Treball Final de Carrera

Realització d'una aplicació de gestió de xarxes per a la xarxa sensefils de Manresa GuifiBages

Carles Bruguera i Artero

Enginyeria Tècnica d'Informàtica de Gestió

Directora: Mª Dolors Anton i Solà

Vic, setembre de 2007

Resum de Treball Final de Carrera

Enginyeria Tècnica d'Informàtica de Gestió

Títol : Realització d'una aplicació de gestió de xarxes per a la xarxa sensefils de Manresa GuifiBages
Paraules clau : Plone, orientació a objectes, xarxes, wireless, programari lliure
Autor : Carles Bruguera i Artero
Direcció : Mª Dolors Anton i Solà
Data : Setembre de 2007

Resum

En els últims anys, la popularitat de les xarxes sensefils (WIFI) ha anat en augment a un ritme incansable. Des de petits aparells instal·lats a les cases amb aquesta tecnologia com a complement dels routers d'accés a internet instal·lats per diverses companyies, fins a empreses fent petits desplegaments per comunicar entre si les seves seus.

Al marge d'aquests escenaris, s'ha produït un fenomen social d'acolliment d'aquesta tecnologia a nivell mundial, en forma del que coneixem com a xarxes ciutadanes / xarxes lliures / xarxes socials. Aquestes xarxes han estat possibles gràcies a diverses raons que han fet assequible a col·lectius de persones,tant els aparells com els coneixements necessaris per a dur a terme aquestes actuacions.

Dintre d'aquest marc, al Bages, concretament a Manresa, es va començar a desenvolupar una d'aquestes xarxes. Les decisions d'aquesta xarxa d'utilitzar exclusivament hardware i software de codi obert, i determinats aspectes tècnics de la xarxa, ha comportat que la xarxa fos incompatible amb alguns de les aplicacions de gestió de xarxes existents, desenvolupades per comunitats com guifi.net a Osona.

És per això que per garantir el creixement, la supervivència i l'èxit d'aquesta xarxa en el temps, és indispensable poder comptar amb una eina de gestió que s'adigui a les característiques de GuifiBages.

L'objectiu principal d'aquest treball es dotar a la xarxa GuifiBages de les eines necessàries per poder gestionar tota la informació referent a la estructura de la seva xarxa, tant per facilitar l'accés a nous usuaris sense molts coneixements tècnics, com per facilitar nous desplegaments /reparacions / modificacions de la xarxa d'una manera automàtica.

Com a conclusió d'aquest treball, podem afirmar que les avantatges que proporcionen tecnologies com Plone, faciliten enormement la creació d'aplicacions de gestió de continguts en entorn web. Alhora, l'ús de noves tècniques de programació com AJAX o recursos com els que ofereix Google, permeten desenvolupar aplicacions web que no tenen res a envejar al software tradicional. D'altra banda, voldríem destacar l'ús exclusiu de programari lliure tant en els paquets de software necessaris pel desenvolupament, com en el sistema operatiu i programes dels ordinadors on s'ha dut a terme, demostrant que es poden desenvolupar sistemes de qualitat sense dependre de programari privatiu.

Degree Project Summary

Technical Software Engineering Degree

Title : Development of a network management application for Manresa's wireless network GuifiBages
Keywords : Plone, object oriented development, networks, wireless, open source software
Author : Carles Bruguera i Artero
Director : Mª Dolors Anton i Solà
Date : September 2007

Overview

On last years, wireless networks have been growing in popularity in a crazy way. From little Internet routers, packed with wireless technology delivered by some ISP, to company networks, allowing them to link together isolated headquarters.

Side to this examples, there's been a social adoption of this technology by the so-called free networks. Those networks have been possible for several reasons, that delivered to groups of people, even the hardware and the knowledge required to make this kind of networks.

On this scenario, specifically on Manresa, one of this networks started growing. Some of the decisions token about the exclusive use of open-source hardware and software, and some technical stuff on network management, made that network incompatible with many of the network management solutions out there like the one from guifi.net at Osona.

That's why, with the goal of keep growing and survival of this network along time, is essential to count on a management tool designed specifically for Guifibages

The main goal of this project is to provide Guifibages with the right tools to manage all structure related network information, to make easy for new non-techie users to join the network, and to make easy new network deployments / fixings / modifications in a automated way.

In conclusion ,we can state that advantages provided by technologies like Plone, makes web content management systems easier do develop. In addition, the use of new programming methods like AJAX and resources like the ones offered by Google, shows us that web developed applications have nothing to envy with regard of regular desktop software. As a closure, we want to remark the use of open source software on all the tools needed during the development, as in operative systems an other software installed on computers involved in this project, proving that is possible to develop high level software without depending on closed-software solutions.

Índex

Capítol 1

Objectius

1.1 Referent a Guifibages

1. Que l'aplicació generada cobreixi totes les necessitats bàsiques del col·lectiu guifibages per poder gestionar la informació de la xarxa.
2. Que els coneixements tècnics referents a la xarxa requerits per parts de l'usuari final (no administrador) siguin els mínims.
3. Que els administradors de guifibages puguin usar l'aplicació per definir i configurar nous trams de xarxa automàticament, així com mantenir els existents.

1.2 Referent al futur de l'aplicació

1. Crear un model de dades el més genèric possible.
2. Tenir en compte paràmetres presents a guifi.net, per tal de fer l'aplicació compatible.

1.3 Referent al disseny

1. Fer servir els coneixements adquirits com a administrador i desenvolupador d'algunes de les parts de l'aplicació web que es troba en producció a guifi.net
2. Prendre dedicions partint de valorar en quins punts guifi.net podria millorar, i quines parts li manquen, per tal que el resultat d'aquest projecte pugui ser una base vàlida per una futura migració.

Capítol 2

Requeriments

2.1 D'on surt la idea

La idea neix a la ciutat de Manresa l'any 2005, on un col·lectiu de persones interessades per les noves tecnologies, comencen un projecte comú per a difondre l'ús de les tecnologies de xarxes sensefils als ciutadans(concretament l'estàndard 802.11bg). Aquest projecte va ser subvencionat per l'Ajuntament de Manresa, sota el format de 5 jornades teòriques i pràctiques, on es van donar a conèixer, des d'aspectes físics de la tecnologia, ones electromagnètiques, antenes, etc., fins als aspectes informàtics que fan que funcioni tot plegat, com conceptes generals sobre xarxes, adreçament, encaminament. La part pràctica d'aquestes jornades va estar encarada a configurar equips wifi que posteriorment formarien part de la actual xarxa guifibages, i en algunes jornades també es va procedir a la instal·lació i posada en marxa d'alguns d'aquests equips. Aquests actes es van dur a terme aproximadament durant un any des de la creació del grup.

Fins a l'actualitat, un cop ja acabades les jornades, alguns dels integrants del col-lectiu original i algunes persones més que es van unir al projecte, han vetllat per acabar la construcció de la xarxa que havia estat projectada en un principi, i també pel creixement de la mateixa. Això però, no va resultar tasca fàcil, per diversos aspectes essencials que no és donaven aleshores:

Massa crítica Per fer que una xarxa d'aquestes característiques atregui a la gent, és necessari que al darrera hi hagi un determinat nombre de persones que li donin suport i hi dediquin temps.

Gestió de la xarxa Per tal de poder créixer controladament, s'ha vist que és necessari una gestió descentralitzada de la xarxa a nivell de les persones capacitades per fer-ho : No és gens bo que nomes poques persones siguin capaces de gestionar la xarxa, i/o que es necessitin grans coneixements tècnics per fer-ho.

Interès general És imprescindible que la gent vegi la xarxa com alguna cosa de la qual en pot treure algun profit en algun nivell.

Paral·lelament, cert temps abans del projecte de guifibages, a Osona, un grup de gent va fer un pas semblant al de la gent de Manresa, però amb diferencies significatives:

- A Manresa, el projecte era subvencionat per l'Ajuntament de Manresa, i l'objectiu principal eren les jornades de divulgació; la construcció de la xarxa, era un resultat de les propies jornades. A

Osona no hi havia una subvenció com a tal, sinó que era el propi grup que va començar a muntar la xarxa amb els coneixements justos i els seus recursos.

- A Manresa, la xarxa no responia a una necessitat real dels ciutadans ni de la gent que la va iniciar, en canvi a Osona és tractava d'un intent de fer arribar la banda ampla a llocs on ningú s'havia molestat a fer-la arribar. Un dels incentius doncs era l'accés a Internet mitjançant ampla banda, cosa que que a la ciutat de Manresa no era un problema.

- Els primers èxits de funcionament a Osona, van fer que la xarxa ràpidament s'entengués de Gurb, on es va iniciar, fins a d'altres municipis veïns. Això juntament amb la necessitat que comentava anteriorment, va ser el que va fer que la massa crítica i l'interès general pel projecte no tardés a créixer. La manca d'aquest dos aspectes, ha sigut sens dubte un dels motius pels quals la xarxa guifibages no ha crescut gaire des de que va ser creada.

- A partir de cert nivell de creixement, a Osona de seguida es va veure la necessitat d'una eina per gestionar les dades de la xarxa, concretament adreços ip, i dades geogràfiques. Es va optar per desenvolupar una eina de gestió en entorn web, que ha sofert molts canvis des de la primera versió, aixi com dues grans migracions de dades entre versions. A guifibages, la gestió tot i ser via web també, no era automatitzada, i per tant més subjecte a l'error humà.

- Un dels patrocinadors de guifibages era Catux (Associació d'usuaris de GNU/Linux de la Catalunya central) el que juntament amb la convicció de la majoria dels components de guifibages, feia un requisit imprescindible l'ús de programari lliure en tots els aspectes de la xarxa. A Osona, els equips que s'utilitzaven es modificaven amb un software que per part de la gent de Manresa van decidir no utilitzar perquè no era 100% programari lliure. Això juntament amb d'altres desavinences en quant a aspectes tècnics de com estructurar la xarxa, van fer que guifibages optés per una solució tècnica prou diferent com perquè les eines desenvolupades per a guifi.net a Osona, no servissin per a gestionar la xarxa de Manresa.

Arribats a aquest punt avui en dia, abans de l'inici de l'aplicació de que es objecte aquest treball, ens trobem amb una xarxa (guifibages) que funciona, amb una vintena d'usuaris regulars, i uns 40 registrats però que necessita d'una eina per poder gestionar la xarxa, per poder créixer controladament. En certa manera, en molts aspectes ha estat un peix que es mossegava la cua. No es veia la necessitat immediata d'una aplicació a mida per gestionar la xarxa, fins que aquesta no agafés un volum determinat, però és difícil d'arribar a aquest volum sense les eines adequades.

És d'aquí doncs, d'on surt la idea/necessitat de l'aplicació de gestió per la xarxa sense-fils guifibages.

2.2 Orígen dels requeriments

Els requeriments inicials que es fan des de guifibages són exclusivament per a crear una aplicació per satisfer les seves necessitats. D'altra banda, amb l'evolució que ha portat la aplicació de guifi.net, s'han vist canvis radicals, en el model de dades i en el propi codi de l'aplicació, cada vegada que es volia introduir algun canvi o alguna millora a la xarxa. No tenir previstos alguns aspectes, ha comportat encallaments importants en la xarxa d'Osona, i solucions que molta gent considera "pedaços"per no tenir un disseny consistent de base. És per això que apart dels requeriments de guifibages, també es tindran en compte els requeriments per tal d'evitar que l'aplicació resultant tingui aquest tipus de problemes en un futur, i que el model de dades sigui capaç de contenir la informació tant de la solució tècnica de guifibages com de guifi.net[1].

2.2.1 Requeriments de guifibages

- Guardar tota la informació referent a la xarxa : jerarquia de xarxa, protocols d'encaminament, rangs d'adreces i adreces IP.
- Guardar tota la informació geogràfica de la xarxa: zones geogràfiques, nodes (ubicacions).
- Representació gràfica de la informació geogràfica : sistema de mapes per representar nodes i enllaços entre nodes.
- Guardar tota la informació referent als aparells utilitzats per la xarxa: models, firmwares, SO's, interfícies de xarxa/radio, i els paràmetres d'aquestes.
- Interfícies necessàries estil *assistent* per poder donar d'alta els nodes clients.
- Interfícies necessàries per poder donar d'alta qualsevol d'aparell de manera genèrica (només administradors).
- Auto-generar la informació i fitxers necessaris per poder configurar els aparells de la xarxa.

2.2.2 Requeriments extres derivats de guifi.net

- Poder definir qualsevol aparell nou i la seva estructura interna sense haver de modificar codi existent.
- Diferenciar a nivell de les interfícies d'usuari qui és administrador i qui és usuari normal, fent servir com a criteri, els coneixements que cada usuari pot tenir.
- Simplificar al màxim les interfícies en dos aspectes :
  1. Reduir al màxim les recàrregues de les pàgines.
  2. Reduïr al màxim les dades a demanar als usuaris.

[1] Veure secció 8.1 Treball futur

2.3 Especificació de requeriments

A continuació explicarem en detall els conceptes que es presenten en els requeriments, i que ens serviran per a poder fer-ne l'anàlisi posteriorment. Distingirem entre dues categories per classificar els elements d'interès del domini:

2.3.1 Part física de la xarxa

- **Zona** : Una zona és una determinada àrea geogràfica, que pot coincidir amb territoris existents(comarques, ciutats) o ser-hi completament independent. Una zona pot tenir d'altres zones associades, indefinidament, seguint una jerarquia en forma d'arbre. D'una zona en necessitem saber el nom, l'extensió, i un nom curt que ens servira de prefix pels nodes.

- **Node** : Un node és una ubicació geogràfica, un punt al mapa. Un node pertany a una única zona. Del node necessitem saber la seva latitud i longitud per poder-lo ubicar en un mapa, la seva alçada respecte el terra (NO respecte al nivell del mar), una adreça de contacte de la persona encarregada, un nom, i una abreviació del nom, amb caràcters de 7 bits. Un node es troba en un estat determinat per l'estat dels aparells que conté. Un node és el lloc on s'instal·len aparells, pot tenir diversos aparells o no tenir-ne cap.

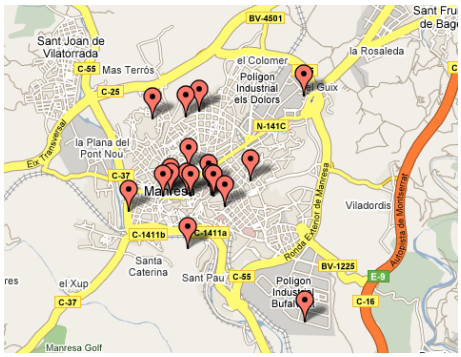


Figura 2.1: Zona corresponent a Manresa, amb nodes marcats

- **Aparell** : Un aparell és un dispositiu de xarxa, que pot tenir 1 o més interfícies de xarxa, de diversos tipus, i cada una d'aquestes interfícies pot tenir interfícies virtuals associades, també de diversos tipus. Quan una interfície X tingui interfícies virtuals , X serà anomena interfície "master "de les virtuals. No poden existir interfícies virtuals sense una master. De l'aparell necessitem saber quantes interfícies te i de quins tipus, les interfícies virtuals, de quines interfícies físiques *master* depenen, quin paper juga l'aparell dins la xarxa, en quin estat de funcionament es troba l'aparell, de quin model és, i quin firmware o sistema operatiu utilitza.

- **Firmware** : Un firmware és el programari que porta un aparell per permetre el seu funcionament. A vegades un mateix equip, pot ser modificat amb diversos firmwares, modificant aixi certs aspectes del seu funcionament. Cada firmware o SO, té una manera específica de configurar-se, independentment del hardware que tingui a sota. Els modes a que es poden configurar les interfícies sensefils, depenen del firmware i no de la interfície en si.

- **Interfícies**: Una interfície de xarxa és l'element físic d'un equip que permet la comunicació amb un altre equip a través d'un medi, que pot ser cablejat o sensefils. De cada interfície que forma un aparell, necessitem saber l'adreça MAC única, i si esta habilitada per funcionar. A més les interfícies que proporcionin tecnologia sensefils, necessitem saber, referent a la senyal que emeten: ssid, banda, canal, mode, potència, referent a l'antena que utilitzen: angle, elevació guany, i tipus. Les interfícies virtuals, poden ser de 4 tipus, cablejades , sensefils, virtuals o lògiques. Al seu temps, les interfícies cablejades i sensefils és diu que són interfícies físiques. Qualsevol altra no ho és.
  - **interfícies cablejades** : les que fan servir qualsevol tecnologia guiada per transmetre les dades.
  - **interfícies sensefils**: les que fan servir qualsevol tecnologia no guiada per transmetre les dades.
  - **Virtuals** : interfícies que no existeixen físicament, sinó com a una representació informàtica en la configuració. Depenen d'una interfície física, però poden diferir en la configuració amb aquesta. Una interfície física pot tenir cap o múltiples interfícies virtuals.
  - **Lògiques** : interfícies virtuals, però que no depenen duna interfície física per existir. Poden ser independents, com els túnels, o poden estar compostes d'altres interfícies, com els ponts.
    - **Ponts** : són interfícies lògiques que agrupen diverses interfícies físiques, i la única configuració del pont afecta a aquestes. Una interfície física nomes pot ser part d'un pont a la vegada. Un pont ha de tenir com a mínim una interfície física.
    - **Túnels** : són interfícies lògiques per unir lògicament xarxes que no estan unides físicament,a través d'una segona xarxa portadora. Un túnels només pot estar associat a un altre túnel.

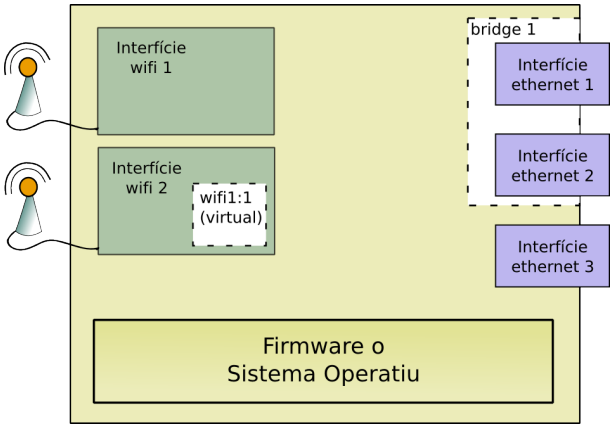


Figura 2.2: Diagrama d'un aparell amb diverses interfícies

Les interfícies físiques han de contenir informació sobre si permeten la creació d'interfícies virtuals i quina quantitat. El alguns casos en que una interfície física està associada a una virtual o a un pont, pot interessar prohibir l'assignació d'una adreça a aquella interfície per motius de configuració de l'aparell. Per tant cal indicar si permetem o no assignar adreça ip a una interfície física.

2.3.2 Part lògica de la xarxa

S'han de poder definir principalment 2 tipus de xarxes segons l'enrutament que utilitzin, OSPF i OLSR (Adhoc).

Xarxes Adhoc

Les xarxes adhoc tindran un rang associat, del qual podran agafar subrangs i adreces. Una xarxa adhoc te associat un canal i un essid únic per tots els aparells que en formaran part. Una xarxa adhoc necessita distingir els subrangs que va gastant en 3 tipus o particions: particions master, particions lliures, i particions ovpenvpn.

- Les particions masters guardaran els rangs que s'utilitzaran per la part sensefils de la xarxa en els aparells denominats "masters olsr", contindran sempre un subrang de la xarxa adhoc i no repetit en cap altra partició.
- Les particions lliures guardaran els rangs que s'utilitzaran per la part sensefils de la xarxa en els aparells denominats "clients olsr", contindran sempre un subrang de la xarxa adhoc i no repetit en cap altra partició.
- Les particions openvpn guardaran els rangs que s'utilitzaran per connectar aparells "masters olsr"o "clients olsr"a traves de túnels entre punts de la troncal, coincidiran sempre amb un subrang de la xarxa adhoc i no repetit en cap altra partició.

Les adreces ip's que s'assignin a cada partició, hauran de ser adreces que estiguin dins del subrang assignat a la partició. La màscara utilitzada en aquestes adreces, serà sempre independent de la màscara de la partició associada. Les ip's pertanyents a particions lliures i master, adoptaran la mascara del rang de la xarxa adhoc, les ip's pertanyents a particions openvpn, adoptaran sempre una màscara /32.

Xarxes OSPF

Les xarxes OSPF, segueixen una jerarquia basada en àrees. Hi ha una àrea troncal, que ha d'estar en contacte amb la resta de àrees, cada una amb un identificador únic diferent de 0 (l'indentificador de la troncal). Cada una d'aquestes àrees te un conjunt de rangs OSPF associats. Cada un d'aquests rangs, pot tenir un tipus associat, per determinar l'ús que s'en farà: rangs públics per la connexió de clients, i rangs privats per enllaçar aparells entre si.

Cada rang estarà partit en xarxes, de qualsevol mida sempre que sigui un subrang del rang OSPF al que pertany. Les ip's pertanyents a aquestes xarxes, tindran les mateixes característiques que la resta d'ip's, però la mascara que utilitzaran serà la de el rang immediatament superior al que pertanyin.

Les xarxes adhoc, utilitzen les xarxes OSPF com troncal per comunicar diversos trams aïllats. Tot i que no és un requeriment tècnic d'aquest tipus de xarxa, ni afecta en la jerarquia d'aquestes, per claredat d'organització, es dirà que el rang de les xarxes adhoc, al igual que els rangs utilitzats per les àrees OSPF, formaran part d'un únic rang o conjunt de rangs, agrupat com a àrees BGP, protocol d'encaminament necessari per comunicar grans àrees OSPF entre si. Una xarxa OLSR mai es comunicara directament amb una altra xarxa OLSR si no és a traves d'una troncal OSPF.

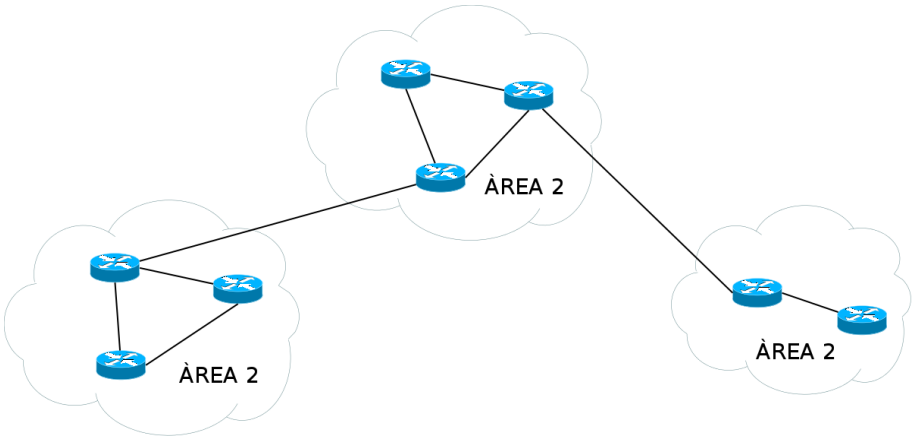


Figura 2.3: Diagrama d'una xarxa OSPF

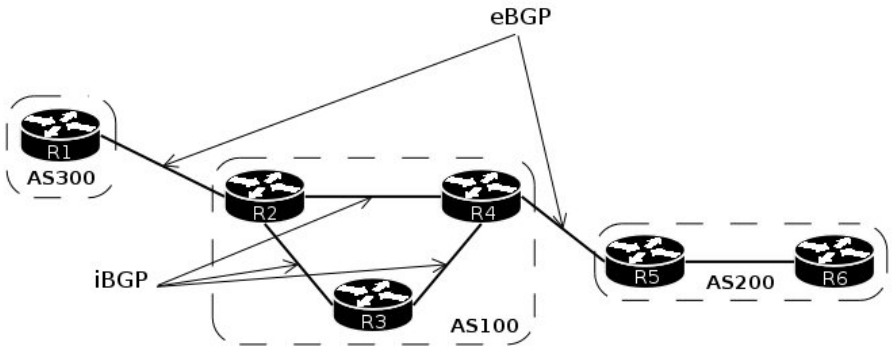


Figura 2.4: Diagrama d'una xarxa BGP

El protocol BGP ocupa el punt més alt en la jerarquia de xarxa. Una àrea BGP és coneix com a AS, i te un identificador únic associat, i un o mes rangs del que es poden servir els altres nivells de la xarxa, esmentats anteriorment. Cada AS engloba tota una xarxa OSPF, on hi poden haver routers que només formin part de la xarxa OSPF, i d'altres que comuniquin el protocol OSPF amb el BGP. Al protocol BGP se l'anomena i funciona de diferent manera en funció de entre quins aparells funcioni: iBGP si esta comunicant aparells dins un mateix AS, i eBGP si comunica aparells de diferents AS.

2.3.3 Rols dels aparells dins la xarxa

Com hem dit anteriorment, un aparell pot tenir diverses interfícies. En funció de la combinació dels tipus de xarxes a les quals estan associades cada una de les interfícies d'un aparell, direm que un aparell juga un determinat rol dins la xarxa:

1. **Router Intern** : Un aparell el qual només està a dins d'una àrea OSPF, i no limita amb cap d'altra.
   - Té com a mínim una interfície amb una ip d'una xarxa ospf a qualsevol àrea.
2. **Router ABR** : Un aparell que fa d'ABR entre una àrea OSPF X i la troncal.
   - Té com a mínim una interfície amb ip de l'àrea X.
   - Té 1 interfície amb ip de la àrea 0 (troncal).

3. **Border BGP** : Un aparell que comunica diverses àrees BGP entre si.
   - Té com a mínim una interfície amb ip d'alguna àrea 0 (troncal).
   - Té com a mínim una interfície comunicada amb un altre Border BGP.
4. **Master OLSR** : Un aparell que uneix una xarxa adhoc OLSR amb la troncal OSPF, i es pot comunicar amb altres masters a través de túnels.
   - Té 1 interfície amb una ip d'una partició master.
   - Pot tenir cap o múltiples interfícies túnels amb una ip d'una partició openvpn.
   - Té 1 interfície amb una ip d'una xarxa OSPF.
5. **Client Freifunk** : Un aparell que forma part d'una xarxa adhoc OLSR.
   - Té una interfície amb una ip d'una partició lliure.
   - Pot tenir cap o múltiples interfícies túnels amb una ip d'una partició openvpn.
6. **Client AP** : Un aparell que connecta amb un AP d'una xarxa OSPF en mode client.
   - Té una interfície amb una ip del rang de l'ap-

2.3.4 Restriccions per enllaçar aparells

Un enllaç entre dos aparells representa una unió a nivell físic mitjançant cable o ones, i a nivell lògic, usant adreces d'una mateixa subxarxa. Dos aparells es podran enllaçar entre ells, si compleixen una serie de requisits, que depenen de l'ubicació física de l'aparell, el tipus d'enllaç i el tipus i configuració de les seves interfícies de xarxa:

- Només es podran fer enllaços entre interfícies si a nivell físic comparteixen el tipus: es poden donar els casos següents
  - les interfícies són físiques, i ambdues son del mateix tipus (cablejades o sensefils).
  - una o les dues interfícies són virtuals i el seu master/s coincideixen en tipus.
  - una o les dues interfícies són ponts, i aquest pont/s te alguna interfície comuna en tipus.
- Els túnels són l'excepció de la restricció anterior, ja que es poden comunicar independentment del tipus de les altres interfícies.
- Només es podran fer enllaços de cable entre interfícies d'aparells del mateix node.
- Si la interfície que enllacem ja te algun enllaç, el mode que tingui assignat limitarà les possibilitats d'enllaç.
- Si la interfície que enllacem no te cap enllaç, podem autoconfigurarla si la configuració de la interfície remota ens ho permet
- En cap cas un enllaç podrà modificar la configuració bàsica d'una interfície remota.

2.3.5 Funcionalitat dels mapes.

Els mapes han de complir amb les següents funcions:

- Eina per afegir nodes a la xarxa sense saber-ne les coordenades numèriques.
- Veure tots els nodes de la xarxa en una regió concreta així com els enllaços entre nodes.
- Comprovar distàncies entre nodes existents i coordenades arbitràries de la xarxa.
- Comprovar distàncies entre coordenades arbitràries de la xarxa.
- Comprovar perfils d'elevació entre coordenades arbitràries de la xarxa.

Capítol 3

Anàlisi del Sistema

3.1 Model Dinàmic

3.1.1 Casos d'ús

Per tal de poder determinar les funcions que tindrà el sistema, ens basarem en l'elaboració d'un diagrama de casos d'ús (Figura 3.1). Per cada cas d'ús, detallarem quins són els passos que ha de dur a terme el sistema. De l'anàlisi dels casos d'ús n'extraurem l'informació necessària per determinar els mètodes que faran falta incloure en cada objecte del sistema.

3.1.2 Descripció dels casos d'ús

- **Alta de zona nova**

  Descripció: Crear una zona nova i guardar-ne les dades entrades.

  Pre-condicions: No n'hi han.

  Flux Principal:

  1. Buscar coordenades i zoom de la zona pare i centrar-hi el mapa.
  2. Validar que no hi hagi cap zona amb el mateix nom al mateix nivell.
  3. Guardar la zona.

  Fluxos Alternatius:

  1. Si no hi ha zona pare, o la zona pare no te coordenades, centrarem el mapa a un valor per defecte.

  Post-condicions: Si la validació és correcte, zona nova creada, sinó retornar error referent al camp incorrecte .

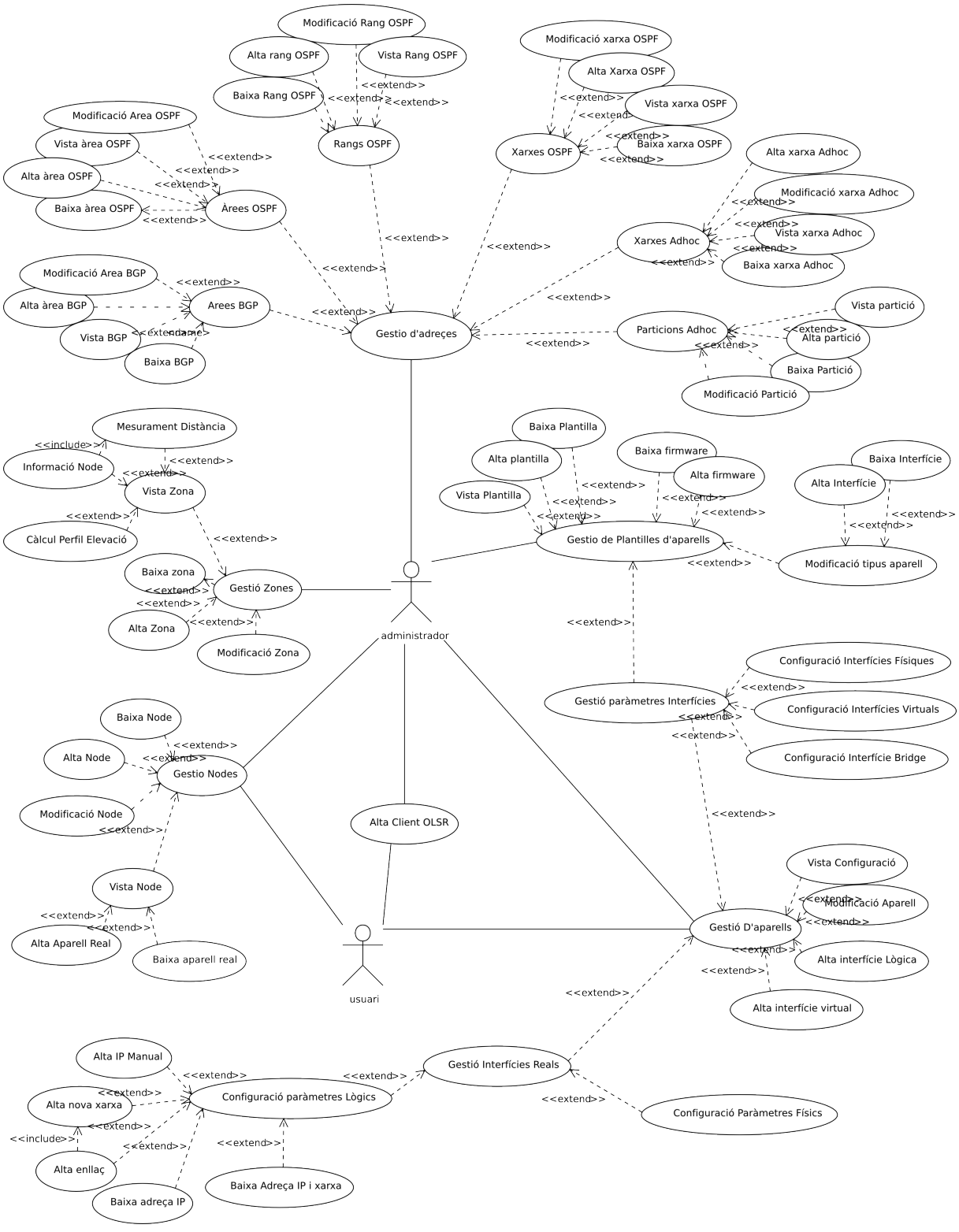


Figura 3.1: Model dinàmic. Diagrama de casos dús

- **Modificació d'una zona**

  Descripció: Editar una zona existent i guardar els canvis entrats.

  Pre-condicions: Ha d'existir la zona.

  Flux Principal:

  1. Llegir coordenades i zoom de la zona, i centrar-hi el mapa.
  2. Validar les dades modificades.
  3. Guardar la zona.

  Fluxos Alternatius:

  1. Si la zona no te coordenades posades, centrarem el mapa amb les coordenades de la zona pare, i sinó a un valor per defecte.

  Post-condicions: Si la validació és correcte, canvis a la zona guardats, sinó retornar error referent al camp incorrecte.

- **Vista de Zona**

  Descripció: Es mostra el mapa i les dades d'una zona.

  Pre-condicions: Ha d'existir la zona.

  Flux Principal:

  1. Es mostra el mapa centrat a les coordenades i zoom de la zona.
  2. Es dibuixen tots els nodes que hi han dins les coordenades visibles del mapa, encara que siguin d'altres zones.
  3. Es dibuixen tots els enllaços entre aparells, com a enllaços entre nodes.
  4. Es mostra una llista de les zones filles del nivell immediatament inferior.
  5. Es mostra una llista dels nodes d'aquella zona.

  Fluxos Alternatius:

  1. Si la zona no tingués coordenades assignades, s'agafaran valors per defecte.

  Post-condicions: Mapa de la zona amb les dades associades representades.

- **Baixa de Zona**

  Descripció: S'esborra una zona.

  Pre-condicions: Ha d'existir la zona.

  Flux Principal:

  1. Comprovar si hi han nodes o altres zones dins la zona.
  2. Esborrar la zona.

  Post-condicions: Si no hi han elements dins la zona, zona esborrada.

- **Informació node**

  Descripció: Es mostra informació d'un node al clicar-lo.

  Pre-condicions: Ha d'existir la zona amb com a mínim un node.

  Flux Principal:

1. Es calcula la distància entre un marcador mòbil i el node.
2. Es recullen les dades del node.
3. Es mostren les dades del node en una finestra emergent.
4. Es dibuixa una línia entre el node i el marcador mòbil.

Post-condicions: Finestra emergent amb les dades del node, i la distància al marcador.

- **Mesurament de distància**

  Descripció: Es calcula la distància entre dos punts del mapa cada cop que es mogui un marcador mòbil.

  Pre-condicions: Ha d'existir una zona .

  Flux Principal:

  1. Es llegeix les coordenades dels dos marcadors.
  2. Es calcula la distància entre els marcadors.
  3. Es dibuixa una línia entre els marcadors.
  4. Es mostra la distància entre es marcadors.

  Post-condicions: Línia dibuixada entre els dos marcadors i distància entre ells.

- **Càlcul perfil d'elevació**

  Descripció: Es mostra el perfil d'elevació entre dos punts del mapa al clicar un marcador mòbil.

  Pre-condicions: Ha d'existir una zona .

  Flux Principal:

  1. Es llegeix les coordenades dels dos marcadors.
  2. Es calcula la distància entre els marcadors.
  3. Es demana al servidor una imatge amb el perfil d'elevació.
  4. Es mostra la imatge i la distància.

  Post-condicions: Finestra emergent amb el perfil d'elevació i la distància entre els dos punts.

- **Alta Node**

  Descripció: Afegir un node dins una zona.

  Pre-condicions: Ha d'existir una zona on crear el node.

  Flux Principal:

  1. Buscar coordenades de la zona pare i centrar-hi el mapa.
  2. Validar que no hi hagi cap node amb el mateix nom a la mateixa zona.
  3. Validar que no hi hagin caràcters no permesos al nom curt del node.
  4. Guardar el node.

  Fluxos Alternatius:

  1. Si s'ha creat el node directament del mapa de la zona: centrar i marcar el node al mapa i mostrar les coordenades.

Post-condicions: Si la validació és correcte, node nou creat, sinó retornar error referent al camp incorrecte.

- **Modificació Node**

  Descripció: Editar un node existent i guardar els canvis entrats.
  Pre-condicions: Ha d'existir el node.
  Flux Principal:

  1. Llegir coordenades del node, centrar-hi el mapa i marcar-hi un punt.
  2. Validar les dades modificades.

  Post-condicions: Si la validació és correcte, canvis al node guardats, sinó retornar error referent al camp incorrecte.

- **Vista de Node**

  Descripció: Es mostra el node i les seves dades.
  Pre-condicions: Ha d'existir el node.
  Flux Principal: Es mostren les dades del node, així com una llista dels aparells i enllaços de cada tipus que hi ha entre cada aparell i aparells d'altres nodes.

  1. Es mostren les dades del node.
  2. Es mostra una llista dels aparells d'aquell node, amb l'adreça principal de cadascun i el tipus d'aparell.
  3. Es mostra una llista dels enllaços de cada aparell del node a aparells d'altres nodes.
  4. Es mostra una porció del mapa de la zona amb les coordenades del node marcades.

  Post-condicions: Pàgina amb totes les dades referents a un node.

- **Baixa de node**

  Descripció: S'esborra un node.
  Pre-condicions: Ha d'existir el node.
  Flux Principal:

  1. Comprovar si hi han aparells a dins del node.
  2. Esborrar el node.

  Post-condicions: Si no hi han aparells dins el node, node esborrat.

- **Alta aparell real**

  Descripció: Crear un aparell nou en un node.
  Pre-condicions: Ha d'existir un node i ha de ser propietat de l'usuari.
  Flux Principal: entre interfícies

  1. Es fa una copia al node de l'aparell contenidor de la plantilla d'aparell escollida, amb el nom introduït per l'usuari.
  2. Es copien les referències entre interfícies de l'aparell contenidor de la plantilla al nou aparell.

3. Es crea una referència entre l'aparell i la plantilla d'aparell.

Fluxos Alternatius:

1. Si el nom introduït per l'usuari ja pertany a un aparell del mateix node, s'afegira un sufix numèric al nom.
2. Si l'usuari no ha entrat un nom per l'aparell, es generà un nom compost del nom curt del node i un sufix numèric.

Post-condicions: Aparell nou creat, amb totes les interfícies clonades de la plantilla d'aparell escollida.

- **Modificació Aparell**

  Descripció: Modificar les dades d'un aparell existent.
  Pre-condicions: Ha d'existir l'aparell.
  Flux Principal:

  1. Es validen les dades modificades.
  2. S'activen les interfícies seleccionades.
  3. Es guarden els canvis a l'aparell.

  Post-condicions: Si la validació és correcte, canvis a l'aparell guardats, sinó retornar error referent al camp incorrecte.

- **Baixa Aparell**

  Descripció: Esborrar un aparell i les seves interfícies.
  Pre-condicions: Ha d'existir l'aparell.
  Flux Principal:

  1. Es comprova que l'aparell no tingui cap interfície associada a una ip.
  2. S'esborra la referència a la plantilla d'aparell.
  3. S'esborren totes les interfícies de l'aparell, i l'aparell en si.

  Post-condicions: Si les interfícies de l'aparell no tenen cap dependència, aparell esborrat.

- **Alta interfície lògica**

  Descripció: Afegir una interfície lògica a un aparell.
  Pre-condicions: Ha d'existir l'aparell. Si l'interfície lògica és un bridge, l'aparell ha de tenir com a mínim una interfície física .
  Flux Principal:

  1. Es genera el nom de l'interfície en base a el nom de l'aparell + nom del tipus d'interfície + sufix numèric.
  2. Es crea l'interfície segons el tipus escollit.

  Post-condicions: Interfície lògica nova creada, sense configurar.

- **Alta interfície virtual**

  Descripció: Afegir una interfície lògica a una interfície física d'un aparell.

  Pre-condicions: Ha d'existir l'aparell, i ha de tenir com a mínim una interfície física que permeti virtuals, i que hi hagi lloc per una altra virtual .

  Flux Principal:

  1. Es genera el nom de l'interfície en base a el nom de l'aparell + nom del tipus d'interfície + sufix numèric.
  2. Es crea l'interfície segons el tipus escollit.
  3. Es crea una referència entre l'interfície virtual i la master.

  Post-condicions: Interfície virtual nova creada, sense configurar.

- **Vista Configuració**

  Descripció: Es genera la configuració necessària per configurar un aparell.

  Pre-condicions: Ha d'existir un aparell .

  Flux Principal: A partir de les dades generades al crear un enllaç, es genera la configuració necessària per configurar l'aparell triat, i es mostra per pantalla.

  1. Es comprova que l'aparell i les seves interfícies tinguin el mínim de paràmetres entrats per poder generar la configuració.
  2. Es genera la configuració en funció de l'aparell, les interfícies i firmware que té, i la xarxa o xarxes de les que forma part.
  3. Es mostra la configuració.

  Post-condicions: Si l'aparell esta preparat per configurar-se, configuració generada.

- **Configuració paràmetres físics**

  Descripció: Es configuren els paràmetres no referents a adreçament d'una interfície.

  Pre-condicions: Ha d'existir l'interfície .

  Flux Principal:

  1. Es validen les dades entrades.
  2. Es guarda els canvis a l'interfície.

  Fluxos Alternatius:

  1. Si algun dels camps de l'interfície esta en funció d'un altre, es llegeixen els valors necessaris del servidor, i s'actualitza el formulari quan calgui.

  Post-condicions: Si la validació és correcte, paràmetres físics de l'interfície guardats, sinó retornar error referent al camp incorrecte.

- **Alta IP Manual**

  Descripció: L'usuari vol introduir una adreça ip manualment.

  Pre-condicions: Ha d'existir una interfície i aquesta ha de ser configurable .

  Flux Principal:

1. Es valida el format de la ip.
2. Es busca la xarxa associada a la ip/màscara.
3. Es crea la ip dins la xarxa.
4. Es crea una referència entre la ip i l'interfície.

Fluxos Alternatius:

1. Si la xarxa no existeix, es busca si hi ha un rang existent per a la xarxa.
2. Si existeix un rang per la xarxa, es crea la xarxa i es continua normalment.

Post-condicions: Si existeix un rang i xarxa possibles per la ip, i la ip és vàlida, ip creada i referenciada a l'interfície, sinó retornar error explicant el motiu.

- **Alta nova xarxa**

  Descripció: Es crea una nova xarxa per disposar-ne de les adreces.
  Pre-condicions: Ha d'existir l'interfície que volem configurar, com a mínim una àrea BGP i una àrea ospf amb rangs o bé una xarxa adhoc .
  Flux Principal:

  1. Es tria la xarxa BGP i es demana al servidor la llista de àrees OSPF i xarxes OLSR d'aquella àrea BGP.
  2. Es tria o bé una àrea OSPF o bé una xarxa OLSR.
  3. Es tria una mida i tipus de rang en el cas de l'àrea OSPF, o un tipus de partició en el cas de les xarxes OLSR.
  4. Es demana al servidor una xarxa i una adreça que compleixi amb les dades entrades.
  5. Es crea la xarxa nova.
  6. Es crea una adreça nova dins la xarxa.
  7. Es crea una referència entre l'adreça nova i l¡interfície.

  Fluxos Alternatius:

  1. Si el rang o partició que escollim no té lloc per crear-ne més xarxes, no es mostrarà l'opció d'assignar una ip.
  2. Si la xarxa OLSR que escollim no te cap partició lliure, crearem una partició nova del tipus indicat si hi ha lloc.

  Post-condicions: Si existeix un espai lliure per crear la xarxa en funció de les opcions escollides, xarxa creada amb una ip assignada a l'interfície.

- **Alta Enllaç**

  Descripció: Es crea un enllaç entre les interfícies seleccionades.
  Pre-condicions: Els dos aparells a enllaçar han d'existir i han de tenir com a mínim una interfície física. Aquesta ha de ser configurable, tenir-ne de virtuals configurables, o amb l'opció de crear-ne virtuals noves a sobre.
  Flux Principal:

1. Es tria un aparell remot de la llista, i es demana al servidor una llista de les interfícies físiques disponibles per enllaçar.
2. Es tria una interfície remota de la llista, i es demana al servidor una llista de les opcions possibles per a fer l'enllaç.
3. Es tria una opció de la llista, i es demana al servidor la ip o ip's necessàries per configurar l'enllaç.
4. Es crea la ip.
5. Es crear una referència entre l'adreça nova i l'interfície.

Fluxos Alternatius:

1. Si no hi han opcions d'enllaç, no es mostrarà la opció d'assignar una adreça per finalitzar l'enllaç.
2. Si la opció d'enllaç es amb una interfície virtual remota i aquesta no existeix, es crearà la virtual abans d'assignar les ips.
3. Si l'interfície remota no te ip ni xarxa, es mostrà el formulari d'alta de xarxes per crear-ne una de nova.
4. Si al configurar la ip de l'interfície la xarxa no existeix, primer es crearà la xarxa i després es crearan 2 ips i les seves referències a les interfícies remota i local. .

Post-condicions: .
Si existeix una opció vàlida per enllaçar les dues interfícies , ip/s creades i assignades a les interfície/s .

- **Baixa Adreça IP**
  Descripció: S'esborra l'adreça assignada a l'interfície.
  Pre-condicions: Ha d'existir una interfície amb una adreça IP associada.
  Flux Principal:

  1. Es comprova que no hi hagin interfícies remotes que depenguin de la ip de l'interfície.
  2. S'esborra la referència entre la ip i l'interfície.
  3. S'esborra la ip.

  Post-condicions: Si la ip no tenia dependències amb altres interfícies, ip i referència cap a l'interfície esborrades, sinó error explicant el motiu.

- **Baixa adreça IP i xarxa** *Descripció:* S'esborra l'adreça assignada a l'interfície i la xarxa a la que pertany.
  Pre-condicions: Ha d'existir una interfície amb una adreça IP associada.
  Flux Principal:

  1. Es comprova que la ip sigui la única de la xarxa.
  2. S'esborra la referència entre la ip i l'interfície.
  3. S'esborra la ip.

4. S'esborra la xarxa.

Post-condicions: Si la ip era la única de la xarxa, ip, referència cap a l'interfície i xarxa esborrades.

- **Alta firmware**

  Descripció: Es crea un firmware nou.
  Pre-condicions: No n'hi han .
  Flux Principal:

  1. Es comprova que no hi hagi cap més firmware amb el mateix nom.
  2. Es guarden les dades del nou firmware.

  Post-condicions: Si el nom donat no existeix, firmware creat, sinó mostrar error.

- **Baixa firmware**

  Descripció: S'esborra un firmware.
  Pre-condicions: ha d'existir el firmware .
  Flux Principal:

  1. Es comprova que no hi hagi cap plantilla d'aparell que utilitzi aquell firmware.
  2. S'esborra el firmware.

  Post-condicions: Si el firmware no té cap dependència, firmware esborrat.

- **Alta plantilla d'aparell**

  Descripció: Es crea una plantilla d'aparell nova.
  Pre-condicions: Ha d'existir com a mínim un firmware .
  Flux Principal:

  1. Es comprova que no hi hagi cap més plantilla amb el mateix nom.
  2. Es crea una plantilla nova amb el nom donat.
  3. Es crea una referència entre la plantilla i el firmware.
  4. Es crea un aparell nou contenidor a dins de la plantilla.

  Post-condicions: Si el nom donat no existeix, plantilla d'aparell amb aparell contenidor creats, sinó mostrar error.

- **Alta interfície**

  Descripció: Es crea una interfície nova en l'aparell contenidor d'una plantilla d'aparell.
  Pre-condicions: Ha d'existir una plantilla d'aparell creada amb un aparell contenidor buit.
  Flux Principal:

  1. Es comprova que no hi hagi una altra interfície amb el mateix nom dins l'aparell contenidor de la plantilla.
  2. Es crea l'interfície amb el nom i el tipus escollit.

  Post-condicions: Si no hi havia cap interfície amb el mateix nom, interfície nova creada, sinó mostrar error.

- **Baixa Plantilla d'aparell**

  Descripció: S'esborra una plantilla d'aparell.

  Pre-condicions: Ha d'existir la plantilla d'aparell.

  Flux Principal:

  1. Comprovar que no hi hagi cap aparell real associat a aquesta plantilla.
  2. Esborrar la referència al firmware.
  3. Esborrar recursivament totes les interfícies de l'aparell contenidor, aquest inclòs.

  Post-condicions: Si la plantilla no té cap dependència, plantilla d'aparell esborrada.

- **Vista Plantilla d'aparell**

  Descripció: Es mostra una plantilla d'aparell.

  Pre-condicions: Ha d'existir la plantilla d'aparell.

  Flux Principal:

  1. Mostrar dades de la plantilla.
  2. Mostrar llista de les interfícies físiques de l'aparell contenidor i els seus tipus.
  3. Mostrar llista de les interfícies lògiques de l'aparell contenidor, els seus tipus, i la interfície/s a les que estan associades.
  4. Mostrar llista de les interfícies virtuals de l'aparell contenidor, els seus tipus, i quina és la interfície master.

  Post-condicions: Pàgina amb totes les dades referents a una plantilla d'aparell.

- **Configuració Interfícies Físiques**

  Descripció: Es configura els paràmetres d'una interfície física.

  Pre-condicions: Ha d'existir l'interfície i ha de ser física .

  Flux Principal:

  1. Es validen les dades entrades.
  2. Es guarden a l'interfície.

  Post-condicions: Si les validacions són correctes, interfície física configurada, sinó mostrar error associat al camp.

- **Configuració Interfície Bridge**

  Descripció: Es configura els paràmetres d'una interfície física.

  Pre-condicions: Ha d'existir l'interfície i ha de ser un bridge .

  Flux Principal:

  1. Es mostra una llista de les interfícies físiques que no formen part de cap bridge.
  2. Es guarda una referència del bridge a cada una de les interfícies seleccionades.

  Post-condicions: Interfícies assignades al bridge.

- **Configuració Interfícies Virtuals**

  Descripció: Es configura els paràmetres d'una interfície virtual.

  Pre-condicions: Ha d'existir l'interfície i ha de ser virtual, les interfícies físiques existents a la plantilla han d'estar configurades .

  Flux Principal:

  1. Es mostra una llista de les interfícies físiques que poden acceptar virtuals d'aquell tipus.
  2. Es guarda una referència de la interfície a la interfície física *master* seleccionada.

  Post-condicions: Interfície master assignada a la virtual.

- **Alta client OLSR**

  Descripció: Es crea un aparell i un enllaç d'aquest a una xarxa OLSR.

  Pre-condicions: Ha d'existir un node i una xarxa OLSR amb un rang assignat, i com a mínim una plantilla d'aparell vàlida per una xarxa OLSR .

  Flux Principal:

  1. Comprovar si hi ha lloc per una adreça IP.
  2. Es fa una copia al node de l'aparell contenidor de la plantilla d'aparell escollida, amb el nom generat en base al nom curt del node.
  3. Es copien les referències entre interfícies de l'aparell contenidor de la plantilla al nou aparell.
  4. Es crea una adreça ip en una partició de tipus bàsica, que coincideixi amb el valor de dhcp triat per l'usuari.
  5. Es busca la primera interfície wireless de l'aparell creat.
  6. Es crea una referència entre l'adreça ip creada i l'interfície wireless trobada

  Fluxos Alternatius:

  1. Si no existeix cap partició bàsica amb adreces lliures , o no n'hi ha cap que coincideixi amb el valor triat per l'usuari, la crearem si hi ha lloc.

  Post-condicions: Si hem pogut trobar o crear una partició bàsica adient, aparell creat, ip creada i referenciada a la interfície wireless de l'aparell.

- **Alta àrea BGP**

  Descripció: Es dona d'alta una nova àrea BGP.

  Pre-condicions: No n'hi han.

  Flux Principal:

  1. Es busca un identificador AS per l'àrea no usat i s'ofereix a l'usuari.
  2. Es comprova l'identificador AS per si s'ha modificat.
  3. Es comprova que el rang no se solapi amb d'altres rangs BGP.
  4. Es guarda l'àrea BGP.

  Post-condicions: Si l'AS i el rang són vàlids, àrea BGP creada, sinó retornar error.

- **Modificació àrea BGP**

  Descripció: Es modifiquen les dades d'una àrea BGP.

  Pre-condicions: Ha d'existir l'àrea BGP.

  Flux Principal:

  1. Es comprova que el rang modificat continua sent un rang que englobi totes les subxarxes possibles que hi hagin per sota el BGP.
  2. Es comprova que el rang modificat no se solapi amb d'altres rangs BGP.
  3. Es guarda l'àrea BGP.

  Post-condicions: Si el rang són vàlids, àrea BGP modificada, sinó retornar error.

- **Vista àrea BGP**

  Descripció: Es mostren les dades d'una àrea BGP.

  Pre-condicions: Ha d'existir l'àrea BGP.

  Flux Principal:

  1. Es mostren l'AS i els rangs de l'àrea.
  2. Es llisten totes les àrees OSPF que hi hagin dins l'àrea BGP.

  Post-condicions: Pàgina amb les dades associades a una àrea BGP.

- **Baixa Àrea BGP**

  Descripció: S'esborra una àrea BGP i tots els seus fills.

  Pre-condicions: Ha d'existir l'àrea BGP.

  Flux Principal:

  1. Es comprova que no hi hagi cap element fill dins l'àrea BGP.
  2. S'esborrarà totes les referències a interfícies que puguin tenir les adreces ip de l'extrem de la jerarquia.
  3. S'esborraran recursivament tot els objectes per sota de l'àrea BGP, inclosa.

  Fluxos Alternatius:

  1. Si hi han àrees OSPF dins l'àrea BGP, s'avisarà que de continuar, s'esborraran juntament amb tot el que continguin.

  Post-condicions: Àrea BGP i fills esborrats.

- **Alta àrea OSPF**

  Descripció: Es dona d'alta una nova àrea OSPF.

  Pre-condicions: Ha d'existir una àrea BGP.

  Flux Principal:

  1. Es busca un identificador per l'àrea no usat i s'ofereix a l'usuari.
  2. Es comprova l'identificador per si s'ha modificat.
  3. Es guarda l'àrea OSPF.

Post-condicions: Si l'identificador és vàlid, àrea OSPF creada, sinó retornar error.

- **Modificació àrea OSPF**

  Descripció: Es modifiquen les dades d'una àrea OSPF.

  Pre-condicions: Ha d'existir l'àrea OSPF.

  Flux Principal:

  1. Es comprova l'identificador d'àrea nou.
  2. Es guarda l'àrea OSPF.

  Post-condicions: Si l'identificador és vàlid, àrea OSPF modificada, sinó retornar error.

- **Vista àrea OSPF**

  Descripció: Es mostren les dades d'una àrea OSPF.

  Pre-condicions: Ha d'existir l'àrea OSPF.

  Flux Principal:

  1. Es mostren l'identificador de l'àrea.
  2. Es llisten totes els rangs OSPF que hi han dins l'àrea.

  Post-condicions: Pàgina amb les dades associades a una àrea OSPF.

- **Baixa Àrea OSPF**

  Descripció: S'esborra una àrea OSPF i tots els seus fills.

  Pre-condicions: Ha d'existir l'àrea BGP.

  Flux Principal:

  1. Es comprova que no hi hagi cap element fill dins l'àrea BGP.
  2. S'esborrarà totes les referències a interfícies que puguin tenir les adreces ip de l'extrem de la jerarquia.
  3. S'esborraran recursivament tot els objectes per sota de l'àrea OSPF, inclosa.

  Fluxos Alternatius:

  1. Si hi han Rangs OSPF dins l'àrea OSPF, s'avisarà que de continuar, s'esborraran juntament amb tot el que continguin.

  Post-condicions: Àrea OSPF i fills esborrats.

- **Alta rang OSPF**

  Descripció: Es dona d'alta un nou rang OSPF.

  Pre-condicions: Ha d'existir una àrea OSPF.

  Flux Principal:

  1. Es comprova que el rang no se solapi amb d'altres rangs BGP i que sigui subrang de l'àrea BGP.
  2. Es guarda el rang OSPF.

Post-condicions: Si el rang és vàlid, rang OSPF creat, sinó retornar error.

- **Modificació rang OSPF**

  Descripció: Es modifiquen les dades d'un rang OSPF.

  Pre-condicions: Ha d'existir el rang OSPF.

  Flux Principal:

  1. Es comprova que el rang modificat continua sent un rang que englobi totes les subxarxes possibles que hi hagin per sota el rang OSPF actual.
  2. Es comprova que el rang modificat no se solapi amb d'altres rangs OSPF i que sigui subrang de l'àrea BGP.
  3. Es guarda el rang OSPF.

  Post-condicions: Si el rang és vàlid, rang OSPF modificat, sinó retornar error.

- **Vista rang OSPF**

  Descripció: Es mostren les dades d'un rang OSPF.

  Pre-condicions: Ha d'existir el rang OSPF.

  Flux Principal:

  1. Es mostra el tipus i el rang del rang OSPF.
  2. Es llisten totes els xarxes OSPF que hi han dins el rang.

  Post-condicions: Pàgina amb les dades associades a un rang OSPF.

- **Baixa rang OSPF**

  Descripció: S'esborra una rang OSPF i tots els seus fills.

  Pre-condicions: Ha d'existir el rang OSPF.

  Flux Principal:

  1. Es comprova que no hi hagi cap element fill dins el rang OSPF.
  2. S'esborrarà totes les referències a interfícies que puguin tenir les adreces ip de l'extrem de la jerarquia.
  3. S'esborraran recursivament tot els objectes per sota del rang OSPF, inclòs.

  Fluxos Alternatius:

  1. Si hi han xarxes OSPF dins del rang OSPF, s'avisarà que de continuar, s'esborraran juntament amb tot el que continguin.

  Post-condicions: rang OSPF i fills esborrats.

- **Alta xarxa OSPF**

  Descripció: Es dona d'alta una nova xarxa OSPF.

  Pre-condicions: Ha d'existir un rang OSPF.

  Flux Principal:

1. Es comprova que el rang no se solapi amb d'altres rangs BGP i que sigui subrang del rang OSPF.
2. Es guarda la xarxa OSPF.

Post-condicions: Si el rang és vàlid, xarxa OSPF creada, sinó retornar error.

- **Modificació xarxa OSPF**

  Descripció: Es modifiquen les dades d'un xarxa OSPF.

  Pre-condicions: Ha d'existir la xarxa OSPF.

  Flux Principal:

  1. Es comprova que el rang modificat continua sent un rang que englobi totes les adreces ips creades dins la xarxa OSPF actual.
  2. Es comprova que el rang modificat no se solapi amb d'altres xarxes OSPF i sigui subrang del rang OSPF.
  3. Es guarda la xarxa OSPF.

  Post-condicions: Si el rang és vàlid, xarxa OSPF modificada, sinó retornar error.

- **Vista xarxa OSPF**

  Descripció: Es mostren les dades d'un xarxa OSPF.

  Pre-condicions: Ha d'existir la xarxa OSPF.

  Flux Principal:

  1. Es mostra el rang de la xarxa OSPF.
  2. Es llisten totes les adreces IP que hi han dins la xarxa OSPF.

  Post-condicions: Pàgina amb les dades associades a un xarxa OSPF.

- **Baixa xarxa OSPF**

  Descripció: S'esborra una xarxa OSPF i totes les seves adreces.

  Pre-condicions: Ha d'existir la xarxa OSPF.

  Flux Principal:

  1. Es comprova que no hi hagi cap adreça dins la xarxa OSPF.
  2. S'esborrarà totes les referències a interfícies que puguin tenir les adreces ip de la xarxa OSPF.
  3. S'esborraran recursivament tot els objectes per sota de la xarxa OSPF, inclosa .

  Fluxos Alternatius:

  1. Si hi han adreces IP dins de la xarxa OSPF, s'avisarà que de continuar, s'esborraran.

  Post-condicions: xarxa OSPF i adreces esborrades.

- **Alta xarxa Adhoc**

  Descripció: Es dona d'alta una nova xarxa Adhoc.

  Pre-condicions: Ha d'existir un rang BGP.

  Flux Principal:

1. Es comprova que el rang no se solapi amb d'altres xarxes Adhoc o rangs OSPF i que sigui subrang del rang BGP.
2. Es guarda la xarxa Adhoc.

Post-condicions: Si el rang és vàlid, xarxa Adhoc creada, sinó retornar error.

- **Modificació xarxa Adhoc**

  Descripció: Es modifiquen les dades d'un xarxa Adhoc.

  Pre-condicions: Ha d'existir la xarxa Adhoc.

  Flux Principal:

  1. Es comprova que el rang modificat continua sent un rang que englobi totes les particions que hi hagin dins la xarxa Adhoc.
  2. Es comprova que el rang modificat no se solapi amb d'altres xarxes Adhoc o rangs OSPF i sigui subrang del rang BGP.
  3. Es guarda la xarxa Adhoc.

  Post-condicions: Si el rang és vàlid, xarxa Adhoc modificada, sinó retornar error.

- **Vista xarxa Adhoc**

  Descripció: Es mostren les dades d'un xarxa Adhoc.

  Pre-condicions: Ha d'existir la xarxa Adhoc.

  Flux Principal:

  1. Es mostra el rang de la xarxa Adhoc.
  2. Es llisten totes les particions dins de la xarxa Adhoc agrupades per tipus.

  Post-condicions: Pàgina amb les dades associades a un xarxa Adhoc.

- **Baixa xarxa Adhoc**

  Descripció: S'esborra una xarxa Adhoc i totes les seves adreces.

  Pre-condicions: Ha d'existir la xarxa Adhoc.

  Flux Principal:

  1. Es comprova que no hi hagi cap partició dins la xarxa Adhoc.
  2. S'esborrarà totes les referències a interfícies que puguin tenir les adreces ip de l'extrem de la jerarquia.
  3. S'esborraran recursivament tot els objectes per sota de la xarxa Adhoc, inclòsa .

  Fluxos Alternatius:

  1. Si hi han particions dins de la xarxa Adhoc, s'avisarà que de continuar, s'esborraran.

  Post-condicions: xarxa Adhoc i fills esborrats.

- **Alta partició Adhoc**

  Descripció: Es dona d'alta una nova partició Adhoc.

  Pre-condicions: Ha d'existir una xarxa Adhoc.

  Flux Principal:

1. Es comprova que el rang no se solapi amb d'altres particións ni xarxes OSPF i que sigui subrang de la xarxa Adhoc.
2. Es guarda la partició Adhoc.

Post-condicions: Si el rang és vàlid, partició Adhoc creada, sinó retornar error.

- **Modificació partició Adhoc**

  Descripció: Es modifiquen les dades d'un partició Adhoc.
  Pre-condicions: Ha d'existir la partició Adhoc.
  Flux Principal:

  1. Es comprova que el rang modificat continua sent un rang que englobi totes les adreces ips creades dins la partició Adhoc actual.
  2. Es comprova que el rang modificat no se solapi amb d'altres particions ni xarxes OSPF i sigui subrang de la xarxa Adhoc.
  3. Es guarda la partició Adhoc.

  Post-condicions: Si el rang és vàlid, partició Adhoc modificada, sinó retornar error.

- **Vista partició Adhoc**

  Descripció: Es mostren les dades d'un partició Adhoc.
  Pre-condicions: Ha d'existir la partició Adhoc.
  Flux Principal:

  1. Es mostra el rang de la partició Adhoc.
  2. Es llisten totes les adreces IP que hi han dins la partició Adhoc.

  Post-condicions: Pàgina amb les dades associades a un partició Adhoc.

- **Baixa partició Adhoc**

  Descripció: S'esborra una partició Adhoc i totes les seves adreces.
  Pre-condicions: Ha d'existir la partició Adhoc.
  Flux Principal:

  1. Es comprova que no hi hagi cap adreça dins la partició Adhoc.
  2. S'esborrarà totes les referències a interfícies que puguin tenir les adreces ip de la partició Adhoc.
  3. S'esborraran recursivament tot els objectes per sota de la partició Adhoc, inclosa .

  Fluxos Alternatius:

  1. Si hi han adreces IP dins de la partició Adhoc, s'avisarà que de continuar, s'esborraran.

  Post-condicions: partició Adhoc i adreces esborrades.

3.2 Model Estàtic

Hi ha un aspecte de l'especificació en que s'ha reflexat directament sobre el model, que és la diferenciació entre les dues part de la xarxa, la part física del sistema, des de les zones a les interfícies de xarxa, i la part lògica, englobant totes les classes necessàries per representar els diferents protocols d'enrutament i adreces. El nexe d'unió entre aquestes dues parts, serà una única relació 1..1 entre la classe interfície de xarxa i la classe adreça ip. Amb aquesta relació es podran deduir i calcular certes dades, que d'haver creat relacions extres haguessin resultat redundants. Pel mètode de desenvolupament triat, també ens interessa mantenir les associacions entre objectes al mínim, i intentar modelar tot el que sigui possible en base a agregacions o composicions.

A l'especificació es parlava de donar d'alta aparells nous de forma genèrica, i també de definir qualsevol aparell nou i la seva estructura interna, sense haver de modificar el codi existent. Mirem això amb detall, ja que quan parlem d'un aparell nou, ens podem referir a :

- Un aparell que donem d'alta i configurem els seus paràmetres per tal de aconseguir-ne la configuració.
- Un *tipus* d'aparell que no s'ha utilitzat mai a la xarxa, que pot ser una combinació d'aparell i firmware nous, o només un nou firmware amb un tipus d'aparell existent, i per tant no existeix com a tal, i tampoc existeix el codi necessari per configurar-lo.

Mirant la solució actual de guifi.net, ambdós casos tenen certs problemes en l'actualitat: El primer cas, implica que cada vegada que volem donar d'alta un aparell, n'hem de donar d'alta també manualment les interfícies, essent nosaltres directament que controlem la validesa de els elements que estem donant d'alta. Per altra banda, davant d'un tipus nou d'aparell, hi han diversos elements que s'hauran de donar d'alta a la base de dades, modificar les interfícies d'usuari i programar el codi corresponent per generar la configuració per aquella combinació d'aparell i firmware. Actualment apart d'aquestes modificacions, totes les configuracions automàtiques les genera una sola funció, carregada de condicionals, per tenir en compte totes les possibilitats. A mesura que es van afegint nous elements, s'està tornant molt difícil de mantenir.

Hem dissenyat doncs una part del model, que dona sortida a aquestes dues situacions amb un sol concepte nou, *plantilles d'aparells* tenint en compte les següents premisses:

- Una plantilla d'aparell, serà una combinació d'un aparell fictici, i un firmware. Aparell serà representat per la mateixa classe que un aparell real, i per tant tindra els mateixos atributs, mètodes i classes agregades.
- A aquest aparell fictici se li afegiran totes les possibles interfícies físiques que pot tenir, així com s'indicara quines interfícies virtuals es poden crear sobre de les físiques. Això servira de plantilla cada vegada que s'hagi de crear un aparell real d'aquell tipus, ja que només n'haurem de fer una copia. D'aquesta manera, també s'evita el poder definir aparells irreals; la responsabilitat recau en crear les plantilles d'aparells correctament
- Tot el codi referent a la generació de configuracions, serà encapsulat en les classes referents als firmwares, ja que les diferències més grans entre una configuració i una altra, estan en funció de

en quin firmware s'han de posar. Les variacions en la configuració relatives a l'aparell, es poden generalitzar en la majoria de casos, ja que simplement depenen de la quantitat de interfícies de xarxa que tingui un aparell concret.

- Crear un model per les plantilles d'aparells que pretengui no haver d'afegir codi quan hi hagin modificacions és impossible, ja que al introduir un firmware nou, necessàriament s'haurà d'escriure el codi que generi les configuracions. De totes maneres, a nivell de les interfícies i del comportament que se n'espera. S'ha dissenyat el model de manera que es puguin afegir nous tipus d'interfícies de xarxa i configurar-ne el seu comportament gràficament, afegint elements nous al diagrama i relacionat-los amb les interfícies UML marcadores genèriques que s'han afegit. Amb les interfícies marcadores, podrem tractar les classes segons els interfícies que implementin, en comptes de segons de quina classe siguin.

NOTA de nomenclatura : Com es pot veure en el diagrama UML següent i en posteriors, tots els noms de classes atributs i mètodes, estan escrits en anglès. Algunes raons que justifiquen aquest fet:

- Hi han molts dels conceptes representat que tenen el seu origen en mots anglesos.
- Al intentar escriure certs noms en català, al no poder fer ús d'accents, hi ha noms que poden semblar ambigus.
- Per estètica i comoditat, alhora de programar, és agraït no veure diferents idiomes barrejats. Ja que la sintaxis del llenguatge és propera a l'anglès, es fa servir aquest per anomenar els diferents elements.
- Pensant en el futur de l'aplicació, si es volgués posar a disposició de tercers, els elements de codi en anglès permeten que un public més ampli entengui el producte.

De totes maneres en la descripció del diagrama de classes, hi ha una relació dels noms de les classes amb el seu equivalent en català.

3.2.1 Diagrama de classes

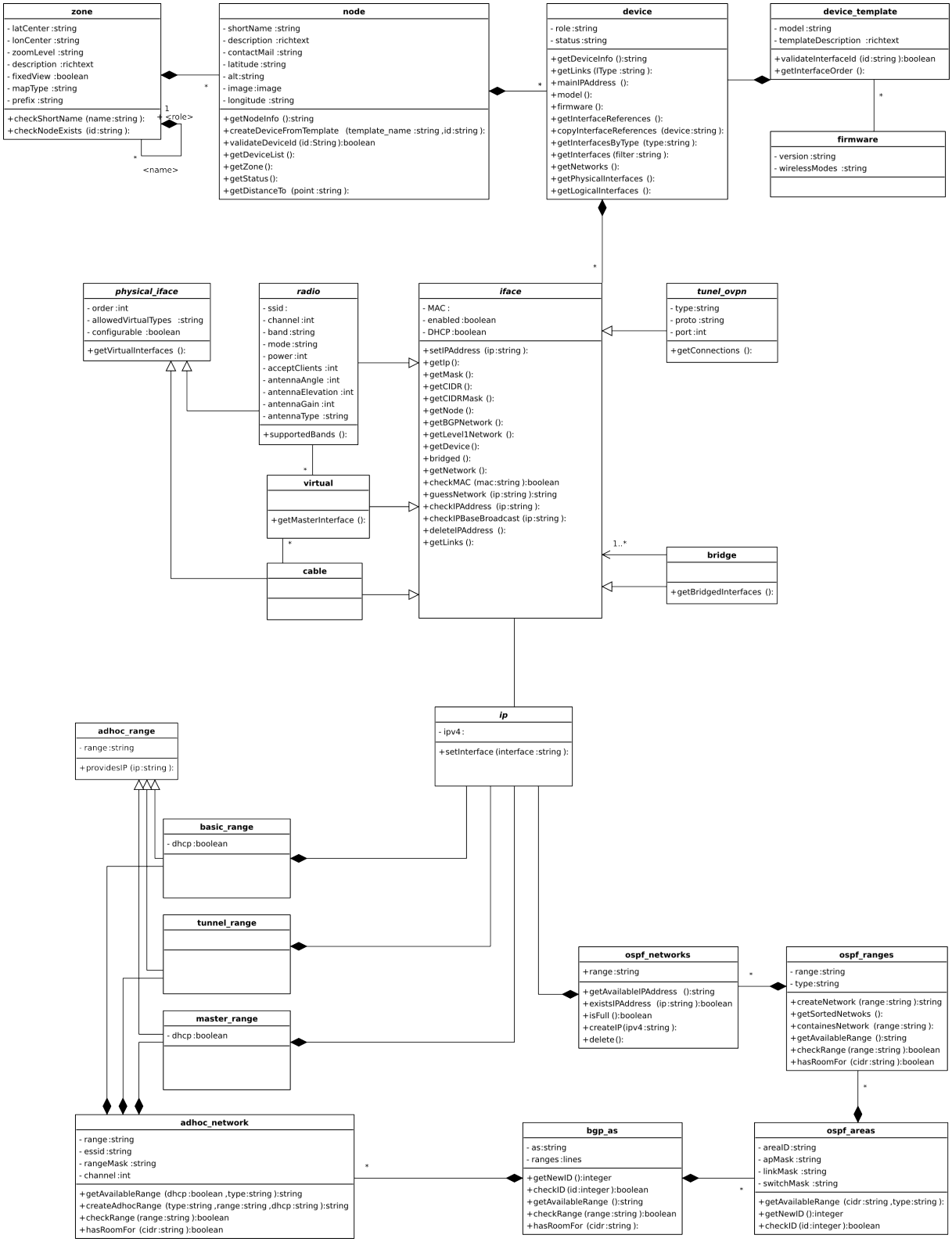


Figura 3.2: Model estàtic. Diagrama de classes

3.2.2 Descripció del diagrama de classes

Zona *[zone]*

Descripció : Representa una àrea geogràfica

Atributs :

- `latCenter`: Latitud del centre de la zona; `string`
- `lonCenter`: Longitud del centre de la zona; `string`
- `zoomLevel`: Magnitud a la qual s'ha de mostrar el mapa; `string`
- `description`: Text opcional per fer una petita descripció de la zona; `richtext`
- `fixedView`: Determina si s'ha de calcular l'extensió de la zona en funció dels nodes que conté; `boolean`
- `mapType`: Determina el tipus de mapa que es mostra inicial ment a la zona; `string`
- `prefix`: Text que s'oferirà com a prefix del nom d'un node dins aquesta zona; `string`

Mètodes :

- `checkShortName(name:string)`: Comprova que el nom curt no tingui cap caràcter invàlid, retorna cert si és correcte.
- `checkNodeExists(id:string)`: Comprova que no hi hagi cap node amb l'identificador `id`, retorna cert si en troba algun.

Node *[node]*

Descripció : Un punt en un mapa, que pot contenir aparells a dins

Atributs :

- `shortName`: Nom curt amb caràcters de 7 bits, que s'usarà com a prefix en els aparells del node; `string`
- `description`: Text opcional per fer una petita descripció del node; `richtext`
- `contactMail`: Adreça de contacte del responsable del node; `string`
- `latitude`: Latitud a la que es troba el node; `string`
- `longitude`: Longitud a la que es troba el node; `string`
- `alt`: Alçada del node respecte el nivell del terra; `integer`
- `image`: Fotografia opcional del node; `image`

Mètodes :

- `getNodeInfo()`: Retorna un diccionari amb les dades del node
- `createDeviceFromTemplate(template_name:string,id:string)`: Fa una copia completa de la plantilla `template_name`, creant un aparell nou amb identificador id

- `validateDeviceId(id:string)`: Comprova que no hi hagi cap aparell amb l'identificador id, retorna cert si no n'hi ha cap
- `getDeviceList()`: Retorna una llista amb l'identificador i el nom de cada aparell que hi ha en aquell node
- `getZone()`: Retorna l'objecte que representa la zona on pertany el node
- `getStatus()`: Calcula l'estat del node en funció de l'estat dels diferents aparells que conté
- `getDistanceTo(point:tuple)`: Calcula la distancia del node a un punt representat per una tupla (latitud,longitud)

Aparell *[device]*
Descripció : Representa un dispositiu de xarxa, el qual pot tenir un nombre determinat de interfícies, i esta associat a una plantilla
Atributs :

- `role`: Indica el rol que juga l'aparell a la xarxa; `string`
- `status`: Indica en quin estat de funcionament es troba l'aparell; `string`

Mètodes :

- `getDeviceInfo()`: Retorna informació de l'aparell en forma de diccionari
- `getLinks(IType:interface)`: Calcula tots els enllaços existents a l'aparell, filtrats segons una interfície marcadora, retorna una llista amb detalls de cada enllaç
- `mainIPAddress()`: Retorna l'adreça ip principal de l'aparell, en funció de la configuració de les seves interfícies
- `model()`: Retorna el model de l'aparell, consultant l'atribut de la plantilla que té associada
- `firmware()`: Retorna el firmware de l'aparell, consultant l'atribut de la plantilla que té associada
- `getInterfaceReferences()`: Retorna un diccionari amb les interfícies de l'aparell, i de cada una, les interfícies que tingui associades, si n'hi han
- `copyInterfaceReferences(device:string)`: Replica les referències entre interfícies en un aparell nou, prenent com a base les referències entre interfícies d'un altre aparell existent
- `getInterfacesByType(type:string)`: Retorna una llista d'interfícies de l'aparell, que el portal_type sigui `type`
- `getInterfaces(filter:list)`: Retorna una llista d'interfícies de l'aparell, que no siguin de cap dels tipus dins de filter
- `getNetworks()`: Retorna una llista de totes les xarxes que esta presents en alguna de les interfícies de l'aparell
- `getPhysicalInterfaces()`: Retorna una llista de totes les interfícies del tipus IWired o IWireless de l'aparell

- `getLogicalInterfaces()`: Retorna una llista de totes les interfícies del tipus ILogical de l'aparell

Relacions :

- `template`: Associa una aparell a una plantilla d'aparell

Interfície de xarxa *[iface]*
Descripció : Representa una interfície de xarxa genèrica, super-classe de la resta d'interfícies
Atributs :

- `MAC`: Adreça de màquina única de l'interfície de xarxa; `string`
- `enabled`: Marcador per determinar si volem l'interfície activada o no al moment de generar-ne la configuració; `boolean`
- `DHCP`: Marcador per determinar si volem configurar un servidor DHCP en la interfície al moment de generar-ne la configuració; `boolean`

Mètodes :

- `setIPAddress(ip:string)`: Estableix una referència entre la interfície i ip
- `getIp()`: Retorna la adreça assignada a la interfície
- `getMask()`: Retorna la màscara de la interfície en format decimal
- `getCIDR()`: Retorna la ip i la màscara de la interfície en notació CIDR
- `getCIDRMask()`: Retorna la màscara de la interfície en notació CIDR
- `getNode()`: Retorna el node al qual pertany l'aparell que conté la interfície
- `getBGPNetwork()`: Retorna la xarxa BGP a la qual pertany en última instància la ip de l'interfície
- `getLevel1Network()`: Retorna la xarxa a la qual pertany directament la ip de la interfície
- `getDevice()`: Retorna l'aparell que conté la interfície
- `bridged()`: Determina si la interfície forma part d'un bridge o no
- `getNetwork()`: Retorna la xarxa a la qual pertany directament la ip de la interfície
- `checkIPAddress(ip:string)`: Comprova si la ip està en un format vàlid
- `checkIPBaseBroadcast(ip:string)`: Comprova que la ip no coincideixi ni amb l'adreça base ni amb l'adreça broadcast, en funció de la màscara
- `deleteIPAdress()`: Esborra una adreça ip i les seves referències a la interfície
- `geLinks()`: Calcula tots els enllaços existents cap a l'interfície,i retorna una llista amb detalls de cada enllaç

Relacions :

- `iface_ip`: Indica quina ip esta assignada a aquesta interfície

Interfície de xarxa física *[physical_iface]*

Descripció : Representa una interfície de xarxa genèrica física (cablejada o sensefils)

Atributs :

- `order`: Numero identificant la priorat de la interfície respecte la resta d'interfícies físiques de l'aparell; `int`
- `allowedVirtualTypes`: diccionari que conté quines interfícies virtuals es permeten crear sobre la interfície física, quantes, i en quines condicions; `dict`
- `configurable`: Marcador per determinar si permetem que es pugui assignar una adreça ip a la interfície ; `boolean`

Mètodes :

- `getVirtualInterfaces()`: Retorna llista de les interfícies virtuals de les quals n'és l'interfície master

Radio *[radio]*

Descripció : Representa una interfície wireless genèrica

Atributs :

- `ssid`: Nom visible de la xarxa wireless; `string`
- `channel`: Canal d'emissió; `string`
- `band`: Banda de freqüències a la que treballa; `string`
- `mode`: Mode d'operació de l'interfície, referent a la topologia de la xarxa; `string`
- `power`: Potència d'emissió; `integer`
- `acceptClients`: Determina si la interfície permet connexions clients, pel contrari és dedicada; `boolean`
- `antennaAngle`: Direcció en graus a on esta orientada l'antena; `integer`
- `antennaElevation`: Numero en graus d'elevació de l'antena; `integer`
- `antennaGain`: Guany de l'antena en dBm; `integer`
- `antennaType`: Tipus d'antena; `integer`
- *Hereta* `order,allowedVirtualTypes,configurable` de physical_iface
- *Hereta* `MAC,enabled,DHCP` de iface

Mètodes :

- `supportedBands`: Retorna una llista de les bandes suportades per la radio
- *Hereta* mètodes de iface
- *Hereta* mètodes de physical_iface

Relacions :

- `linked_to`: Indica quines interfícies virtuals sensefils té

Cable *[cable]*
Descripció : Representa una interfície ethernet genèrica
Atributs :

- *Hereta* `order,allowedVirtualTypes,configurable` de physical_iface
- *Hereta* `MAC,enabled,DHCP` de iface

Mètodes :

- *Hereta* mètodes de iface
- *Hereta* mètodes de physical_iface

Relacions :

- `linked_to`: Indica quines interfícies virtuals té

Virtual *[virtual]*
Descripció : Representa una interfície virtual sobre una interfície física
Atributs :

- *Hereta* `MAC,enabled,DHCP` de iface

Mètodes :

- *Hereta* mètodes de iface

Relacions :

- `linked_to`: Indica quina és l'interfície master

Pont entre interfícies *[bridge]*
Descripció : Representa una interfície lògica que uneix com a una de sola d'altres interfícies del mateix aparell
Atributs :

- *Hereta* `MAC,enabled,DHCP` de iface

Mètodes :

- `getBridgedInterfaces()`: Retorna una llista de les interfícies que formen part del pont
- *Hereta* mètodes de iface

Relacions :

- `linked_to`: Indica quines interfícies formen part del bridge

Tunel OpenVPN *[tunel_ovpn]*

Descripció : Representa una interfície lògica que pot comunicar 2 o més interfícies distants sense un enllaç físic comú

Atributs :

- `type`: Determina si el túnel es de tipus enrutat(tun) o broadcast(tap); `string`
- `proto`: Determina el protocol a usar per la comunicació (tcp o udp); `string`
- `port`: Determina el port usat pel protocol per la comunicació; `int`
- *Hereta* `MAC,enabled,DHCP` de iface

Mètodes :

- `getConnections()`: Retorna una llista de les connexions desde altres interfícies túnel cap a ella
- *Hereta* mètodes de iface

Plantilla d'aparell *[device_template]*

Descripció : Descriu les característiques d'un aparell real i la seva configuració no relacionada amb la xarxa,serveix com a base per crear aparells i les seves interfícies

Atributs :

- `model`: Model de l'aparell, i.e nom comercial; `string`
- `templateDescripcion`: Descripció breu de les característiques de la plantilla; `richtext`

Mètodes :

- `validateInterfaceId(id:string)`: Comprova si ja hi ha una plantilla amb el mateix identificador.
- `getInterfaceOrder()`: Determina l'ordre de totes les interfícies en funció de la prioritat de les interfícies físiques

Relacions :

- `firmware`: Associa una plantilla d'aparell amb un firmware

Sistema Operatiu o Firmware *[firmware]*

Descripció : Classe genèrica per representar el software que permet el funcionament d'un aparell concret.

Atributs :

- `version`: Verisó del firmware; `string`
- `wirelessModes`: llista dels modes d'operació wireless suportats pel firmware; `list`
- `templateDescripcion`: Descripció breu de les característiques de la plantilla; `richtext`

Mètodes :

- `validateInterfaceId(id:string)`: Comprova si ja hi ha una plantilla amb el mateix identificador.

- `getInterfaceOrder()`: Determina l'ordre de totes les interfícies en funció de la prioritat de les interfícies físiques

Sistema Autònom BGP *[bgp_as]*
Descripció : Representa el conjunt de adreços d'una gran area BGP
Atributs :

- `as`: Indentificador per l'AS; `string`
- `ranges`: llista de rangs disponibles per a fer subnetting amb nivells inferiors; `lines`

Mètodes :

- `getNewID()`: Ha de retornar un identificador no usat per una àrea BGP
- `checkID()`: Ha de comprovar la validesa d'un identificador d'una àrea BGP
- `checkRange(range:string)`: Ha de comprovar la validesa d'un rang BGP

Àrees OSPF *[ospf_areas]*
Descripció : Representa el conjunt de rangs d'una àrea OSPF
Atributs :

- `areaID`: Identificador de l'àrea OSPF; `string`
- `apMask`: mida de màscara a utilitzar en punts d'accés dins aquesta àrea, en notació CIDR; `string`
- `linkMask`: mida de màscara a utilitzar en enllaços punt a punt dins aquesta àrea, en notació CIDR; `string`
- `switchMask`: mida de màscara a utilitzar en enllaços per cable multi-punt amb switch dins aquesta àrea, en notació CIDR; `string`

Mètodes :

- `getAvailableRange(cidr:string,type:string)`: Si hi ha lloc, retorna el rang d'una xarxa "ospf_networks"del tipus i mida indicats i l'identificador del "ospf_ranges"on s'ha de crear.
- `getNewID()`: Ha de retornar un identificador no usat per una àrea OSPF
- `checkID()`: Ha de comprovar la validesa d'un identificador d'una àrea OSPF

Rangs OSPF *[ospf_ranges]*
Descripció : Representa un rang d'adreços del que pot disposar una àrea OSPF
Atributs :

- `range`: Adreça base i màscara del rang, en notació CIDR; `string`
- `type`: Tipus del rang segons l'ús que se li vulgui donar; `string`

Mètodes :

- `createNetwork(range:string)`: Crea una xarxa "ospf_networks"dins el rang OSPF

- `getSortedNetworks()`: Retorna una llista ordenada de petit a gran de totes les xarxes "ospf_networks" que conté el rang OSPF
- `containsNetwork(range:string)`: comprova si el rang donat és subrang del rang OSPF
- `checkRange(range:string)`: Ha de comprovar la validesa d'un rang: ha de ser subrang de BGP i no solapar-se amb altres rangs ospf
- `hasRoomFor(cidr:string)`: Ha de comprovar si hi ha lloc en el rang per una subrang de mida `cidr`

Xares OSPF *[ospf_networks]*
Descripció : Representa un a subxarxa d'un rang OSPF, de la qual es poden assignar adreces ip
Atributs :

- `range`: Adreça base i màscara del rang, en notació CIDR; `string`

Mètodes :

- `createNetwork(range:string)`: Crea una xarxa "ospf_networks" dins el rang OSPF
- `getSortedNetworks()`: Retorna una llista ordenada de petit a gran de totes les xarxes "ospf_networks" que conté el rang OSPF
- `containsNetwork(range:string)`: comprova si el rang donat és subrang del rang OSPF
- `getAvailableIPAddress()`: Ha de retornar una adreça ip disponible de la xarxa OSPF
- `existsIPAddress(ip:string)`: Ha de comprovar si la ip donada ja existeix a la xarxa OSPF
- `isFull()`: Ha de comprovar si hi ja hi han totes les adreces ip assignades
- `createIP(ipv4:string)`: Ha de crear una adreça ip dins la xarxa OSPF
- `delete()`: Ha d'esborrar la xarxa OSPF

Xarxes adhoc OLSR *[adhoc_network]*
Descripció : Representa la configuració base d'una xarxa OLSR i el rang de que disposa per fer particions
Atributs :

- `range`: rang d'adreces disponible per fer particions; `string`
- `essid`: Nom visible de la xarxa wireless que s'usarà per tots els aparells que formin part de la xarxa OLSR; `string`
- `channel`: Canal en que emetran tots els aparells que formin part de la xarxa OLSR; `string`
- `rangeMask`: mida de la màscara per defecte a utilitzar a L'hora de fer noves particions, en notació CIDR; `string`

Mètodes :

- `getAvailableAdhocRange(dhcp:boolean,type:string)`: Retorna una partició del tipus indicat que tingui adreces lliures, o el rang per crear-ne una de nova, si no n'hi han

- `createAdhocRange(type:string,range:string,dhcp:boolean)`: Crea un "adhoc_range"del tipus i paràmetres indicats
- `checkRange(range:string)`: Ha de comprovar la validesa d'un subrang: ha de ser subrang de l'area BGP, i no es pot solapar amb d'altres xarxes adhoc
- `hasRoomFor(cidr:string)`: Ha de comprovar si hi ha lloc a la xarxa per una partició de mida `cidr`

Rang d'adreçes Adhoc *[adhoc_range]*
Descripció : Objecte genèric que representa una partició o subconjunt del rang d'una xarxa OLSR
Atributs :

- `range`: Rang d'adreces en notació CIDR; `integer`

Mètodes :

- `providesIP(ip:string)`: Determina si és possible assignar la ip a la partició, en funció de les ip's ocupades i el tipus de partició.
- `getAvailableIPAddress()`: Ha de retornar una adreça ip disponible de la xarxa
- `existsIPAddress(ip:string)`: Ha de comprovar si la ip donada ja existeix a la xarxa
- `isFull()`: Ha de comprovar si hi ja hi han totes les adreces ip assignades
- `createIP(ipv4:string)`: Ha de crear una adreça ip dins la xarxa
- `delete()`: Ha d'esborrar la xarxa

Partició bàsica *[basic_range]*
Descripció : Representa un conjunt d'adreces disponibles per a aparells freifunk clients
Atributs :

- `dhcp`: Determina si hem de deixar lloc per adreces dinàmiques contigues quan assignem adreces noves; `boolean`
- *hereta* `range` de adhoc_range

Mètodes :

- *hereta* mètodes de adhoc_range

Partició OpenVPN *[tunnel_range]*
Descripció : Representa un conjunt d'adreces disponibles per a fer túnels entre aparells freifunk
Atributs :

- *hereta* `range` de adhoc_range

Mètodes :

- *hereta* mètodes de adhoc_range

Partició Master *[master_range]*

Descripció : Representa un conjunt d'adreces disponibles per a aparells freifunk connectats a la troncal OSPF

Atributs :

- `dhcp`: Determina si hem de deixar lloc per adreces dinàmiques contigues quan assignem adreces noves; `boolean`
- *hereta* `range` de adhoc_range

Mètodes :

- *hereta* mètodes de adhoc_range

Adreça IP *[ip]*

Descripció : Representa una adreça genèrica de xarxa

Atributs :

- `ipv4`: Adreça ip i màscara de xarxa en notació CIDR; `string`

Mètodes :

- `setInterface(interface:object)`: Assigna la interfície de xarxa indicada a l'adreça IP

Relacions :

- `ip_iface`: Indica quina interfície esta assignada a aquesta ip

Capítol 4

Disseny del Sistema - Tecnologia

4.1 Maquinari

Per determinar quin hardware és necessari per a hostatjar l'aplicació, hi han dos punts de vista a tenir en compte: L'entorn on s'explotarà l'aplicació en un principi, i l'entorn on es faria servir donada una futura migració de guifi.net.

El criteri a utilitzar serà el nombre de peticions per segon que volem que el servidor ens proporcioni un cop en producció.

4.1.1 Explotació a GuifiBages

S'espera un ús local de l'aplicació, entenent com a local els usuaris de la xarxa a Manresa, sense un accés massiu de l'aplicació via Internet. Les dades que s'hi guardaran seràn només de la xarxa de Manresa. Tot això fa que no necessitem ni molt ample de banda ni molta capacitat de processament i emmagatzemament. Basant-nos en el rendiment que han donat les maquines utilitzades per dur a terme el desenvolupament, podríem marcar el hardware mínim a ser utilitzat en :

- 1 Intel Pentium 4 2.6 Ghz
- 1 Gb RAM
- 80 Gb Disponibles disc dur Serial Ata

Aquesta configuració ens dona un rendiment de 2,8 peticions per segon acceptades pel servidor, suficient per les necessitats del primer cas. En quant a capacitat de disc, faran falta mínim unes 10 gigues, sumant les necessitats del Sistema Operatiu, l'aplicació, i la base de dades

4.1.2 Explotació a guifi.net

Per aquest cas, agafarem dades del rendiment de l'actual aplicació de guifi.net, i les característiques del maquinari on esta allotjada:

- 2 Intel Xeon 2.8 Ghz
- 1 Gb RAM

- Discs durs 250 GB Serial ATA en Raid-0

Aquesta maquina dedicada exclusivament a servir l'aplicació de guifi.net (Apache + PHP + Mysql) actualment suporta un transit mensual de 80 Gb, unes 600.000 visites, i unes 3.000.000 d'accessos. Mesurant el rendiment en les mateixes condicions que l'anterior cas, aquesta maquina esta suportant aquest transit amb un rendiment de 3.54 peticions per segon.

Per poder fer una extrapolació dels requeriments de hardware necessaris per a poder suportar aquest transit amb la aplicació que hem desenvolupat, cal tenir diverses coses en compte:

- **El software servidor :** És difícil de dir si una solució és més rapida que l'altra. Per una banda la solució actual a guifi.net te 3 capes de funcionament, que serien Apache + PHP + Mysql en contra de la única capa de Zope. Per altra banda però, la complexitat de Zope fa que en general una instal·lació de Zope bàsica sigui d'execució més lenta que una solució LAMP.

- **El SGBD :** L'aplicació de guifi.net és l'evolució de una petita aplicació de gestió a mida que ha anat creixent, i que ara es troba amb problemes difícils de resoldre amb la solució tecnològica en que esta programada. A nivell de rendiment un problema a destacar seria la Base de dades. S'utilitza un SGBD Relacional (MySQL) per gestionar un model de dades que és intrínsecament orientat a objectes, el que ha portat, tot i intents de optimització, que consultes molt frequents i necessàries a la BD siguin altament costoses en temps de processament. En Zope l'us de una base de dades OO, ha reduït molt aquest cost.

- **Les interfícies d'usuari :** Aquest punt, més que una qüestió del programari utilitzat, tracta sobre el disseny de les interfícies. L'aplicació de guifi.net es base en interfícies dinàmiques, que un cop estan carregades necessiten de una recarrega amb la conseqüent petició de tota la pàgina al servidor. Les parts més utilitzades d'aquesta aplicació són pàgines amb gran volum de dades, de les quals més de la meitat de la informació que arriba al navegador amb cada recarrega és idèntica. En aquest projecte, s'ha optat per utilitzar AJAX en les interfícies susceptibles de ser recarregades diverses vegades, per evitar aquesta càrrega innecessària i nomes carregar les dades que necessitessin ser actualitzades.

Seria extremadament complexe comparar quantitativament els dos sistemes per poder extreure'n conclusions vàlides per decidir una configuració de hardware. Qualitativament podríem afirmar que la manera en que s'ha optimitzat el disseny de l'aplicació, i l'ús de una BDOO, compensa la falta de rapidesa d'execució del sistema Zope en comparació amb LAMP, i per tant l'aplicació molt possiblement tindria un rendiment similar un cop instal·lat en una maquina amb iguals característiques. També s'ha de dir que l'arquitectura de Zope permet l'utilització de sistemes de cache per accelerar, que fàcilment poden augmentar de 3 a fins a 100 peticions per segons el rendiment d'un servidor, i sumat al seu disseny modular, permet distribuir la carrega de processament separant els seus components en diverses instàncies repartides en diverses màquines.

4.1.3 Conclusions

Per concloure doncs, una configuració per poder donar un alt rendiment a nivell de peticions per segon, constaria de 2 maquines de característiques mínimes a les del servidor actual de guifi.net, una per allotjar Zope, i l'altra per fer el l'acceleració via cache de les peticions entrants, i possiblement altres serveis secundaris.

4.1.4 Maquinari usat en la realitat

Actualment l'aplicació es troba en el mateix servidor que s'ha utilitzat com a entorn de producció durant el desenvolupament. És una maquina allotjada a l'Escola Politècnica Superior d'Enginyeria de Manresa, on hi han allotjats altres projectes. En un futur, està previst canviar l'aplicació a un servidor dedicat, fent servir les mateixes instal·lacions de la UPC.

Les característiques del hardware on està allotjat són

- 2 Intel Pentium 4 2.8 Ghz
- 1 Gb RAM
- 138 Gb disponibles de Disc dur

4.2 Software

L'objectiu general que es persegueix amb l'aplicació de que es objecte aquest projecte és en definitiva un gestor de continguts personalitzat a un model de dades concret i relativament complexe. Aquests gestors són anomenats amb les sigles CMS (Content Management System). En els requeriments ens demanen que l'aplicació s'ha de desenvolupar fent servir el CMS Plone. Utilitzar Plone implica fer servir i conèixer tot el que hi ha darrera de Plone: el servidor d'aplicacions Zope i el llenguatge de programació Python, totes eines amb llicència de codi obert.

Tot i que és un requeriment bàsic del projecte, a continuació m'agradaria justificar el perquè aquestes eines són les adequades per al projecte, passant per definir alguns conceptes i explicar d'on provenen algunes coses.

4.2.1 CMS

Per definició un CMS és una aplicació web dissenyada per facilitar a usuaris sense coneixements tècnics sobre web ni sobre el propi CMS, afegir, editar i en definitiva gestionar els continguts d'un portal web. Un CMS no només ha de facilitar l'ús i la gestió de continguts, sinó que en segon pla, ha de mantenir índexs per cercar informació, gestionar usuaris, permisos, aspectes de seguretat, generar automàticament elements de navegació pel portal, i moltes altres coses. Dins aquesta definició, hi han moltes solucions disponibles al mercat que s'hi podrien englobar.

Una característica que tenen molts CMS és que pràcticament tots permeten fins un cert grau de personalització sense haver de escriure una sola linia de codi. Hi hauran inevitablement grans diferencies entre CMS, tant a nivell de funcionalitat com de complexitat, però actualment és una de les característiques bàsiques.

Des d'un punt de vista d'usuari, el que marcarà la diferència entre CMS, serà la usabilitat de les seves interfícies i la corba d'aprenentatge per a aprendre a utilitzar el CMS bàsic. Des d'un punt de vista de desenvolupador, el que marca la diferència entre CMS, és amb què ens trobem un cop passem la línia del que es pot fer sense escriure codi, i veiem les eines que ens proporciona el CMS per poder desenvolupar components personalitzats. Des del mateix punt de vista, també són molt importants aspectes com el llenguatge de programació utilitzat, l'orientació a objectes, quins sistemes de bases de dades podem utilitzar amb aquell CMS, internacionalització (i18n)...

Els CMS que compleixen amb garanties aquestes premisses, solen tenir una arquitectura robusta, basada en un servidor d'aplicacions. Aquest terme va sorgir de la plataforma Java, concretament amb l'anomenat J2EE. Això va proporcionar una plataforma per crear tot tipus d'aplicacions basades en web amb estructura servidor-client, centralitzant les dades i el codi assegurant-ne així la integritat amb independència de qui o des d'on accedís a les dades. Més enllà del java, el qual en el seu moment va representar un increment en productivitat respecte altres llenguatges com C, avui en dia són molts els que opinen que Java ha crescut tant desmesuradament, que ha reduït la productivitat de la que podia presumir respecte altres llenguatges, i ara com ara no és la millor eina per desenvolupar aplicacions web.

Una de les raons per afirmar això és la següent: *el codi d'una aplicació, és llegeix més vegades que no pas s'escriu.* És a dir, en un llenguatge de programació, un dels aspectes més important és que el codi sigui fàcilment llegible i intel·ligible, ja que escriure s'escriurà una vegada, però inevitablement, un mateix o altes persones l'hauran de llegir per modificar-lo o ampliar-lo, i l'han d'entendre. Algú podrà

dir que per això existeixen els comentaris al codi, però personalment crec que codi comentat línia a línia més aviat posa traves a entendre'l bé.

Una de les maneres més obvies de aconseguir que un llenguatge compleixi això, és reduir-ne al màxim la verbositat. I això és una de les bases de la filosofia del llenguatge de programació Python. No és doncs extrany que entre els sistemes CMS més avançats que es poden trobar avui en dia, apareguin noms com Django, Rails, TurboGears o Zope, tots ells programats en Python.

4.2.2 Python + Zope + Plone

Plone és un CMS pogramat sobre la base del servidor d'aplicacions Zope. Plone és un producte de Zope, la funció del qual és aportar una capa que facilita d'utilització de la potència de Zope com a CMS, posant a disposició una API pròpia independent de l'API de Zope. És a dir, que tot i que Plone pot semblar que només sigui una extensió de Zope, en realitat el que proporciona és tot un framework de desenvolupament separat de Zope, però aprofitant tota la potència d'aquest de rerefons.

Zope

Zope esta compost per ZODB, una base de dades transaccional orientada a objectes, que emmagatzema tot des dels continguts, passant per codi i templates HTML dinàmics, fins a connexions a bases de dades relacionals o sistemes basats en fitxers. Un altre component de Zope és ZServer que permet accedir als continguts a traves de diversos protocols com HTTP, FTP, WebDAV o XML-RPC, o pot ser utilitzat fent servir servidors web de tercers com Apache o similars. L'extensibilitat de Zope és possible gràcies al mecanisme de Productes i de Zclasses, que permet incorporar des de simples classes noves a complexos components.

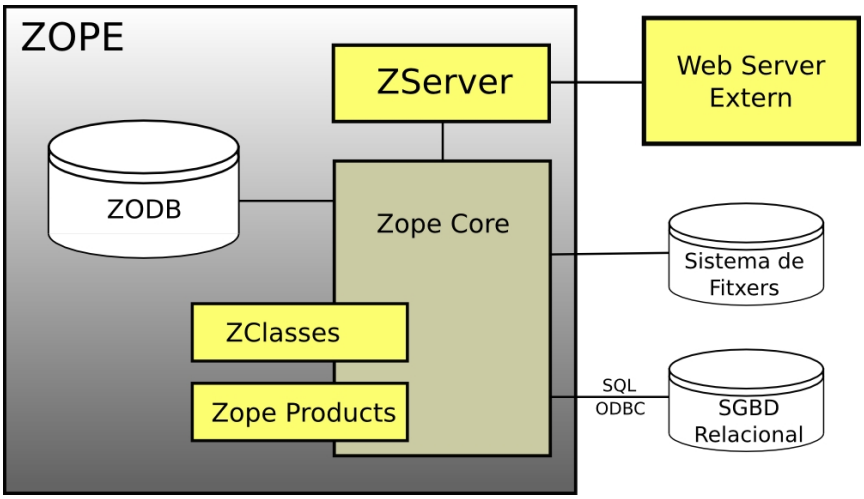


Figura 4.1: Estructura dels components de Zope

La extrema orientació a objectes de Zope el fa molt més potent i flexible que altres sistemes com PHP o ASP que es basen en arxius. Permet una clara separació de les dades, codi i presentació, té un sistema de seguretat i permisos molt potent integrat, controls d'accés, eines de cerca. Tota la arquitectura de Zope esta pensada per complir amb les especificacions de W3C, i amb el model de desenvolupament Web, permetent així poder delegar el desenvolupament d'una aplicació web a experts en disseny, bases de dades

i gestors de continguts separadament, amb un grau molt alt d'independència entre ells.

El fet que Zope sigui tal com és recau en adonar-se que el Web com a tecnologia ha set en definitiva sempre orientat a objectes. Si ens hi fixem, una URL d'una web, no és res més que un camí cap a un objecte dins una jerarquia contenidora, i el protocol HTTP és un mecanisme per enviar missatges a aquests objectes. Basat en aquest simple concepte, les URL de Zope sempre reflecteixen la jerarquia on pertany l'objecte que estem visualitzant.

Hi han diferents aspectes que fan de Zope un sistema molt potent: Un d'ells és sens dubte que qualsevol cosa que es pugui configurar modificar o afegir a Zope es pot fer a traves de la interfície de gestió via web ZMI (Zope Management Interface). Altres punts forts són el model de compartició de dades, en particular l'Adquisició [1], i suport de XML complet.

Plone

Algunes de les característiques més importants de Plone:

- Fàcil d'instal·lar - disposa d'instal·ladors preparats per a les diferents plataformes per a les que esta disponible.
- Fàcil d'utilitzar - A l'equip de Plone hi han reconeguts experts en usabilitat d'interficies, que han fet que sigui un producte extremadament facil d'usar.
- Internacional - La interfície esta traduïda a més de 35 idiomes, i hi han incloses eines per gestionar continguts multilingües.
- Estàndard i neutral - Plone segueix els estàndards d'accessibilitat i usabilitat marcats per organismes com el W3C o altres, així com pot operar amb la majoria de SGBDR, ja siguin de codi obert o comercials, i pot funcionar el multitud de plataformes.
- Programari Lliure - El fet de estar llicenciat sota GNU, et dona el dret d'usar Plone sense pagar per ell, modificar-lo i millorar-lo. Això és una de les claus de la gran evolució de Plone: els centenars de desenvolupadors de tot el món que contribueixen al projecte, i un gran nombre d'empreses especialitzades, que utilitzen Plone, i hi contribueixen sota les mateixes condicions de la GNU.
- Extensible - Hi ha una llarga llista de productes addicionals per estendre la funcionalitat de Plone, gràcies en gran mesura al comentat al punt anterior

Python

Algunes dels aspectes més destacables de Python:

- Sintaxis clara i entenedora : Python elimina els delimitadors innecessaris que contenen la majoria de llenguatges de programació com **{} () begin end ;** i en lloc d'això fa servir la indentació de codi com a delimitador. En Python indentar el codi no és una opció estètica, és part de la sintaxis, el que fa que un codi Python sempre serà llegible en aquest sentit.

[1] exemple a secció 6.2.2

- Introspecció : És refereix a la capacitat que te un programa escrit en Python de poder obtenir informació d'un objecte : que és, que pot fer, o que conté entre altres coses. Podem conèixer les funcions que te un objecte sense conèixer la seva definició, o accedir a funcions i/o atributs dels que en desconeixem el nom. És molt útil entre d'altres coses a l'hora d'aprendre el llenguatge i per depurar codi.

- Tipus de dades d'alt nivell : apart dels tipus base, prescindeix de els clàssics arrays i utilitza llistes, diccionaris i tuples, cadascun d'ells molt optimitzat per diferents tasques i amb una gran ventall de funcionalitats i mètodes.

- Llibreries: té una extensa llista funcions a la llibreria bàsica que cobreix una gran funcionalitat sense importar mòduls externs. Apart te un sistema de mòduls molt extensible, amb milers de llibreries de tercers. El fet que sigui programari lliure influeix molt en la quantitat de les llibreries que van apareixent.

- Mode d'execució : Python tant pot ser compilat com interpretat, el que li dona molta flexibilitat en funció de l'objectiu pel qual estem programant i de la complexitat del mateix.

- Plataforma : Esta disponible pels principals Sistemes Operatius, Linux/Unix, OS/2, MacOS, Windows i altres. Inclús pot ser executat en .NET o en una maquina virtual de java. Qualsevol codi pot ser executat sense canvis entre plataformes.

4.2.3 Com funciona tot plegat

La següent figura descriu gràficament la cohesió de aquests 3 components Python, Zope i Plone i com es relacionen entre ells:

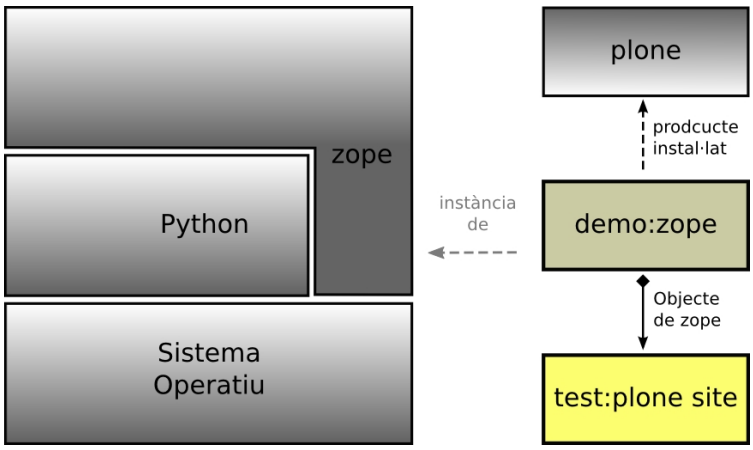


Figura 4.2: Funcionament de Zope i Plone

Com hem comentat anteriorment, Zope esta escrit quasi exclusivament en Python, excepte algunes parts escrites en C++ que accedeixen al SO directament. Això però no ens afecta per explicar el funcionament de Zope.

Zope basa en seu funcionament en instàncies. Cada instància de Zope, és l'execució del propi Zope, associada a un port d'escolta HTTP, per defecte el 8080. És poden crear i executar diverses instàncies de Zope en una mateixa màquina, totes independents l'una de l'altra. això vol dir que cada instància és una

rèplica completa de tot el Zope, incloent una rèplica de Zserver i de la BDOO ZDB per cada instància. Hi han altres configuracions possibles, com per exemple l'utilització de ZEO, que serveix per afegir una capa extra al Zope separant la base de dades de la resta de components, de manera que les diferents instàncies de Zope, comparteixen una sola ZODB.

En el diagrama, la instància correspon a l'objecte anomenat *demo:zope*. Amb això ja tenim un Zope complet i funcionant. Per la part de Plone, seguim un procés que te certes similituds: Plone és un producte de Zope i com a tal, s'ha d'instal·lar dins la estructura de Zope. Com a producte que és, Plone pot dependre i de fet depèn d'altres productes, alguns del propi nucli de Zope i d'altres creats per completar el framework de Plone, com a mòduls separats.

Arribats a aquest punt tenim una instància de Zope funcionant, amb Plone instal·lat. Plone però ara no és més que un conjunt de objectes del quals Zope en pot disposar, però dels que no n'existeix cap instància. Per fer funcionar Plone hem de crear una instància de Plone (*plonesite*), contingut dins de l'arrel de la jerarquia d'objectes de Zope. D'aquesta manera tenim un Plone anomenat *test* funcionant sobre la instància *demo:zope* de Zope.

Seguint aquest procés, hi han diversos escenaris possibles on podem tenir 1 Zope + 1 instància Plone funcionant de manera dedicada, 1 Zope + N instàncies Plone o tb N Zope + N Plone, tot en funció de les nostres necessitats.

Plone per si sol, permet gestionar continguts en forma de carpetes, documents i events. Per extendre Plone afegint nous tipus de continguts,existeix un modul anomenat ArcheTypes, inclòs amb el paquet de Plone. A continuació veurem com ArcheTypes ens permetrà definir nous tipus de continguts per Plone.

ArcheTypes

ArcheTypes és un framework dins el propi Plone per desenvolupar nous tipus de continguts per un projecte Plone. La majoria de CMS proporcionen eines per afegir nous tipus de continguts, que normalment requereixen un coneixement de com funciona tot el sistema. ArcheTypes proporciona un sistema simple que no necessita de més coneixement que la pròpia sintaxis. ArcheTypes es basa en objectes, definits amb un sintaxis pròpia, que un cop en funcionament, queden integrats amb Plone, generant automàticament formularis d'edició i interfícies de visualització pels continguts.

ArcheTypes consta de 2 parts: ATContentTypes, que contenen una serie de classes base a partir de les quals, mitjançant herència, podem crear les nostres. Apart tenim els ATProducts. Un ATProduct (ArcheTypeProduct) és un conjunt que fitxers que compleixen els requeriments dels productes de Zope, i que utilitzen ATContentTypes i la sintaxis d'ArcheTypes per definir com és el producte. La estructura bàsica d'un Producte és la següent :

```
- README.txt
- __init__.py
- config.py
- permissions.py
- content
    - __init__.py
    - content.py
- Extensions
    - Install.py
```

```
- skins
    - instantmessage
- tests
    - __init__.py
    - base.py
    - test_setup.py
```

De tota aquesta estructura, que de base sol ser pràcticament idèntica, ens interessa el fitxer content/content.py. En aquest arxiu es on hi haurà la definició dels nous tipus de continguts. Per veure l'estructura necessària per definir un nou tipus de contingut ens basarem en la seguent porció de codi com a exemple:

Llistat 4.1: ArcheTypes

```
1  from AccessControl import ClassSecurityInfo
2  from Products.Archetypes.atapi import *
3  from Products.guifibages.radio import radio
4  from Products.guifibages.config import *
5
6
7  ##code-section module-header #fill in your manual code here
8  ##/code-section module-header
9
10 schema = Schema((
11
12     IntegerField(
13         name='diversity',
14         widget=IntegerField._properties['widget'](
15             label='Diversity',
16             label_msgid='guifibages_label_diversity',
17             i18n_domain='guifibages',
18         )
19     ),
20
21     IntegerField(
22         name='distancia',
23         widget=IntegerField._properties['widget'](
24             label='Distancia',
25             label_msgid='guifibages_label_distancia',
26             i18n_domain='guifibages',
27         )
28     ),
29
30 ),
31 )
32
33
34
35 atheros_schema = BaseSchema.copy() + \
36     getattr(radio, 'schema', Schema(())).copy() + \
37     schema.copy()
38
39
40 class atheros(BaseContent, radio):
41     """
42 ␣␣␣␣"""
43     security = ClassSecurityInfo()
44     __implements__ = (getattr(BaseContent,'__implements__',()),) + (getattr(radio,'__implements__',())
           ,)
45
```

```
46      # This name appears in the 'add' box
47      archetype_name = 'atheros'
48
49      meta_type = 'atheros'
50      portal_type = 'atheros'
51      allowed_content_types = [] + list(getattr(radio, 'allowed_content_types', []))
52      filter_content_types = 0
53      global_allow = 0
54      #content_icon = 'atheros.gif'
55      immediate_view = 'base_view'
56      default_view = 'base_view'
57      suppl_views = ()
58      typeDescription = "atheros"
59      typeDescMsgId = 'description_edit_atheros'
60
61      _at_rename_after_creation = True
62
63      schema = atheros_schema
```

Podem veure que hi han quatre parts diferenciades:

- Línies 1-4 : En la primera importem els mòduls necessaris que necessitarem més endavant
- Línies 10-31 : En la segona definim "schema"que representa el conjunt d'atributs que tindrà aquell objecte, les propietats de cada atribut, i molt important, el "widget"que s'utilitzara per representar-lo visualment. Fent una analogia de les conegudes IDE's de desenvolupament de Borland, un widget seria un equivalent a un dels molts components VCL disponibles com caixes de text, llistes desplegables, taules, etc.
- Línies 35-37 : En la tercera part, es construeix l'schema definitiu, heretant del bàsic BaseSchema, i de altres si és el cas. En l'exemple, podem veure com adquireix els atributs de la classe *radio* apart de la base. En certa manera, podem considerar que aquí es fa herència de interfícies, ja que la pròpia definició de com serà la interfície (quins widgets utilitzarà), esta inclosa en l'schema.
- Línies 40-63 : En la quarta part, es defineix la classe com a tal. Per fer-ho, s'han de definir tota una llista de variables obligatòries i algunes d'opcionals, que determinen diversos aspectes del nou objecte, des d'aspectes visuals (icones, textos), com de comportament (permisos, filtres). Notar que un dels atributs d'aquesta nova classe, és l'schema que hem definit anteriorment. Plone, coneixent la estructura d'un producte basat en ArcheTypes, sabrà doncs on buscar la informació en la definició d'un objecte per tal de poder-lo representar. A l'exemple no hi han definicions de funcions, però en cas que n'hi haguessin, es col·locarien just a continuació de la declaració dels atributs.

Un Producte pot constar únicament d'un nou tipus de contingut, o bé tenir-ne diversos, amb relacions més o menys complexes. Segurament com en d'altres entorns de programació, un cop un projecte arriba a un cert punt de complexitat, és necessari de fer servir tècniques o eines més avançades per tal de dur a bon port el projecte. És aquí on la forta orientació a objectes de Zope + Plone + ArcheTypes, permet l'existència d'una eina indispensable per a la creació i manteniment de productes complexos. Aquesta eina s'anomena ArchGenXML i a continuació veurem de que tracta.

ArchgenXML

Per parlar d'aquesta part, hem de veure que ArchGenXML és simplement un programa que duu a terme una tasca concreta, però el procés complet depèn d'altres programes i/o tecnologies. Aquestes tecnologies són UML, i XML/XMI, i els programes, Editor UML XMI, i el propi ArchGenXML. Per veure com s'integren tots aquests components, donem un cop d'ull al següent diagrama:

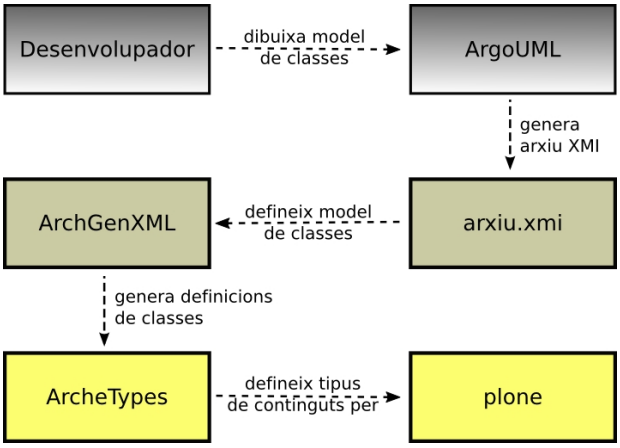


Figura 4.3: Procés de desenvolupament usant ArchGenXML

Com hem dit abans, donada la orientació a objectes del sistema que utilitzem, és obvi de fer servir UML per definir el sistema. L'interessant és com aquí s'aprofiten diverses tecnologies per complementar un diagrama UML, per convertir el conjunt del procés en una eina CASE a mida per construir productes Zope per Plone basats en ArcheTypes.

Primer de tot el desenvolupador dibuixa el diagrama de classes per definir el projecte en que esta treballant. En tot moment, s'ha de tenir en compte quins elements de UML estan suportats per ArchGenXML, ja que més endavant aquest els haurà d'entendre. Per dibuixar l'UML s'ha de tenir clars alguns conceptes bàsics de com interpretarà les classes i les relacions entre classes l'ArchGenXML:

Els dos tipus base amb que treballa Plone són carpetes i documents. Les carpetes poden contenir documents i altes carpetes, i els documents nomes poden estar a dins de carpetes. Això es correspon a una composició en UML. Per defecte, tota classe al final d'una jerarquia de classes composades serà tractada com a document, i tota classe que tingui una composició serà tractada com a carpeta. Les relacions que es poden definir entre classes en UML, l'ArchGenXML les interpreta en una sola direcció: és a dir que si tenim una relació 1 a N per exemple, amb navegabilitat cap a les 2 bandes, només generà una sola relació a la classe des d'on s'ha creat la relació. És a dir que si volem poder navegar a traves de les classes utilitzant una relació en les 2 direccions, hem de crear 2 relacions entre les classes, una en cada direcció.

Sabent això, podem dissenyar el diagrama de classes fent servir el mètode UML normalment. Per modificar les propietats de les classes i relacions, es poden associar parells de variable:valor a nivell de classe, atribut, mètode, i relacions. Els valors possibles d'aquests variables i una llista extensa d'aquestes, forma part de la documentació de l'ArchGenXML[2].

Per tal que tota aquesta feina de dibuixar arribi a bon port, l'editor UML que utilitzem ha de complir un requeriment bàsic: ha de poder exportar el diagrama en format xmi. XMI (XML Metadata

[2]http://plone.org/documentation/tutorial/archgenxml-getting-started/

Interchange) és un format basat en XML que permet exportar qualsevol tipus de diagrama UML en un estàndard reconegut. ArchGenXML necessita el diagrama de classes en aquest format per tal de poder funcionar. Hi han diversos editors disponibles que compleixen amb l'especificació XMI, alguns d'ells de codi obert, com ArgoUML.

Un cop tenim el diagrama acabat i exportat en XMI, passarem aquest arxiu a ArchGenXML, del qual obtindrem de resultat, l'estructura de fitxers d'acord a l'especificació d'un producte Zope i la definició de totes les classes noves amb els seus "schemas", tal com havíem vist en la secció dels ArcheTypes. ArchGenXML també s'ocupa de generar els fitxers necessaris per la instal·lació automàtica de les noves classes.

Hem aconseguit doncs, a partir de la definició d'un model UML, l'exportació a XMI i l'acció de ArchGenXML, crear tota la estructura necessària per tenir un producte funcional. Fins al moment no hem escrit ni una sola línia de codi. El gran avantatge de tot plegat és que permet el desenvolupament de continguts fins a un cert grau de complexitat, sense haver de escriure gens de codi, ni de construir cap tipus de interfície visual.

A partir però de cert grau de complexitat, voldrem funcionalitats que el model de ArcheTypes no ens pot aportar per si sol, o voldrem construir interfícies complexes de vista i edició que no estiguin lligades directament a un objecte, o amb comportaments especials. En aquest punt, és on entra el desenvolupament en Plone pròpiament dit. No obstant, no és incompatible amb el que s'ha explicat fins ara, sinó que es complementari. La base que obtenim amb ArchGenXML es extensible per adaptar-la a el que necessitem, i ArchGenXML tindrà en compte tots els canvis que puguem fer al codi que ell genera, no sobreescrivint mai res del que modifiquem. En el futur podem fins i tot modificar el diagrama de classes, i regenerar el producte, i ArchGenXML tindrà en compte la base que hi havia generada, els canvis que hem introduït manualment, i el nou model UML a partir del qual estem treballant en aquell moment.

4.2.4 Adaptació del model estàtic

Tot aquest conjunt d'eines exposades anteriorment, constitueixen un entorn de desenvolupament molt específic. Com a tal, necessita de certes adaptacions respecte un model estàtic tradicional, ja que al usar UML com a llenguatge base per la eina ArchGenXML, hem d'introduir i modificar certs elements en el model de dades. En aquesta secció mostrarem el perquè d'aquests elements, així com el diagrama de classes ampliat definitiu, en el que s'inclouen tots els elements i classes noves necessàries per la generació del codi.

Interfícies

Una interfície és una element abstracte no instanciable, que en el cas de l'entorn escollit, pot contenir declaracions de funcions i atributs. L'objectiu d'aquestes interfícies és el de que una o diverses classes les *realitzin*, és a dir que n'implementin els seus mètodes. La diferència amb una classe abstracta normal amb herència cap a altres classes, recau en que el que s'hereta és justament només una definició de les funcions i no la seva implementació. Això permet a classes diferents, però amb semblant comportament, compartir definicions de funcions, amb diferent implementació. Al no heretar cap implementació, no cal sobreescriure cap funció com es faria en herència normal. Això comporta certes avantatges a nivell de programació, ja que es pot accedir a mètodes amb el mateix nom i resultats, de classes amb implementacions diferents.

També es fa ús de les interfícies, com a interfícies *marcadores*. Això vol dir que independentment de si conté o no declaracions de funcions, la interfície en sí és un element utilitzat per *marcar* una classe amb una propietat o capacitat concreta [3]. Es fa ús d'això a nivell de programació, demanant a la classe si proveeix una determina interfície o no.

Descripció de les interfícies

IArea

- `getNewID()`: Ha de retornar un identificador per una àrea
- `checkID()`: Ha de comprovar la validesa d'un identificador d'una àrea

IRange

- `getAvailableRange()`: Ha de retornar un subrang disponible per ser usat en nivells inferiors
- `checkRange(range:string)`: Ha de comprovar la validesa d'un subrang
- `hasRoomFor(cidr:string)`: Ha de comprovar si hi ha lloc en un rang per un subrang de mida `cidr`

INetwork

- `getAvailableIPAddress()`: Ha de retornar una adreça ip disponible de la xarxa
- `existsIPAddress(ip:string)`: Ha de comprovar si la ip donada ja existeix a la xarxa
- `isFull()`: Ha de comprovar si hi ja hi han totes les adreces ip assignades

[3] Exemple d'ús d'interfícies marcadores al punt 7.2.2

- `createIP(ipv4:string)`: Ha de crear una adreça ip dins la xarxa
- `delete()`: Ha d'esborrar la xarxa

IXMLExportable

Defineix si una classe és exportable en format XML

- `getXMLAttributes()`: Ha de retornar els atributs de la classe preparats per ser exportats

IWired

Defineix si una classe representa una interfície de xarxa cablejada

IWireless

Defineix si una classe representa una interfície de xarxa sensefils

ILogical

Defineix si una classe representa una interfície Lògica

IVirtual

Defineix si una classe representa una interfície de xarxa virtual

- `getMasterInterface()`: Ha de retornar la interfície de xarxa master a la que pertany

IWiredVirtual

Defineix si una classe representa una interfície de xarxa virtual, concretament sobre una interfície de xarxa cablejada

IWirelessVirtual

Defineix si una classe representa una interfície de xarxa virtual,concretament sobre una interfície de xarxa sensefils

Diagrama de classes ampliat

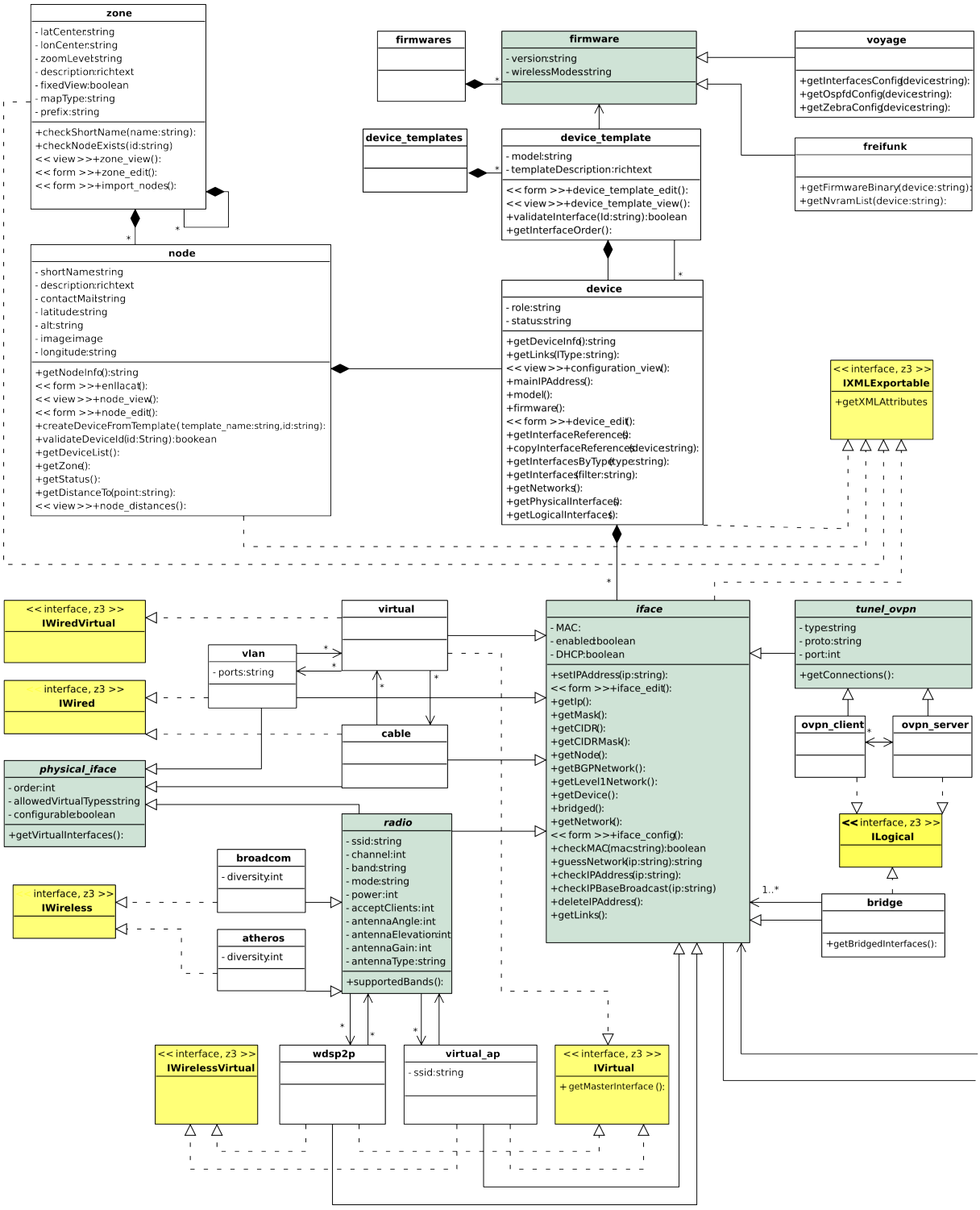


Figura 4.4: Diagrama UML 1. Part física de la xarxa

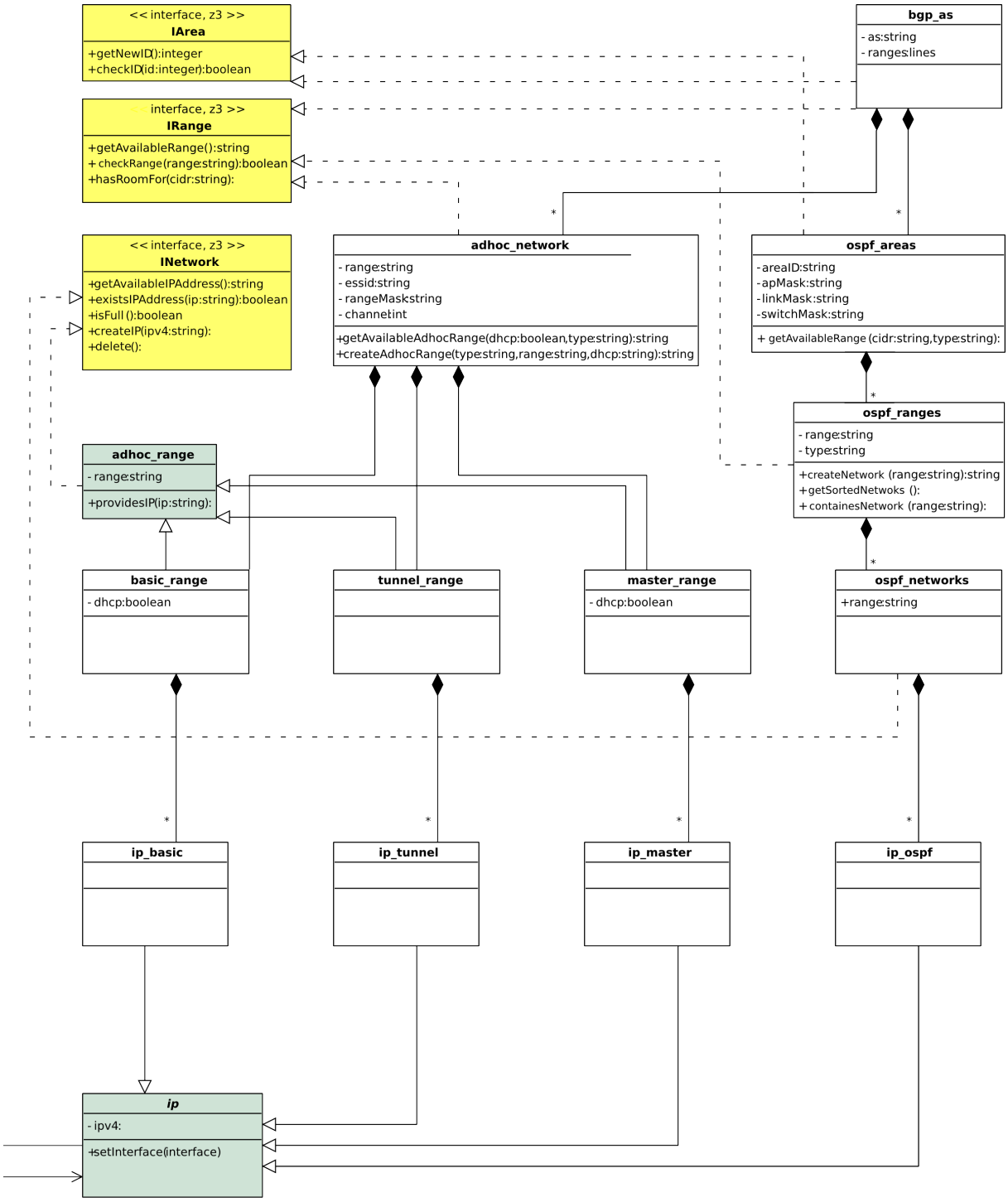


Figura 4.5: Diagrama UML 2. Part lògica de la xarxa

Canvis realitzats al diagrama de classes original

1. Les següents classes del diagrama, han guanyat una realització cap a l'interfície IXMLExportable, per tant totes elles implementaran el mètode `getXMLAttributes()`:
   - Zone
   - Node
   - Device
   - Iface
2. Degut a la limitació de les referències en ArcheTypes, les relacions presents entre les diferents subclasses de interfícies de xarxa, s'han duplicat, per tal de poder tenir la relació accessible en les dues direccions, en els casos que s'ha considerat que aquest fet podria comportar una pèrdua d'eficiència alhora de accedir a informació a través de les referències. Aquest canvi tot i només afectar a la implementació, està present en el diagrama de classes, ja que aquest es fa servir de base per generar el codi.
3. La classe virtual que representava una interfície virtual genèrica, ara passa a ser una interfície virtual sobre interfícies cable, per diferenciar-les de les interfície virtuals *virtual_ap* que tindran propietats diferents:
4. En totes les subclasses de *Iface*, s'han afegit realitzacions cap a interfícies marcadores amb l'objectiu de classificar les classes amb un tipus, per sobre de la classificació que ja s'obté amb la generalització en si mateixa.
5. Alguns mètodes han set agrupats en interfícies de manera que els seus mètodes seràn implementats per
   - Interfície *Iarea* : bgp_as, ospf_areas
   - Interfície *IRange* : bgp_as, ospf_ranges, adhoc_network
   - Interfície *INetwork* : adhoc_range, ospf_networks
6. S'ha fet una generalització sobre la classe ip, distingint així diferents classes d'ip en funció de quin tipus de xarxa pertanyen. Aquest punt també és una qüestió d'implementació, ja que aconseguim optimitzar algunes consultes, al obtenir informació del tipus de xarxa de la ip directament accedint a la ip, sense tenir que consultar en quina tipus de xarxa esta l'objecte.

Descripció dels elements nous del diagrama de classes ampliat

Broadcom *[broadcom]*
Descripció: Representa una interfície de xarxa amb chipset broadcom
Atributs :

- `diversity`: Indica a quin dels dos connectors d'antena de la radio esta connectada l'antena; `string`
- *Hereta* `ssid,channel,band,mode,power,acceptClients,antennaAngle,antennaElevation ,antennaGain,antennaType` de radio

- *Hereta* `MAC,enabled,DHCP` de iface

Mètodes :

- *Hereta* mètodes de radio
- *Hereta* mètodes de iface
- *Hereta* mètodes de physical_iface

Atheros *[atheros]*
Descripció : Representa una interfície de xarxa amb chipset atheros
Atributs :

- `diversity`: Indica a quin dels dos connectors d'antena de la radio esta connectada l'antena; `string`
- *Hereta* `ssid,channel,band,mode,power,acceptClients,antennaAngle,antennaElevation ,antennaGain,antennaType` de iface
- *Hereta* `MAC,enabled,DHCP` de iface

Mètodes :

- *Hereta* mètodes de radio
- *Hereta* mètodes de iface
- *Hereta* mètodes de physical_iface

WDS punt-a-punt *[wdsp2p]*
Descripció : Representa una interfície virtual sobre una interfície wireless, per fer connexions punt a punt usant WDS
Atributs :

- *Hereta* `MAC,enabled,DHCP` de iface

Mètodes :

- *Implementa* `getMasterInterface()` de IVirtual
- *Hereta* mètodes de iface

Relacions :

- `linked_to`: Indica quina es l'interfície master

Punt d'accés Virtual *[virtual_ap]*
Descripció : Representa una interfície virtual sobre una interfície wireless, per fer de punt d'accés per a clients
Atributs :

- `ssid`: Nom visible de la xarxa wireless, que substitueix al de la supernals radio ; `string`
- *Hereta* `MAC,enabled,DHCP` de iface

Mètodes :

- *Implementa* `getMasterInterface()` de IVirtual
- *Hereta* mètodes de iface

Relacions :

- `linked_to`: Indica quina és l'interfície master

VLAN *[vlan]*
Descripció : Representa una interfície ethernet associada a 1 o diversos connectors físics representats per numeros
Atributs :

- `ports`: una llista dels connectors físics des dels quals es pot accedir a la interfície; `string`
- *Hereta* `order,allowedVirtualTypes,configurable` de physical_iface
- *Hereta* `MAC,enabled,DHCP` de iface

Mètodes :

- *Hereta* mètodes de iface
- *Hereta* mètodes de physical_iface

Relacions :

- `linked_to`: Indica quines interfícies virtuals té

Virtual *[virtual]*
Descripció : Representa una interfície virtual que té una interfície cable com a master
Atributs :

- *Hereta* `MAC,enabled,DHCP` de iface

Mètodes :

- *Implementa* `getMasterInterface()` de IVirtual
- *Hereta* mètodes de iface

Relacions :

- `linked_to`: Indica quina és l'interfície master

Tunel OpenVPN Servidor *[ovpn_server]*
Descripció : Representa a un tunel OpenVPN que accepta connexions de túnels OpenVPN Clients
Atributs :

- *Hereta* `MAC,enabled,DHCP` de iface
- *Hereta* `type,proto,port` de tunel_ovpn

Mètodes :

- *Hereta* mètodes de iface
- *Hereta* mètodes de tunel_ovpn

Relacions :

- `linked_to`: Indica les conexions a client que te el servidor

Tunel OpenVPN Servidor *[ovpn_client]*
Descripció : Representa a un túnel OpenVPN que pot fer una única connexió a un OpenVPN Servidor
Atributs :

- *Hereta* `MAC,enabled,DHCP` de iface
- *Hereta* `type,proto,port` de tunel_ovpn

Mètodes :

- *Hereta* `getConnections()` de tunel_ovpn
- *Hereta* mètodes de iface
- *Hereta* mètodes de tunel_ovpn

Relacions :

- `linked_to`: Indica quin és el servidor d'aquest client

Voyage Linux *[voyage]*
Descripció : Representa un Sistema Operatiu basat en debian anomenat Voyage Linux. Conté les funcions necessàries per configurar-lo
Atributs :

- *Hereta* `version,wirelessModes` de firmware

Mètodes :

- `getInterfacesConfig(device:object)`: Retorna el fitxer de configuració necessari per configurar les interfícies de xarxa de l'aparell `device`
- `getOspfdConfig(device:object)`: Retorna el fitxer de configuració necessari per configurar l'enrutament OSPF de l'aparell `device`
- `getZebraConfig(device:object)`: Retorna el fitxer de configuració necessari per configurar l'enrutament general de l'aparell `device`

OpenWRT Freifunk *[freifunk]*
Descripció : Representa un firmware basat en openWRT anomenat Freifunk. Conté les funcions necessàries per configurar-lo
Atributs :

- *Hereta* `version,wirelessModes` de firmware

Mètodes :

- `getFirmwareBinary(device:objecte)`: Retorna un fitxer binari amb un freifunk ja configurat per a l'aparell `device`
- `getNvramList(device:objecte)`: Retorna la llista de paràmetres necessaris per configurar el freifunk manualment

Adreça IP OSPF *[ip_ospf]*
Descripció : Representa una adreça d'una xarxa OSPF
Atributs :

- *hereta* `ipv4` de ip

Mètodes :

- *hereta* mètodes de ip

Adreça IP bàsica *[ip_basic]*
Descripció : Representa una adreça d'una partició bàsica
Atributs :

- *hereta* `ipv4` de ip

Mètodes :

- *hereta* mètodes de ip

Adreça IP túnel *[ip_tunnel]*
Descripció : Representa una adreça d'una partició OpenVPN
Atributs :

- *hereta* `ipv4` de ip

Mètodes :

- *hereta* mètodes de ip

Adreça IP master *[ip_master]*
Descripció : Representa una adreça d'una partició master
Atributs :

- *hereta* `ipv4` de ip

Mètodes :

- *hereta* mètodes de ip

Capítol 5

Disseny del Sistema - Interficies

5.1 Criteris de disseny

5.1.1 Usabilitat

Hi ha dos criteris en els que ens hem basat per dissenyar les interfícies principals:

- L'experiència d'usuari.
- L'aplicació actual de guifi.net.

Quan parlem d'usuari ens referim a l'usuari sense coneixements tècnics que vol unir-se a la xarxa. Per tal d'aconseguir això, hem d'aconseguir que la manera de demanar les dades a l'usuari vagi acompanyada de descripcions explicatives, i quan sigui possible, fer servir valors per defecte,o fins o tot no demanar algunes dades, que no són imprescindibles pel correcte funcionament de l'aplicació, i que poden ser informades posteriorment, si l'usuari ho creu convenient i té els coneixements per fer-ho.

Quan tinguem moltes dades per recollir de l'usuari, ja sigui un usuari simple, o algú amb coneixements tècnics, procurarem 2 coses:

- Agrupar les dades que hem de recollir en seccions segons la naturalesa de les mateixes.
- Dividir el formulari de recollida de dades en diversos formularis, si es considera convenient quan hi hagi moltes dades, o siguin radicalment diferents.

Amb això pretenem aconseguir evitar el que passa actualment amb l'aplicació de guifi.net: Són molts els usuaris que usen aquesta aplicació per primera vegada i abandonen en l'intent d'afegir-se a la xarxa, i la majoria tenen la sensació de que no saben que els demanen. Parlant amb alguns d'aquests usuaris, ens vam adonar que no es que no sàpiguen la majoria de dades que se'ls hi demana, sinó que es demana massa informació, i tota d'una plegada, i també en alguns punts es mostra massa informació. Per tant l'objectiu d'aquests dos punts es justament evitar aquest problema.

5.1.2 Estil visual

En totes les interfícies s'han fet servir els estils CSS per defecte que acompanyen a Plone, així com també s'ha respectat l'estil visual de construir les interfícies i d'ordenar els elements visualment. Això s'ha pogut

fer examinant l'estructura en com estan escrits els templates de Plone, per fer-los servir de referència. Considerem que ja que no som dissenyadors, la opció més encertada és fer servir les pautes que han marcat els experts en usabilitat que han dissenyat les interfícies de Plone.

5.1.3 Dinamisme

Abans exposàvem la necessitat de dividir formularis en diverses parts per aconseguir no sobrecarregar l'entrada de dades de cara l'usuari, però dividir un formulari en diverses parts, a vegades pot ser contraproduent per diverses raons. Si es recullen dades que no depenen de cap factor aliè a les pròpies dades, no te cap mena de complicació recollir aquestes dades en diversos formularis, ja que no mantenen una relació. En canvi quan les dades que entrem depenen de dades ja existents, o fins i tot de les pròpies dades que estem entrant, es pot complicar molt.

Un bon exemple és el afecta a una de les interfícies principals d'aquest projecte, que gestiona les dades associades a les interfícies de xarxa d'un aparell. Part de la problemàtica que presenta aquesta interfície, i la solució que se n'extreu, pot ser aplicada a altres parts del projecte, per tant farem una descripció genèrica del problema:

- S'han d'entrar moltes dades que cada una depèn del valor de l'ultima que s'ha entrat.
- El volum total de les dades de l'aplicació és massa gros com per carregar-les totes de cop en la pàgina, per consultar-les posteriorment sense recarregar.
- En el procés d'anar entrant aquestes dades, totes les que hem entrat són susceptibles d'haver canviat mentre estàvem entrant la resta, i necessiten de validacions entre entrada i entrada.

Una possible solució és dividir l'entrada de dades en diversos formularis, lo qual permetria la validació entre recarregues de cada pàgina, però en aquest exemple concret, farien falta fins a 6 recarregues de la pàgina per poder entrar totes les dades: això apart de lent, genera una càrrega innecessària al servidor, i hi ha més risc de fallades i pèrdues de dades a causa de hipotètics talls o fallades de connexió.

Amb l'experiència de guifi.net, podem dir que això es converteix en un problema bastant sovint. Un d'aquests processos d'alta pot allargar-se desenes de minuts degut a les lentes recarregues, i molt sovint s'ha de tornar a començar si una recarrega ha consumit massa temps per executar-se. Amb això vull justificar el raonament, ja que no es un cas hipotètic, sinó molt real.

Per solucionar aquest cas concret, la solució recau en dissenyar interfícies dinàmiques, utilitzant una tècnica de programació anomenada AJAX.

5.1.4 Interfícies genèriques i especialitzades

Dos conceptes que són diferents, però que tenen una relació en el context següent:

L'aplicació de guifi.net actual, en el procés dels dos canvis que ha sofert, ha patit un procés de "generalització"sobretot a nivell del model de dades, però que malauradament conserva masses excepcions en funció de certes variables. Apart del model de dades, les interfícies també han seguit aquest camí de generalització, provocant un problema que descrivim a continuació:

Molt sovint, quan es vol fer un model de dades molt genèric, s'acaba per tenir que introduir alguns conceptes que poden resultar abstractes, dades extres, i les interfícies tendeixen a complicar-se. Això ha

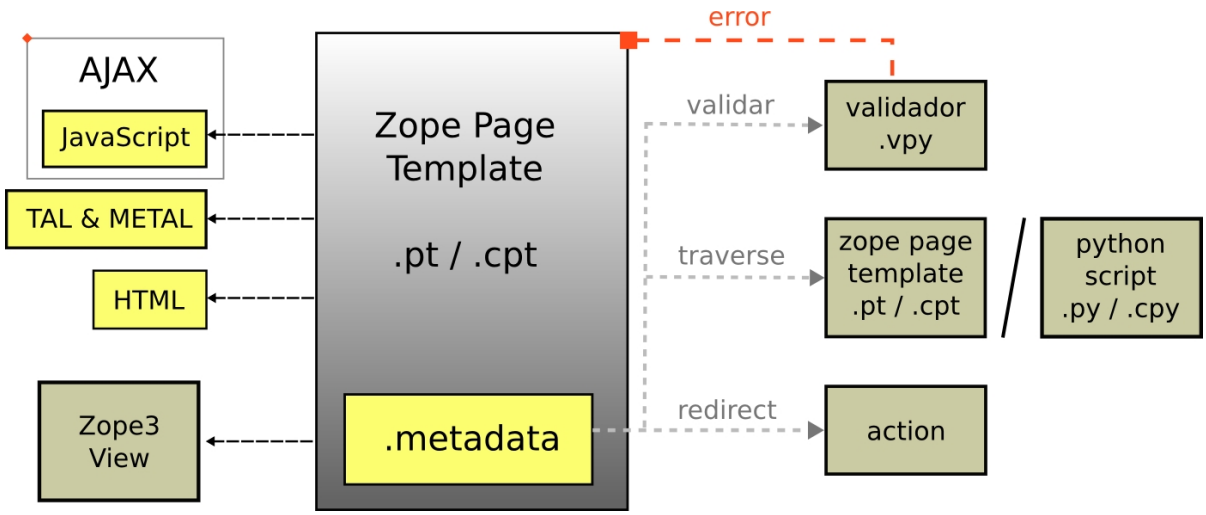
desembocat en que totes les interfícies per entrar dades a guifi.net són les mateixes tant si ets un usuari simple com un administrador. Això esta relacionat amb el que es deia al principi de demanar les dades justes, facilitant la feina als usuaris.

Actualment, pràcticament ningú sense coneixements mínims de xarxes i/o informàtica es capaç de utilitzar algunes de les interfícies més complexes de guifi.net per aquest motiu: són massa genèriques. Hi ha un problema que penja entre un model de dades que vol ser molt genèric però no ho és, i unes interfícies que en alguns casos haurien de ser més especialitzades.

Per solucionar aquest punt, en general el que s'ha fet és: Crear les interfícies genèriques, que són indispensables, però a més dissenyar interfícies especialitzades destinades als usuaris simples, per facilitar-los la feina, i evitar-los entrar en contacte amb conceptes que ni coneixen ni en tenen cap necessitat per l'objectiu que persegueixen, que es connectar-se a la xarxa.

5.2 Mecanismes de funcionament de les interfícies

Hi han diversos components que prenen part en el funcionament de les interfícies, alguns són presents sempre, altres només en certs escenaris, però tots tenen un estret lligam. A continuació s'explicarà el funcionament d'aquests components, acompanyat d'un diagrama per poder veure més clarament la relació entre els diferents components.



Estructura mecanisme de funcionament de les interficies en plone

5.2.1 Zope Page Templates

Zope Page Templates (ZPT a partir d'ara), són un eina per generar pagines web. Permeten elaborar el disseny de la pàgina fent servir les eines que el dissenyador usi normalment, sense que el codi que permet la part dinàmica de la pàgina influeixi en el procés de disseny. El llenguatge utilitzat per aconseguir això s'anomena TAL i basa la seva sintaxis en XML. D'aquesta manera, al nostre document HTML, li incorporem el namespace de TAL `xmlns:tal="http://xml.zope.org/namespaces/tal` amb el que aconseguim que el document resultant sigui un XHTML, que complirà amb els estàndard HTML i amb la sintaxis de TAL.

5.2.2 TAL (Template Attribute Language)

TAL esta basat en tags i atributs, exactament igual que HTML, amb el que s'aconsegueix una fusió perfecte amb l'HTML. Una de les idees bàsiques darrera la programació de templates amb TAL, es refereix a la substitució: omplir, modificar o substituir per complet tags o atributs HTML mitjançant una determinada sintaxis. Veiem un exemple senzill:

Posem per cas que hem dissenyat una pàgina en HTML, la qual te una capçalera, el contingut de la qual ha de ser diferent en funció des d'on es carregui la pagina. El codi HTML per la capçalera seria:

```
<h1> Titol </h1>
```

per tal de transformar aquesta capçalera en un template dinàmic faríem el següent

```
<h1 tal:content="nom_variable"> Títol </h1>
```

Amb *tal:content* ens estem referint al contingut del tag, és a dir tot el que hi hagi entre la obertura `<h1>` i el tancament del tag `</h2>`. TAL omplirà aquest espai amb el valor de *nom_variable*, substituint qualsevol altre text que hi hagués dins el TAG. En aquest cas és pot veure que el text `"Títol"` desapareixerà, per lo que pot semblar inútil de escriure'l. Aquí però s'ha de tenir en compte el punt de vista del dissenyador que per crear l'HTML, necessita posar text per poder veure com queda el disseny. Com es pot veure doncs, el llenguatge de templates, no interfereix amb la feina del dissenyador ni amb la del programador.

En el cas de tags que continguin atributs, el procediment és semblant: posem per cas un link a una ULR, en que el link és generat dinàmicament: el codi HTML seria el següent:

```
<a href=" http.//url.com">text pel link</a>
```

El template corresponent seria:

```
<a tal:content="variable_text" tal:attributes="href variable_url">text pel link</a>
```

Tenim un canvi del text del tag igual que en l'exemple anterior, i a més tenim el paràmetre *attributes* que esta indicant que substitueixi el valor de l'atribut *href* del tag `<a>` pel valor de *variable_ULR*. En aquest cas, més que substituir, el que fa es afegir l'atribut, ja que el tag per si sol no el porta. És un dels casos en que un element pot ser-hi o no, ja que no afecta en el disseny.

Altres elements de la sintaxis són les repeticions i els condicionals. Agafant l'exemple anterior, suposem que volem fer una llista de links nomes si es compleix una condició concreta. Veiem l'exemple:

```
<tal:block condition="variable_condicio" repeat="link variable_links">
<a tal:content="link/text" tal:attributes="href link/url">text pel link</a>
</tal:block>
```

Com a primer element tenim el `tal:block`, que crea un context que és valid fins al tancament del `</tal:block>`. No s'executarà res d'aquest context si no es compleix la condicio definida per *variable_condició*. Si es compleix, el *repeat* s'ocupara de iterar tantes vegades com elements tingui la variable *varaible_links* col·locant cada cop el valor corresponent en la variable *link*. En cada iteració renderitzarà tot el que hi hagi dins del bloc on estigui el *repeat*, substituint les variables que fagin falta en cada punt. Per tal que aquest mecanisme funcioni, les variables que utilitzem en els *repeat* han de ser en forma de llista Python d'elements, i els elements *link/text* i *link/url*, fan referència a un diccionari Python. En general, podem dir que per construir una estructura bàsica iterativa en que cada element contingui diversos valors, utilitzarem com a variable una llista de diccionaris, que podria tenir una forma com aquesta:

```
[ {"text" : valor_text , "url" : valor_url} ,
 ...
 {"text" : valor_text , "url" : valor_url} ,
]
```

Evidentment, en funció de la complexitat, podem fer llistes de llistes de diccionaris, diccionaris de llistes, o altres combinacions, tot depèn del que vulguem aconseguir.

Hi han altres elements tel llenguatge TAL com son *define* per definir variables, *replace* per substituir un tag sencer pel valor d'una variable. La potència dels ZPT però, no depèn en la complexitat de la seva sintaxis, sinó tot el contrari, en saber combinar tots aquests simples elements per poder crear les

interfícies que necessitem. Per complementar a TAL, les expressions que s'escriurien on en els exemples hem posat simplement noms de variables, tenen la seva pròpia sintaxis proporcionada per TALES.

5.2.3 TALES (TAL Expression Syntax)

TALES descriu la manera com podem escriure expressions que proporcionen i/o manipulen dades per a utilitzar amb el llenguatge TAL. Amb TALES podem escriure expressions per:

- Accedir a una estructura de dades com si fos un cami :

  ```
  "a/b/c"
  ```

- Determinar si existeix una variable dins un estructura:

  ```
  "exists : a/b/c"
  ```

- Negar variables:

  ```
  "not:variable"
  ```

- Definir valors com a cadenes o barreges de cadenes i variables:

  ```
  "string: cadena de text"
  "string: cadena de text ${nom_variable}"
  ```

- Executar codi Python:

  ```
  "python: linia de codi Python"
  ```

I disposa de una seria de variables amb significats específics com són:

nothing Valor que representa res, NULL.

repeat Variables referents a la iteració en curs.

here Representa l'objecte sobre el qual s'aplica el template.

template L'objecte que representa el template.

user L'usuari que ha executat el template.

request L'objecte amb les variables referents a la petició que ha executat el template.

5.2.4 Obtenció i processament de les dades als templates

Quan utilitzem ZPT podem fer-ho amb dos objectius: pagines de visualització o formularis d'edició. Ambdós casos tenen en comú el llenguatge TAL per definir els templates i el mètode per obtenir les dades necessaris pel seu funcionament. En el cas dels formularis, veurem que hi han alguns mecanismes extres que hi prenen part.

Alguns llenguatges com PHP, basen la construcció del codi HTML de les pagines en barrejar porcions de codi HTML amb porcions de codi que genera codi HTML, amb el que resulta línies i línies de codi difícil de llegir i mantenir. Hi han també mecanismes de templates per PHP, la majoria dels quals però tenen una sintaxis que s'han d'afegir elements extranys dins l'HTML que fa que no es puguin fer servir les eines de disseny web habituals per poder construir el disseny i mantenir-lo.

Una de les bases de la filosofia de Zope i Plone és separar codi de disseny, i amb el cas del disseny ho aconsegueix amb el mecanisme de Zope Views.

Obtenció - Zope Views

Zope Views no és res més que objectes Python no persistents, associats a una pagina HTML mitjançant els ZPT, que contenen tots els mètodes necessaris per obtenir i formatar les dades que necessita mostrar un template. La idea doncs, és encapsular tota la lògica relacionada amb la visualització de dades en unes classes separades. Amb això s'aconsegueix l'objectiu de mantenir separat codi i disseny, i a més s'aconsegueix separar el codi en 2 seccions: codi de visualització de dades, i el codi de les classes principals, mantenint així les classes del projecte netes de cap mena d'element relacionat amb el disseny o la visualització.

S'ha de dir però, que ens alguns casos justificats, es considera correcte obtenir dades directament desde el template, sense definir una funció en una View que ens la proporcioni. Aquest casos solen donar-se quan necessitem certes dades en un format que ja ens el proporciona directament alguna funció dels objectes implicats en aquell template. També quan simplement volem accedir a algun atribut de un objecte que no necessita cap formatació. En aquests casos, es pot utilitzar l'expressió TALES `python:` per executar el codi necessari per obtenir les dades. Aquestes expressions però, es recomana que no siguin mai de més d'una línia.

Processament de dades

Quan utilitzem ZPT per construir simples pàgines per mostrar dades, no necessitem res més excepte el mecanisme de templates. En canvi quan construïm formularis d'edició, necessitarem processar i validar dades que s'entrin a traves d'ells, i donar continuïtat als formularis un cop enviats. Podem diferenciar 3 casos que poden passar un cop tenim les dades d'un formulari:

- Validar-les.
- Guardar-les.
- Processar-les.

En qualsevol dels 3 casos el template ha de saber cap on redirigir el flux de l'execució. Per fer això, cada template te associat un fitxer amb idèntic nom al template, acabat amb l'extensió *.metadata*. Aquest fitxer conté informació de cap a on re dirigir el template en funció del resultat de l'enviament del formulari: Aquestes opcions poden ser úniques o múltiples, en funció de la quantitat de `<forms>` i /o botons que contingui la pàgina. Per cada un, es pot definir un validador .vpy i una accio a dur a terme.

A tall d'exemple, el contingut d'un arxiu .metadata podria ser el següent:

```
[default]
title=iface_edit

[validators]
validators..ModifyInterfaceLogic = validate_iface_logic_modify
validators..SaveInterfaceLogic = validate_iface_logic_save
validators..DeleteIP = validate_iface_ip_delete
validators..DeleteIPNetwork = validate_iface_ip_network_delete

[actions]
action.success..SaveInterface=traverse_to:string:iface_save
```

```
action.success..SaveInterfaceLogic=traverse_to:string:iface_logic_save
action.success..ModifyInterfaceLogic=traverse_to:string:iface_logic_modify
action.success..DeleteIP=traverse_to:string:iface_ip_delete
action.success..DeleteIPNetwork=traverse_to:string:iface_ip_network_delete
action.success=redirect_to_action:string:view
action.failure..SaveInterfaceLogic=traverse_to:string:iface_edit
action.failure..ModifyInterfaceLogic=traverse_to:string:iface_edit
```

Aquí podem veure com hi ha definit un validador en funció de les maneres en que pot ser enviat el formulari. El text que hi ha ha continuació de `validators..` es defineix en el template del formulari, com a atribut del boto al qual volem assignar aquell validador. A les accions, podem veure també en funció del boto executat (en aquest cas el text després de `success..`) cap a on s'enviara el flux d'execució. Els noms de les accions són cadenes de text definides després de `string:` i l'extensió es determina dinàmicament en funció del tipus d'script que s'estigui executant.

Controlled Python Scripts

Aquests scripts com el seu nom indica, són fitxers amb codi Python, però que no estan associats a cap classe. És per això que es recomana que el codi dins aquests scripts, no sigui molt extens, sinó que fagi ús de funcions definides a les classes principals del producte. L'objectiu és el de processar les dades que provenen d'un formulari, ja sigui per guardar-les a la base de dades, o per efectuar qualsevol altre operació. Un cop acabada la execució d'un d'aquests scripts, es passa el control al *.metadata* associat, que determinarà cap a on continua l'execució. Si en funció del boto executat, a continuació se n'ha d'executar algun altre, o bé mostrar un altre formulari, l'acció a dur a terme serà l'anomenada `traverse_to`. Aquesta acció el que fa és assegurar que les variables del formulari es mantenen al llarg de l'execució següent. Si estem en l'ultim punt d'execució i ja no necessitem cridar més templates ni scripts, l'acció que associarem en aquell punt serà la de `redirect_to`.

La programació d'aquests scripts és codi Python normal, excepte una la capçalera que ha de contenir unes determinades declaracions, per exemple:

```
## Script (Python) "iface_logic_save"
##bind container=container
##bind context=context
##bind namespace=
##bind script=script
##bind state=state
##bind subpath=traverse_subpath
##parameters=section,iface_mac, iface_ip, iface_mask=0, iface_network, iface_network_create,
##title=Alta d'un node
```

L'element més important d'aquesta capçalera és la variable *parameters*, amb la qual indiquem quines variables del formulari amb que estem treballant volem passar a l'script. És util per poder definir valors per defecte, com es veu a `iface_mask=0`, valors que només s'assignaran en cas de que la variable no existís en la petició. D'aquesta manera podem controlar casos especials que podrien acavar en una inconsistència i provocar un error.

Validadors

Un validador és un Controlled Python Script amb extensió *.vpy*, configurat com a tal en el *.metadata* d'un template. És l'encarregat de comprovar que les dades entrades en un formulari segueixen uns criteris determinats. El validador fa ús d'una variable "state"on es pot canviar l'estat de cada variable del formulari en cas que la validació falli, així com adjuntar un missatge d'error associat al camp. Aquest script retornara el control de l'execució al template original en cas de que es produeixi algun error de validació, o a l'acció corresponent definida en el *.metadata*.

Quan es produeix un error, el template que rep de nou el flux d'execució, ha d'estar programat en conseqüència per tal de poder mostrar el missatges provinents del validador. Així doncs, la programació del template en TAL, ha de tenir en compte sempre la variable state, i generar la pagina en funció si és una primera execució o si prové d'un validador.

Tal com hem dit abans, el codi dins d'aquests arxius es mantindrà al mínim possible. D'aquesta manera, mantenim el codi associat a guardar dades net de validacions, i tenim funcions específiques per validar que s'utilitzaran en aquests scripts validadors.

Un exemple de validador seria :

```
## Controlled Python Script "validate_iface_save"
##bind container=container
##bind context=context
##bind namespace=
##bind script=script
##bind state=state
##bind subpath=traverse_subpath
##parameters=iface_mac
##title=Valida paràmetres físics de l'interfície


if (context.checkMAC(iface_mac.upper())==False) and (iface_mac!=''):
    state.setError('iface_mac', 'La MAC no és correcta, repassa-la')

if state.getErrors():
    state.set(portal_status_message='Corregeix els errors marcats més avall.')
    return state.set(status='failure')
else:
    return state
```

Com es pot veure si falla la comprovació que duu a terme la funció `checkMAC`, s'associa un error a la variable `iface_mac`, amb un missatge d'error. Al final si s'han produït errors, es retorna l'estat de "failure", acompanyat d'un missatge a nivell global indicant que hi han hagut errors.

JavaScript

JavaScript és un llenguatge interpretat, semblant a Java pero NO orientat a objectes. L'ús principal de JavaScript es pot veure integrat amb pagines web, en forma de porcions de codi les quals són executades

en navegador un cop s'ha carregat la pàgina. L'objectiu d'aquest llenguatge dins una pàgina és la de poder executar petits programes en entorn web, però en l'aplicació client, aportant així un cert grau de dinamisme a les pàgines web.

En aquest projecte s'ha utilitzat JavaScript en dues seccions diferenciades:

Mapes basats en Google Maps : L'API que proporciona Google per poder fer aplicacions a mida basades en els seus mapes esta escrita en JavaScript, i per tant és el llenguatge que obligatòriament hem hagut d'utilitzar per programar aquest part.

Interfície dinàmica d'assignació de xarxes a interfícies : En aquesta secció s'ha utilitzat JavaScript per programar part de la interfície, però no aïlladament sinó dins el context d'AJAX.

AJAX

AJAX respon a l'acrònim Asynchronous JavaScript And XML. No és un llenguatje de programació nou, sinó més aviat una tècnica de desenvolupament per crear aplicacions interactives i dinàmiques en entorn web. Les aplicacions AJAX estan escrites usant JavaScript, per tant s'executen al navegador, i manté una comunicació asíncrona amb el servidor en segon pla. D'aquesta manera s'aconsegueix poder fer canvis en una pàgina sense haver de recarregar-la. Apart del llenguatge amb que se'n fa ús, AJAX és possible gràcies a la combinació de diverses tecnologies existents:

- XHTML per programar la pagina web i CSS per especificar-ne el disseny.
- DOM (Document Object Model) per poder interaccionar amb l'estructura de la pàgina generada. S'hi accedeix a traves de JavaScript.
- XMLHttpRequest és un objecte que actualment proporcionen tots els navegadors més importants, que permet l'intercanvi de dades asíncronament amb el servidor web.
- XML com a format de les dades que s'intercanvien client i servidor.

5.3 Exemples d'interfícies

A continuació presentarem exemples d'algunes de les interfícies, segons el context en que treballen:

- Interfícies auto-generades
- Interfícies personalitzades
  - Vista i edició de plantilles d'aparells
  - Vista i edició de Zones
  - Exportació XML de dades
  - Configuració d'interfícies reals d'aparells

5.3.1 Interfícies auto-generades

Són les interfícies que genera ArchGenXML automàticament a partir del model UML, en les que no hem tocat ni una sola línia de codi per crear-les. Aquestes interfícies tenen en comú que fan servir un template ZPT comú, amb el que s'aconsegueix que només modificant aquest template puguem canviar l'aspecte i/o el comportament d'una interfície. Un altre aspecte comú és que mostren les dades d'un únic objecte a la vegada, així com una llista dels objectes continguts. Com es podrà veure amb l'exemple següent, es tracta d'una simple llista d'elements, cadascun representat amb el widget que li correspon per al seu tipus de dades.

Interfícies per a gestionar plantilles d'aparells

Posem com a exemple la visualització (Figura 5.1) i edició de plantilles d'aparells (Figura 5.2), per poder-la comparar amb la versió personalitzada que mostrarem més endavant.

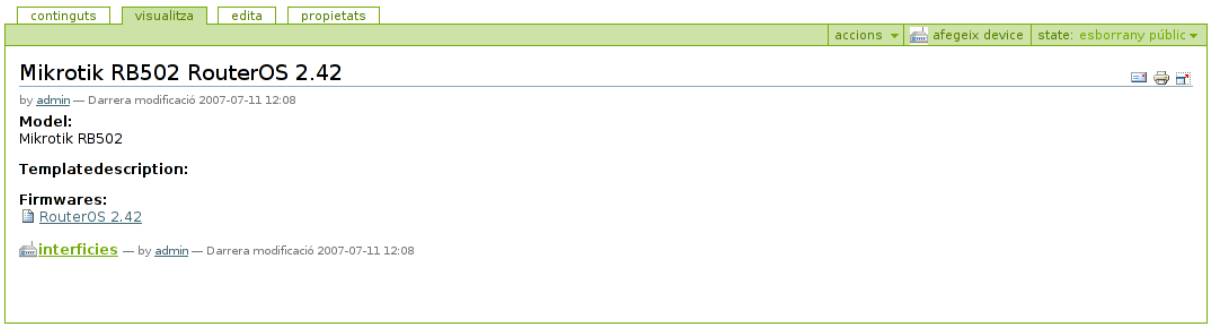


Figura 5.1: Exemple d'interfície de visualització auto-generada

5.3.2 Interfícies personalitzades

Són interfícies que no tenen que estar lligades a un sol objecte, sinó que estan programades per representar el que vulguem dins el context on s'executen. La personalització inclou tant el codi necessari per construir els templates, com el codi per guardar i validar les dades. D'aquest tipus en veurem 4 exemples diferents en relació als elements que contenen

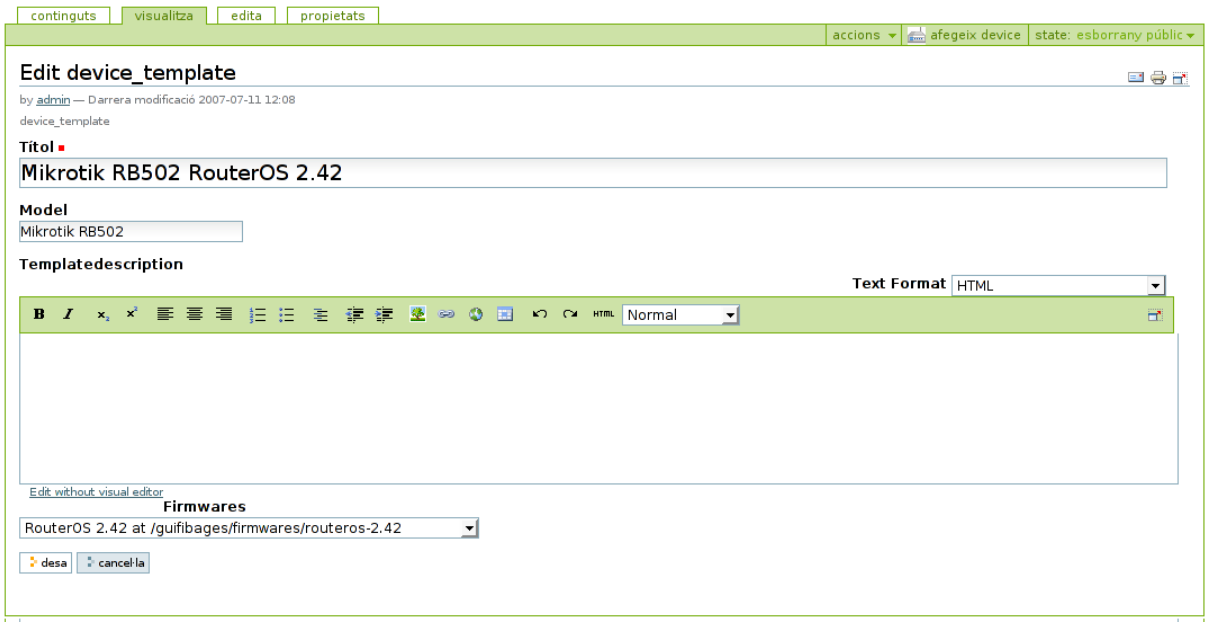


Figura 5.2: Exemple d'interfície d'edició auto-generada

Interfícies per a gestionar plantilles d'aparells

Aquest exemple ens serveix per comparar-lo amb el de la secció anterior. Una plantilla d'aparell és un objecte que conté 1 objecte del tipus aparell, que al seu temps conté un cert nombre de interfícies. Amb una interfície auto-generada, per visualitzar i editar cada nivell d'aquesta jerarquia, fa falta que accedim a la interfície concreta de cada objecte, perdent així moltes vegades la visió global del que estem gestionant.

Figura 5.3: Visualització d'una plantilla d'aparell

En aquest cas doncs podem veure que en la visualització (Figura 5.3), apart de les dades del propi objecte, es mostra una secció amb tot el contingut dels objectes fills dins la jerarquia, convenientment

classificats, i amb enllaços directes a cada element.

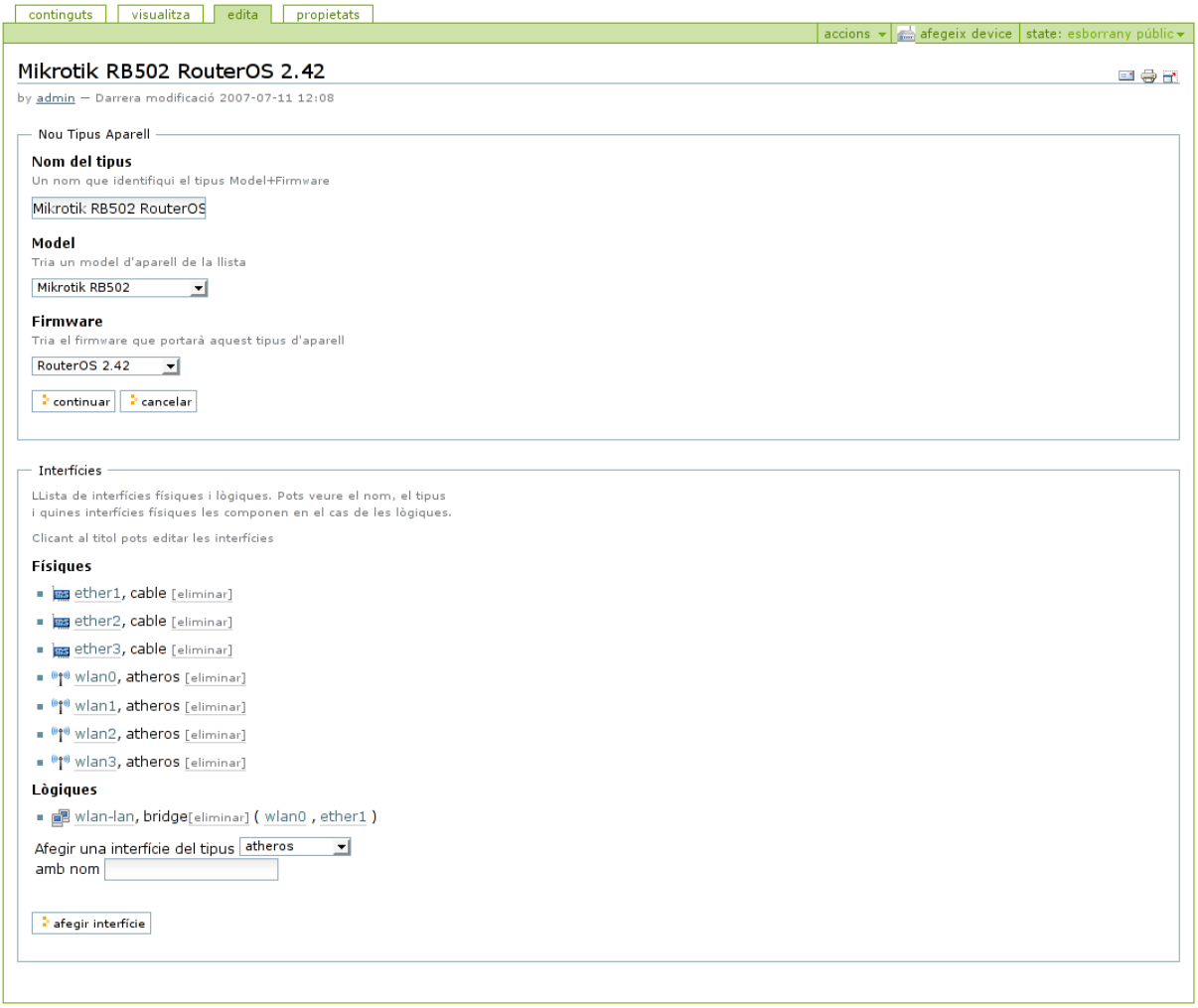


Figura 5.4: Edició d'una plantilla d'aparell

La interfície corresponent a l'edició, és molt semblant a la de visualització, mostrant els controls necessaris per poder modificar o afegir els diferents elements relacionats amb la plantilla d'aparell. Cada tipus diferent d'interfície de xarxa, té associada una interfície d'edició diferent si s'escau. Les dues interfícies següents, són exemples del que trobarem al accedir a alguna de les interfícies individualment per editar-les. Notar que aquestes interfícies no son dependents de la primera, sinó que com que estan associades a l'objecte, hi podem accedir-hi des de qualsevol lloc.

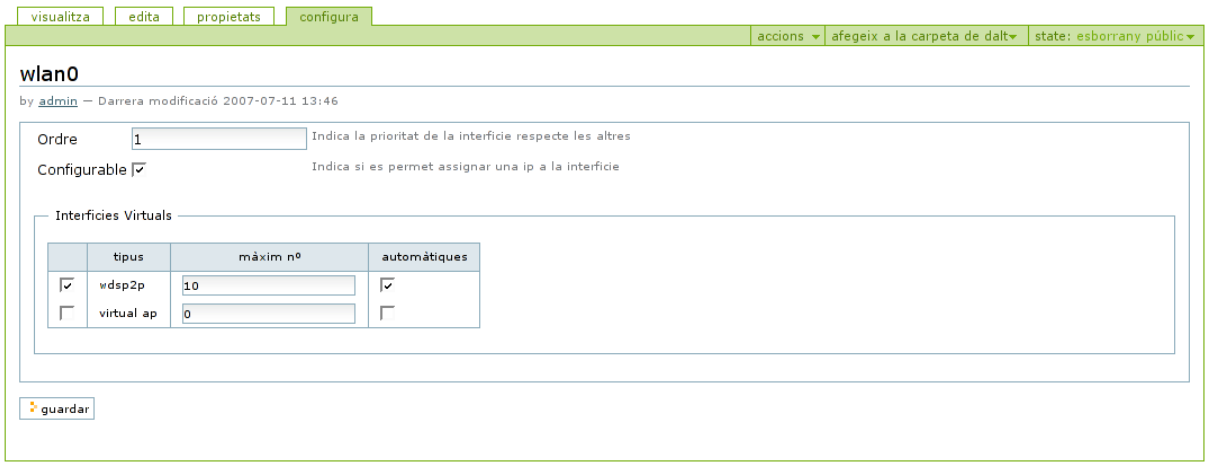


Figura 5.5: Configuració d'una interfície física

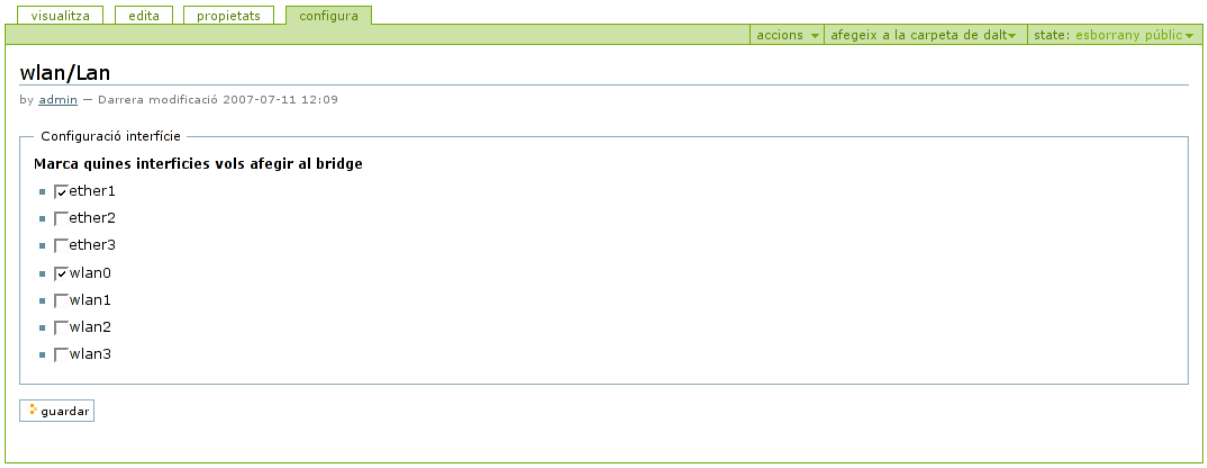


Figura 5.6: Configuració d'un pont entre interficies

Interfícies per a gestionar zones

En aquests exemples, es mostra l'ús de JavaScript i de l'API de GoogleMaps, per mostrar d'una forma visual les dades d'una zona geogràfica. Apart de les funcions per defecte dels mapes de Google, s'ha afegit certa funcionalitat necessària pel projecte.

En la interfície de vista (Figura 5.7) l'usuari pot veure gràficament l'extensió d'una zona geogràfica sobre un mapa real, així com tots els nodes que es troben dins aquesta extensió, i els enllaços que puguin existir entre ells. Complementaria al mapa, hi ha una llista de les zones filles dins la actual, i una llista dels nodes que pertanyen directament a aquesta zona, si n'hi han.

L'interfície de vista esta pensada per poder interactuar amb el mapa, de 3 maneres diferents:

- Desplaçar i fer zoom utilitzant el ratolí per tal de poder veure altres punts del mapa.
- Visualitzar distàncies entre elements, ja siguin nodes, o punts arbitraris del mapa utilitzant els marcadors mòbils. (Els nodes són els marcadors vermells, i els mòbils els de color groc i verd.)
- Comprovar perfils d'elevació entre punts del mapa, mitjançant els marcadors mòbils. (Figura 5.8)

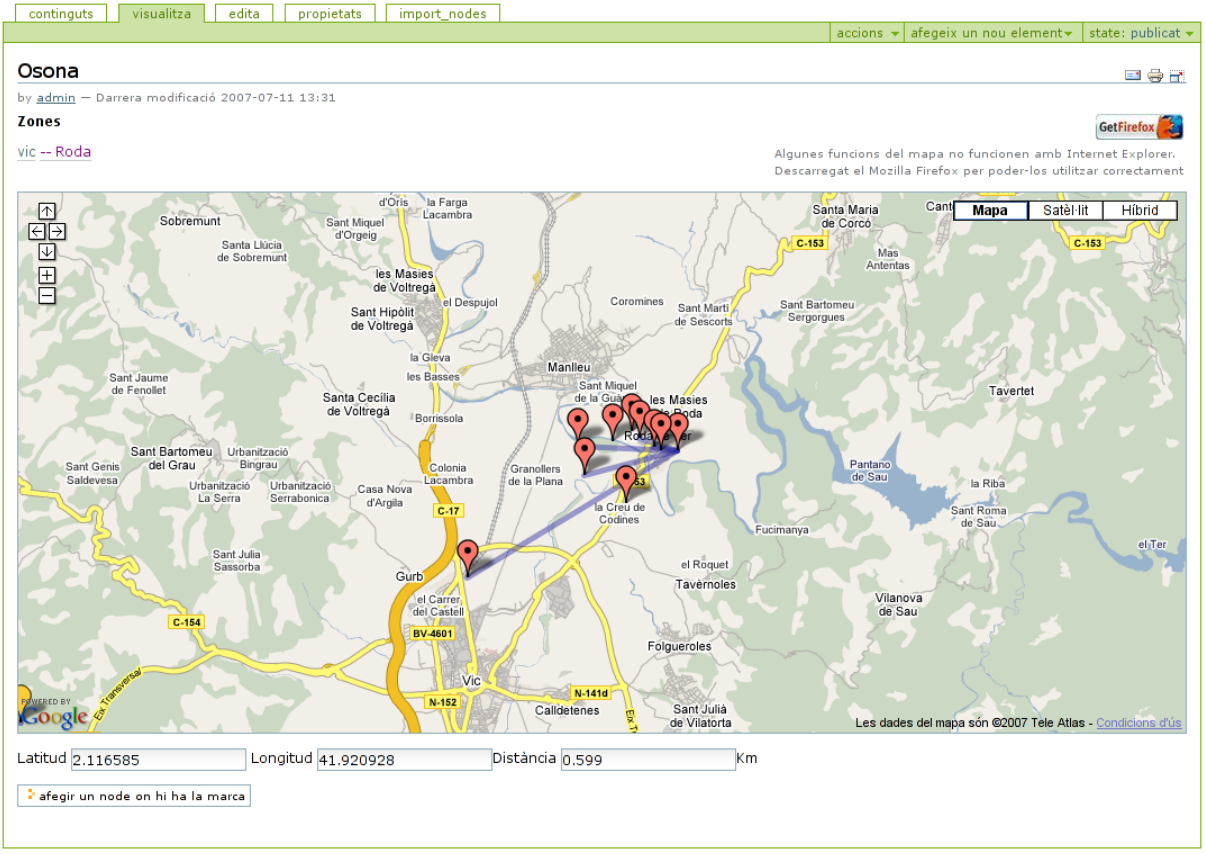


Figura 5.7: Visualització d'una zona

I per últim en la interfície d'edició i alta d'una zona (Figura 5.9) és pot veure com un lligam entre els formularis de text i el mapa mitjançant JavaScript, fa que puguem modificar el valor dels camps, simplement utilitzant el ratolí per desplaçar el mapa al lloc i zoom exacte que vulguem. També es permet la modificació manual de cada camp, ja que es validen tots ells independentment de com s'hagin entrat.

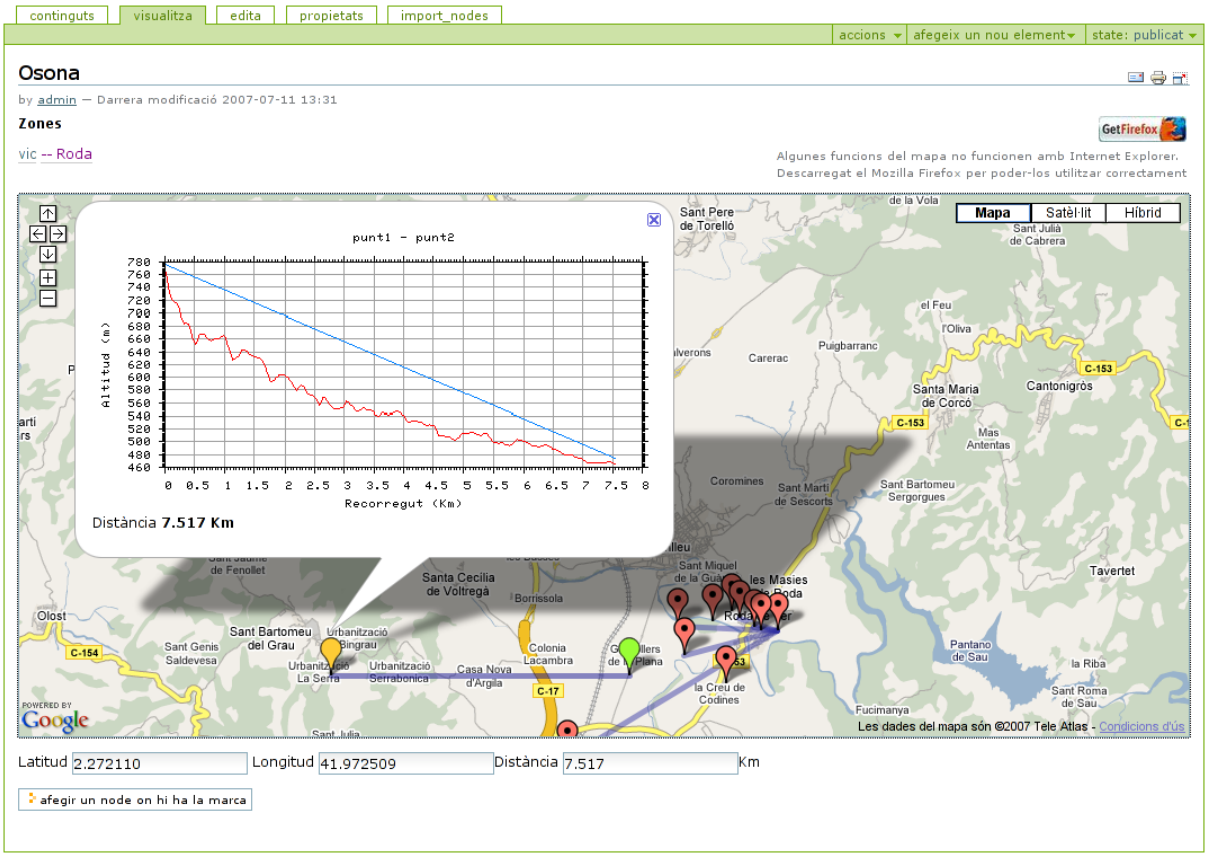


Figura 5.8: Mesura de distàncies i perfils dins d'una zona

Configuració d'interfícies reals d'aparells

Aqui és on és fa ús de la tecnologia AJAX explicada en capítols anteriors. L'interfície (Figura 5.10) consta de diversos elements, en un primer moment ocults, que es van activant en seqüencia a mesura que l'usuari va triant les diferents opcions que se li mostren.

Podem veure 2 parts diferenciades, la primera és on es configuren els paràmetres físics de l'interfíciea. Aquesta part és pràcticament estàtica, excepte per la llista desplegable corresponent al canal, que esta en funció del valor del camp Banda. La segona part, correspon als paràmetres d'adreçament IP i enrutament de la interfície, i és completament dinàmica, en funció de l'opció escollida.

En la figura 5.11 podem veure una porció de la interfície ,corresponent al final de la seqüència produïda després de prémer el boto "Assignar xarxa nova". Cada una de les llistes, apareix al triar un valor de la llista immediatament superior, resultant com a final l'aparició del botó "assigna una xarxa i una ip en aquest punt". En cada un d'aquests passos, és on entre en joc AJAX: per cada valor que triem en una de les llistes desplegables, la pàgina fa una petició XMLHttpRequest al servidor, el qual li retorna les dades que necessita per construir la següent llista de valors. Un exemple de les dades que retorna el servidor, corresponent a la tria d'un valor de la primera es pot veure al llistat 5.1 .Cada una de les llistes que necessita de dades del servidor, te associada una funció programada fer la petició corresponent, i per interpretar l'XML retornat pel servidor per omplir la llista de valors.

Per últim, la porció de la interfície corresponent a triar la opció de "Enllaça a un altre aparell"(Figura

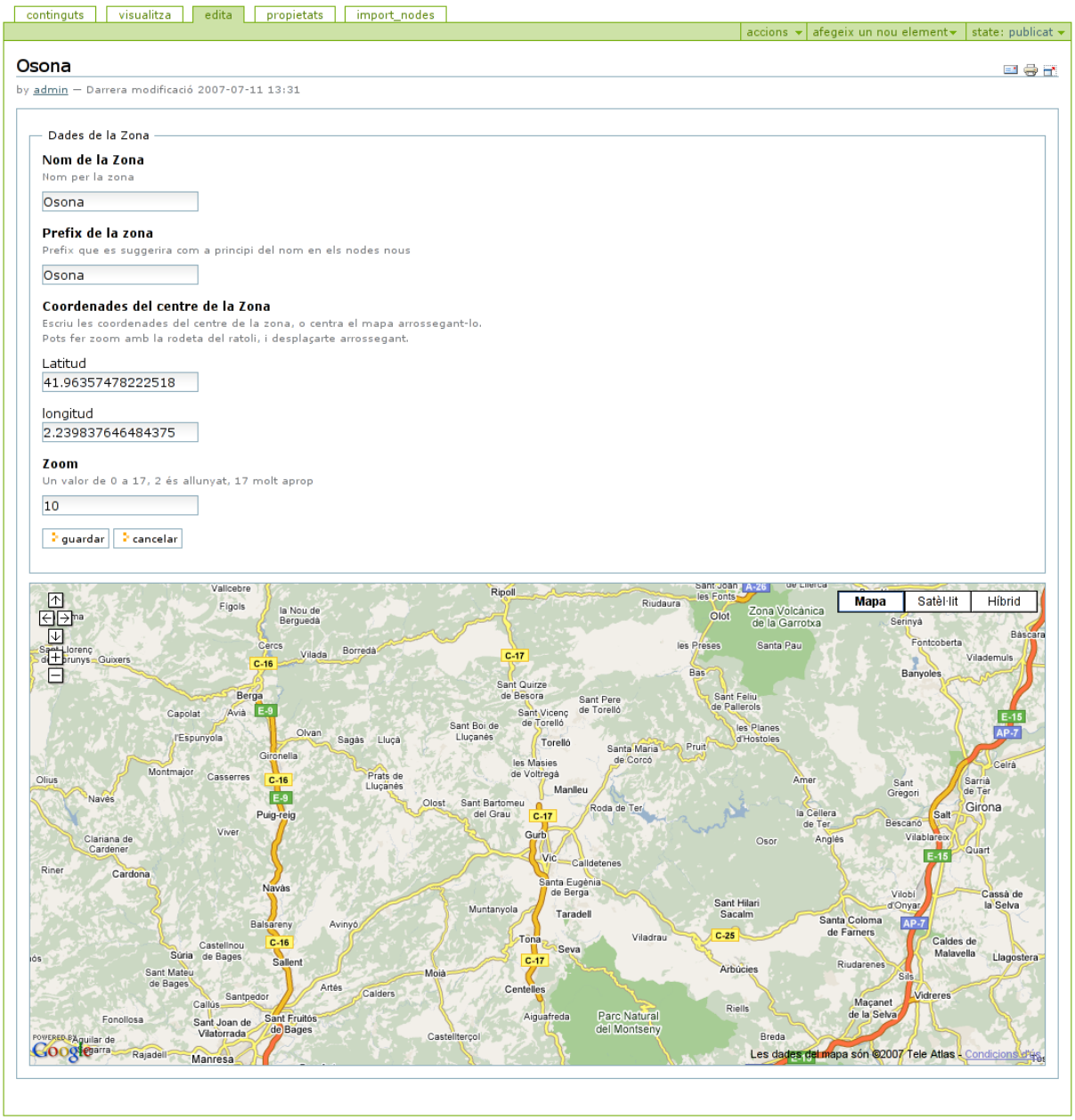


Figura 5.9: Edició d'una zona

5.12), seguint el mateix mecanisme AJAX, però amb les opcions corresponents per l'acció. Es pot veure que hi ha dues columnes amb opcions, la segona de les quals permet assignar una xarxa nova com en l'interfície anterior, però en aquest cas associada a un enllaç. Aquesta segona columna només apareix si és necessària, en funció dels valors que s'hagin triat a la columna de l'esquerra.

Llistat 5.1: Resposta a un XMLHttpRequest solicitant arees OSPF

```
<xml version="1.0">
<level1_networks>
    <level1_network title="Troncal" area_id="0"
                    type="ospf_areas" id="troncal"></level1_network>
    <level1_network title="Area␣OSPF␣Roda" area_id="1"
                    type="ospf_areas" id="area-ospf-roda"></level1_network>
</level1_networks>
</xml>
```

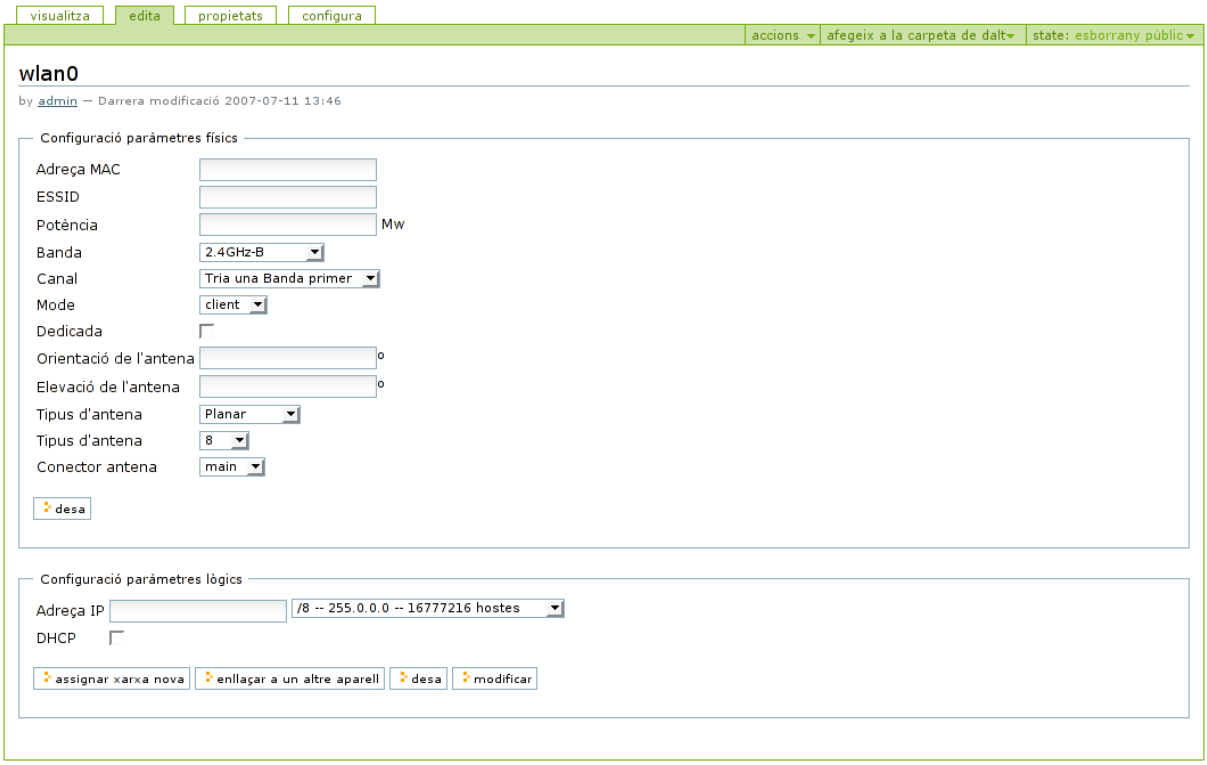


Figura 5.10: Edició d'una interficie wireless

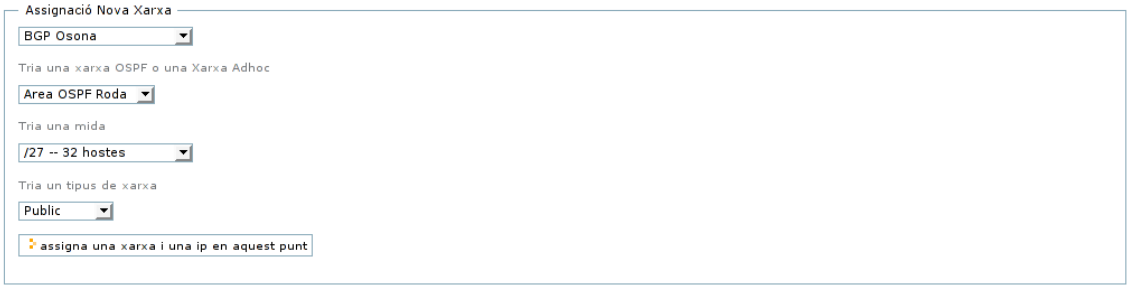


Figura 5.11: Configuració d'una xarxa nova

Figura 5.12: Enllaç a un altre aparell

Exportació XML de dades

Aquesta és una interfície diferent a les altres, únicament per visualització, formada exclusivament per text pla, sense cap element gràfic. L'objectiu és el de proporcionar recursivament les dades d'un punt

qualsevol de la jerarquia d'objectes, formatada en XML [1].

Llistat 5.2: Exportació XML d'un node

```
<xml version="1.0">
    <node lat="41.970595" lng="2.301378" id="RodaSUpernode2" title="RodaSupernode2">
        <device type="device" id="RodaSUpernode2device1" title="RodaAP3">
            <interface type="cable" id="ether1" name="ether1"></interface>
            <interface type="cable" id="ether2" name="ether2"></interface>
            <interface type="cable" id="ether3" name="ether3"></interface>
            <interface type="atheros" id="wlan0" name="wlan0"></interface>
            <interface type="atheros" ip="10.138.33.193/27" id="wlan1" name="wlan1"></interface>
            <interface type="atheros" id="wlan2" name="wlan2"></interface>
            <interface type="atheros" id="wlan3" name="wlan3"></interface>
            <interface type="bridge" id="wlan-lan" name="wlan/Lan">
                <bridged_interface name="wlan0"></bridged_interface>
                <bridged_interface name="ether1"></bridged_interface>
            </interface>
        </device>
        <device type="device" id="rodaap1" title="RodaAP1">
            <interface type="cable" id="ether1" name="ether1">
            <interface type="cable" id="ether2" name="ether2">
            <interface type="cable" id="ether3" name="ether3">
            <interface type="atheros" id="wlan0" name="wlan0">
            <interface type="atheros" ip="10.138.33.225/27" id="wlan1" name="wlan1">
            <interface type="atheros" id="wlan2" name="wlan2">
            <interface type="atheros" id="wlan3" name="wlan3">
            <interface type="bridge" id="wlan-lan" name="wlan/Lan">
                <bridged_interface name="wlan0"></bridged_interface>
                <bridged_interface name="ether1"></bridged_interface>
            </interface>
        </device>
    </node>
</xml>
```

[1] Detalls sobre la implementació a la secció 7.2.2

5.4 Dificultats

Hi han certs aspectes relacionats amb alguns dels llenguatges i tècniques exposades en aquesta secció, que ens han comportat certa dificultat a l'hora d'implementar les interfícies. La dificultat recau bàsicament, en que alguns dels conceptes i filosofies d'aquestes tecnologies, són radicalment diferents d'altres sistemes que havíem utilitzat anteriorment com pot ser PHP. La dificultat ha estat en gran mesura, la corba d'aprenentatge que requereix Zope i Plone i en particular els ZPT.

Estàvem acostumats massa sovint a barrejar disseny i contingut, amb aquest entorn, això no és acceptable. El resultat però és positiu, ja que al entendre el funcionament de tot plegat, produeix un canvi en la manera de pensar i abordar la programació de interfícies en qualsevol altre context. La dificultat inicial que pot comportar dissenyar un sistema separant completament visualització, codi i contingut, és veu recompensada amb escreix, un cop s'arriba a un grau alt de complexitat, ja que simplifica tot el procés.

Una altre dificultat trobada és que amb la versió estable de Plone que hem utilitzat per desenvolupar el projecte (2.5.3), no hi havia cap mena de suport per desenvolupar interfícies amb AJAX. Per tant la part d'interfícies programada amb AJAX, ha quedat deslligada en certa manera del mecanisme de formularis de Plone, ja que aquest mecanisme es basa en la recàrrega de les pagines per poder fer validacions i processar dades. Per suplir això, s'ha delegat part de la feina en forma de codi JavaScript d'aquestes interfícies, i en validacions al final de tot el procés.

Això ha comportat trencar en un punt unes de les premisses que planteja Plone, que és que MAI pot haver-hi una funció que retorni codi HTML (tornem a la filosofia de no barrejar codi i visualització). De totes maneres, la funció en qüestió que retorna codi HTML, no forma part del codi principal de l'aplicació, sino que s'ha quedat en el context del template que conté la interfície AJAX.

Capítol 6

Disseny del Sistema - Base de dades orientada a objectes

La base de dades usada per la realització d'aquest projecte és una base de dades orientada a objectes, anomenada ZODB, que està inclosa dins el paquet de Zope [1]. S'ha triat utulitzar aquesta base de dades, ja que pel metode de desenvolupament escollit, era necessari l'us d'aquesta base de dades. Apart, donada la forta orientació a objectes del model que es representa en aquest projecte, una base de dades orientada a objectes era la opció més natural.

6.1 Zope Object DataBase

ZODB és un sistema de persistència transparent per a objectes del llenguatge python. Està basat en el sistema de persistència d'objectes de Python, al que afegeix mecanismes de concurrència, resolució de conflictes, replicació, cache i transaccions. A diferència de les bases relacionals en el qual és necessari un llenguatge per poder inserir un objecte dins una base de dades com pot ser SQL, els objectes s'afegeixen a la zodb tal com són, sense necessitar cap transformació. Una bona manera de entendre com funciona ZODB per emmagatzemar les dades, és pensar en un sistema de fitxers organitzat en carpetes i arxius formant una jerarquia, en que en comptes de carpetes i arxius, tenim objectes (contenidors, i continguts).

El funcionament de ZODB es podria resumir amb la següent explicació:

Es transforma un objecte en persistent, mitjançant herència de la classe Persistence.persistent . Aquest objecte llavors pot ser guardat i llegit directament d'un medi físic, i queda integrat dins la base de dades corresponent. ZODB s'encarrega de detectar canvis en un objecte, de manera que quan s'executa alguna funció que modifica un objecte guardat a la ZODB, aquest objecte quede marcat com a "brut". En un moment determinat després de la modificació, s'executa una petició per part del programa a ZODB, que fa que els canvis queden guardats a disc. Aquest procés és el que s'anomena confirmar una transacció. Les transaccions, des de que s'inicien al modificar un objecte, poden ser cancel·lades o fins i tot recuperar l'estat últim dels objectes afectats que hi havia avans de confirmar-la.

Amb un mecanisme de transaccions s'assegura que una base de dades no quedi corrupte per culpa de fallades de software o hardware. Un mecanisme de transacció pot tenir 4 propietats, de les quals ZODB

[1] Veure la secció 3.2.2

proporciona les 4:

Atomicitat Qualsevol canvi fet a les dades durant una transacció es du a terme íntegrament o es descarta. O bé tots els canvis es duen a terme, o cap. Si un programa falla després d'haver fet modificacions però sense haver confirmat la transacció, tots els canvis són eliminats.

Consistència Significa que la base de dades s'assegura que cada transacció executa només canvis vàlids en relació a l'estat de la base de dades.

Aïllament Donats dos processos, en transaccions diferents, cap dels dos processos pot veure els canvis que provoca l'altre fins que no es confirmin les transaccions.

Durabilitat Assegura que un cop una transacció s'ha confirmat, una fallada posterior no provocarà la pèrdua o la corrupció de les dades involucrades.

6.2 ZODB en Plone

Per utilitzar la base de dades ZODB en Plone no ens cal saber-ne ni el funcionament, ni pràcticament la existència, ja que Plone proporciona el seu propi mecanisme a través d'una API per poder guardar i recuperar objectes de la base de dades transparentment. Dividirem aquest mecanismes en 2 segons serveixin per accedir o per guardar les dades:

6.2.1 Escriptura

Tal com hem vist anteriorment, Plone basa el funcionament de productes basats en ArcheTypes en crear objectes nous a mida, que anomenarem *content_types*. Per tal de poder crear instàncies d'aquests objectes i manipular els seus atributs utilitzarem una eina anomenada portal_factory, el ReferenceEngine, i els mutators.

Portal_factory

Portal factory manté informació de tots els *content_types* disponibles dins Plone, i és l'encarregat de crear els objectes dins la base de dades. Conté també informació sobre la jerarquia permesa, limitant així la creació de determinats *content_types* dins d'altres. Portal_factory és una eina en forma d'objecte no persistent, que es pot instanciar en qualsevol moment en un programa quan necessitem crear un objecte. La sintaxis per crear un objecte seria:

```
id = self.generateUniqueId("nom")
objecte = self.invokeFactory(type_name="nom_del_tipus", id=id, atribut=valor)
```

El procés és tant simple com generar un identificador únic mitjançant les eines de Plone com a l'exemple, o utilitzant alguna funció pròpia per generar els id's. Amb aquest identificador podem executar la funció invokeFactory que tenen tots els objectes instanciables de Plone, indicant-li com a paràmetres obligatoris el tipus de *content_type* que volem crear dins l'objecte pare (representat per *self* a l'exemple), i l'identificador. Com a paràmetres opcionals, podem especificar els valors de cada un dels atributs de l'objecte si s'escau.

La manera com esta dissenyat portal_factory, fa que mitjançant el mecanisme de les transaccions, els canvis produïts per invokeFactory, no es confirmin fins al final de la funció o script on esta inclosa la comanda.

Mutators

Els mutators són funcions creades automàticament al definir un nou content_type, que permeten l'accés als atributs d'un objecte. Per definició la forma d'un mutator és la següent: Donat un objecte de nom "objecte"amb un atribut de nom "atribut"el mutator corresponent serà una funció objecte.setAtribut(valor), es a dir la paraula clau "set"més el nom del *content_type* amb la primera lletra en majúscula. Els canvis produïts pels mutators, estan subjectes a les transaccions de la mateixa manera que el portal_factory.

ReferenceEngine

Per a guardar les referències entre objectes, es fa servir una tècnica diferent: Per poder crear una referència entre objectes, cal que els dos objectes existeixin i hem de saber el nom de la relació que estem creant,

d'acord amb com hem definit la relació al diagrama UML. Un cop sabem aquestes dades, si volem crear una referència de nom "relacionat_amb"de l'objecteA a l'objecteB, farem el següent:

```
objecteA.addReference(objecteB,relationship='relacionat_amb')
```

Recordar que aquest mecanisme de referències només permet l'accés en una sola direcció. És a dir que en l'exemple anterior, no ni ha manera de saber que objecteA te una relació amb objecteB, excepte que consultant explícitament les referències de objecteA.

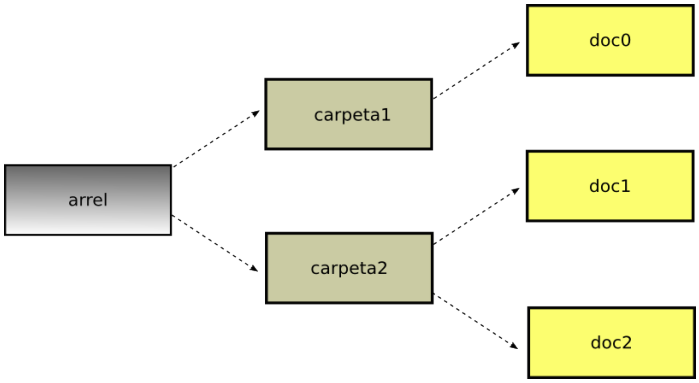
6.2.2 Lectura

Per accedir a les dades, hi han diversos sistemes en funció del context i del resultat que volguem obtenir. Distingirem en dos grups:

- Accés directe als objectes
- Accés indirecte mitjançant catàleg

Accés directe

Per l'exemple, suposem que tenim una jerarquia formada pels següents objectes:



Suposem que en un punt del programa s'ha executat una funció de l'objecte arrel. En el codi d'aquesta funció, l'objecte arrel serà representat per *self*. si coneixem els noms dels objectes que conté arrel, podem accedir-hi de les seguents maneres, sent totes vàlides en qualsevol context, donant com a resultat l'obecte directament:

```
Doc1 = self.carpeta1.doc1
carpeta2 = self["carpeta2"]
carpeta2 = getattr(self,"carpeta2")
doc2 = getattr(self["carpeta2"],"doc2")
```

En casos en que no coneixem a priori els noms dels objectes als que volem accedir, podem usar la funció `contentItems()`, que ens retornarà una llista dels objectes que conté, en que cada element serà una tupla amb 2 elements: el tipus de l'objecte i l'objecte: Per exemple:

```
resultat = self.contentItems()
> [ ('carpeta', <objecte carpeta a /arrel/carpeta1>), ('carpeta', <objecte carpeta a /arrel/carpeta2>)]
```

```
resultat = self.carpeta1.contentItems()
> []
```

Per accedir a aquests elements podem usar iteradors per recórrer la llista, o accedir als elements,mitjançant un index numèric, tant per la llista com per la tupla:

```
carpeta1 = resultat[0][1]
```

si volguéssim simplement conèixer el tipus de l'objecte

```
tipus = resultat[0][0]
> 'carpeta'
```

Aquesta funció pot ser costosa innecessàriament en funció del que necessitem fer, ja que retorna els objectes directament. Hi ha una funció amb idèntic funcionament anomenada `contentIds()`, que en comptes de retornar els objectes retorna els identificadors dels objectes, que podem usar posteriorment per accedir a ells si ens cal.

```
resultat = self.contentIds()
> [ ('carpeta', 'carpeta1'), ('carpeta', 'carpeta2')]

carpeta1 = self[resultat[0][1]]
```

Per poder accedir a un objecte pare, podem importar la funció `aq_parent()` de la llibreria `Acquisition`, per tal de poder recuperar el pare d'un objecte:

```
arrel = aq_parent(carpeta1)
```

I per últim, per poder accedir als objectes referenciats a traves del ReferenceEngine, hem de consultar l'objecte en qüestió, per comprovar si te referències amb la funció getRefs() aquesta funció la podem limitar segons el nom de la referència que busquem, afegint el paràmetre getRefs(relationship="nom_relacio"). Aquesta funció retorna directament una llista en que cada element és un objecte referenciat.

Accés amb catàleg

Existeix una eina anomenada *portal_catalog*, que s'encarrega de mantenir index optimitzats dels continguts de les bases de dades. Podríem dir que és l'equivalent d'una consulta SQL, però no pel funcionament, sinó pel concepte de fer una consulta sobre un conjunt extens de dades. El catàleg funciona a base de indexar uns valors base de cada objecte com poden ser l'identificador, el títol, la posició dins la jerarquia (path), i el tipus, i a més podem definir quins atributs extra de cada objecte volem afegir al catàleg. D'aquesta manera, les consultes al catàleg no es fan en base a un objecte concret, sinó al conjunt de les dades indexades al catàleg. Podem filtrar-hi optimitzar els resultats d'una cerca al catàleg,en funció dels paràmetres que li indiquem per cercar. Els més habituals són per identificador (*id*), per jerarquia (*path*), o per tipus (*portal_type*). El resultat d'una cerca, és una llista de *brains*, un tipus d'objecte especial, que inclou tota la informació present al catàleg d'aquell objecte, així com una funció per accedir al objecte real. Per exemple :

```
resultat1 = portal_catalog.searchResults(portal_type="carpeta")
resultat2 = portal_catalog.searchResults(portal_type="document")
resultat3 = portal_catalog.searchResults(portal_type="document", path="/arrel/carpeta1")
```

Després d'això, *resultat1*,ens retornarà una llista amb 2 *brains*, corresponents a les 2 carpetes del diagrama; *resultat2* contindrà els 3 documents del diagrama, i *resultat 3*, només el doc0, ja que l'hem restringit per *path*. notar que totes les cerques són recursives, si no s'indica el contrari.

6.2.3 Elements UML i la seva traducció

Al treballar amb una base de dades orientada a objectes, com es desprèn de les explicacions d'aquest capítol, es pot veure que una de les grans avantatges d'aquest sistema és el fet de no haver de programar un mòdul de traducció de dades com es faria si utilitzéssim una base de dades relacional. Els objectes queden guardats transparentment a la base de dades. Això, afegit a la tècnica usada per desenvolupar la base del projecte [2], fa que sigui important entendre com es comporta cada element UML del model, en relació a l'accés de les dades

Composicions Són potser l'element més usat i més bàsic, ja que representen la jerarquia d'objecte. Una composició A $\diamondsuit \longrightarrow$ B, significarà que l'objecte A serà contenidor de l'objecte B. Plone interpretarà qualsevol objecte que pugui contenir a 1 o més objectes com a BaseFolder, que serà la classe directa de la que heretarà l'objecte, A en aquest cas. En el cas de B, si no té cap altre classe composada, B serà creat en base a la classe BaseContent. Tot objecte composat, només podrà ser creat dins la classe contenidora on estigui definit.

Generalitzacions Totes les classes d'un model seran *content_types* diferents, amb un shema Archetypes separat, independentment si són classes normals, superclasses o sublasses, a nivell d'accés de dades no es notarà cap diferència en aquest sentit.

Associacions Les referències d'una classe a una altra, són guardades dins cada objecte, i s'accedeixen a través d'ell (vist a la secció anterior)

Herència ArchGenXML suporta herència simple i herència múltiple, El resultat d'heretar funcions i/o atributs, es tradueix en la definició dels *content_types*, que un cop instal·lats al Plone, crea els objectes amb tots els atributs heretats. Un canvi en l'herència de les classes, implica que hem de reinstal·lar aquell *content_type* al Plone, per tal que els canvis es vegin reflectits.

[2] Veure apartat ArchGenXML de la secció 2.2.3

6.3 Avantatges i inconvenients

Com tot sistema, presenta avantatges que hem anat comentant, així com algun inconvenient. S'ha de tenir molt clar abans d'escollir usar una base de dades orientada a objectes, el model de dades que hem de guardar, així com quin tipus d'accés necessitem sobre les dades. En el nostre cas, el model es per naturalesa jeràrquic, i s'adapta perfectament a una BDOO, i la majoria dels accessos a les dades és fan seguint la jerarquia.

Posant un exemple genèric:

- Volem accedir a un objecte que esta al final d'una determinada jerarquia coneguda amb 5 nivells, cada un corresponent a un tipus diferent de dades i amb una desena d'elements. Per accedir a l'element buscat, en cada nivell de la jerarquia, tan sols tenim accés als elements d'aquell nivell per continuar, per tant a cada nivell com a molt farem 10 accessos a la base de dades. Així fins a arribar al final. En una base de dades relacional, típicament, tots els elements de cada tipus de dades anomenat abans, formarien part d'una sola taula. Això vol dir que les successives consultes per arribar al final de la mateixa jerarquia, han de tractar amb totes les dades de cada taula, encara que conceptualment estan fora de la jerarquia on estem treballant.

En un context amb un gran volum de dades i una marcada jerarquia, esta clar que una base de dades orientada a objectes serà molt més eficaç en aquest tipus de consultes.

El problema amb les BDOO, en concret amb la ZODB, arriba quan el tipus de consulta que necessitem fer no segueix la jerarquia definida en el model de dades. Per exemple, en el projecte d'ha definit la jerarquia $\longrightarrow$**zona**$\longrightarrow$**node**$\longrightarrow$**aparell**$\longrightarrow$**interfície**. Si volem fer una cerca de tots els aparells d'un cert tipus, no tindríem més remei que recórrer totes les possibles branques on hi podria haver aparells, buscant i llistant totes les coincidències. En una base de dades relacional, simplement hauríem d'accedir a la taula específica que contingues en aquest cas els elements *aparell*, i llistar els elements coincident. Aquí es veu clar que els accessos que faria una BDOO serien molts més que els que faria una BD Relacional.

Per solucionar aquest tipus d'accessos, Plone conta amb l'eina portal_catalog comentada anteriorment. En certa manera, podem pensar en el conjunt dels índexs que conté el catàleg, com en una base de dades relacional, que conté un subconjunt de la base de dades OO. El portal_catalog, no només és útil en casos com l'exposat a l'exemple, sinó que des dels desenvolupadors de Plone recomananen l'us extensiu del portal_catalog per accedir a les dades. Això pot portar a confusions, que poder fer arribar a plantejar el perquè fer servir aquesta mena de "segona base de dades"per accedir als objectes. La raó és molt simple:

- Agafem com a context un objecte concret, del qual necessitem saber quins altres objectes hi són continguts. Dels atributs que necessitem d'aquesta llista d'objectes, en dependrà l'us o no del portal_catalog. Si sabem del cert que hem d'accedir a atributs de tots els objectes, i aquests atributs no estan indexats al portal_catalog, llavors en podem prescindir, ja que un accés directe als objectes serà igual o més ràpid. En canvi si només hem d'accedir a uns quants d'aquests objectes, o bé els atributs que necessitem estan indexats al portal_catalog, llavors l'accés a través d'aquest serà molt més ràpid i eficient que un accés directe als objectes.

Capítol 7

Implementació

7.1 Algorismes desenvolupats

7.1.1 Assignació de subrangs

Una de les parts més crítiques de l'aplicació és la encarregada d'assignar adreces ip automàticament a les diferents interfícies de xarxa que en reclamin. Aquest procés implica diverses parts algunes amb més complexitat que les altres. Premem com a exemple el següent diagrama:

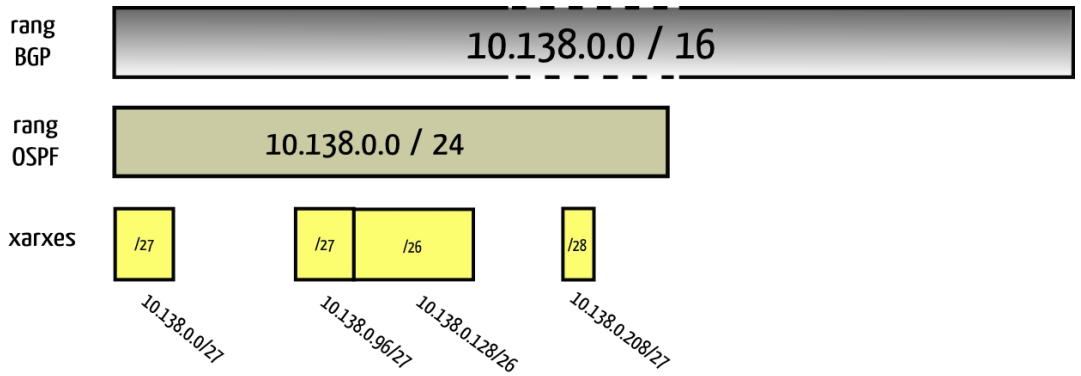


Figura 7.1: Exemple de subnetting

Podem veure com, partint d'un rang inicial /16 n'agafem un de més petit /24 (256 adreces). Al nivell de xarxes tornem a partir el rang immediatament superior en trossos de diverses mides, cadascun amb la numeració de xarxa que li correspon. El problema recau en que a cada nivell de la jerarquia de xarxa, quan partim els rangs assignats a cada nivell en subrangs més petits, hem d'assegurar que els nous subrangs són un subconjunt del rang superior, i que la numeració i la posició d'aquests rangs sigui correcte segons la màscara de xarxa que triem. No n'hi ha prou en cercar un espai buit on hi hagin suficients adreces disponibles com per encabir un subrang de la mida que necessitem, sinó que a més ha de ser un subnetting vàlid. Agafant el nivell de xarxes de l'exemple anterior, veiem un exemple del que hem d'evitar:

Les línies discontinues verticals de la figura 7.2 indiquen els espais vàlids als quals podem col·locar una subxarxa /26. Podem veure que tot i que entre les dues subxarxes /27 existents hi havia espai suficient per encabir una /26, aquesta no coincideix amb un subnetting vàlid. Aquesta xarxa invàlida si existís,

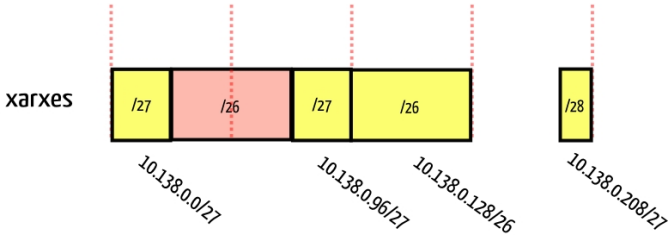


Figura 7.2: Exemple d'assignació d'un rang invàlid

hauria de tenir com a rang 10.138.0.32/26, el que significa que és una xarxa amb 64 adreces compreses entre 10.138.0.32 i 10.138.0.95. Si fem el càlcul sobre 10.138.0.32/26, veurem que en el subnetting real i vàlid, 10.138.0.32 és una ip compresa entre 10.138.0.0 i 10.138.0.63 .

Per poder fer els càlculs necessaris per resoldre situacions com les que ens hem trobat abans, hem definit un mètode concret. Abans de res però, primer necessitem una manera de representar una adreça de manera que puguem fer-hi càlculs, com ara sumar-hi una mida d'un rang. Per poder fer això hem de transformar l'adreça ip en un nombre enter. Una adreça ip de la forma x.x.x.x està formada per 4 nombres cadascun comprès entre 0 i 255, la combinatòria dels quals ens dona un màxim de 4.294.967.296 adreces possibles, per tant cada adreça ip pot ser representada com un nombre enter comprès entre 0 i 4.294.967.295. si calculem la representació binària d'una adreça ip, per exemple 10.138.0.32, obtenim el següent nombre:

$$00001010\ \ 10001010\ \ 00000000\ \ 00100000$$

on cada conjunt de 8 bits representa cada un dels nombres de 0 a 255 de l'adreça. Binàriament, la manera més fàcil de transformar aquest nombre en el que volem, es aplicar desplaçament de N bits cap a l'esquerra (24,16,8 i 0 respectivament) a cada una de les parts, convertir-les a decimal i sumar-les. amb l'exemple anterior el resultat seria:

$$\underbrace{00001010}_{24}\ \underbrace{10001010}_{16}\ \underbrace{10001010}_{8}\ \underbrace{10001010}_{0}$$

$$\underbrace{167772160\quad 9043968\quad 0\quad 32}_{176816160}$$

De la mateixa manera, podem obtenir el resultat contrari, fent desplaçament de bits cap a la dreta, utilitzant els mateixos valors. En binari el càlcul es veu molt clar, però per simplificar el procés i no haver de fer les diverses conversions a binari, és molt més còmode fer els càlculs directament amb els nombres en decimal. Així doncs, per transformar els 4 nombres de la ip 10.138.0.32 a una representació entera, utilitzarem la següent formula:

$$10(2^{24}) + 138(2^{16}) + 0(2^{8}) + 32$$

Algorisme utilitzat

El mètode utilitzat per calcular la posició i numeració d'una subxarxa, esta compost per un algorisme que té en compte tots els possibles casos que es poden donar, i dóna solució a cada un d'ells, fent servir com a base la representació entera d'una adreça ip.

A continuació es detalla aquesta llista possible de casos:

Donat un rang R de mida /24 (256 adreces) del qual volem obtenir una xarxa de mida X, amb X $\leq$ que /24:

1. **No hi ha cap subxarxa dins el rang R**

   Solució : Agafem el primer subrang disponible de mida X, que en aquest cas, coincideix amb l'adreça base de R substituint la mascara per X.

   Exemple : Rang 10.138.30.0/24, busquem una /27 el rang resultant serà 10.138.30.0/27.

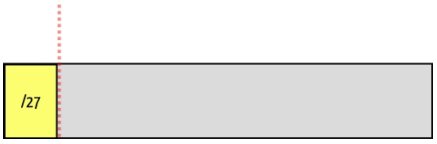


Figura 7.3: Cas 1

2. **Hi han xarxes Solució :** En aquest cas, ens limitarem a mirar si tenim espai al principi del rang per col·locar una subxarxa de mida X.

   (a) **Hi ha lloc per la xarxa que busquem al principi del rang**

   Solució : Agafem el primer subrang disponible de mida X, que en aquest cas, coincideix amb l'adreça base de R substituint la mascara per X.

   Exemple : Rang 10.138.30.0/24, amb la 10.138.30.32/27 ocupada, busquem una /27 el rang resultant serà 10.138.30.0/27.

Figura 7.4: Cas 2a

   (b) **Hi ha lloc al principi del rang, però no és suficient**

   Solució : Continuar al punt 3 si només hi ha una xarxa o al punt 4 si n'hi ha més d'una.

   Exemple : Rang 10.138.30.0/24, amb la 10.138.15.0/28 ocupada, busquem la /27 següent el rang resultant serà 10.138.30.32/27.

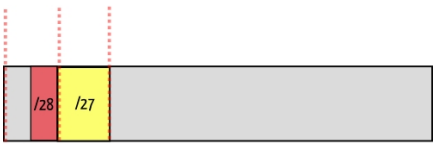


Figura 7.5: Cas 2b

   (c) **No hi ha lloc al principi del rang**

   Solució : Continuar al punt 3 si només hi ha una xarxa o al punt 4 si n'hi ha més d'una.

   Exemple : Rang 10.138.30.0/24, amb la 10.138.0.0/27 ocupada, busquem la /27 següent el rang resultant serà 10.138.30.32/27.

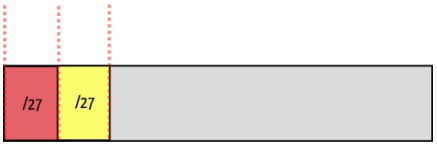


Figura 7.6: Cas 2c

3. **Només hi ha una xarxa** Aquest cas només es donarà, si abans s'ha donat 2.b o 2.c
   Solució : Agafem el primer subrang disponible de mida X, que trobem a la dreta de la xarxa ocupada, sempre que hi hagi lloc i respectem el subnetting.

   (a) **Si la mida de la primera xarxa de R, R1 és $\geq$ X, podem donar la següent xarxa de mida X directament, ja que és segur que respectarà el subnetting**
   Solució : El rang serà el valor enter de R1 + la mida de R1.
   Exemple : Rang 10.138.30.0/24, amb la 10.138.30.32/26 ocupada, busquem una /27 següent el rang resultant serà 10.138.30.64/27.

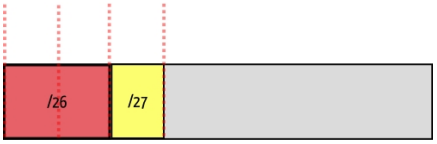


Figura 7.7: Cas 3a

   (b) **si la mida de la primera xarxa de R, R1 és < X, podem donar la següent xarxa de mida X, tenint en comte la mida X**
   Solució : El rang serà el valor enter de R1 + la mida de X.
   Exemple : Rang 10.138.30.0/24, amb la 10.138.30.0/28. ocupada, busquem una /27 següent el rang resultant serà 10.138.30.32/27.

Figura 7.8: Cas 3b

   Fins aqui, es pot veure que s'ha especificat molt detalladament els casos que es podien donar. La raó d'això és optimització. Els dos punts següents es basen en iterar a través de tots els elements que conté un rang, amb lo qual en funció de la quantitat d'elements, l'algorisme podria ser costós. Tant detall és doncs un mecanisme per evitar entrar en els següents casos si es pot fer un tractament més simple del problema.

4. **Hi ha més d'una xarxa** Aquest cas només es donarà, si abans s'ha donat 2.b o 2.c
   Solució : Es tracta de recórrer tots els subrangs de R, ordenats segons el nombre enter, per parelles, (R[i-1] , r[i]), comprovant si hi ha espai per encabir la nova xarxa entre les 2. Per que es

doni això, la resta del nombre enter de R[i] - nombre enter de R[i-1] - mida[i-1] ha de ser > que X, si no hi ha espai, passem a la següent parella.
Es poden donar els casos següents:

(a) **Si la mida de R[i-1] ≥ X, podem encabir perfectament la nova xarxa en l'espai entre R[i-1] i R[i].**
Solució : El rang serà el valor enter de R[i-1] + la mida de R[i-1].
Exemple 1: Rang 10.138.30.0/24, amb la R[i-1] = 10.138.30.0/27 ocupada, i la R[i] = 10.138.30.64/27 ocupada busquem una /27 següent el rang resultant serà 10.138.30.32/27.
Exemple 2: Rang 10.138.30.0/24, amb la R[i-1] = 10.138.30.0/26 ocupada, i la R[i] = 10.138.30.96/27 ocupada busquem una /27 següent el rang resultant serà 10.138.30.65/27.

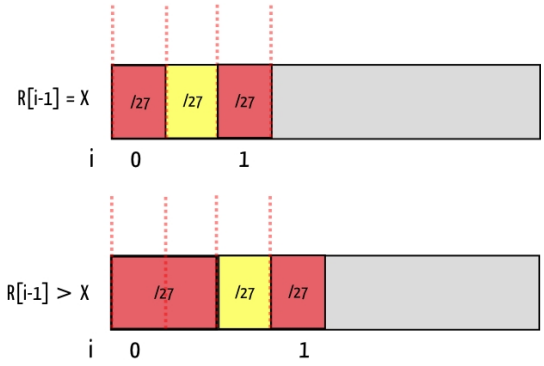


Figura 7.9: Cas 4a

(b) **Si la mida de R[i-1]< que X, cal validar que l'espai entre les 2 respecta el subnetting respecte la mida X**
Solució : Si el resultat de (restar R[i] - R[i-1]- X) < X, vol dir que que tot i que podria haver-hi prou espai, aquest no respecta el subnetting, si el resultat és més ≥ X, llavors podem col·locar la xarxa entre R[i-1] i R[i].
Exemple 1: Rang 10.138.30.0/28, amb la R[i-1] = 10.138.30.0/28 ocupada, i la R[i] = 10.138.30.48/28 ocupada busquem una /27 següent però l'espai no ens deixa colocar la xarxa sense trencar el subnetting.
Exemple 2: Rang 10.138.30.0/28, amb la R[i-1] = 10.138.30.0/28 ocupada, i la R[i] = 10.138.30.65/27 ocupada busquem una /27 següent el rang resultant serà 10.138.30.32/27.

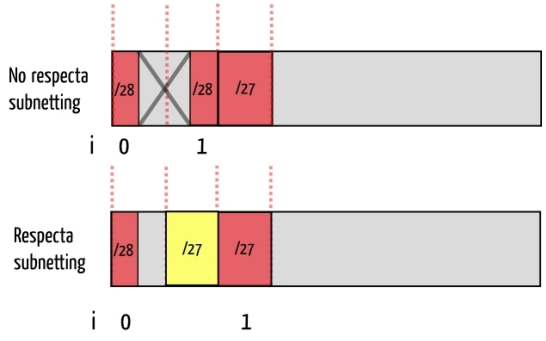


Figura 7.10: Cas 4b

5. **Hi ha més d'una xarxa** Aquest cas només es donarà, si després de recórrer totes les parelles de R, no s'ha trobat cap espai disponible

   Solució : Es tracta de comprovar si hi ha espai després de l'ultima xarxa ocupada, i si aquest respecta el subnetting.

   Es poden donar els casos següents:

   (a) **La nova xarxa és més petita o igual que R[últim]**

   Solució : Si el resultat de R[últim]+ mida R[últim] + X -1, és $\leq$ que l'enter que representa a R + la mida de R, el rang serà R[últim]+ mida R[últim].

   Exemple 1: Rang 10.138.30.0/24, amb R[últim] = 10.138.30.192/27 ocupada, busquem una /27 següent el rang resultant serà 10.138.30.224/27.

   Exemple 2: Rang 10.138.30.0/24, amb R[últim] = 10.138.30.224/27 ocupada, no queda gens d'espai.

Figura 7.11: Cas 5a

   (b) **La nova xarxa és més gran que R[últim], per tant encara que hi hagi espai, hem de comprovar el subnetting**

   Solució : Si el resultat de R[últim]+ 2X -1, és $\leq$ que l'enter que representa a R + la mida de R, el rang serà R[últim]+ mida R[últim].

   Exemple 1: Rang 10.138.30.0/24, amb R[últim] = 10.138.192.0/28 ocupada, busquem una /27 , el rang resultant serà 10.138.224/27.

   Exemple 2: Rang 10.138.30.0/24, amb R[últim] = 10.138.30.224/28 ocupada, queda espai, però no és suficient.

Figura 7.12: Cas 5b

El codi encarregar d'implementar aquest algorisme esta encapsulat dins la funció getAvailableRange:

Llistat 7.1: funció Products.sirona.ospf_ranges.getAvailableRange

```
security.declarePublic('getAvailableRange')
def getAvailableRange(self,cidr):
  """Determina si hi ha lloc per una xarxa de mida cidr, i en cas afirmatiu
     retorna el rang que haurem de fer servir per crear una nova xarxa
  """
  import pdb; pdb.set_trace()
  import operator
  import Products.sirona.ipaddr as ipaddr

  mida_cidr=ipaddr.NETWORK_SIZE[cidr]

  ospf_range_cidr=self.getRange().split('/')[1]
  ospf_range_net=self.getRange().split('/')[0]
  ospf_range=ipaddr.network(ospf_range_net,ospf_range_cidr,ipaddr.DEMAND_NONE)
  ospf_range_intrep=ospf_range.network_intrep
  ospf_range_broadcast_intrep=ospf_range.broadcast_intrep

  intrep=None

  intreps = self.getSortedNetworks()


  if cidr>=ospf_range_cidr:
    if len(intreps)==0:
        intrep=ospf_range_intrep
        else:
          espai_inici=intreps[0][0]-ospf_range_intrep
          if ((intreps[0][0]>ospf_range_intrep) and (espai_inici>=mida_cidr)):
              intrep=ospf_range_intrep
          else:
              if len(intreps)==1:
                if intreps[0][1]<=cidr:
                  if (intreps[0][0]+intreps[0][2]+mida_cidr) <= ospf_range_broadcast_intrep:
                    intrep=intreps[0][0]+intreps[0][2]
                  else:
                    nin=ipaddr.intrep_to_dotted_decimal(intreps[0][0])
                    v_intrep=ipaddr.network(nin,cidr,ipaddr.DEMAND_NONE)
                  if (v_intrep.network_intrep+mida_cidr+mida_cidr) <= ospf_range_broadcast_intrep:
                    intrep=v_intrep.network_intrep+mida_cidr
                  else:
                    i = 1
                    while ((i<len(intreps)) and (intrep==None)):
                      sep = intreps[i][0]-intreps[i-1][0]-intreps[i-1][2]-1
                      if sep >= mida_cidr:
                        if intreps[i-1][1]<=cidr:
                          intrep=intreps[i-1][0]+intreps[i-1][2]
                        else:
                          nin=ipaddr.intrep_to_dotted_decimal(intreps[i-1][0])
                          v_intrep=ipaddr.network(nin,cidr,ipaddr.DEMAND_NONE)
                          if (intreps[i][0]-v_intrep.network_intrep-mida_cidr) >=mida_cidr:
                            intrep=intreps[i-1][0]+mida_cidr
                            i=i+1;
                    i=i-1;
                    if intrep==None:
                      if intreps[i][1]<=cidr:
                        if (intreps[i][0]+intreps[i][2]+mida_cidr-1) <=ospf_range_broadcast_intrep:
                          intrep=intreps[i][0]+intreps[i][2]
                        else:
                          nin=ipaddr.intrep_to_dotted_decimal(intreps[i][0])
```

```
60                          v_intrep=ipaddr.network(nin,cidr,ipaddr.DEMAND_NONE)
61                          if (v_intrep.network_intrep+mida_cidr+mida_cidr-1) <=
                                ospf_range_broadcast_intrep:
62                            intrep=v_intrep.network_intrep+mida_cidr
63
64    if intrep==None:
65      return None
66    else:
67      rang= ipaddr.intrep_to_dotted_decimal(intrep)+'/'+cidr
68    return rang
```

7.1.2 Enllaçar aparells

Per tal de poder crear enllaços entre diferents aparells de la xarxa, s'ha dissenyat una interfície dinàmica [1], que permet escollir amb quin aparell i amb quina interfície s'enllaça. Com hem comentat a l'especificació de requeriments, hi ha una serie de regles que s'han de complir alhora de poder enllaçar un aparell amb un altre. El resultat d'aplicar aquestes regles, és un conjunt d'opcions, cadascuna amb una acció a dur a terme si és escollida.

Com a primer pas, la llista d'interfícies a mostrar, a les quals es pot enllaçar queda limitada de la següent manera, en funció del tipus d'interfície que estiguem configurant en aquell moment:

- **Interficies Wireless :** Es farà llista de totes les interfícies Wireless de l'aparell remot, així com de tots els Bridge que continguin com a mínim una interfície Wireless
- **Interficies Wired :** Es farà llista de totes les interfícies Wired de l'aparell remot, així com de tots els Bridge que continguin com a mínim una interfície Wired
- **Interficies Bridge :** Es farà llista de totes les interfícies de l'aparell remot que tinguin el mateix tipus que alguna de les interfícies presents en el Bridge local, així com de tots els Bridge remots que continguin com a mínim una interfície del mateix tipus que alguna de les interfícies presents en el Bridge local.

Un cop aplicat aquest filtre, per determinar quines opcions són possibles en cada cas, segons la interfície que s'esculli de la llista anterior, s'ha construït el següent diagrama per veure en conjunt, com es pren les decisions l'algorisme:

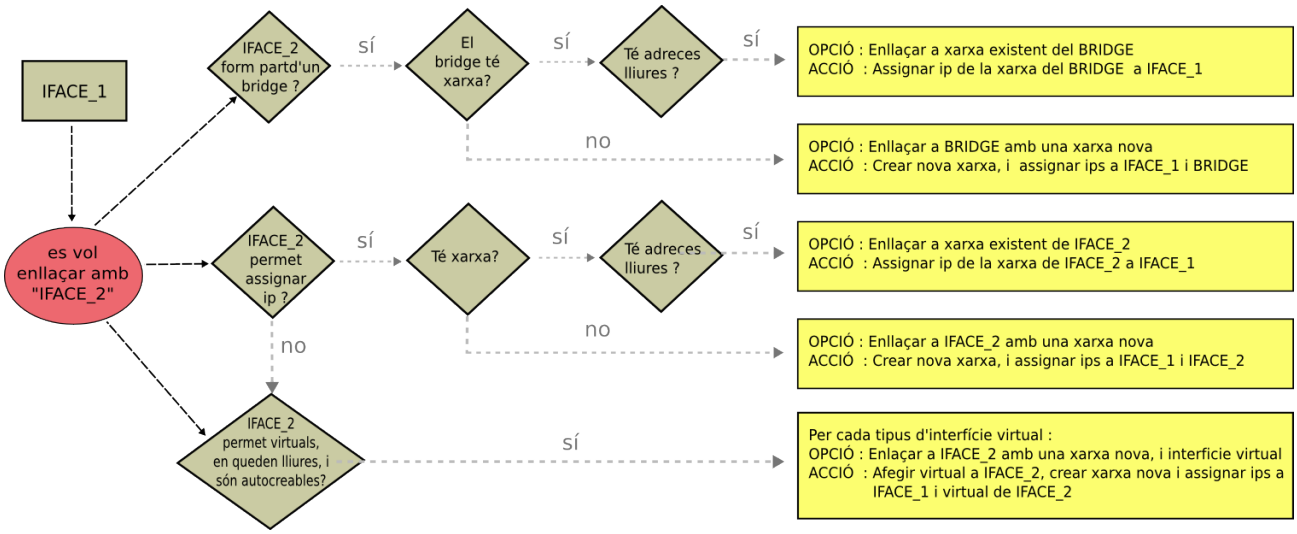


Figura 7.13: Diagrama d'enllaç d'aparells

[1] Veure 4.3.2, secció Configuració d'interfícies reals d'aparells

El codi encarregar d'implementar aquest algorisme esta encapsulat dins la funció getLinkOptions:

Llistat 7.2: funció Products.sirona.browser.getLinkOptions

```
def getInterfaceLinkOptions(self,path):
  """Determina de quines maneres podem enllaÃ§ar amb la interficie remota que hem triat
  """
  from Products.sirona.IVirtual import IVirtual
  from Products.sirona.IWired import IWired
  from Products.sirona.IWireless import IWireless

  options=[]

  remote_iface=self.context.portal_catalog(path=path)[0].getObject()
  local_iface=self.context

  remote_bridge = remote_iface.bridged()
  remote_device=remote_iface.getDevice()

  if hasattr(remote_iface,'configurable'):
    remote_iface_is_configurable=remote_iface.getConfigurable()
  else:
    remote_iface_is_configurable=True
    virtual_reference={'virtual':0,'wdsp2p':0,'virtual_ap':0}
    if IWired.providedBy(remote_iface) or IWireless.providedBy(remote_iface):
      virtuals=[]
      virtuals = [i['id'] for i in remote_iface.getVirtualInterfaces()]
      remote_iface_virtual_count = len(virtuals)
      if remote_iface_virtual_count>0:
        for v in virtuals:
           type=remote_device[v].portal_type
           if IVirtual.providedBy(remote_device[v]):
             virtual_reference[type]=virtual_reference[type]+1

    if remote_iface_is_configurable:
      network = remote_iface.getNetwork()
      if network:
        if not network.isFull():
          ip=network.getAvailableIPAddress()
          options.append({'option':'JOIN_NETWORK',
                          'network':network.id,
                          'remote_iface_id':remote_iface.id})
        else:
          options.append({'option':'JOIN_NETWORK_CREATE',
                          'remote_iface_id':remote_iface.id})

    if not IVirtual.providedBy(remote_iface):
      if hasattr(remote_iface,'allowedVirtualTypes'):
         if remote_iface.getAllowedVirtualTypes()=='':
           remote_iface.setAllowedVirtualTypes('[]')
           remote_iface_config=eval(remote_iface.getAllowedVirtualTypes())
         else:
           remote_iface_config=False

         if remote_iface_config:
           for config in remote_iface_config:
             if (config['auto']) and (config['limit']>virtual_reference[config['type']]):
                 remote_path='/'.join(remote_device.getPhysicalPath())
                 options.append({'option':'JOIN_VIRTUAL_CREATE',
                                 'remote_path':remote_path,
                                 'remote_iface_id':remote_iface.id,
                                 'remote_iface_virtual_type':config['type']})

```

```
60            for v in virtual_reference.keys():
61              if virtual_reference[v]>0:
62                 remote_virtual_ids=[f['id'] for f in remote_iface.getVirtualInterfaces()]
63                 for remote_v in remote_virtual_ids:
64                    iface=remote_device[remote_v]
65                    network= iface.getNetwork()
66                    if network:
67                      if not network.isFull():
68                        remote_path='/'.join(iface.getPhysicalPath())
69                        ip=network.getAvailableIPAddress()
70                        options.append({'option':'JOIN_VIRTUAL_NETWORK',
71                                        'remote_path':remote_path,
72                                        'remote_iface_id':remote_iface.id,
73                                        'network':network.id,
74                                        'remote_iface_virtual_id':iface.id})
75                     else:
76                        remote_path='/'.join(remote_device.getPhysicalPath())
77                        options.append({'option':'JOIN_VIRTUAL_NETWORK_CREATE',
78                                        'remote_path':remote_path,
79                                        'remote_iface_id':remote_iface.id,
80                                        'remote_iface_virtual_id':iface.id})
81    return options
```

7.1.3 Llistar enllaços

Partint com a millora directe sobre el model guifi.net, es va decidir no incloure en el model de dades el concepte d'enllaç entre aparells com a objecte. Hi ha diversos raonaments que ens han portat a prescindir de l'element enllaç:

- Enllaç existeix només com a concepte en el "món físic", representat físicament per cables, o ones
- Un enllaç, per definició, existeix, si hi han dos interfícies de xarxa que comparteixen adreces ip d'una mateixa xarxa.
- Representar un enllaç doncs com a una referència entre interfícies, vol dir haver de mantenir una doble referència entre elles en tot moment.
- Una interfície que esta enllaçada a un altre aparell, tindrà sempre una adreça ip assignada.

Així doncs, no només és innecessari definir un enllaç, sinó que és redundant. Quan necessitem obtenir informació sobre els enllaços d'un aparell, ens limitarem a calcular-los en base a consultar de quines xarxes forma part aquell aparell, i quins altres aparells hi han en aquestes xarxes.

De fer això se n'encarrega la següent funció, que genera una llista de les interfícies amb les que té algun enllaç, indicant atributs tant de la pròpia interfície, com de l'aparell i al node on pertanyen:

Llistat 7.3: funció Products.sirona.iface.getLinks

```
1     security.declarePublic('getLinks')
2     def getLinks(self):
3         """ retorna les dades dels enllaÃ§os associats amb la interficie
4         """
5         from Acquisition import aq_parent
6         from Products.sirona.IWireless import IWireless
7         from Products.sirona.IWirelessVirtual import IWirelessVirtual
8         links=[]
9         network=self.getNetwork()
10        if network:
11            if self.getRefs('iface_ip')[0].portal_type=='ip_ospf':
12                ips =network.contentItems()
13                if len(ips)>1:
14                    selfip=self.getCIDR()
15                    for ip in ips:
16                        if ip[1].getIpv4()!=selfip:
17                            if ip[1].getRefs():
18                                # si ip[1].getRefs() no retorna res en aquest punt vol
                                      dir que s'ha esborrat un aparell
19                                # pero encara existeix la ip que tenia associada
20                                remote_iface=ip[1].getRefs()[0]
21                                remote_device = remote_iface.getDevice()
22                                remote_node=remote_iface.getNode()
23
24                                if IWireless.providedBy(self) or IWirelessVirtual.
                                      providedBy(self):
25                                    selfmode=self.getMode()
26                                else:
```

```
27                                  selfmode=''
28                              if IWireless.providedBy(remote_iface) or IWirelessVirtual
                                    .providedBy(remote_iface):
29                                  remotemode=remote_iface.getMode()
30                              else:
31                                  remotemode=''
32
33                              if (selfmode!='client') or ((remotemode=='ap') and (
                                    selfmode=='client')):
34                                  links.append({'device_id':remote_device.getId(),
35                                            'device_name':remote_device.title,
36                                            'node':remote_node.id,
37                                            'node_title':remote_node.title,
38                                            'local_iface':self.title,
39                                            'local_ip':selfip,
40                                            'remote_iface':remote_iface.title,
41                                            'remote_ip':remote_iface.getCIDR()})
42          if links:
43              return links
44          else:
45              return None
```

7.2 Exemples de codi

7.2.1 Expressions regulars

Sempre que ha set possible, quan s'ha hagut de fer alguna validació de cadenes de text relativament complexa, s'ha optat per utilitzar expressions regulars, en comptes d'escriure codi per analitzar una cadena de text. Hi ha alguns casos, en que l'expressió regular resultant, és molt obvia i d'altres en què és més complicada respecte a un mètode tradicional. En qualsevol dels casos, una expressió regular és sempre una solució molt més elegant.

Per tal de mostrar una pinzellada d'aquestes expressions, posarem dos exemples de codi, corresponents a 2 validacions de cadenes, de diferent complexitat:

- **Validació d'una adreça MAC**

  Aquest cas es força senzill: una adreça mac és de la forma `$$:$$:$$:$$:$$:$$` on cada `$` pot ser un caràcter hexadecimal 0-9 i A-F. Veiem que hi han 6 grups de 2 caràcter cada un, amb un punt que separa cada grup, en total 5 punts. Si partim el problema en dos, tenim un primer grup de dos caràcters de la forma `$$`, seguits de 5 grups de la forma `:$$`. Per tant el que hem de fer es una doble validació buscant aquesta combinació de 1+5 repeticions, i en cada una, comprovar que els caràcters siguin correctes, i n'hi hagin dos en cada grup `([0-9A-F]2)`.

  Llistat 7.4: Validació de MAC's

```
1     security.declarePublic('checkMAC')
2     def checkMAC(self,mac):
3         """comprova si la mac esta en format correcte
4         """
5         import re
6         regex = re.compile("^([0-9A-F]{2}){1}(:[0-9A-F]{2}){5}$")
7         if regex.match(mac):
8             return True
9         else:
10            return False
```

- **Validació d'una adreça IP**

  Per tal de validar una adreça ip, l'expressió va resultar una mica més complicada: una adreça ip és de la forma `$.$.$.$` on cada `$` pot ser un valor decimal de 0 a 255. Com que les expressions regulars serveixen per tractar text, cada un dels 4 valors de la ip, ha de ser tractat com a cadena, i com a tal, tindrà 1,2 o 3 caràcters en funció del valor numèric. En aquest cas, potser una solució via codi tradicional, separant les 4 parts, convertint el text en enter i validant numèricament si estava comprès entre 0 i 255 hagués estat més senzilla. Definir una expressió regular per aconseguir-ho, però va ser un repte que ens vam proposar.

  La solució te una part semblant a l'anterior, ja que divideix el problema en dues parts, 3 grup de entre 1 i 3 caràcters de la forma `$$$.` i 1 de la forma `$$$`. En cada grup, per poder validar si el nombre esta entre 0 i 255, farem una llista de tots els casos possibles en que la combinació de caràcters és vàlida :

  - **25[0 -5]** Un numero que comenci per 25 i tingui un caràcter entre 0 i 5 al darrera

– **2[0 -4][0 -9]** Un numero que comenci per 2 , seguit d'un caràcter entre 0 i 4, i un tercer entre 0 i 9

– **[01]?[0 -9][0 -9]?** Aquest compren els nombres entre e 0 i el 199, al tenir el primer i l'ultim com a opcionals (?)

Llistat 7.5: Validació de IP's

```
1   security.declarePublic('checkIPAddress')
2   def checkIPAddress(self,ip):
3       """Comprova que el format de la ip sigui correcte
4           x.x.x.x essent cada x un numero entre 0 i 255
5       """
6
7       import re
8       regex = re.compile("^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.)
            {3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$")
9       if regex.match(ip):
10          return True
11      else:
12          return False
13      pass
```

7.2.2 Exportació XML

La raó d'exportar les dades de l'aplicació en format XML, ve donada per la necessitat de tenir un sistema de representar les dades, independent d'una tecnologia de base de dades concreta com és ZODB, o una base de dades relacional. Tenir un esquema XML ben definit per exportar les dades, permet poder crear aplicacions secundaries que s'alimentin de les dades de l'aplicació, sense tenir accés directe a la base de dades, sense conèixer-ne els mecanismes, i evitant així canvis davant de modificacions en el model de dades de l'aplicació original. Dins l'entorn de guifi.net, ja s'estan usant aquestes exportacions de dades, per fer funcionar els sistemes de monitorització descentralitzats. Una altre de les coses per les quals es vol donar ús a poder disposar de les dades en format XML, es per poder importar i exportar dades entre diferents aplicacions. El model XML que hem dissenyat, exporta els elements de la part física de la xarxa.

Tots els elements possibles d'una exportació XML, queden representats en el següent exemple, ignorant els valors innecessaris, per poder veure mes clarament els elements:

Llistat 7.6: Jerarquia d'elements XML

```
<zone center_latitude="" center_longitude="" id="" title="">
   (...)
   <zone center_latitude="" center_longitude="" id="" title="">
       <node lat="" lng="" id="" title=""></node>
           <device type="" id="" title="">
               <interface type="" id="" name=""></interface>
                   <virtual_interface type="" id="" name=""></virtual_interface>
                       <link node="" device="" type="" interface="" ip=""/>
               <interface type="" id="" name=""></interface>
                   <link node="" device="" type="" interface="" ip=""/>

               <interface type="bridge" id="" name=""></interface>
                   <bridged_interface name=""></bridged_interface>
                   <link node="" device="" type="" interface="" ip=""/>
```

En la major part, aquesta jerarquia coincideix amb la jerarquia del model de dades, i a més hi han elements calculats com són `<link>` i altres que accedits a traves de referències com `<virtual_interface>`. En l'exemple, es veu la estructura sencera, però amb l'exportació el que es volia era poder exportar qualsevol element, des de `<zone>` com a l'exemple, fins a només un `<device>`. També es volia aconseguir re-utilitzar cada part, de manera que cada exportació de cada element fos independent de les altres, però no es repetís codi.

Per poder aconseguir aquesta fita, s'ha implementat seguint un dels conceptes que marca Zope3, i que comença a ser utilitzable en cert nivell a Zope2: els adaptadors. Els adaptadors, a grans trets, són un mecanisme que permet transformar qualsevol objecte en un altre, afegint-li certa funcionalitat,a través d'un tercer objecte que és l'adaptador pròpiament dit. En el cas que ens ocupa això ens permetrà definir un adaptador per convertir qualsevol classe en exportable. Com que tenim clars quins elements volem exportar i quins no, no farem servir aquest mecanisme directament, però si que deixarem el camí preparat per a poder-lo usar en un futur.

Per tal que un objecte sigui exportable doncs, haurem de fer el següent:

En el model de dades, l'objecte ha d'estar associat a la interficie IXMLExportable, que indicara que aquell objecte és exportable, i el proveirà amb la funció `getXMLAttributes` la qual haurà de ser programada per retornar una llista de parelles atribut/valor, (Llistat 7.7 línies 5-6) que volem exportar

d'aquell objecte, així com una llista d'objectes de cada tipus immediatament inferior en la jerarquia (Llistat 7.7 línies 7-9).

Llistat 7.7: funció Products.sirona.iface.getXMLAttributes

```
1  security.declarePublic('getXMLAttributes')
2  def getXMLAttributes(self):
3      """
4      """
5      data =  {'id':self.getId(),
6              'title':self.title,
7              'physical':[self[i['id']] for i in self.getPhysicalInterfaces()],
8              'logical':[self[i['id']] for i in self.getLogicalInterfaces()],
9              'bridges':[self[i] for i in self.getInterfacesByType('bridge')],}
10     return data
```

Les dades que retorna aquest codi, són interpretades per un template ZPT, que formata els atributs en format XML (Llistat 7.8 línies 4-5), amb el tag corresponent a aquell objecte, i crida altre cop el procés, per cada llista d'objectes(Llistat 7.8 línies 6-10), fent ús de la funció `view.getXML()`.

Llistat 7.8: template xml_device_view.pt

```
1  <device type="device"
2          tal:define="context python:options['context'];
3                      atts context/getXMLAttributes"
4          tal:attributes="id atts/id;
5                          title atts/title">
6      <tal:block condition="atts/physical" repeat="ifs atts/physical">
7          <tal:block replace="structure python:view.getXML(ifs)" />
8      </tal:block>
9      <tal:block condition="atts/bridges" repeat="ifs atts/bridges">
10         <tal:block replace="structure python:view.getXML(ifs)" />
11     </tal:block>
12 </device>
```

La funció `view.getXML()` es crida un cop per cada element de les llistes d'objectes que ha retornat la funció anterior `getXMLAttributes()`, passant com a paràmetre l'objecte de la llista actual en cada iteració. Això fa que des de el template, es torni a repetir per cada un d'aquests objectes, tot el procés, generant com a resultat el codi XML corresponent a cada element de l'objecte. Això com es pot veure serà recursiu, de manera que mentre hi hagin objectes que tinguin elements exportables, els templates s'aniran cridant els uns als altres fins al final de la jerarquia. getXML basa la crida del template que executarà en el tipus d'objecte pel qual ha estat invocada la funció, per tant els noms dels templates seguiran el patró `xml_nomdeltipus_view.pt`

Llistat 7.9: funció Products.sirona.browser.XML_View.getXML

```
1  def getXML(self,new_context):
2      """
3      """
4      from Products.Five.browser import pagetemplatefile
5
6      self.template = pagetemplatefile.ZopeTwoPageTemplateFile('./templates/xml_'+
           new_context.portal_type+'_view.pt')
7      return self.template(context=new_context)
```

Per últim, per engegar aquest mecanisme, només cal cridar qualsevol objecte que sigui exportable, afegint darrera la seva URL la paraula clau `xml_view`, el qual farà que es cridi un template genèric anomenat `xml_view.pt` que executarà tot aquest procés descrit, començant pel l'objecte en qüestió.

Llistat 7.10: template xml_view.pt

```
<xml version="1.0" tal:define="resp request/response;
                               h python:resp.setHeader('Content-Type', 'text/xml');
                               ">

<tal:block replace="structure python:view.getXML(context)" />

</xml>
```

Capítol 8

Conclusions

8.1 Treball futur

Com a resultat d'aquest projecte, en els últims mesos s'ha exposat el contingut del mateix en diverses ocasions. En concret durant la trobada internacional de xarxes lliures SAX2007 celebrada a Sant Bartomeu del Grau, es va aprofitar el dia dedicat a parlar sobre desenvolupament de l'actual aplicació de guifi.net, per presentar aquest projecte, i valorar si interessava continuar-ne el desenvolupament per a una futura migració del sistema actual cap a aquest.

Dels participants a la trobada, un grup d'ells es va interessar pel projecte, i es va començar a parlar sobre les línies de treball necessàries per complementar el treball. Del resultat d'aquestes trobades, s'ha creat un grup de desenvolupament, sota el nom clau de "Sirona", accessible públicament a través dels següents recursos:

- **Llista de correu :** https://llistes.projectes.lafarga.org/cgi-bin/mailman/listinfo/guifi-sirona
- **Repositori de codi (subversion) :** http://projectes.cpl.upc.edu/svn/sirona
- **Gestor del projecte (trac) :** http://projectes.cpl.upc.edu/trac/sirona
- **Entorn de proves :** http://guifi.cpl.upc.edu

A partir d'aquest punt, des d'aquest grup s'intentarà completar Sirona amb els elements que li manquen per poder fer una primera migració de dades de guifi.net, i testejar-ne el funcionament.

8.2 Millores

En la línia del treball comentat abans, hi ha una serie de aspectes nous a desenvolupar dins l'aplicació:

- Suport per tots els aparells usats a guifi.net i els seus mètodes de configuració.
- Suport per definir, afegir i gestionar serveis de la xarxa.
- Suport per afegir i gestionar usuaris dels serveis de la xarxa, usant LDAP.
- Integrar el sistema de gràfiques descentralitzat existent a l'aplicació.

- Definir un nou DTD per a les exportacions XML que compleixi amb qualsevol dels punts anteriors.

I com a millores del projecte en l'estat actual:

- Optimitzar el tractament de dades als mapes per a grans volums de dades.
- Polir les interfícies d'administració del portal.
- Implementar un mètode de cerca de camins sobre la estructura de xarxa guardada, amb l'objectiu per exemple, de determinar el servei més proper a un node.
- Implementar un sistema d'actualització automàtica dels aparells de la xarxa.
- Implementar importació de dades a partir de XML.

8.3 Conclusions

En quant a la xarxa:

- Les primeres proves amb dades reals a l'aplicació, i la prova de les configuracions automàtiques, ha donat bons resultats, amb el que s'esta treballant per complimentar totes les dades a l'aplicació, i fer alguns modificacions a la xarxa existent per posar el sistema en producció.
- S'ha vist que una aplicació de gestió és indispensable per a la supervivència d'una xarxa d'aquestes característiques.
- És molt complexe per no dir impossible d'intentar definir un model de dades prou genèric que sigui capaç de poder representar qualsevol estructura de xarxa.

En quant al software usat:

- És possible desenvolupar una eina de qualitat exclusivament amb programari lliure.
- L'ús de CMS extensibles com Plone, i llenguatges d'alt nivell com Python enriqueix tant el producte final com el treball del programador, i n'augmenten la productivitat.
- L'ús de noves tecnologies lligades al fenomen "WEB 2.0"com AJAX en un entorn web, milloren substancialment l'experiència d'usuari.

Capítol 9

Glossari

- **Adhoc**: Mètode de connexió de xarxes, sovint associat a dispositius sensefils. No necessita d'una estació base a on connectar-se ja que qualsevol dispositiu en mode adhoc, pot connectar-se a qualsevol altre dispositiu en mode adhoc que estigui dins la seva cobertura.
- **AP**: De l'anglès *access point*. Aparell sensefils que proporciona accés a d'altres dispositius sensefils a una xarxa, i fa de punt comú de comunicació entre ells i altres xarxes.
- **API**: De l'anglès *Application Programming Interface*, és una interfície de codi que un programa informàtic, sistema operatiu o llibreria proporciona per accedir a serveis implementats en ells, de manera que siguin accessibles per programes de tercers.
- **Backend**: Es diu de la part dels sistemes informàtics encarregades de processar informació obtinguda per la part *frontend* del mateix.
- **BGP**: De l'anglès *Border Gateway Protocol*, és un protocol d'enrutament que manté índexs i taules de xarxes, per facilitar la interconnexió d'aquestes. És el protocol més important dins l'enrutament a Internet.
- **Canal**: Index numèric amb el qual es classifiquen els diferents rangs de freqüència als que operen els dispositius sensefils.
- **CIDR**: De l'anglès *Classless Inter-Domain Routing*, és la manera més nova d'interpretar i representar una adreça ip i la seva màscara de xarxa.Es basa en l'adreça ip, seguida d'una / i el numero de bits prefixats de l'adreça.
- **Client**: Mode de connexió de xarxes sensefils, en el qual és necessari la presència d'un AP per poder-se comunicar amb d'altres dispositius de la xarxa.
- **DTD**: De l'anglès *Document Type Definition*, és un llenguatge per definir l'estructura de documents escrits amb llenguatges de marques com XML.Un DTD expressa la sintaxis correcte que ha de tenir un determinat document.
- **SSID**: De l'anglès *Service Set Indetifier*, és una cadena de text de retransmesa en el context d'un punt d'accés o d'una xarxa adhoc, que identifica els paquets retransmesos com a part d'una xarxa. Tots els aparells d'una mateixa xarxa han de compartir el mateix SSID.

- **Frontend**: És la part d'un sistema informàtic que interactua directament amb l'usuari.
- **Ip**: És una adreça única que es fa servir per identificar els diferents dispositius presents en una xarxa.
- **LAMP**: Es refereix al conjunt de programes de codi obert que es fan servir conjuntament per desenvolupar alguns sistemes web.(Linux Apache Mysql PHP).
- **LDAP**: De l'anglès *Lightweight Directory Access Protocol*, és un protocol per consultar i modificar directoris de serveix en entorns de xarxa. Els directoris de serveis solen consistir en conjunts de noms organitzats alfabeticament i /o jeràrquicament, sovint usats per serveis d'autenticació.
- **Màscara**: Numero que identifica la mida d'una subxarxa o subrang d'adreços de xarxa.
- **Namespace**: Més concretament un XML Namespace, és un context en el qual es defineixen un conjunt d'elements formant un vocabulari. L'ús del namespace identifica aquest vocabulari, permetent evitar ambigüitats quan hi han elements de diferents vocabularis amb el mateix nom.
- **OLSR**: De l'ànglès *Optimized Link State Routing protocol*, es refereix a un protocol d'enrutament que permet la comunicació entre dispositius d'una xarxa adhoc que no estan connectats directament l'un amb l'altre.
- **OSPF**: De l'anglès *Open Shortest Path First*, és un protocol d'enrutament un nivell per sota de BGP, que s'encarrega de dirigir el trànsit d'una xarxa en funció del camí mes curt representat pel graf d'aquesta.
- **Rang/Subrang**: Un conjunt d'adreces ip contigues, representades per una ip i una màscara. Un subrang és un subconjunt d'un rang superior.
- **Router**: Dispositiu que s'encarrega de comunicar diferents segments de xarxa, cadascun amb rangs d'adreces diferents, mitjançant algun protocol d'enrutament.
- **Subnetting**: Es diu quan s'agafa un rang de xarxa i es parteix jeràrquicament en subrangs més petits.
- **W3C**: De l'anglès *World Wide Web Consortium*, és l'organisme internacional principal que s'encarrega de definir els estàndards per la www.

Llistats

Índex de figures

Bibliografia

[1] Zope Documentation, en línia, 2007, http://www.zope.org/Documentation/

[2] ZPT Reference, en línia, 2007, http://www.zope.org/Documentation/Books/6Edition/AppendixC.stx

[3] Documentació Plone CMS, en línia, 2007, http://www.plone.org/documentation

[4] Introduction to Plone API, en línia, 2007, http://www.ifpeople.net/fairsource/material/apiPlone

[5] Expresiones Regulares, en línia, 2007, http://www.ignside.net/man/php/regex.php

[6] Style Guide for Python Code, en línia, 2007, http://www.python.org/dev/peps/pep-0008/

[7] Python Documentation, en línia, 2007, http://www.python.org/doc/topics/

[8] Dive Into Python, Python from novice to pro, en línia, 2007, http://www.diveintopython.org/

[9] Google Maps API Documentation, en línia, 2007, http://www.google.com/apis/maps/documentation/

[10] Google Maps API Tutorial, en línia, 2007, http://www.econym.demon.co.uk/googlemaps/

[11] Client-Side JavaScript Reference, en línia, 2007, http://docs.sun.com/source/816-6408-10/

[12] Javascript tutorial - DOM nodes and tree, en línia, 2007, http://www.howtocreate.co.uk/tutorials/javascript/dombasics

[13] Posicionamiento DIV i CSS, en línia, 2007, http://www.ignside.net/man/css/

[14] HTML 4.01 / XHTML 1.0 Reference, en línia, 2007, http://www.w3schools.com/tags/default.asp

[15] XML Tutorial, en línia, 2007, http://www.w3schools.com/xml/default.asp