel PLC i traduïdes a impulsos elèctrics digitals o analògics. Aquest control el realitza un operari de producció, és sols un control de producció.



El sistema de control actual va més enllà, a part del control de producció del sistema també està el control de l'estat del sistema. Aquestos dos controls poden estar completament lligats i ser administrats per un mateix operari o poden estar separats administrativament (però relacionats) per la complexitat del sistema o perquè el control de l'estat es realitza des d'un punt de vista que engloba tota la línia de producció o, fins i tot, tot el maquinari de l'empresa.

La evolució del control de producció arriba amb la connexió amb la resta de sistemes: control d'estoc, optimització del temps per a planificació, centralització de dades (es necessiten menys recursos per al control de producció), i major grau d'automatització.

Per altra banda apareix el control de l'estat del sistema. Aquest és un punt important perquè ajuda a previndre incidents de forma automatitzada i més fiable, deixant a banda anteriors mètodes basats en rutes: revisions planificades pel sistema però que es feien manualment, amb veredictes basats en la observació humana i amb un temps entre revisions normalment massa llarg.

Ara el control de l'estat també és en temps real i es recullen tota classe de dades: consum elèctric, relació entre matèria prima consumida i producte final produït, temperatura i altres

dades específiques de cada sistema: al cas de la siroperia està el cabdal en cada tram de les canonades per a detectar fugues o embossaments, al cas de variadors es podria mesurar la potència en cada instant, en cas de transportadors la velocitat,...

El control de l'estat pot ser sols monitor o interactuar amb el sistema per a poder implementar accions correctives en temps real. Però tot depèn si el sistema a controlar te sistemes de recolzament ja instal·lades o simplement per seguretat sols es requereix monitoritzar (dues interaccions completament diferents que apliquen a un mateix sistema pot arribar a avariar completament el sistema).

Mitjançant l'històric de dades del sistema de control el sistema es capaç de dependre el funcionament i així localitzar un mal funcionament del sistema, informant sobre una futura avaria, fuga, baix rendiment d'algun dels components del sistema.

Amb aquestes dades històriques també es pot fer una previsió d'avaries, parades de producció, reposats utilitzats (i el seu cost), consum del sistema... que ajuda a l'empresa a fer previsió de gastos, càlcul del cost per unitat produïda, càlcul de temps estimat de funcionament del sistema per a poder optimitzar la planificació,...

Tenint la monitorització en temps real i la comunicació amb la resta de sistemes, baix una avaria que no s'haja pogut detectar i que no es pot solucionar (per exemple el sistema té controls de l'estoc de material i pot saber si el component del sistema es pot reparar o no) la industria 4.0 pot prendre una decisió automàtica per a canviar la planificació i fabricar altres productes que no necessiten del sistema avariat per a ser produïts.

Aquest sistema de control fa que l'esser humà pren menys decisions i per tant la cadena de producció fallarà menys per errors humans. Al cas de la siroperia l'ordre de producció es carrega automàticament, està més controlat els nivells dels dipòsits,...

I2OT (Indústria Internet Of Things)

El IOT és la connexió de qualsevol objecte a la xarxa, be per a saber la seva ubicació, el seu propietari, recopilar informació al voltant de l'objecte,... i poder accedir a tota aquesta informació des de qualsevol lloc.

Per exemple es pot establir una comunicació entre el despertador, la cafetera i les llums de casa per a que quan sone el despertador, aquest comunique a la cafetera que comence a fer cafè i encendre la llum del bany, i quan s'ix del bany que s'encenga la llum de la cuina.

Doncs be, el I2OT és aplicar aquest concepte a la Indústria per a millorar el funcionament de la cadena de producció.

En quan a connectivitat el concepte varia una mica. El IOT a un entorn no industrial es pot comunicar sense problemes amb tecnologies inalàmbriques però a un entorn industrial no és el millor mètode de comunicació.

La comunicació via ethernet és interessant inclòs es pot dir que és la millor forma per a comunicar-se però també té un sobre cost respecte a altres tecnologies.

Encara que cada empresa és diferent, un bon model de connectivitat pot ser el següent:

- Connectar varis sensors i actuadors (esclaus) a un concentrador (màster) mitjançant Modbus/Profibus,...
- El concentrador s'ocupa de processar aquesta informació i és l'encarregat de connectar-se al món per a enviar/rebre dades.
- Aquestes dades son emmagatzemades/tractades a un servidor

L'equilibri entre esclaus i màsters és important, el tenir un concentrador amb molt esclaus és perillós però el tenir un concentrador per esclau suposa un sobre cost.

La comunicació entre els concentradors i el servidor es pot establir de vàries formes, per exemple utilitzant webservices (REST, SOAP). Encara que últimament el protocol MQTT està agafant força.

Open-source, Open-hardware i Arduino

Si es combina el IOT i DIY s'obté els conceptes Open-source i Open-hardware. Una plataforma que engloba aquestos conceptes és la plataforma Arduino (o *duino). Sense grans coneixements d'electrònica i de programació es pot introduir al món del IOT en uns pocs passos, i això ja és posar un peu a la Indústria 4.0

Arduino[1] es basa en una sèrie de targetes amb microcontrolador de la família Atmel, les quals són compatibles entre si (cadascuna té les seves característiques) i es programen en el llenguatge C.

Les característiques genèriques d'aquestes plaques són:

- Entrades/eixides digitals.
- Entrades/eixides analògiques.
- Ports de comunicació: Serial, SPI, I2C.
- Segons plaques, alimentació a 3v o 5v. Pareix ser que en un futur s'estandarditzarà l'alimentació a 3v per la tendència d'anar reduint la grandària dels circuits electrònics.

Aquestes plaques es poden connectar a la xarxa via ethernet, wifi, bluetooth, USB,... per a enviar la informació dels sensors connectats o per a rebre ordres i accionar els actuadors.

El tenir connectivitat wifi, un consum elèctric de 3v i ser un dispositiu de xicotetes dimensions el fa idoni per a ser utilitzat en el IOT.

Com es disposen dels esquemes elèctrics de les plaques Arduino, es poden adaptar a les nostres necessitats i crear la nostra placa sense tindre nocions de soldar, enviant els dissenys a empreses les quals s'encarregaran de realitzar aquesta funció.

Al mercat hi ha tota mena de sensors i actuadors compatibles amb aquesta plataforma, que es comuniquen pels ports mencionats anteriorment.

Amb un bon repositori de llibreries i esquemes elèctrics (Internet) es pot donar la opció a qualsevol persona de poder adaptar ella mateixa el seu propi sistema de IOT.

[1] A l'anex I hi ha més informació relacionada amb Arduino

Indústria 4.0 a la petita empresa

Aquest model també es pot extrapolar a un altre nivell: la petita empresa (comerços, empreses de serveis,...) encara que no s'apliquen totes les característiques de la Indústria 4.0 sí poden passar a ser 'empreses smart'.

A una petita empresa no es té la mateixa infraestructura que a una gran empresa però es pot adaptar igualment el model Indústria 4.0

Si be el tema de l'estoc de matèria prima està controlat ja per un sistema informàtic en la majoria de les empreses, aquest es pot millorar: enviant automàticament les comandes de material, realitzant un històric de reposició que pot ajudar, per exemple, a identificar les èpoques de més consum i poder controlar les despeses,...

Depenent del tipus d'empresa també es pot millorar en altres aspectes com el control d'estoc de producte venut, utilització de realitat virtual per a prototipats, automatització/optimització de l'agenda segons la demanda (equivalent a la planificació de la Indústria 4.0),...

Però en general on més es pot millorar a la petita empresa és a l'àmbit energètic ja que no sol haver una monitorització del consum energètic, a part de la implantació d'un sistema domòtic.

Sense oblidar que el més important per a tenir una visió real del funcionament del negoci és interconnectant tota la informació recopilada (Big data).

Tota millora té un cost econòmic que no sempre s'està preparat per a afrontar per part de les empreses, i més si no està relacionat directament (i a curt termini) en l'augment de guanys i o productivitat. El paradigma d'estalvi i optimització de recursos és igual a menys despeses i per tant més benefici encara no està prou arrelat a la figura de l'empresari i menys si el benefici s'obté a mitjà o llarg termini.

Hui en dia hi ha varis sistemes tancats amb preus abusius per a poder realitzar aquestes funcions. Inclòs s'han dissenyat nous protocols de comunicació, com pot ser el KNX, entre els grans fabricants per a tenir més compatibilitat entre ells mentre ja hi ha altres creats anteriorment que compleixen aquesta funció en sistemes industrials i es poden aplicar també a aquest àmbit.

Però uns dels principis en els que es basa la Indústria 4.0 és IOT i DIY, el que significa que un sistema deu poder ser heterogeni, escalable i adaptable a les necessitats de l'empresa amb un cost raonable. També deu ser un sistema que la pròpia empresa puga gestionar i no haja de dependre d'empreses alienes.

Hi ha diferents sistemes per a implantar els sistemes de control:

- Sols es vol monitoritzar: Aquesta opció és la més econòmica per a adaptar l'empresa a 4.0 i sempre es pot escalar les funcionalitats. Hi ha diferents nivells de profunditat (monitoritzar cada aparell elèctric o agrupar-los), i la modificació de la estructura (cablejat elèctric,...) és mínima.
- Monitoritzar i interactuar: Amb aquesta opció s'obté el mateix que la primera però a més a més es pot interactuar amb els aparells, podent-los posar en marxa o apagar-los remotament. Aquesta opció és menys econòmica que l'anterior i es necessita modificar la estructura que hi ha, i a canvi ens dona un major control de l'empresa
- Nova instal·lació: Les noves instal·lacions són les més senzilles d'adaptar ja que el cost econòmic de fer una instal·lació antiga i una 4.0 és casi la mateixa. Amb l'avantatge de

poder centralitzar totes les connexions a un quadre (o varis, depenent de la grandària de l'empresa). En aquest cas sempre seran per a monitoritzar i interactuar amb el sistema.

Entre les dos primeres opcions hi ha múltiples combinacions i nivells de profunditat per a adaptar-se a una empresa.

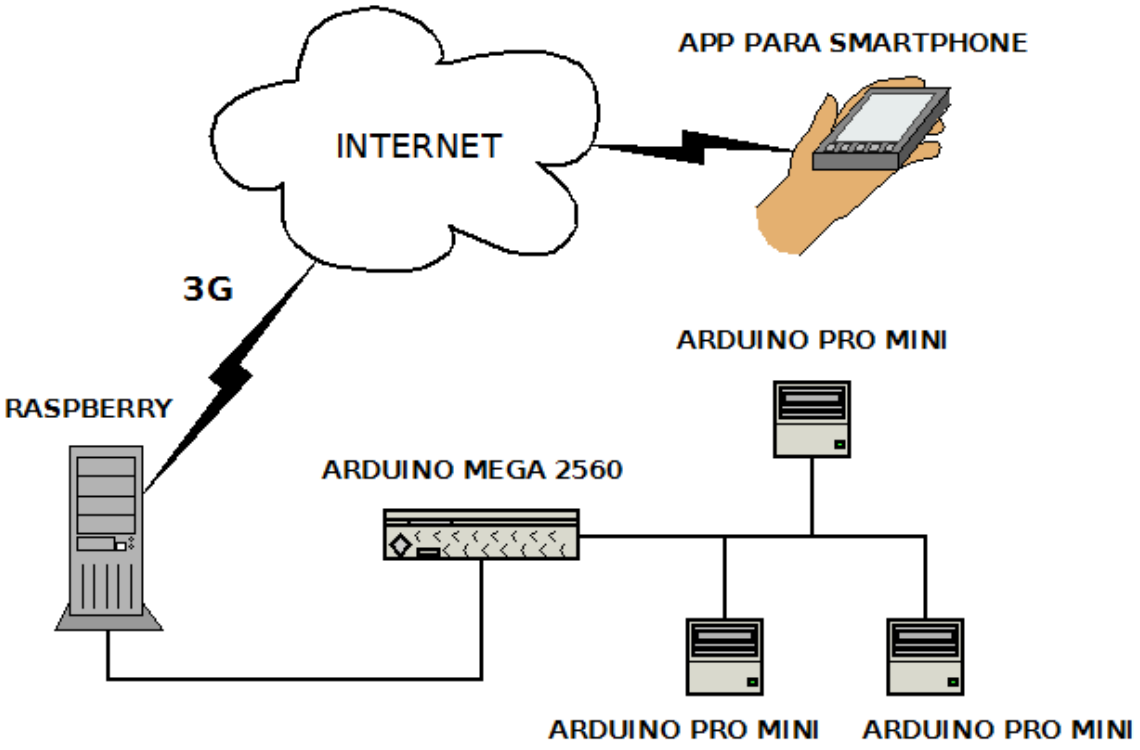
Amb la tercera opció es vol centralitzar i facilitar la configuració de les connexions. El cablejat de tots els actuadors i sensors està centralitzat a un quadre elèctric, on cada cable està identificat i connectat a la entrada/eixida corresponent del PLC. Podent configurar les accions a realitzar i minimitzant els elements a instal·lar. Per exemple no es necessita instal·lar (i cablejar) commutadors per a poder encendre les mateixes llums des de punts separats, un altre exemple es que no es necessita de sensors per a poder identificar quins llums o quins endolls estan en marxa ja que es pot comprovar a nivell intern del PLC.

Cas pràctic

En aquest treball s'adapta el concepte de Indústria 4.0 a una clínica de fisioteràpia. Els objectius són:

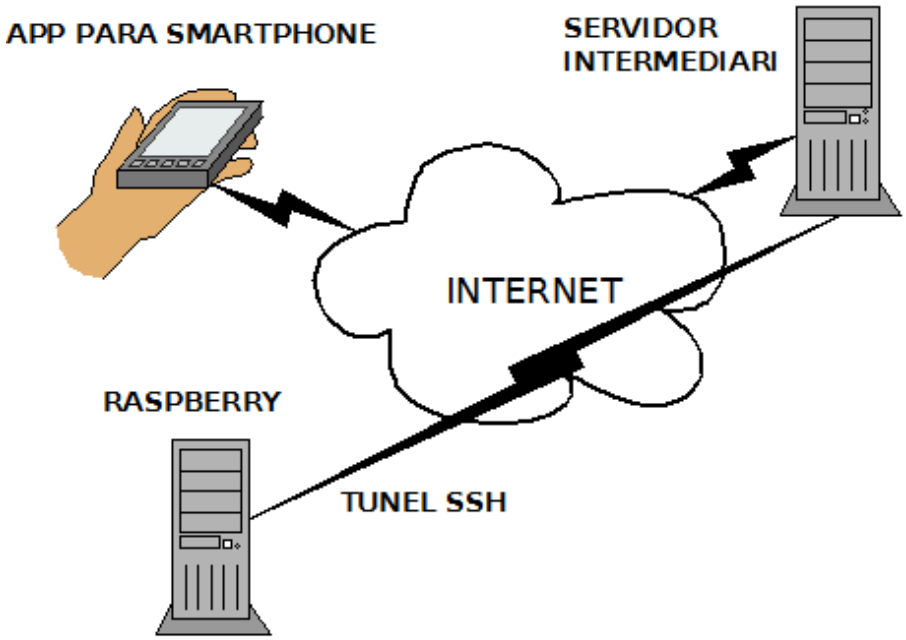
- Estalvi energètic.
- Mantenir un clima ambient idoni per al treball.
- Control de presència.
- Automatitzar l'agenda de cites. En aquest punt la empresària no vol un sistema automàtic per a que el client puga agafar cita sinó un sistema de recordatori mitjançant el whatsapp cara el client.
- Control automàtic del reg.

Aplicant el model de connectivitat comentat al I2OT l'esquema inicial era el següent:



- 3 equips esclaus, que són els encarregats de comunicar amb els sensors/actuadors.
    - Esclau 1:
        - Sensors finestres del pati interior.
        - Sensors portes d'aquesta zona.
        - Sensor temperatura interior.
        - Sensors de consum elèctric de llums, aire condicionat i radiador.
        - Relés encarregats d'obrir/tancar finestres del pati interior
    - Esclau 2:
        - Sensors finestres de l'exterior.
        - Sensors portes d'aquesta zona.
        - Control d'accés.
    - Esclau 3:
        - Control Reg.
        - Sensor llum exterior.
        - Sensor temperatura exterior.
- 1 equip principal (màster), que s'ocupa de comandar els equips d'esclaus.
- 1 servidor on S'emmagatzema els històrics i comunica amb l'exterior (mitjançant 3g).
- App per a poder comandar el sistema des del mòbil.

En la versió 2 s'ha modificat l'apartat de comunicació a l'exterior ja que la companyia telefònica a la que pertany la targeta SIM per a la connectivitat 3g assigna IP's privades i no poden ser accedides des del exterior. S'ha afegit un element més: un servidor intermediari al qual s'ha configurat un port que redirigeix la comunicació via túnel SSH al sistema.

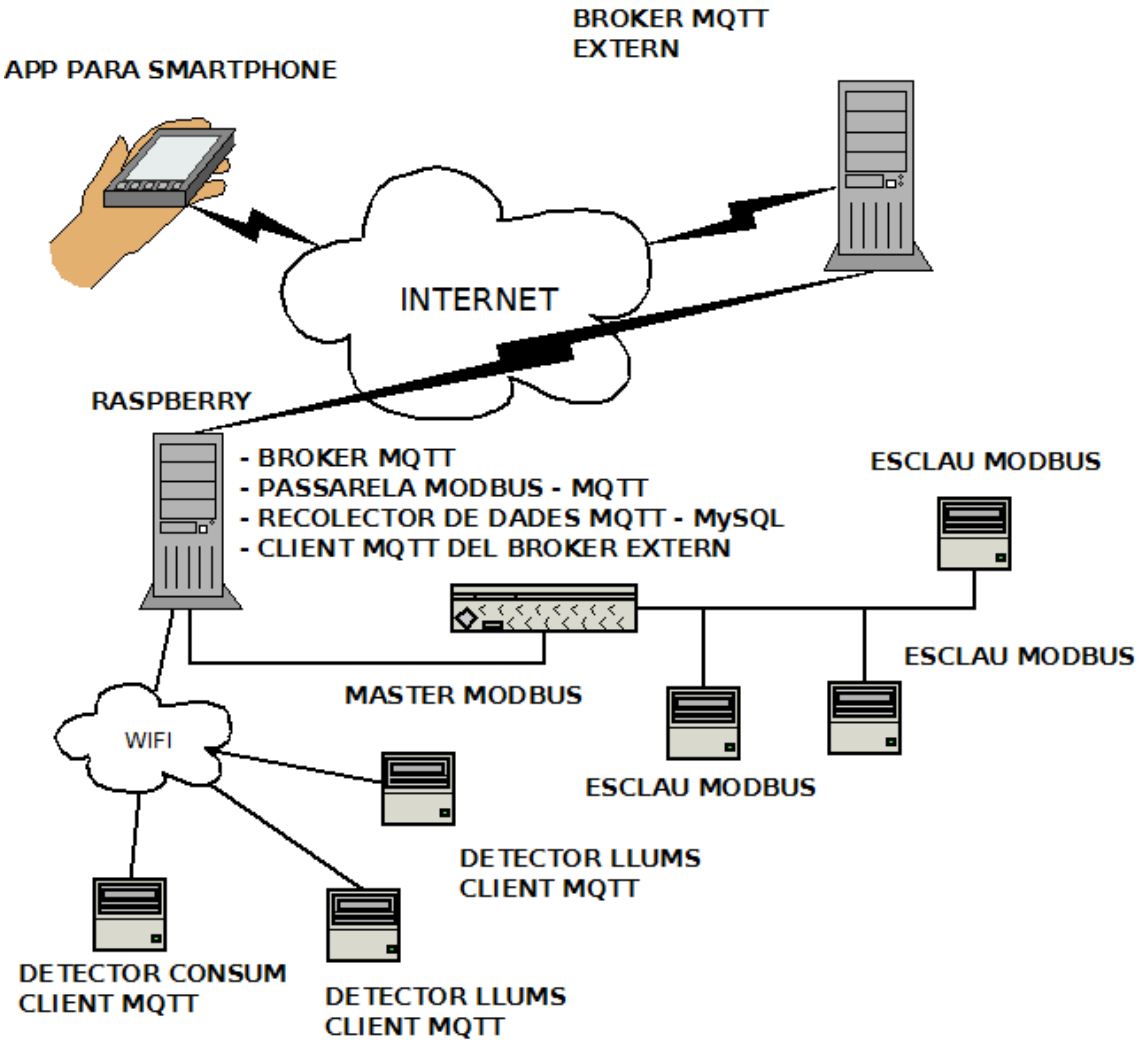


Després de vàries setmanes de funcionament es detecten problemes en la comunicació entre el sistema i l'aplicació exterior. El túnel SSH necessita estar sempre obert i pel pas del temps aquesta connexió es perd o es queda blocat. Es programa un *daemon* encarregat de monitoritzar la connexió i reiniciar-la en cas que es perda, inclòs es configura al *crontab* que es reinicie la connexió cada hora. S'aconseguix que funcione millor però no és una comunicació fiable.

Es decidix buscar un altre mètode de comunicació, en compte d'accedir des de l'exterior (client) al sistema (servidor), configurar el servidor intermediari (que fins ara feia de passarel·la) com a servidor extern del sistema siguent el propi sistema client d'aquest, igual que l'aplicació.

Amb aquesta nova estructura la comunicació entre les parts es fa més senzilla i també fa que el sistema siga més adaptable, si no es requerix de l'accés extern es pot obviar el servidor extern i deixar la connexió directa al sistema domòtic.

Per a dur a terme aquesta nova estructura de comunicació en la versió final s'ha implementat el protocol MQTT. També s'ha implementat el protocol MQTT internament deixant aquesta configuració final:



- Broker MQTT i client MQTT per a comunicar cap al servidor extern, passarel·la per a comunicar amb el equip màster i registrador de dades.
- 1 equip principal (màster), que s'ocupa de comandar els equips d'esclaus
- 3 equips esclaus, que són els encarregats de comunicar amb els sensors/actuadors.
    - Esclau 1:
        - Sensors finestres del pati interior.
        - Sensors portes d'aquesta zona.
        - Sensor temperatura interior.
        - Sensors de consum elèctric d'aire condicionat i radiador.

        - Relés encarregats d'obrir/tancar finestres del pati interior
    - Esclau 2:
        - Sensors finestres de l'exterior.
        - Sensors portes d'aquesta zona.
        - Control d'accés.
    - Esclau 3:
        - Control Reg.
        - Sensor llum exterior.
        - Sensor temperatura exterior.
- 2 equips clients MQTT per a controlar el estat dels llums
- 1 equip client MQTT amb sensor de consum al quadre elèctric
- App per a poder comandar el sistema des del mòbil.

La implementació del protocol MQTT al sistema ha sigut parcial per a poder demostrar el funcionament d'un sistema heterogeni amb diferents protocols de comunicació. Encara que sols s'utilitze hardware arduino, els protocols de comunicació utilitzats són estàndard i per tant compatible 100% en altre hardware que puga comunicar amb aquestos protocols.

Per a la instal·lació es necessita el següent material:

- 1x Raspberry.
- 1x Arduino Mega 2560.
- 3x Arduino Pro Mini.
- 1x Dongle 3G.
- 1x Perifèric de xarxa per a Arduino, ENC28J60.
- 1x Rellotge RTC.
- 4x Perifèric de comunicació RS485, MAX485.
- 3x Comptadors elèctrics no invasius, SCT013-30.
- 2x Sensor de temperatura, DS18B20.
- 8x switch magnètics (per a detectar portes/finestres obertes).
- 1x lector RFID.
- 1x led RGB per a indicar estats del lector RFID.
- 14 x relé, (actuadors).
- 2x electrovàlvula.
- Diversos components electrònics (resistències, condensadors, connectors,..).
- 3x xip ESP8266.
- 1x xip ADS1115

Parts del sistema

Encara que l'usuari pot tenir l'experiència d'un únic sistema amb el que interactua, aquest està dividit en varis mòduls o seccions com s'ha vist a les figures anteriors.

Finalment el punt d'automatitzar l'agenda no s'arriba a desenvolupar fins a posar en productiu, el motiu és perquè l'empresa vol tenir un contacte més personalitzat amb els clients. Per tant no entre en detall però sí faig una breu explicació de la solució plantejada:

Hi ha un client de whatsapp en python anomenat yowsup. Aquest permet enviar missatges, adjunts... per comandaments. Igual que el client whatsapp dels smartphones, es deu registrar a un número de telèfon per a poder utilitzar-lo (i sols pot haver un client whatsapp per número de telèfon).

Després hi ha una API de google que permet accedir al calendari del compte de gmail.

Juntant aquestes dos aplicacions/utilitats es pot crear un sistema per a enviar recordatoris automàtics via whatsapp als clients citats un dia abans.

Sistema central

El sistema central està dividit en quatre parts:

- Una primera part, desenvolupada en phyton, és un client MQTT que està sempre en marxa.
- Una segon part, desenvolupada en php, s'encarrega d'interpretar els missatges rebuts pel client i executar les ordres corresponents.
- La tercera part s'encarrega de gestionar la comunicació de missatges MQTT.
- I la quarta i última part és la base de dades, on es guarden tots els registres.

En un principi la segona part estava destinada a 'conduïr' el sistema mentre que el client MQTT sols s'ocuparia de registrar dades a la base de dades. Però després de varies proves detecte que el client MQTT de PHP no suporta la càrrega, així que destine la primera part per a realitzar la funció de client MQTT (és més robust) i la segona part la destine per a realitzar la resta de funcions.

La comunicació mqttwarn – disparador d'events es realitza cridant directament l'aplicació i passant-li els paràmetres corresponents.

La comunicació disparador d'events – mqttwarn es realitza via MQTT perquè és el canal estàndar de comunicació del mqttwarn.

Client MQTT

L'aplicació mqttwarn s'ocupa d'interpretar els missatges MQTT i reenviar aquesta informació al disparador d'events. Per una part s'encarrega de comunicar l'usuari amb el sistema i per l'altra part realitza la funció d'unió de tots els elements del sistema domòtic junt amb el disparador d'events: monitors MQTT, base de dades i subsistema arduino comunicat per modbus. L'aplicació encarregada de monitoritzar el funcionament de mqttwarn és SupervisorD.

Disparador d'events

Aquesta aplicació es complementa amb mqttwarn. Reb la informació de mqttwarn i executa l'acció corresponent. És l'encarregat d'enviar les dades a l'usuari, registrar les dades a la base de dades i comunicar amb el subsistema modbus. La instal·lació dels requisits per a que funcione el script es troba a l'annexe IV.

Broker MQTT

És l'eix central de la comunicació del sistema, l'encarregat de comunicar totes les parts mitjançant el seu sistema de publicació/subscripció. El broker utilitzat és el software mosquitto al qual se li ha configurat els paràmetres que es mostren a continuació al fitxer /etc/mosquitto/mosquitto.conf

```
allow_anonymous false
password_file /home/vicent/projectes/pfg/mosquitto/usuaris.conf
acl_file /home/vicent/projectes/pfg/mosquitto/topics.conf
```

Amb aquesta parametrització s'aconseguix un nivell de seguretat usuari/password relacionat en cada tòpic, evitant que es puga accedir al sistema amb altres dispositius o a altres tòpics no definits.

Aquest software també inclou comunicació segura (SSL/TSL) però com els dispositius utilitzat no la soporten no s'ha configurat.

Base de dades

És on guarda el sistema tots els registres que siguen necessaris. Aquesta base de dades consta de 3 taules:

- Últims registres: En aquesta taula sols hi ha un valor per tag, siguent l'últim que s'ha registrar amb aquest tag.
- Registres: En aquesta taula es guarden tots els registres de tots els tags. Qualsevol registre rebut és guardat en aquesta taula. És una taula amb un volum d'informació considerable perquè es registren dades cada pocs segons.
- Registres per hora: Aquesta taula té els registres processats de la taula anterior. Cada hora la base de dades agrupa els registres de cada tag, inserta el sumatori en aquesta taula i els elimina de l'anterior. Ací és on apareixen els consums acumulats per hores, el temps acumulat de dispositius en marxa per trams d'una hora màxim,... És aquesta taula la que s'utilitza per a mostrar les gràfiques a l'usuari.

Depenent de les necessitats del sistema es pot crear una nova taula on s'agrupen les dades en trams més grans.

Subsistema comunicat via Modbus[2]

A la primera versió aquest sistema estava dissenyat per a fer totes les funcions: Interactuar amb l'aplicació de control, recopilar les dades dels sensors i controlar els actuadors.

Els tres Arduinos esclaus estan connectats al Arduino Master mitjançant un bus de 2 fils. Per la ubicació dels esclaus, l'esclau de la zona d'entrada té un cable dedicat per a ell i els altres dos comparteixen cablejat. El Arduino Master és l'encarregat de comunicar-se via ethernet (protocol REST) amb la resta del sistema domòtic i és on està la càrrega computacional, els esclaus són els encarregats de comunicar-se amb els perifèrics (actuadors/sensors).

Màster

El màster és l'element que interactua amb l'exterior del subsistema i l'encarregat de controlar tots els actuadors/sensors associats als seus esclaus, es centralitza tota la càrrega computacional al master. El primer apartat el realitza amb la interfície REST i el segon amb la programació del sistema després d'un anàlisis del funcionament dels equips a controlar.

Finalment també dispon d'una interfície web per a configurar els paràmetres referents a la xarxa, modbus, hora i temporitzadors.

Interfície REST

S'ha elegit aquest protocol perquè en un principi la comunicació del sistema era directa amb una aplicació web i la capacitat del dispositiu és limitada. Aleshores aquest protocol s'adapta a les necessitats inicials.

Aquesta interfície està definida com es mostra a continuació:

OP.	URI	VALS	FUNCIÓ
GET	/dades	-	Retorna un string en JSON amb tots els valors: { "relesJSON":[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1], "portesJSON":[0,0,1,0,0,1,1,1,0,0,1,1,1,1,1,1], "ctsJSON":[0,0,21,0],

[2] A l'annex II hi ha més informació relacionada amb Modbus

			"tempJSON":[26,32.50,false,false,29,24], "estatJSON":[false,false,true,false,false,false,false,true,true,false,false,false,false,false,false], "alarmaJSON":[0], "tempsJSON":[2016,8,25,12,36,47], "comunicaJSON":[0,0,1,1] }
POST	/rele	x=y	Canvia el estat del relé indicat x: indica el xip que comanda els relés. - 0 i 1 estan a l'esclau 1 - 2 està a l'esclau2 y: indica el relé a canviar l'estat.
	/configura		Guarda la configuració de paràmetres del subsistema
	/temp		Configura valors màxims i mínims de calefacció - Mínima: 0=valor temp - Màxima: 1=valor temp
	/calf		Activa/Desactiva l'estat automàtic de la calefacció
	/dadesReg		Reb el número de programa de reg a la 'y' i torna JSON amb la configuració
	/guardarReg		Reb la configuració d'un programa de reg i la guarda.
	/reset		

Comandament de finestres

Les finestres tenen un motor de dos sentits. En manual el motor gira (cap a un sentit o cap a l'altre) mentre el polsador està polsat. A més a més no es pot accionar el motor cap als dos sentits alhora.

En automàtic s'ha configurat una exclusió per a que no es puga accionar al mateix temps el relé corresponent a tancar i el relé corresponent a obrir. També s'ha configurat un temporitzador que manté el relé en marxa per a obrir la finestra.

Comandament de Llums

Per a poder comandar l'estat dels llums en manual i en automàtic es fa una commutació entre el relé i l'interruptor del llum (es realitza la mateixa operació quan es vol actuar en un llum des de diferents interruptors.

En versions anteriors s'utilitzaven sensors d'energia connectats als esclaus que detectaven el consum, aleshores si hi havia consum el llum estava encés i si no hi havia consum el llum estava apagat. Aquesta informació era transmesa a l'APP per a mostrar el estat actual dels Llums. Funció que s'ha adjudicat als detectors d'activitat que es comuniquen per MQTT amb el sistema domòtic.

Calefacció

El sistema de calefacció a la clínica està compost per radiadors i per aire condicionat. Aleshores s'ha definit una franja de temperatura configurable des de l'APP, si la temperatura està per baix del llímit inferior s'activa el sistema de calor i si la temperatura està per damunt del llímit superior s'activa el sistema de fred.

Reg

Es pot programar fins dos hores de reg i un temporitzador que indica el temps que està regant. Quan el master comprova que l'hora de reg ha arribat activa el relé corresponent al esclau 3 i quan arriba al temps definit pel temporitzador aleshores el desactiva.

Esclaus

Els esclaus són els encarregats de comunicar-se amb els perifèrics (actuadors/sensors). El seu paper és interpretar les ordres rebudes pel màster i enviar la informació que reben per part dels perifèrics.

Es definix el següent mapa de memòria per a cada esclau:

Esclau 1		Esclau 2		Esclau 3	
00001	ENVIARIR	00001	ALARMA	00001	RELE_MARXA
40001	HDADES_IR	40001	RELE_MARXA	00002	ACTIVAT
40002	LDADES_IR	40002	SENSORS	40001	TEMP.
40003	RELE_MARXA				
40004	SENSORS				
40005	C. ENDOLL				
40006	C. A.C.				
40007	C. LLUM 1				
40008	C- LLUM 2				
40009	TEMP.				

Aquest mapa de memòria és utilitzat pel màster per a poder comunicar-se amb els esclaus i consultar/modificar les dades.

Esclau 1

Aquest esclau està ubicat a la sala interior. És l'encarregat de comandar els relés d'obrir/tancar finestres, encendre els llums i calefacció. El funcionament és senzill:

1. Pren lectura dels sensors i guarda les dades als blocs de memòria corresponents
2. Comunica amb el màster per a refrescar la informació dels blocs de memòria
3. Llig el bloc de memòria RELE_MARXA i actualitza l'estat dels relés.

Esclau 2

Aquest esclau està ubicat a la sala exterior. És l'encarregat de comandar els relés d'obrir/tancar finestres de l'exterior i comandar el lector de targetes RFID.

El lector de targetes funciona de la següent forma: Detecta una targeta, llig el seu ID i comprova si està autoritzada. Si està autoritzada aleshores canvia l'estat de l'alarma.

Si no ha llegit ninguna targeta, la primera targeta que llig es considera màster. La targeta màster s'utilitza per a autoritzar la lectura d'altres targetes. Quan es detecta la targeta màster el lector entra en estat de configuració, aleshores espera la lectura de altres targetes. Quan llig una targeta, si no la té registrada aleshores la registra i si ja la té registrada aleshores la desregistra i queda desautoritzada. Al tornar a detectar la targeta màster acaba el mode configuració.

Al activar l'alarma el led es mostra roig durant uns segons i al desactivar l'alarma es mostra verd. Quan està en mode configuració el led parpelleja colors.

A banda del comandament del lector de targetes el funcionament és el mateix que el de l'esclau 1:

1. Pren lectura dels sensors i guarda les dades als blocs de memòria corresponents
2. Comunica amb el màster per a refrescar la informació dels blocs de memòria
3. Llig el bloc de memòria RELE_MARXA i actualitza l'estat dels relés.

Esclau 3

Aquest esclau està situat a l'exterior i s'encarrega del comandament del reg. El reg s'activa mitjançant un relé que està connectat a una electrovàlvula. Mentre la electrovàlvula reb corrent el reg està habilitat, quan la electrovàlvula deixa de rebre corrent deixa de regar. En aquest cas s'ha posat un temporitzador de 10 minuts i un botó per a poder fer un reg manual.

A part de la seua operativa, el funcionament general és el mateix que als demés esclaus:

1. Pren lectura dels sensors i guarda les dades als blocs de memòria corresponents
2. Comunica amb el màster per a refrescar la informació dels blocs de memòria
3. Llig el bloc de memòria RELE_MARXA i actualitza l'estat dels relés.

Detectors d'activitat

Aquestos detectors són xips ESP8266 que publiquen dades als tòpics MQTT corresponents.

La primera vegada que es posen en marxa actuen com a Hotspot wifi per a poder connectar-se a ells i configurar la xarxa wifi a la qual es connectaran i les dades necessàries per a comunicar amb el broker MQTT (usuari, pass, servidor, port, tòpic,...).

Detectors booleans

Aquestos detectors són els més senzills. Es connecten en paral·lel amb el dispositiu a monitoritzar i quan s'alimenta el dispositiu aleshores també s'alimenta el detector, el qual publica al seu tòpic cada 10 segons una senyal. Quan es desconnecta deixa de publicar.

Els utilitze per a detectar l'estat dels llums (encés o apagat).

Detectors de consum

Aquestos detectors estan sempre actius monitoritzant el consum elèctric amb un sensor CT no invasiu, quan detecta consum ho publica al seu tòpic.

Els utilitze per a monitoritzar el consum general del enllumenat.

Detectors de moviment

Aquestos detectors són igual que els anteriors sols que en compte de tenir un sensor CT tenen un sensor de moviment.

Finalment aquestos sensors no s'han posat en productiu per petició del client. Però el funcionament és el mateix que els altres 2 detectors, quan es detecta moviment es publica un missatge.

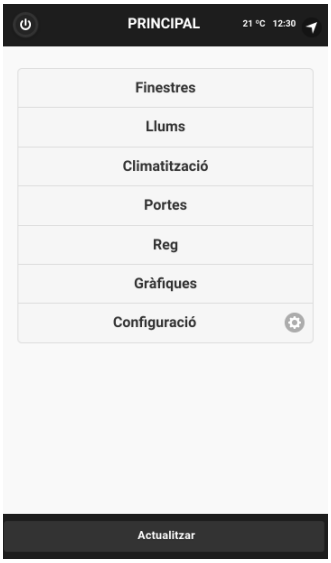
APP (interfície d'usuari)

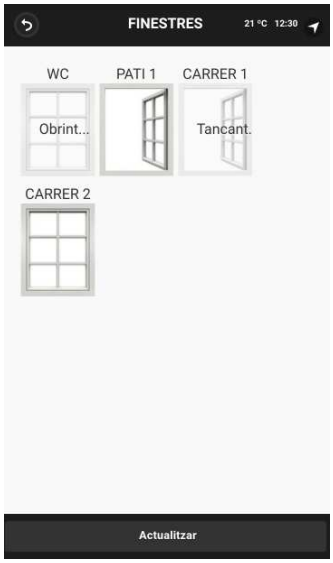
Aquesta aplicació està desenvolupada per a smartphones i es comunica mitjançant missatges MQTT. Com és l' encarregat d'interactuar amb l'usuari, té un apartat per a configurar servidor, usuari i password per a poder connectar amb el sistema central.

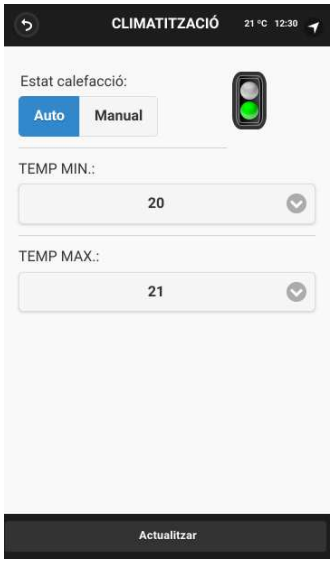
Aquesta aplicació envia peticions d'actualització de dades via MQTT periòdicament i quan l'usuari executa una acció, aquesta és enviada via MQTT al sistema central per a ser processada.

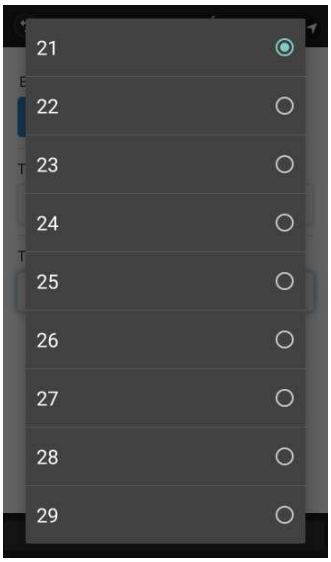
Al estar programada al entorn Cordova ens facilita la migració entre IOs, Windows phone, Android,.... utilitzant tant sols HTML+JS+CSS3.

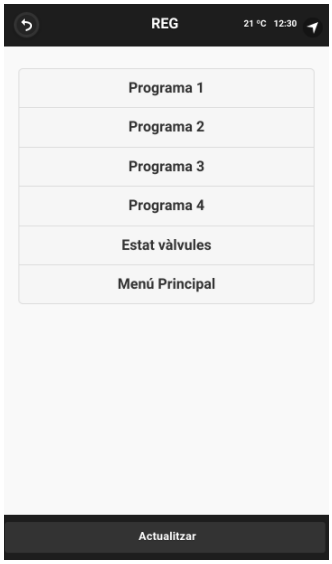
És el nexe entre la persona i el sistema. Té una interfície senzilla amb un menú principal per a poder accedir a cada secció. A continuació es mostres unes imatges de l'APP on es comprova la facilitat d'us i els diferents panels d'interactuació, missatges,...

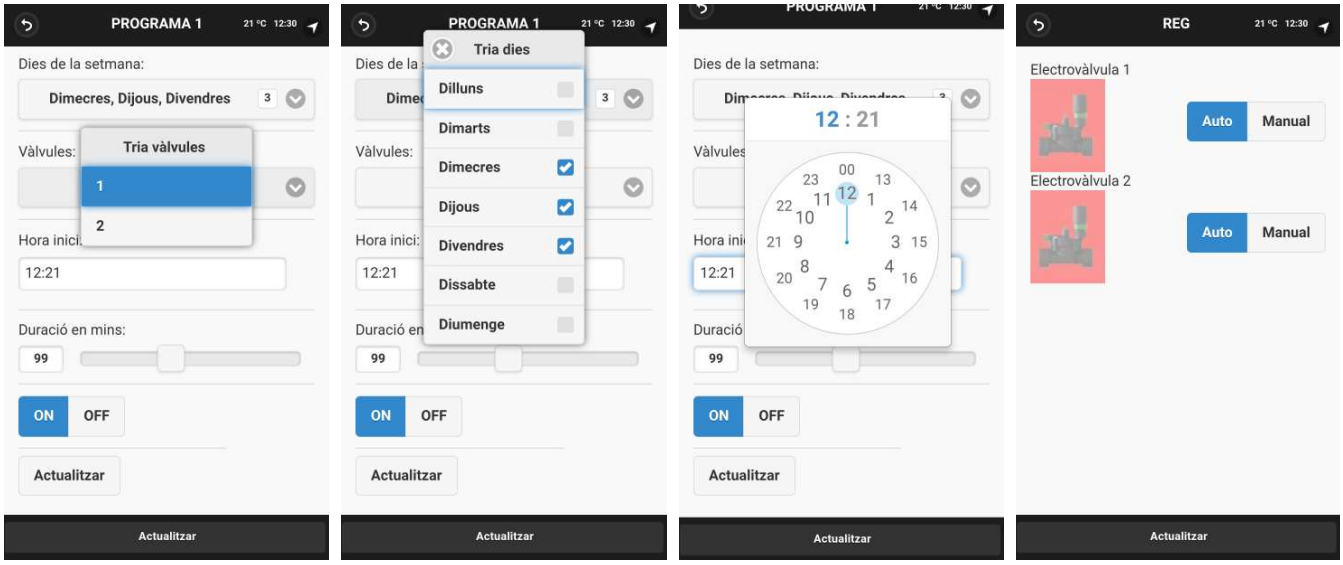


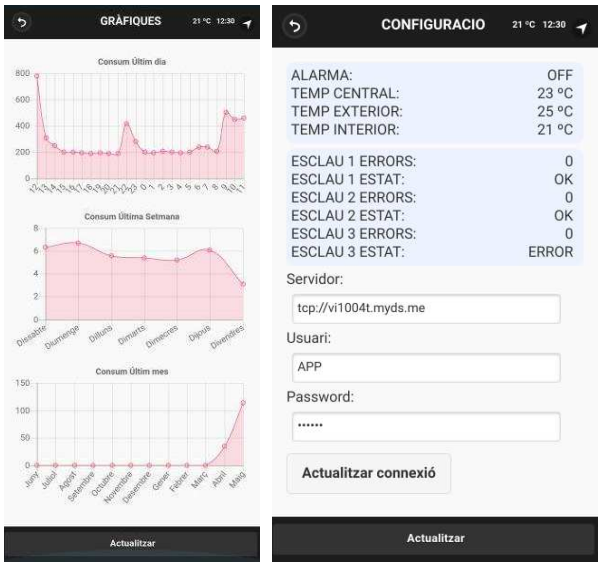












Polzant sobre cada element s'executa l'acció desitjada. Es toca la finestra que es vol obrir o tancar i automàticament s'obri o es tanca, si es vol mantenir una temperatura ambient estable aleshores es definix el rang i s'habilita, el mateix per al reg i finalment es pot veure unes gràfiques amb el consum en kw/h.

Comunicació externa

El que s'ha fet per a poder donar connectivitat externa a l'usuari és duplicar el sistema central. S'ha fet una versió reduïda del sistema habilitant sols la part de comunicació amb l'APP.

Aquest sistema central reduït sols interactua amb les ordres rebudes per part de l'APP, no té accés a l'escriptura de dades a la base de dades. Està composat per:

- Broker MQTT extern amb IP accessible.
- Client mqttwarn intern però configurat per a comunicar amb el broker extern
- Disparador d'events intern però configurat amb el broker extern.

També he fet proves amb un proveïdor d'Internet que assigna IP's públiques. En aquest cas sols es necessita obrir el porte corresponent (en aquest cas 1883) i ja funciona sense problemes (i sense la necessitat d'un servidor extern).

Sistema de comunicació

El protocol utilitzat per a la comunicació del sistema és el MQTT, el client MQTT del sistema reb els missatges, els passa al disparador d'events i, en funció de l'estructura del tòpic i el contingut del missatge, executa l'ordre corresponent. Aleshores he definit unes regles per a estandarditzar la comunicació internament.

La primear part del tòpic defineix l'acció a realitzar per part del disparador d'events

- LOG: Registrar a la base de dades el valor rebut
- ACTUALITZARDADES: Consultar les dades del subsistema modbus i publicar les dades a MQTT amb l'acció LOG.
- APP: Comunica l'usuari amb el sistema.

Tots els elements connectats disposen del seu tòpic, la estructura que tenen és la següent:

/1/2/3/4/.../n-1/n

1. Es l'acció a realitzar: LOG o APP.
2. Ubicació de l'element: Sala1, Sala2, Corredor,...
3. Familia de l'element: Llums, finestres, portes,...
4. Del 4 fins al n-1 es el ID del dispositiu.
5. N: indica la secció: Acció, consum, estat

Per tant la majoria dels elements disposen de 2 tòpics, el que comença en LOG per a disparar l'event de registre de dades i el que comença per APP per a comunicar les dades a l'usuari

Exemples:

```
/LOG/WC/FINESTRES/PATI/1/ACCIO
```

Al rebre el client MQTT nova informació d'aquest tòpic, el disparador d'events guardarà a la base de dades un nou registre amb les dades rebudes

```
/APP/SALA2/FINESTRES/EXTERIOR/ESQUERRA/ACCIO
```

Aquesta informació serà rebuda directament per l'APP, la qual actualitzarà el seu valor per aque l'usuari el puga veure.

L'APP utiliza la següent estrctura de tòpics per a enviar ordres al sistema:

/APP/[ruta|'actualitzar']

d'aquesta forma indica la ruta a utilitzar en la comunicació REST al disparador, passant-li les variables al missatge del tòpic.

Si envia la paraula 'actualitzar' aleshores indica al disparador que consulte totes les dades del subsistema modbus i les publique a MQTT

Exemples:

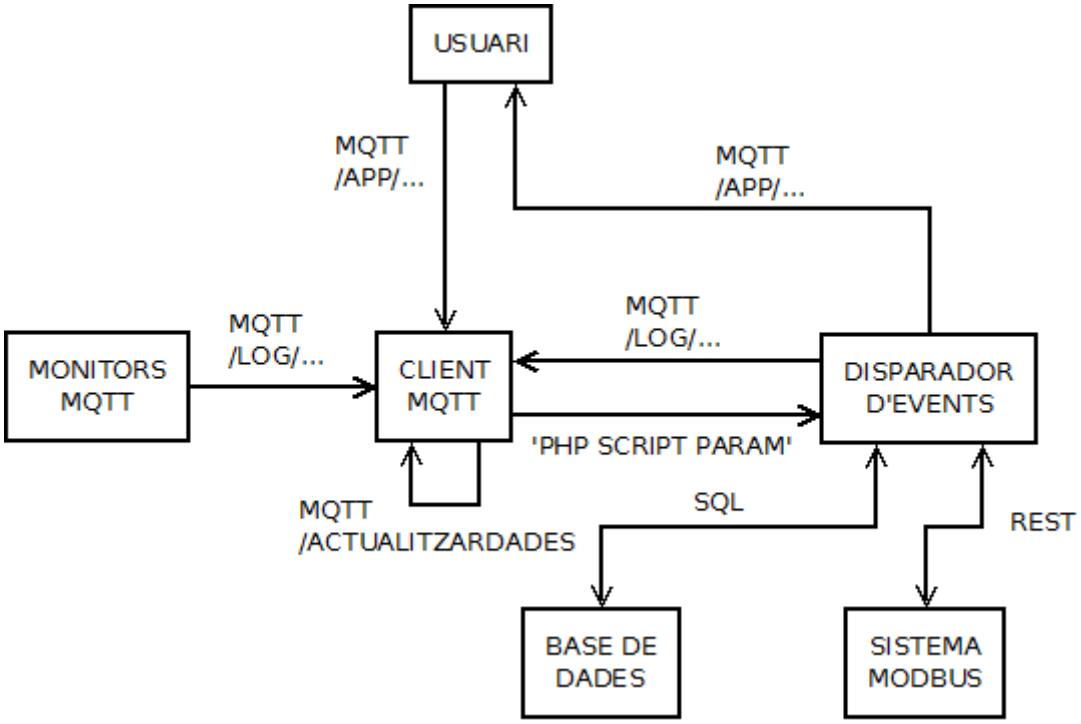
```
/APP/rele amb missatge 0=0
```

El disparador farà la següent consulta REST al subsistema modbus http://ip_arduino/rele?0=0

`/APP/ACTUALITZAR`

El disparador farà la següent consulta REST al subsistema modbus http://ip_arduino/dades

Finalment mqttwarn publica el tòpic /ACTUALITZARDADES per a executar al disparador d'events una consulta de les dades del subsistema modbus, comprovar monitors inactius i guardar a la base de dades les dades que han variat.



El client MQTT li passa al disparador d'events el tòpic i el valor del missatge de la següent forma:

```
php script.php /LOG/WC/FINESTRES/PATI/1/ACCIO@VALOR_MISSATGE
```

I el disparador d'events interpreta el missatge i actua. El missatge està compost per 2 parts separades pel símbol '@'.

Conclusió

En l'època actual què el món s'ha tornat tant competitiu i on qualsevol tendència canvia ràpidament és necessari estar en continua formació i renovació per a poder estar competint a nivell empresarial. El no seguir aquesta premisa el més prompte possible pot fer que el negoci s'estanque i amb el pas del temps desapareixer.

Les grans empreses no tenen cap problema en invertir en tecnologia però als empresaris de les xicotetes empreses els costa més per dos motius: la inversió i la por a fer canvis a l'empresa quan aquesta funciona realment be.

Al fi i al cap la industria 4.0 és posar un nivell de treball superior als ja existents. Es necessiten solucions heterogènies i no els productes que ofereixen les grans empreses als quals s'ha d'adaptar el xicotet empresari: El sistema s'ha d'adaptar a les necessitats de l'empresa i no al contrari.

Existixen les ferramentes adequades per a implementar solucions, però no s'estan creant per a la xicoteta empresa, on el coneixement de la tecnologia és bastant inferior.

Per tant hi ha una manca de servei en aquest aspecte. És important crear un producte adaptable, que siga senzill d'entendre, amb un cost raonable i que els mateixos proveïdors de serveis (empreses d'automatismes en el cas del control energètic, empreses gestore en el cas de ERP's,....) puguen fer d'intermediaris amb les xicotetes empreses. No serà igual de potent/robust que els productes per a grans empreses però sí serà suficient.

Es necessita formar a les empreses proveïdores de la importància de la Industria 4.0 i dotar-les de solucions per a implementar-la.

Una migració a Industria 4.0 suposa una millora en servei al client, suposa destacar entre les demés empreses del sector, suposa invertir en el futur.

Bibliografia

Indústria 4.0

- http://smartcio.es/industria-4-0/
- http://www.rtve.es/
- http://www.eic.cat/noticies/canvi-paradigma-industria-40
- http://www.ecointeligencia.com/2016/01/energia-industria-4-0/
- http://geinfor.com/blog/industria-40/
- Experiència pròpia al Grup DAMM

I2OT

- http://www.forbes.com/sites/jacobmorgan/2014/05/13/simple-explanation-internet-things-that-anyone-can-understand/
- http://wwwen.zte.com.cn/endata/magazine/ztecommunications/2015/2/articles/201507/t20150724_443249.html

Arduino

- https://www.arduino.cc/
- http://www.whatimade.today/esp8266-easiest-way-to-program-so-far/
- http://www.esp8266.com/

Cordova (Disseny APP's)

- https://cordova.apache.org/

Annex I: Arduino

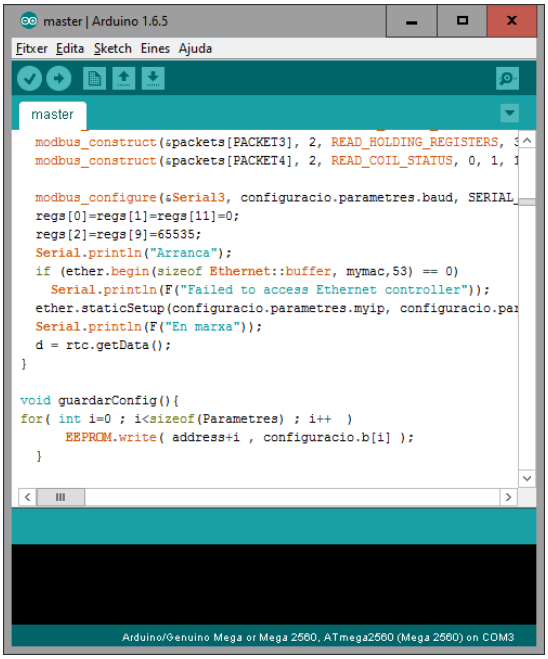
Arduino és una plataforma d'electrònica open-source i està compost tant per software com hardware. És una plataforma realment atractiva perquè és molt fàcil d'utilitzar, és barat i té una corba d'aprenentatge casi nul·la. A dia d'avui hi ha moltes altres empreses que han clonat aquestes plaques electròniques a un preu molt barat, es poden comprar per 2€ o 3€.

La placa per excel·lència per als que s'inicien és la d'Arduino UNO i la seva superior Arduino MEGA 2560. Fa poc que s'han fet lloc al món dels 32bits, inclòs disposen d'una placa per a IoT de 32bits. Però per la meva experiència és millor utilitzar les plaques Aduino Pro Mini per a desenvolupar dispositius IoT senzills i barats (la plataforma 32bits és molt recent encara).

La majoria de les plaques tenen connexió USB compatible tant a Linux com a Mac o Windows, les poques plaques que no tenen connectivitat USB necessiten d'un conversor USB –TTL per a poder programar-les (és el cas de l'Arduino Pro Mini).

Software

La plataforma Arduino facilita software gratuïtament per a poder programar aquestes plaques. Es pot descarregar des de la pròpia web.



A la web també es troba informació sobre les funcions utilitzades, xicotets exemples i tutorials.

Tots els programes contenen la mateixa estructura bàsica:

- Llistat de llibreries a utilitzar
- Definició de variables globals
- Funció setup que sols s'executa al iniciar-se la placa
- Funció loop que s'executa reiteradament.

Després l'usuari pot definir altres funcions i cridar-les des de les funcions setup o loop.

El software de programació té configurat totes les plaques Arduino per a poder programar-les. Al menú Eines està tot el necessari per a configurar la nostra connexió i poder enviar els programes a la placa, posar en marxa el monitor sèrie,...

Les llibreries són necessàries per a poder configurar qualsevol perifèric. Ens faciliten la comunicació en aquestos, reduint-se en molts casos a la cridada a una funció per a obtindre dades sobre ells (sense tenir la necessitat de saber com funciona o es comunica). Es poden gestionar des de la pròpia aplicació (*Sketch -> Include Library -> Manage Libraries...*), indicant el fitxer ZIP (*Sketch -> Include Library -> Add ZIP library*) o copiant la carpeta de la llibreria directament a la carpeta *libraries* dins de l'arrel de la carpeta del programa (es necessita un reinici d'aplicació per a que la detecte). Qualsevol d'aquestes opcions és vàlida i també es poden crear noves llibreries personalitzades.

La majoria dels perifèrics es comuniquen per bus SPI o I2C, els dos presents a les plaques arduino. Aquestos busos ens faciliten molt la integració de diferents perifèrics ja que a un sol port es poden connectar varis perifèrics.

També existeixen perifèrics de connectivitat wifi, ethernet, bluetooth, GPRS/GSM, radiofreqüència, infrarojos,... que ajuden a comunicar les plaques Arduino amb el món.

Hardware

Especificacions tècniques dels productes ofertats actualment:

Nivell Inicial

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P)
	of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Microcontroller	Intel Curie
Operating Voltage	3.3V (5V tolerant I/O)
Input Voltage (recommended)	7-12V
Input Voltage (limit)	7-20V
Digital I/O Pins	14 (of which 4 provide PWM output)
PWM Digital I/O Pins	4
Analog Input Pins	6
DC Current per I/O Pin	20 mA
Flash Memory	196 kB
SRAM	24 kB
Clock Speed	32MHz
LED_BUILTIN	13
Features	Bluetooth LE, 6-axis accelerometer/gyro
Length	68.6 mm
Width	53.4 mm
Weight	34 gr.

Microcontroller	ATmega328
Board Power Supply	3.35 -12 V (3.3V model) or 5 - 12 V (5V model)
Circuit Operating Voltage	3.3V or 5V (depending on model)
Digital I/O Pins	14
PWM Pins	6
UART	1
SPI	1
I2C	1
Analog Input Pins	6
External Interrupts	2
DC Current per I/O Pin	40mA
Flash Memory	32KB of which 2KB used by bootloader
SRAM	2KB
EEPROM	1KB
Clock Speed	8MHz (3.3V versions) or 16MHz (5V versions)
LED_BUILTIN	13

Microcontroller	ATmega32U4
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	20
PWM Channels	7
Analog Input Channels	12
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega32U4)
	of which 4 KB used by bootloader
SRAM	2.5 KB (ATmega32U4)
EEPROM	1 KB (ATmega32U4)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	48 mm
Width	18 mm
Weight	13 g

Microcontroller	ATmega328 *
Board Power Supply	3.35 -12 V (3.3V model) or 5 - 12 V (5V model)
Circuit Operating Voltage	3.3V or 5V (depending on model)
Digital I/O Pins	14
PWM Pins	6
UART	1
SPI	1
I2C	1
Analog Input Pins	6
External Interrupts	2
DC Current per I/O Pin	40 mA
Flash Memory	32KB of which 2 KB used by bootloader *
SRAM	2 KB *
EEPROM	1 KB *
Clock Speed	8 MHz (3.3V versions) or 16 MHz (5V versions)

Nivell millorat

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
LED_BUILTIN	13
Length	101.52 mm
Width	53.3 mm
Weight	37 g

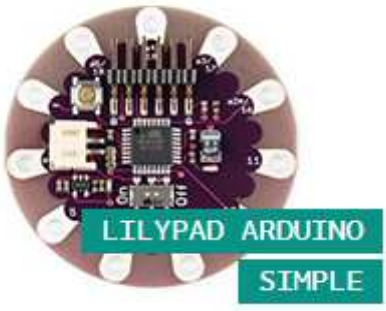
Microcontroller	ATSAMD21G18, 32-Bit ARM Cortex M0+
Operating Voltage	3.3V
Digital I/O Pins	20
PWM Pins	All but pins 2 and 7
UART	2 (Native and Programming)
Analog Input Pins	6, 12-bit ADC channels
Analog Output Pins	1, 10-bit DAC
External Interrupts	All pins except pin 4
DC Current per I/O Pin	7 mA
Flash Memory	256 KB
SRAM	32 KB
EEPROM	None. See documentation
LED_BUILTIN	13
Clock Speed	48 MHz
Lenght	68 mm
Width	30 mm
Weight	12 gr.

IoT

Microcontroller	SAMD21 Cortex-M0+ 32bit low power ARM MCU
Board Power Supply (USB/VIN)	5V
Supported Battery(*)	Li-Po single cell, 3.7V, 700mAh minimum
Circuit Operating Voltage	3.3V
Digital I/O Pins	8
PWM Pins	12 (0, 1, 2, 3, 4, 5, 6, 7, 8, 10, A3 - or 18 -, A4 -or 19)
UART	1
SPI	1
I2C	1
Analog Input Pins	7 (ADC 8/10/12 bit)
Analog Output Pins	1 (DAC 10 bit)
External Interrupts	8 (0, 1, 4, 5, 6, 7, 8, A1 -or 16-, A2 - or 17)
DC Current per I/O Pin	7 mA
Flash Memory	256 KB
SRAM	32 KB
EEPROM	no
Clock Speed	32.768 kHz (RTC), 48 MHz
LED_BUILTIN	6
Full-Speed USB Device and embedded Host	
LED_BUILTIN	6
Lenght	80 mm
Width	55 mm
Weight	32 gr.

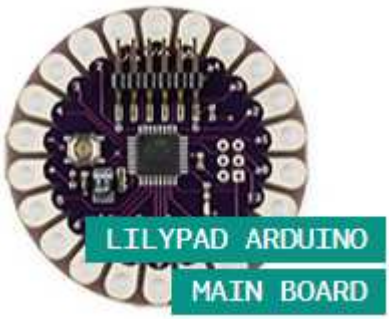
Wearables

Microcontroller	ATtiny85
Operating Voltage	3.3V
Input Voltage	4V-16V
Digital I/O Pins	3
PWM Channels	2
Analog Input Channels	1
DC Current per I/O Pin	20 mA
Absorption	9 mA while running
Flash Memory	8 kB (ATtiny85) of which 2.75 kB used by bootloader
SRAM	512 Bytes (ATtiny85)
EEPROM	512 Bytes (ATtiny85)
Clock Speed	8 MHz
LED_BUILTIN	1
Diameter	27.94 mm



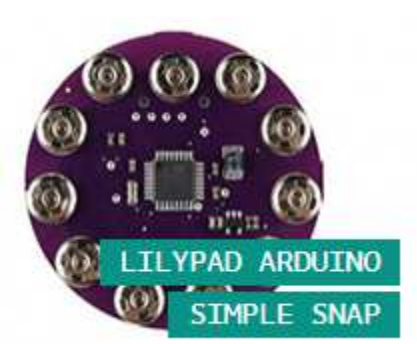
Microcontroller	ATmega328
Operating Voltage	2.7-5.5 V
Input Voltage	2.7-5.5 V
Digital I/O Pins	9
PWM Channels	5
Analog Input Channels	4
DC Current per I/O Pin	40 mA
Flash Memory	32 KB (of which 2 KB used by bootloader)

SRAM	2 KB
EEPROM	1 KB
Clock Speed	8 MHz



Microcontroller	ATmega168 or ATmega328V
Operating Voltage	2.7-5.5 V
Input Voltage	2.7-5.5 V
Digital I/O Pins	14
PWM Channels	6
Analog Input Channels	6
DC Current per I/O Pin	40 mA
Flash Memory	16 KB (of which 2 KB used by bootloader)
SRAM	1 KB
EEPROM	512 bytes
Clock Speed	8 MHz

Microcontroller	ATmega32u4
Operating Voltage	3.3V
Input Voltage	3.8V-5V
Digital I/O Pins	9
PWM Channels	4
Analog Input Channels	4
DC Current per I/O Pin	40 mA
Flash Memory	32 KB (ATmega32u4) of which 4 KB used by bootloader
SRAM	2.5 KB (ATmega32u4)
EEPROM	1 KB (ATmega32u4)
Clock Speed	8 MHz



Microcontroller	ATmega328
Operating Voltage	2.7-5.5 V
Input Voltage	2.7-5.5 V
Digital I/O Pins	9
PWM Channels	5
Analog Input Channels	4
DC Current per I/O Pin	40 mA
Flash Memory	32 KB (of which 2 KB used by bootloader)
SRAM	2 KB
EEPROM	1 KB
Clock Speed	8 MHz
Radius	18 mm

Més informació a https://www.arduino.cc/

Annex II: MODBUS

És un protocol de comunicació industrial que fou desenvolupat en 1979 per a fer possible la comunicació entre dispositius d'automatització. En un principi es va implementar com a protocol de nivell d'aplicació amb la finalitat de transferir dades per una capa serial, Modbus s'ha expandit per a incloure implementació a través de protocols serial, TCP/IP i UDP.

A través de serial existeixen dos tipus de Modbus: Modbus RTU i Modbus Ascii, la gran diferència entre els dos tipus és que el segon envia les dades en format ASCII el qual fa que siga més llegible però també fa que a nivell de màquina tinga que processar més dades. També hi ha una altra diferència: RTU realitza el inici i fi de trames per temps mentre que ASCII ho detecta per caràcters (":" inicia trama i CRLF finalitza trama).

És un protocol màster – esclau, hi ha un únic màster i pot haver varis esclaus (teòricament fins a 247. El màster és el que inicia la comunicació i l'esclau respon. A entorns industrials el normal és que el màster fos un HMI i l'esclau un PLC.

El màster inicia la comunicació enviant una trama amb el número d'esclau al qual es dirigeix, la funció i paràmetres d'aquesta, i finalment el CRC.

11 01 0013 0025 0E84

L'esclau contesta amb una altra trama que conté el número de l'esclau, la mateixa funció, les dades relatives a la funció i finalment el CRC.

11 01 05 CD6BB20E1B 45E6

Cada funció té els seus paràmetres, que s'expliquen més endavant.

Aquest protocol es compon per bancs de registres i per funcions. L'esclau guarda la informació als bancs de registres i el màster utilitza les funcions per a llegir/escriure dades en aquests bancs.

Registres

Hi ha quatre bancs amb un màxim de 9999 registres. Es poden definir de la següent forma:

Número de registre	Tipus	Nom de la taula	Us
1-9999	R/W	Discrete Output Coils	Eixides Digitals
10001-19999	R	Discrete Input Contacts	Entrades Digitals
30001-39999	R	Analog Input Registers	Entrades Analògiques
40001-49999	R/W	Analog Output Holding Registers	Eixides Analògiques

Els registres digitals ocupen 1 bite mentre que els analògics 1 word (2bytes).

(*)Alguns dispositius utilitzen la nomenclatura de prefix + offset. Els prefixes es defineixen com 0, 1, 3 i 4 encara que realment corresponen a 00001, 10001, 30001, 40001. Mentre que el offset està comprés al rang 0000 fins 270E (0 fins 9998).

Funcions

I les funcions més bàsiques són:

Còdi de la funció	Acció	Taula a la que afecta
01 (01 hex)	Llegir	Discrete Output Coils

05 (05 hex)	Escriure	Discrete Output Coils
15 (15 hex)	Multiple escritura	Discrete Output Coils
02 (02 hex)	Llegir	Discrete Input Contacts
04 (04 hex)	Llegir	Analog Input Registers
03 (03 hex)	Llegir	Analog Output Holding Registers
06 (06 hex)	Escriure	Analog Output Holding Registers
16 (16 hex)	Múltiple escriptura	Analog Output Holding Registers

Paràmetres de les funcions

Màster

Els paràmetres de les funcions no són els mateixos al màster i a l'esclau. A la part del màster totes les funcions tenen com a primer paràmetre el número de registre del qual comencen, un WORD (0000 fins 270E) que segons la funció que s'utilitzi ja s'assigna el prefix corresponent.

Funcions 1,2,3,4: El següent WORD indica el número de registres a llegir a partir del definit anteriorment.

Funcions 5 i 6: El següents WORD és la dada a escriure. En el cas de la funció 5 sols pot ser 0xFF00 (ON) o 0x0000 (OFF).

Funcions 15 i 16: S'indica el número de bytes amb dades i després es representen tots aquests bytes a la trama.

Esclau

Funcions 1,2,3,4: Al primer BYTE s'indica el número de bytes amb dades i després es representen tots aquests bytes a la trama.

Funcions 5 i 6: Al primer WORD s'indica el número de registre de la dada, seguit d'un altre WORD amb la dada escrita

Funcions 15 i 6: Al primer WORD s'indica el número de registre de la primera dada i al següent WORD el número de registres escrits.

URLS:

- http://www.ni.com/white-paper/52134/es/
- http://www.simplymodbus.ca/FAQ.htm
- https://code.google.com/archive/p/simple-modbus/

Annex III: Llibreries utilitzades a l'Arduino

Per a facilitar l'accés i comunicació amb els diferents perifèrics es mostren les llibreries utilitzades:

Targeta de xarxa ENC28J60: EtherCard

Url: http://jeelabs.org/pub/docs/ethercard/

Llibreria necessària per a poder comunicar Arduino amb el xip ENC28J60 encarregat de comandar la comunicació ethernet, aquest és un xip barat i amb una connexió màxima de 10Mbits/s (suficient per al tràfic que es genera). Es pot iniciar la tarjeta de xarxa amb configuració DHCP o IP fixa.

El codi necessari per a iniciar correctament el perifèric és el següent:

Abans de qualsevol funció:

```
#include <EtherCard.h>
byte Ethernet::buffer[800];
static byte mymac[] = { 0x74,0x69,0x69,0x2D,0x30,0x31 };
static byte myip[] = { 192,168,1,203 };
static byte mygw [] = { 192,168,1,1 };
static byte mydns [] = { 192,168,1,1 };
BufferFiller bfill;
```

A la funció setup():

```
if (ether.begin(sizeof Ethernet::buffer, mymac,53) == 0)
    Serial.println(F("Failed to access Ethernet controller"));
ether.staticSetup(myip, mygw, mydns);
```

Segons la documentació, el paràmetre de la funció ether.begin que està en roig fa referència al PIN CS que canvia segons el model de la placa Arduino:

	ARDUINO UNO	ARDUINO MEGA
VCC	3,3V	3,3V
GND	GND	GND
SCK	13	52
SO	12	50
SI	11	51
CS	8	53

El que es fa en aquestes funcions és primer definir les variables per al buffer utilitzat i la configuració inicial IP, després a la funció setup() inicialitzar la targeta assignant-li una MAC i se li assigna la configuració IP. Per a més exemples es pot veure els mateixos exemples que porta la llibreria.

Amb aquesta llibreria es pot actuar com a client web, la mateixa funció que un navegador d'internet a un PC:

La funció a utilitzar és:

```
ether.browseUrl("/m2m/", "home.html" , "www.elmeuserver.es", my_callback);
```

Indicant el directori, el fitxer, el servidor i la funció a cridar amb les dades rebudes

La funció my_callback (que es pot utilitzar el nom que es vulga) rep com a paràmetres l'estat, el offset i el tamany de la informació que hi ha al buffer predefinit.

```
static void my_callback (byte status, word off, word len) {}
```

També pot fer funció de servidor, el punt dèbil del servidor és que accepta tots els paquets que vagen dirigits a ell independentment del port al que s'envie. Per a actuar com a servidor es té que crear un switch (en aquest cas un if else niat al loop principal on evalue tots els paquets rebuts al buffer i segons la capçalera que tinga s'actue d'una forma o altra:

Al loop():

```
word len = ether.packetReceive();
  word pos = ether.packetLoop(len);
    // check if valid tcp data is received
    if (pos) {
        bfill = ether.tcpOffset();
        char* data = (char *) Ethernet::buffer + pos;
// receive buf hasn't been clobbered by reply yet
        if (strncmp("GET /hola", data, 9) == 0)

            homePage();
        else if (strncmp("GET /configura", data, 14) == 0)

            webConfiguracio();
        else if (strncmp("GET /dades", data, 10) == 0)
        {
          crearJSON();
          }
        else if (strncmp("POST ", data, 5) == 0)
        {
          canvigeneric(data);
        }
        else{
            bfill.emit_p(PSTR(
                "HTTP/1.0 401 Unauthorized\r\n"
                "Content-Type: text/html\r\n"
                "\r\n"
                "<h1>401 Unauthorized</h1>"));
        }
    ether.httpServerReply(bfill.position()); // send web page data
    }
```

S'analitza el principi de cada paquet i segons corresponga es crida a una funció o a altra. Finalment s'envia una resposta al client. Aquesta resposta es generada per les funcions o per defecte una pàgina d'error.

Per la capacitat del buffer, la informació que se li pot enviar al servidor web és poca, cridar a una web enviant-li pocs paràmetres ja que si es sobrepassa el límit es truncarà la informació.

Referent a aquesta limitació, es pot enviar com a resposta més informació de la que cap al buffer però es separarà per paquets, llevant la indicació per defecte de finalització que genera la funció `ether.httpServerReply`.

Per a aquest cas està la funció `ether.httpServerReply_with_flags` en la qual s'especifiquen els flags que es volen enviar, enviant sols a l'últim paquet el flag de finalització:

```
ether.httpServerReplyAck(); // send ack to the request
bfill.emit_p(dades_per_a_generar_el_paquet_a_enviar);
ether.httpServerReply_with_flags(bfill.position(),TCP_FLAGS_ACK_V);
bfill = ether.tcpOffset();
bfill.emit_p(dades_per_a_generar_el_paquet_a_enviar);
ether.httpServerReply_with_flags(bfill.position(),TCP_FLAGS_ACK_V);
bfill = ether.tcpOffset();
bfill.emit_p(dades_per_a_generar_el_paquet_a_enviar);
ether.httpServerReply_with_flags(bfill.position(),TCP_FLAGS_ACK_V|TCP_FLAGS_FI
N_V);
```

La funció `bfill.emit_p` junt a la macro PSTR fan un paper molt important. Per primera la funció `bfill.emit_p` és la encarregada d'omplir el buffer. La macro PSTR s'utilitza per a poder incloure dades variables a la funció `bfill.emit_p`. Per a utilitzar la funcionalitat de la macro s'indica al text mitjançant la nomenclatura que s'indica a la taula el lloc de les variables, després hi haurà tants arguments a la macro com variables al text.

Codi	Tipus de variable
$S	Variables en RAM
$F	Variables en FLASH
$E	Variables en EEPROM
$D	Variables que són ENTERS

Exemple:

```
bfill.emit_p(PSTR(
"HTTP/1.0 200 OK\r\n"
"Content-Type: text/html\r\n"
"\r\n"
"<html><head><meta charset='utf-8' /> </head><body><h3>CONFIGURADOR</h3><form
action='./configura' method='post'><table>"
"<tr><td>IP:</td><td>"
"<input type='number' name='A' value='$D' min='0' max='255' maxlength='3'
size='1'> ."
"<input type='number' name='B' value='$D' min='0' max='255' maxlength='3'
size='1'> ."
"<input type='number' name='C' value='$D' min='0' max='255' maxlength='3'
size='1'> ."
"<input type='number' name='D' value='$D' min='0' max='255' maxlength='3'
size='1'></td></tr>"), myip[0], myip[1], myip[2], myip[3]);
```

En aquest exemple es mostra una part d'un formulari web en la que apareixen les dades de les variables indicades. El primer codi va lligat a la primera variable indicada, el segon a la segon,...

RS485, Modbus master: SimpleModbusMaster

URLs:

- https://code.google.com/archive/p/simple-modbus/
- https://drive.google.com/folderview?id=0B0B286tJkafVSENVcU1RQVBfSzg&usp=drive_web'

Aquesta llibreria s'utilitza per a poder implementar la comunicació MODBUS RTU al dispositiu master.

Com el màster és l'encarregat d'iniciar la comunicació amb els esclaus, abans de posar-se a configurar aquesta comunicació es revisa els registres accessibles i el seu tipus per part dels esclaus. Amb aquesta informació es genera un mapa de memòria que serà l'utilitzat per a parametritzar el master.

El mapa de memòria i comunicació del màster es composa per:

- Registres: El número de registres és igual al número de dades a les que es vol accedir. En cada registre s'emmagatzemarà la informació rebuda dels esclaus i ocupa un espai de unsigned int (WORD).
- Paquets: El número de paquets és igual al número de funcions que s'utilitzarà. És una estructura amb informació referent a la comunicació de la trama definida. El sistema utilitza la informació definida a aquestos paquets per a comunicar amb els esclaus. Informació emmagatzemada:
    - Unsigned char id.
    - Unsigned char function.
    - Unsigned int address.
    - Unsigned int data
    - Unsigned int local_start_address.
    - Unsigned int request.
    - Unsigned int successful_requests.
    - Unsigned int failed_requests.
    - Unsigned int exception_errors.
    - Unsigned int retries.
    - Unsigned char connection.

Les funcions utilitzades són:

modbus_construct()

Es generen les trames a enviar als esclaus. Per a generar aquestes trames el que es fa és inicialitzar els paquets creats anteriorment amb les dades que es definixen en aquesta funció, els arguments de la funció són:

- Packet *_packet: Punter al paquet definit anteriorment.
- unsigned char id: ID del esclau al que va dirigit la trama
- unsigned char function: número de funció del protocol MODBUS a utilitzar
- unsigned int address: Indica el registre del esclau al que fa referència
- unsigned int data: Depenent de la funció a utilitzar pot indicar el número de registres a llegir o les dades a escriure. Està definit al protocol MODBUS.
- unsigned _local_start_address: indica l'index del registre des del qual pot utilitzar per a guardar/llegir la informació.

Exemple:

```
modbus_construct(&packets[PACKET0], 1, PRESET_MULTIPLE_REGISTERS, 0, 3, 0);
```

Indica que la informació referent a aquesta trama es guarda al paquet packets[PACKET0], va dirigit a l'esclau 1, utilitza la funció 16 de Modbus, el primer registre afectat és el 0, afectarà a 3 registres i la informació a utilitzar està a partir del registre 0 del local master (s'accedirà al registre 0, registre 1 i registre 2).

Si ara es vol crear una nova trama que no implica la modificació/lectura d'aquestos registres, s'indicarà a l'últim valor un número superior a 2.

Exemple:

```
modbus_construct(&packets[PACKET1], 1, READ_HOLDING_REGISTERS, 3, 6, 3);
```

S'indica que la informació referent a aquesta trama es guarda al packets[PACKET1], també va dirigit a l'esclau 1, utilitza la funció 3, el primer registre es troba a la posició 3 de l'esclau, es llegirà la informació de 6 registres que començarà a guardar-se a partir del registre 3 local.

modbus_configure()

Configura el tipus de connexió a utilitzar i la inicialitza. Els arguments de la funció són:

- &Serial: Referència al port físic a utilitzar. A l'Arduino UNO sols pot ser &Serial, a l'Arduino Mega &Serial, &Serial1, &Serial2, &Serial3
- Baud: La velocitat de connexió, accepta les velocitats per defecte de serial
- Format de bite: Exemple, 1 start bit, 8 data bits, 2 stop bits seria SERIAL_8N2
- Timeout: Temps màxim que el màster espera per a la resposta d'un esclau
- Polling: Temps 'de descans' entre envio de trames o 'scan rate'.
- retry_count: Valor que indica el número de renitents de la trama abans de donar per fallida la connexió i no tornar a intentar la comunicació de la trama.
- TxEnablePin: Aquest argument sols s'utilitza en cas de comunicació rs485 i indica quin és el pin que controla quan es pot enviar i rebre
- Packets: fa referència al array de paquets definits.
- TOTAL_NO_OF_PACKETS: Indica el nombre de paquets que hi ha al array de paquets.
- Regs: fa referència al array de registres definits.

Exemple:

```
modbus_configure(&Serial3, 18200, SERIAL_8N2, 200, 150, 100, 4, packets,
TOTAL_NO_OF_PACKETS, regs);
```

modbus_update()

Aquesta funció s'utilitza per a actualitzar les dades al sistema, es crida a aquesta funció al loop() principal del sketch. Es pot cridar més vegades si el programa tarda en executar-se tant que es vol refrescar les dades abans, en aquest cas sempre es respectaran els temps de timeout i polling.

Referent als temps, aquesta llibreria es incompatible amb la funció delay(). Qualsevol delay() major de 100ms afecta negativament a la comunicació.

Una vegada explicat per parts la llibreria, es mostra un exemple complet de configuració i execució:

```
#include <SimpleModbusMaster.h>
#define TOTAL_NO_OF_REGISTERS 9
enum
{
  PACKET0, //esclau1: escriptura reles, IR
  PACKET1, //esclau1: lectura portes, ct's, temp
  TOTAL_NO_OF_PACKETS // leave this last entry
};
Packet packets[TOTAL_NO_OF_PACKETS];
```

```
unsigned int regs[TOTAL_NO_OF_REGISTERS];
setup(){
  modbus_construct(&packets[PACKET0], 1, PRESET_MULTIPLE_REGISTERS, 0, 3, 0);
  modbus_construct(&packets[PACKET1], 1, READ_HOLDING_REGISTERS, 3, 6, 3);
  modbus_configure(&Serial3, 18200, SERIAL_8N2, 200, 150, 100, 4, packets,
TOTAL_NO_OF_PACKETS, regs);
}
Loop(){
  modbus_update()
  for(i=0;i<TOTAL_NO_OF_REGISTERS;i++){
    //mostra la info guardada als registres
    Serial.println(reg[i]);
  }
//mostra l'estat de la connexió amb l'esclau de cada paquet
  Serial.println(packets[PACKET0].connection);
  Serial.println(packets[PACKET1].connection);
}
```

RS485, Modbus esclau: ModbusSerial

URL: https://github.com/andresarmento/modbus-arduino

Aquesta llibreria s'utilitza per a poder implementar la comunicació MODBUS RTU al dispositiu esclau.

El dispositiu esclau és el que comunica amb els sensors/actuadors, aleshores és el que definix el mapa de memòria (al qual es connectarà el master per a llegir/modificar valors). Per tant aquesta llibreria ens ajuda a:

- Configurar la connexió
- Indicar el ID d'esclau.
- Crear el mapa de memòria.
- Interactuar amb el mapa de memòria creat.

Encara que a la web de la llibreria ens indica que també servix per al MODBUS IP, sols s'utilitzarà en mode MODBUS RTU

En primer moment es crea un objecte per a poder utilitzar la llibreria i després ja es pot configurar el necessari al setup():

```
ModbusSerial mb;
```

Configurar la connexió: mb.config()

Es configura la connexió indicant el port de comunicació (en aquesta llibreria es pot utilitzar softwareserial a part dels serials per defecte), la velocitat de connexió i el tipus de bits

Exemple:

```
mb.config(&Serial, 38400, SERIAL_8N1);
```

Indicar el ID d'esclau: mb.setSlaveId()

S'indica el Id de l'esclau, aquest serà únic i estarà al rang de 1 – 247

Exemple:

```
mb.setSlaveId(1);
```

Crear el mapa de memòria

Per a crear el mapa de memòria s'utilitzen vàries funcions en funció del tipus de registre:

Tipus de registre	Funció
Coil	addCoil()
Holding Register	addHreg()
Input Status	addIsts()
Input Register	addIreg()

Aquestes funcions es poden utilitzar de dos formes:

- Indicant el número de registre que es crea
  - Ex.: mb.addHreg(1);
- Indicant el número de registre que es crea i inicialitzant-lo amb un valor
  - Ex.: mb.addHreg(1, 354);

Al cas dels Coils i Input Status sols s'admet com a valors 1 o 0 i als altres dos un valor WORD

Les següents funcions s'utilitzen per a accedir a la informació d'un registre ja creat o per a modificar-la:

Tipus de registre	Funció
Coil	Coil()
Holding Register	Hreg()
Input Status	Ists()
Input Register	Ireg()

En aquest cas actuen igual que les funcions anteriors: amb un argument retornen el valor del registre i amb 2 arguments modifiquen el registre amb el valor del segon argument

Interactuar amb el mapa de memòria creat: mb.task ()

Aquesta és la funció que es crida al loop() al principi de cada execució, és la encarregada de controlar les peticions que li arriben del màster.

Una vegada explicat per parts la llibreria, es mostra un exemple complet de configuració i execució:

```
#include <ModbusSerial.h>
const int LAMP1_COIL = 100;
ModbusSerial mb;
Setup(
  mb.config (&Serial, 38400, SERIAL_8N1);
  mb.setSlaveId (10);
  mb.addCoil (LAMP1_COIL);
  //o també podríem fer mb.addCoil (LAMP1_COIL, true);
}
Loop(){
  mb.task ();
  //posem el pin 'ledPin' al mateix estat definit al Coil
  digitalWrite (ledPin, mb.Coil (LAMP1_COIL));
}
```

Rellotge RTC: RTC

URL: https://giltesa.com/2012/09/02/libreria-gds1307-para-rtc

Aquesta llibreria és l'encarregada de comunicar-se amb el rellotge, facilitant-nos la interacció amb ell. A la url hi ha molta més informació, però ací sols ho utilitzem per a poder tindre un registre horari. A banda el RTC també disposa d'un sensor de temperatura que s'utilitza per a conèixer la temperatura del propi arduino. El funcionament és molt senzill:

```
#include <RTC.h>
RTC rtc(DST_ON); //per a canviar automàticament en horari d'estiu
loop(){
  d = rtc.getData(); //reb data i hora actual
  rtc.getTemperature(); //reb temperatura
}
```

Per a ajustar el RTC es pot fer un sketch sense loop, sols amb setup i posar la següent funció:

```
rtc.setDateTime( 2016, 10, 13, 18, 42, 00 );
```

Posant la data al 13/10/2014 i l'hora a les 18:42

JSON: ArduinoJson

URL: https://github.com/bblanchon/ArduinoJson

Aquesta llibreria és l'encarregada de codificar/decodificar dades en format JSON. S'utilitza per a la comunicació entre l'Arduino Master i l'APP, per tant sols es necessita la funcionalitat de codificar dades a JSON.

A la estructura JSON es poden crear tant objectes com arrays de dades i tots dos es poden niar.

L'ús de la llibreria és el següent:

- Reservar espai de memòria.
- Crear l'arbre o estructura.
- Omplir aquesta estructura amb les dades corresponents.
- Generar el string JSON.

Reservar espai de memòria: StaticJsonBuffer<200> jsonBuffer

La reserva d'espai és molt important ja que si no es reserva l'espai necessari el string JSON serà truncat, i s'ha d'anar amb compte en reservar massa espai per les limitacions de memòria de les plaques Arduino.

Hi ha una forma de calcular l'espai necessari abans de fer la reserva de memòria segons la wiki de la web, inclòs es pot reservar dinàmicament però sols s'aconsella per a tamanys superiors a 2kb. Per a reservar la memòria necessària existixen dos macros: JSON_ARRAY_SIZE(n) i JSON_OBJECT_SIZE(n). Al següent exemple es mostra l'us:

```
//tenint en compte que l'estructura és la següent:
//
//{"sensor":"gps","time":1351824120,"data":[48.756080,2.302038]}
//
//es calcula l'espai necessari
const int BUFFER_SIZE = JSON_OBJECT_SIZE(3) + JSON_ARRAY_SIZE(2);
//es reserva l'espai necessari
StaticJsonBuffer<BUFFER_SIZE> jsonBuffer;
```

Crear l'arbre o estructura

La estructura de JSON ve definida amb un objecte o array com a mínim, al qual després es pot niar més objectes o arrays. Les funcions per a crear-los segons si són objecte/array arrel o niats són:

Arrel	Niats
`JsonArray& array = jsonBuffer.createArray();`	`JsonArray& nestedArray = array.createNestedArray();`
`JsonObject& object = jsonBuffer.createObject();`	`JsonObject& nestedObject = array.createNestedObject();`

Als niats es pot posar opcionalment el nom de l'array o objecte:

```
JsonObject& nestedObject = array.createNestedObject("NOM_IDENTIFICATIU");
```

Inserir dades

La forma d'inserir dades és diferent si es tracta d'un objecte o d'un array. Per a inserir en un array s'utilitza la funció

```
nom_del_array.add(valor_a_inserir)
```

Mentre que per a inserir dades a un objecte es fa com a qualsevol objecte:

```
objecte["valor x"] = "Hola!"
```

Generar el string JSON: root.printTo(variable, MAX_JSON);

Finalment es calcula el tamany utilitzat al generar el JSON i es passa a una variable del tipus char amb aquest tamany:

```
int MAX_JSON = root.measureLength()+1;
char text[MAX_JSON];
root.printTo(text, MAX_JSON);
```

Una vegada explicat per parts la llibreria, es mostra un exemple complet de configuració i execució:

```
//calcular i reservar espai necessari
const int BUFFER_SIZE = JSON_OBJECT_SIZE(1) + JSON_ARRAY_SIZE(8);
StaticJsonBuffer<BUFFER_SIZE> jsonBuffer;
//crear estructura
JsonObject& root = jsonBuffer.createObject();
JsonArray& relesJSON = root.createNestedArray("relesJSON");
//omplir les dades
int i =0;
for(i = 0; i<8; i++){
 relesJSON.add(bitRead(reles[0], i));
}
//Generar el string JSON
int MAX_JSON = root.measureLength()+1;
char text[MAX_JSON];
root.printTo(text, MAX_JSON);
```

EEPROM: EEPROM

URL: https://www.arduino.cc/en/Reference/EEPROM

Aquesta llibreria és utilitzada per a poder llegir/escriure dades a la memòria EEPROM del arduino, per a tindre dades no volàtils. És una llibreria que ve per defecte a l'aplicació. El més normal per a interactuar amb ella és crear un objecte i treballar amb ell.

```
struct Parametres{
  bool calefaccio; /**/
  byte TEMP_MAX, TEMP_MIN, myip[4],mymask[4],mygw[4],mydns[4],mymac[6];
  word trele, baud, timeout, polling, retry_count;
  };
union{
  Parametres parametres;
  byte b[sizeof(Parametres)];
  }configuracio;
```

Amb *struct* es definix l'objecte i les seves propietats i amb *union* definim dues formes d'accedir a les dades: una és cridant directament a la propietat i l'altra és apuntant directament al byte al que volem fer referència.

Per a llegir/escriure a la EEPROM s'utilitza aquesta última forma d'accedir a les dades, aleshores nosaltres en les funcions utilitzem l'accés a les propietats de l'estructura i la llibreria l'accés per byte.

Aleshores per a recuperar tota la informació i guardar-la a una estructura es pot utilitzar una funció com aquesta si no volem cap offset a la EEPROM:

```
for( int i=0 ; i<sizeof(Parametres) ; i++  )
  configuracio.b[i] = EEPROM.read(i);
```

(es comprova el tamany de la estructura i es llig byte a byte la informació de EEPROM)

I per a escriure és el mateix:

```
for( int i=0 ; i<sizeof(Parametres) ; i++  )
  EEPROM.write(i , configuracio.b[i] );
```

(s'escriu en cada posició de la EEPROM el contingut de la estructura)

I si es vol accedir al valor de TEMP_MAX des d'una funció, es farà d'aquesta forma:

```
println(configuracio.parametres.TEMP_MAX);
```

Lector de targetes RFID: MFRC522

URL: https://github.com/omersiar/RFID522-Door-Unlock

Aquesta no és una llibreria, simplement una adaptació del codi per a activar/desactivar l'alarma. Aquest sketch permet tenir una targeta *màster* amb la que donar d'alta altres targetes per a que siguen vàlides. Guarda les dades a la EEPROM.

PCF8574, port expansor I/O: PCF8574

URL: https://github.com/skywodd/pcf8574_arduino_library

Llibreria per a interactuar amb el xip PFC8574, que és un port expander de 8 entrades/eixides digitals per I2C.

Per a utilitzar-la primer es definix la direcció del xip al bus i després ja es pot utilitzar qualsevol de les seves funcions:

```
PCF8574 PCF_38(0x38); //definir l'objecte amb dirección 0x38
PCF_38.write8(i); //escriure el valor I (de 8 bites) al xip
PCF_38.write(i, j); // escriure al I/O i el valor j
PCF_38.shiftLeft(); //moure els valors dels I/O cap a l'esquerra
PCF_38.shiftRight(); //moure els valors dels I/O cap a la dreta
PCF_38.toggle(i); //invertir el valor del I/O i
PCF_38.read8(); //llegir tots els valors de tots els I/O
PCF_38.read(i); //llegir el valor del I/O i
```

DS18B20, Sensor temperatura: DallasTemperature

URL: https://github.com/milesburton/Arduino-Temperature-Control-Library

Utilitzem aquesta llibreria per a consultar les dades del sensor de temperatura, ens dona dades 'llegibles' per a nosaltres. Com en totes les anteriors primer es definix l'objecte i després ja es pot consultar:

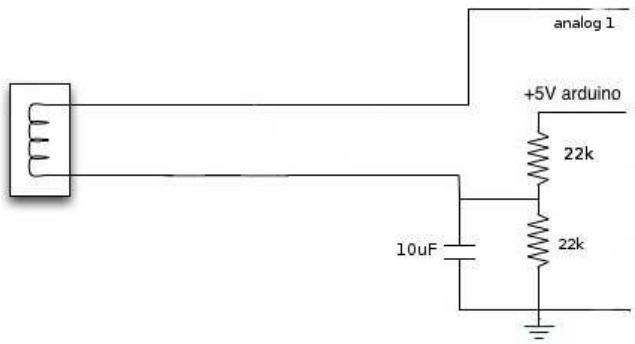
```
#include <OneWire.h>
#include <DallasTemperature.h>
OneWire ourWire(SENSORTEMPERATURA);
DallasTemperature temp(&ourWire);
loop(){
  temp.requestTemperatures();
  int tempcelsius = temp.getTempCByIndex(0);
}
```

SCT013, sensors de consum elèctric: EmonLib

URL: https://github.com/openenergymonitor/EmonLib

Amb aquesta llibreria es pot comunicar amb els sensors no invasius per a mesurar el consum elèctric. Per a interactuar amb aquestos sensors (sensors de la família YHDC SCT-013) es necessita una part hardware fàcil d'implementar per a que la senyal que arriba d'aquestos s'adapte al voltatge acceptat per part d'arduino en les entrades analògiques (0v – 5v).

Per a l'utilitzat al projecte l'esquema és el següent:



Per als sensors YHDC SCT-013-000 es necessita posar una resistència de 33 ohms entre els dos fils del sensor.

Primer es definix l'objecte, s'inicia l'objecte i finalment ja es pot utilitzar per a llegir les dades. Aquesta llibreria és més complexa així que sols mostre el codi utilitzat per al projecte amb el qual es registra els watts que s'estan consumint:

```
#include <EmonLib.h>
EnergyMonitor emon0;
void setup(){
  emon0.current(CT_LLUM1, 30);
}
void loop(){
  int Irms0 = emon0.calcIrms(1480)* 230;
}
```

Depèn de l'ús, aquesta llibreria pot arribar a ser incompatible amb la de l'esclau modbus. Per a poder compatibilitzar les dues es necessita establir un temps entre cada execució d'aquesta. Al cas pràctic del projecte s'aplica un interval de 10 segons:

```
void loop() {
  mb.Hreg(MOD_SENSORS, PCF_01.read8());
  long intervalct = 10000;
  unsigned long  currentMillis = millis();
  if ((unsigned long)(currentMillis - previousMillisct) > intervalct){
    previousMillisct = currentMillis;
    //ACTUALITZAR TEMPERATURA
    temp.requestTemperatures();
    int tempcelsius = temp.getTempCByIndex(0);
    mb.Hreg(MOD_TEMPERATURA, tempcelsius ); //sols es queda en els enteros
    //ACTUALITZAR CONSUMS
    int Irms0 = emon0.calcIrms(1480)* 230;
    mb.Hreg(MOD_CONSUM_ENDOLL, Irms0);
    int Irms1 = emon1.calcIrms(1480)* 230;
    mb.Hreg(MOD_CONSUM_AC, Irms1);
    int Irms2 = emon2.calcIrms(1480)* 230;
    mb.Hreg(MOD_CONSUM_LLUM1, Irms2);
    int Irms3 = emon3.calcIrms(1480)* 230;
    mb.Hreg(MOD_CONSUM_LLUM2, Irms3);
  }
  //COMUNICAR EN MASTER
  mb.task();
  ...
}
```

Amb el desenvolupament dels dispositius satèl·lits de medir consum i biestables es redueix l'us d'aquesta llibreria de 4 medides a sols 1.

PubSubClient: Client MQTT

URL: https://github.com/knolleary/pubsubclient

Aquesta llibreria ens pemet implementar un senzill client MQTT que utilitzarem per a enviar dades al sistema central. Els passos a seguir per a poder configurar un client per a enviar dades són:

Primer es crea l'objecte

```
#include <PubSubClient.h>
WiFiClient espClient;
PubSubClient client(espClient);
```

Es definix al setup el servidor i port al que connectar

```
void setup() {
  client.setServer(mqtt_server, atoi(mqtt_port));
}
```

Es crea una funció per a comprovar si està connectat i si no ho està que es connecte

```
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect("contador", mqtt_user, mqtt_pass)) {
      Serial.println("connected");
      // Once connected, publish an announcement...
      client.publish("outTopic", "hello world");
      // ... and resubscribe
      client.subscribe("inTopic");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}
```

I al loop principal es crida al reconnect per a connectar i una vegada estiga connectat envia el missatge:

```
void loop{
if (!client.connected()) {
    reconnect();
  }
  client.loop();
  client.publish(tòpic_al_que_enviar, missatge_que_es_vol_enviar);
}
```

Adafruit_ADS1015: Port expansor de senyals analògiques (amb més precisió)

URL: https://github.com/adafruit/Adafruit_ADS1X15

Aquesta llibreria ens permet comunicar-nos amb el xip ADS1115. Aquest xip es comunica pel protocol I2C amb altres dispositius (ESP8266 en aquest cas) i conté 4 entrades analògiques d'una precisió de 16 bits. Valors superiors respecte el ESP83266 que sols té 1 entrada analògica amb una precisió de 10 bits.

Aquest xip ens permet configurar el rang a medir dintre dels 16 bits i per tant poder ser més exactes. Ademés ens permet fer dos tipus de medides: medida única, sols utilitza una entrada analògica i té com a referència el voltatge que li entra, i medida diferencial, utilitza 2 entrades analògiques per a realitzar la lectura i es poden medir voltatges negatius. Per a medir el consum es necessita aquest tipus de lectura.

Amb el xip ADS1115 i la medida diferencial no es necessita cap hardware afegit de rectificació per a la lectura dels SCT013 com es comenta en la llibreria EmonLib, i tampoc es pot utilitzar la llibreria EmonLib. Aquest xip sols s'utilitza per als dispositius satèl·lits.

La configuració per a utilitzar les funcions de la llibreria és la següent:

Es crea l'objecte

```
#include <Wire.h>
#include <Adafruit_ADS1015.h>
Adafruit_ADS1115 ads;
```

Al setup es definix la precisió o ganancia

```
void setup(){
  ads.setGain(GAIN_FOUR);          // 4x gain    +/- 1.024V   1 bit = 0.5mV
0.03125mV
  ads.begin();
}
```

I ja es pot comunicar amb el xip, recordar que estarà connectat als pins SCL i SDA corresponents al canal de comunicació I2C

```
void loop(){
  sampleI = (double)ads.readADC_Differential_0_1();
}
```

A la seva web es pot trobar més informació sobre les funcions, declaracions, ajusts de ganància...

URL: https://learn.adafruit.com/adafruit-4-channel-adc-breakouts/programming

WiFiManager: Portal captiu per als dispositius satel·lits

URL: https://github.com/tzapu/WiFiManager

Aquesta llibreria ens ajuda a configurar un portal captiu al dispositiu ESP8266. Així ens permet la configuració 'in situ' del dispositiu, evitant així tenir que reprogramar el dispositiu per a configurar les dades de la xarxa wifi a la que es connecte.

Al iniciar el dispositiu es comprova si hi ha configuració guardada, si no hi ha configuració aleshores es configura el portal captiu des del qual definim:

- SSID
- Password de la xarxa
- Servidor MQTT
- Port MQTT
- Usuari MQTT
- Password MQTT
- Tòpic MQTT

S'ha implementat al ESP8266 un botó de reset per a poder accedir a aquest portal sempre que es vulga canviar la configuració.

Com en la majoria de les llibreries, primer es defineix l'objecte, en aquest exemple també definim paràmetres per defecte

```
#include <DNSServer.h>
#include <ESP8266WebServer.h>
#include <WiFiManager.h>
#include <ArduinoJson.h>
//define your default values here, if there are different values in
config.json, they are overwritten.
char mqtt_server[40] = "192.168.1.240";
char mqtt_port[6] = "1883";
char mqtt_user[34] = "Usuari";
char mqtt_pass[34] = "Password";
char mqtt_topics[255] = "Topics";
#define activarPortal 16
//flag for saving data
bool shouldSaveConfig = false;
//callback notifying us of the need to save config
void saveConfigCallback () {
  Serial.println("Should save config");
  shouldSaveConfig = true;
}
```

I al setup es definix el portal captiu

```
void setup() {
  //clean FS, for testing
  //SPIFFS.format();

  //es comprova si existix o no el fitxer de configuració
  Serial.println("mounting FS...");

  if (SPIFFS.begin()) {
    Serial.println("mounted file system");
    if (SPIFFS.exists("/config.json")) {
      //file exists, reading and loading
      Serial.println("reading config file");
      File configFile = SPIFFS.open("/config.json", "r");
      if (configFile) {
        Serial.println("opened config file");
        size_t size = configFile.size();
        // Allocate a buffer to store contents of the file.
        std::unique_ptr<char[]> buf(new char[size]);

        configFile.readBytes(buf.get(), size);
        DynamicJsonBuffer jsonBuffer;
        JsonObject& json = jsonBuffer.parseObject(buf.get());
        json.printTo(Serial);
        if (json.success()) {
          Serial.println("\nparsed json");

          strcpy(mqtt_server, json["mqtt_server"]);
          strcpy(mqtt_port, json["mqtt_port"]);
          strcpy(mqtt_user, json["mqtt_user"]);
          strcpy(mqtt_pass, json["mqtt_pass"]);
          strcpy(mqtt_topics, json["mqtt_topics"]);

        } else {
          Serial.println("failed to load json config");
        }
      }
    }
  } else {
    Serial.println("failed to mount FS");
  }
  //end read
```

```
  // Es configuren els paràmetres extra que es vol que apareguen
  WiFiManagerParameter custom_mqtt_server("servidor", "mqtt server",
mqtt_server, 40);
  WiFiManagerParameter custom_mqtt_port("port", "mqtt port", mqtt_port, 5);
  WiFiManagerParameter custom_mqtt_user("usuari", "mqtt user", mqtt_user, 32);
  WiFiManagerParameter custom_mqtt_pass("password", "mqtt pass", mqtt_pass,
32);
  WiFiManagerParameter custom_mqtt_topics("topics", "mqtt topics",
mqtt_topics, 255);

  //WiFiManager
  //Local intialization. Once its business is done, there is no need to keep
it around
  WiFiManager wifiManager;

  //set config save notify callback
  wifiManager.setSaveConfigCallback(saveConfigCallback);



  //add all your parameters here
  wifiManager.addParameter(&custom_mqtt_server);
  wifiManager.addParameter(&custom_mqtt_port);
  wifiManager.addParameter(&custom_mqtt_user);
  wifiManager.addParameter(&custom_mqtt_pass);
  wifiManager.addParameter(&custom_mqtt_topics);

Serial.println(digitalRead(activarPortal));
  //Si el botó reset està pressionat aleshores borra la configuració
  if(digitalRead(activarPortal))
    {
      wifiManager.resetSettings();
    }
//En aquest punt es crea el portal captiu i no continua executant
//codi fins que l'usuari guarda les dades
  if (!wifiManager.autoConnect("AutoConnectAP", "password")) {
    Serial.println("failed to connect and hit timeout");
    delay(3000);
    //reset and try again, or maybe put it to deep sleep
    ESP.reset();
    delay(5000);
  }

  //if you get here you have connected to the WiFi
  Serial.println("connected...yeey :)");

  //read updated parameters
  strcpy(mqtt_server, custom_mqtt_server.getValue());
  strcpy(mqtt_port, custom_mqtt_port.getValue());
  strcpy(mqtt_user, custom_mqtt_user.getValue());
  strcpy(mqtt_pass, custom_mqtt_pass.getValue());
  strcpy(mqtt_topics, custom_mqtt_topics.getValue());


  //save the custom parameters to FS
  if (shouldSaveConfig) {
    Serial.println("saving config");
    DynamicJsonBuffer jsonBuffer;
    JsonObject& json = jsonBuffer.createObject();
    json["mqtt_server"] = mqtt_server;
    json["mqtt_port"] = mqtt_port;
    json["mqtt_user"] = mqtt_user;
    json["mqtt_pass"] = mqtt_pass;
    json["mqtt_topics"] = mqtt_topics;
```

```
    File configFile = SPIFFS.open("/config.json", "w");
    if (!configFile) {
      Serial.println("failed to open config file for writing");
    }

    json.printTo(Serial);
    json.printTo(configFile);
    configFile.close();
    //end save
  }
}
```

Aquest és l'exemple seguit als satel·lits però a la seva web es troba més informació per a poder definir una IP, un SSID,...

Annex IV: MQTT (Message Queuing Telemetry Transport)

Com el seu nom indica, MQTT és un protocol de telemetria client – servidor del tipus publish/suscribe. El servidor és el broker al qual els clients es subscriuen al tòpic que els interessa i van publicant la informació. Els altres clients que també estiguen subscrits al mateix tòpic rebran el missatge publicat. Així mateix el protocol és capaç d'utilitzar una connexió segura SSL/TLS, encara que degut a la poca capacitat de processador dels dispositius que utilitzem no serà possible utilitzar aquesta propietat.

Va ser un protocol desenvolupat inicialment per IBM i després lliberat com a codi obert. S'utilitza en moltes aplicacions com per exemple en el Facebook Messenger. Els ports utilitzats són el 1883 per a connexions sense xifrat i 8883

Existeixen 3 nivells de qualitat de servei:

- 0: S'envia el missatge sense confirmació.
- 1: S'envia el missatge una vegada com a mínim fins que es confirma la recepció.
- 2: S'envia el missatge sols una vegada i es confirma que es rep correctament.

Els tòpics es classifiquen en forma d'arbre de la següent forma:

Nivell0/nivell1/nivell2/...

Aquesta estructuració ens permet classificar la informació. També hi ha dos caràcters comodí que ens ajuden a l'accés:

- '+': és un caràcter comodí per a un nivell i es pot posar al mig d'una subscripció
- '#': és un caràcter comodí per a tots els nivells a partir del qual és posat, sempre va al final

Exemple:

Tenint aquesta estructura:

Casa/habitacio1/sensors/llum
Casa/habitacio1/sensors/temperatura
Casa/habitacio1/sensors/llum
Casa/habitacio1/sensors/temperatura
Casa/cuina/sensors/llum

Podem fer les següents subscripcions

- Casa/+/sensors/llum: accés a tots els sensors de llum de casa.
- Casa/#: accés a tots els sensors de casa.
- Casa/Habitacio1/sensors/#: accés a tots els sensors de l'habitacio1.
- Casa/cuina/sensors/llum: accés al sensor llum de la cuina.

Per a donar-li utilitat a aquest protocol en un sistema de control es necessita registrar als tòpics individuals cada 'recolector de dades' en mode escriptura i després registrar als tòpics que siguen necessaris en mode lectura per a poder accedir a les dades i poder tractar-les, funció semblant a la que fa un OPC industrial.

Mosquitto

És el broker més utilitzat, es troba per a tots els S.O. (Linux, MAC, Windows...) amb les següents característiques:

- Accepta connexions SSL/TSL.
- Autentificació per usuari i password.
- Filtrar tòpics: Tòpics disponibles, usuaris autoritzats, tipus de subscripció (R, W, RW).
- Connexió amb altres brokers.

La seva instal·lació a debian és senzilla i per a estar a l'última versió estable per a Debian en aquest cas, cal seguir les instruccions del link:

https://mosquitto.org/2013/01/mosquitto-debian-repository/

Instal·la directament el servei de mosquitto, la ruta del fitxer de configuració és /etc/mosquitto/mosquitto.conf i deixa un fitxer de configuració d'exemple en /usr/share/doc/mosquitto/examples/mosquitto.conf.example

A la seva web té un bon manual on s'explica les opcions de configuració i disposa de utilitats per a fer proves:

- mosquitto_sub: client consola per a registrar-se a tòpics i poder veure en temps real les dades que reben aquests tòpics.

```
mosquitto_sub.exe -h servidor -p port -q 1 -t tòpic -v -u usuari -P password
```

- mosquitto_pub: client consola per a enviar dades a tòpics

```
mosquitto_pub -h servidor -p port -q Nivell_QoS -t tòpic -m missatge -u usuari -P password
```

Enllaços

Web Oficial: https://mosquitto.org/

Client PHP de MQTT: Mosquito-PHP

Mosquito-PHP es basa en les llibreries de client de Mosquitto. Es pot instal·lar manualment baixant el codi font com s'explica a la seva web de Github o es pot instal·lar amb el gestor de repositoris de PHP: PECL.

Es necessita instal·lar prèviament el paquet libmosquitto-dev (en debian) per a poder compilar aquest client.

```
apt-get install libmosquitto-dev
```

Instal·lar els següents paquets per a tindre el repositori de PHP i utilitzar PECL:

```
apt-get install make php5-dev php-pear
```

I ara per a instal·lar aquest client:

```
pecl install Mosquitto-alpha
```

Per a finalitzar la instal·lació s'inclou al php.ini la extensió afegint la següent línia

```
extension=mosquitto.so
```

Per a saber la ruta del php.ini es pot executar a la consola el següent comandament:

```
php -i | grep php.ini
```

Al repositori de github hi ha exemples d'aquest client.

Per a poder comunicar amb els servicis REST del subsistema Modbus utilitze la llibreria phphttpclient, la qual es crida al inici del script.

apt-get install php5-curl

Enllaços

Repositori: https://github.com/mgdm/Mosquitto-PHP

Llibreria client REST: http://phphttpclient.com/

MQTTWARN

Aquesta aplicació és un client de MQTT, el qual es subscriu als tòpics configurats i en rebre dades realitza una acció predefenida. S'utilitza per a serveis de notificació, pot enviar un correu, missatge per Telegram, pushover, twitter,... En aquest treball s'utilitza per a emmagatzemar les dades a una base de dades MySQL. No requereix cap instal·lació, simplement baixar-se del seu repositori al Github i després modificar el fitxer mqttwarn.ini segons les necessitats. Però es necessiten un requisits prèvis ja que l'aplicació corre en phyton:

Instal·lar els requisits:

```
apt-get install python-pip
```

```
pip install paho-mqtt
```

A la web de Github indica cóm configurar cada acció. Cada acció és executada per el seu corresponent script, per tant si sols es va a utilitzar un es poden eliminar els demés fitxers.

En la primera part del fitxer mqttwarn.ini es definixen els paràmetres generals del programa: servidor mqtt i els seus paràmetres, serveis a utilitzar,...

Després es definixen els valors de cada servei (la documentació està a la pròpia web), dins d'aquesta configuració es definixen etiquetes que s'utilitzaran més endavant. També es poden crear àlies d'un servei, modificant així paràmetres generals si és necessari.

Finalment es definixen els tòpics a utilitzar i la seva configuració: Filtres, formats de dades, etiquetes a les quals remet l'acció a realitzar,...

Enllaços

Repositori: https://github.com/jpmens/mqttwarn/