Sistemes ERP-CRM. Explotació i adequació - I

Isidre Guixà Miranda

Sistemes de gestió empresarial

Índex

Introducció

Una vegada efectuada la implantació tècnica d'un ERP, el programari ja hauria d'estar en condicions de ser utilitzat per l'organització per donar-ne sortida a les necessitats. Però això no acostuma a passar, ja que les organitzacions, tot i que en moltes ocasions s'adapten al màxim a l'ERP, tenen necessitats no suportades per l'ERP.

Per aquest motiu, per a tots els ERP es necessiten professionals programadors que siguin capaços d'adequar-los a les necessitats de l'organització. Com que hi ha un gran ventall d'ERP-CRM i és impossible abastar les tecnologies que cadascun d'ells utilitza, hem hagut de decidir-nos per un, per dur a la pràctica allò que cal fer en qualsevol posada en explotació d'un ERP-CRM. El producte escollit ha estat l'ERP-CRM de codi obert OpenERP, en la seva versió 6.1.

Aquesta unitat està destinada a introduir-nos en l'arquitectura MVC (patró model-vista-controlador) facilitada pel *framework* OpenObject en el qual es basa OpenERP i s'ha estructurat en dos apartats.

L'apartat "Explotació i adequació. OpenERP: el model" té un doble objectiu. El primer, de curta durada, introduir els conceptes bàsics que cal tenir en compte en un procés d'explotació i adequació. El segon, de llarga durada, introduir-nos en el disseny del model de dades en OpenObject i la seva implementació en la base de dades PostgreSQL. Com a preàmbul del segon objectiu, es presenta com és la base de dades PostgreSQL corresponent a una empresa gestionada per OpenERP i se'n faciliten diversos mètodes d'accés. Una vegada es conegui la base de dades, cal endinsar-se en el disseny del model i és adequat iniciar-se amb una eina de diagramació UML, capaç de generar el codi Python per OpenObject, per passar posteriorment a descriure directament el model amb el llenguatge Python.

L'altre apartat, "OpenERP: la vista i el controlador", ens endinsa en el disseny, via XML, de les diverses vistes que proporciona OpenERP (formularis, llistes, calendaris, gràfics, menús...) i en el disseny del controlador (mètodes que implementen la lògica de negoci) a través del llenguatge Python.

Amb tot això no es pot dir que ja s'és expert en OpenERP, ni molt menys. S'està en disposició de dissenyar mòduls adaptables a un OpenERP estàndard, però manquen altres funcionalitats facilitades per OpenObject: herència —utilitzada per a l'adaptació de mòduls existents—, seguretat, càrrega massiva de dades, desenvolupament d'assistents, generació de traduccions idiomàtiques, disseny d'informes, disseny de quadres de comandament i altres.

El seguiment d'aquesta unitat pressuposa que l'alumne és coneixedor de:

- La implantació tècnica d'OpenERP. El seu coneixement s'adquireix a la unitat "Sistemes ERP-CRM. Implantació" del mòdul professional *Sistemes de gestió empresarial*.

- El llenguatge XML. El seu coneixement s'adquireix en el mòdul professional *Llenguatges de marques i sistemes de gestió de la informació*.
- La programació orientada a objectes. El seu coneixement s'adquireix en les unitats formatives relatives a programació orientada a objectes del mòdul professional *Programació*.
- El llenguatge Python.

Als apartats "Bibliografia" i "Adreces d'interès" del web hi trobareu referències a materials relacionats amb el llenguatge Python, que us ajudaran a introduir-vos-hi de manera ràpida si sou coneixedors de la programació orientada a objectes.

Per tal d'assolir un bon aprenentatge cal estudiar els continguts en l'ordre indicat, sense saltar-se cap apartat. Quan es fa referència a algun annex del web, cal adreçar-s'hi i estudiar-lo. Una vegada estudiats els continguts del material paper i del material web, s'han de desenvolupar les activitats web.

Resultats d'aprenentatge

En finalitzar aquesta unitat l'alumne/a:

1. Realitza operacions de gestió i consulta de la informació seguint les especificacions de disseny i utilitzant les eines proporcionades pels sistemes ERP-CRM i solucions d'intel·ligència de negocis (BI).
    - Utilitza eines i llenguatges de consulta i manipulació de dades proporcionades pels sistemes ERP-CRM.
    - Genera formularis.
    - Exporta dades.
    - Automatitza les extraccions de dades mitjançant processos.
    - Documenta les operacions realitzades i les incidències observades.
2. Desenvolupa components per a un sistema ERP-CRM analitzant i utilitzant el llenguatge de programació incorporat.
    - Reconeix les sentències del llenguatge propi del sistema ERP-CRM.
    - Utilitza els elements de programació del llenguatge per crear components de manipulació de dades.
    - Modifica components programari per afegir noves funcionalitats al sistema.
    - Integra els nous components programari en el sistema ERP-CRM.
    - Verifica el correcte funcionament dels components creats.
    - Documenta tots els components creats o modificats.

1. Explotació i adequació. OpenERP: el model

L'explotació i adequació de l'ERP d'una organització és una tasca imprescindible, ja que garanteix que el programari es mantingui en condicions de ser utilitzat per l'organització per tal de donar sortida a les seves necessitats. Per poder-ho dur a terme, per una banda cal identificar les necessitats (tasca pròpia de consultors) i, per una altra, tenir un coneixement profund de l'ERP, tant en les funcionalitats que facilita (tasca de consultors i implantadors) com en les qüestions tècniques vinculades a l'ERP (tasca d'analistes i programadors).

La nostra tasca, com a programadors, consistirà en conèixer l'arquitectura de l'ERP i les eines de desenvolupament que s'han d'utilitzar per poder fer el vestit a mida que necessita l'organització. El gran ventall d'arquitectures i d'eines vinculades als ERP fan impossible efectuar un aprenentatge estàndard d'explotació i adequació d'ERP i, per tant, ens centrarem a clarificar els punts claus a tenir en compte i els posarem en pràctica sobre la versió 6.1 d'OpenERP.

L'OpenERP és un programari de gestió empresarial desenvolupat sobre el *framework* OpenObject de tipus RAD (*Rapid Application Development*).

La facilitat dels entorns RAD rau en el fet que el desenvolupament d'aplicacions és molt simple pel programador, de manera que amb poc esforç es pot obtenir aplicacions d'altes prestacions.

L'OpenObject facilita diversos components que permeten construir l'aplicació:

- La capa ORM (*Object Relational Mapping*) entre els objectes Python i la base de dades PostgreSQL. El dissenyador-programador no efectua el disseny de la base de dades; únicament dissenya classes, per les quals la capa ORM d'OpenObject n'efectuarà el mapat sobre el SGBD PostgreSQL.
- Una arquitectura MVC (model-vista-controlador) en la qual el model resideix en les dades de les classes dissenyades amb Python, la vista resideix en els formularis, llistes, calendaris, gràfics... definits en fitxers XML i el controlador resideix en els mètodes, definits en les classes, que proporcionen la lògica de negoci.
- Un sistema de fluxos de treball o *workflows*.
- Dissenyadors d'informes.
- Facilitats de traducció de l'aplicació a diversos idiomes.

Tal com es pot observar, són molts els components d'OpenObject a conèixer per poder adequar l'OpenERP a les necessitats de l'organització, en cas que les funcionalitats que aporta l'OpenERP, tot i ser moltes, no siguin suficients.

El nostre objectiu és conèixer com es dissenya i s'implementa el model de dades en OpenObject. Abans, però, aprofundirem en el coneixement de la base de dades d'una empresa d'OpenERP, amb dues finalitats:

- Conèixer la relació existent entre els seus elements (taules i columnes) i els elements que observem en qualsevol dels formularis i informes d'OpenERP.
- Saber com accedir a les dades de l'empresa atacant directament la base de dades.

Una vegada coneguda l'estructura d'una base de dades d'OpenERP, ens podem endinsar en el disseny del model en OpenERP i ho fem en dues fases:

1. Utilitzem l'eina de diagramació *Dia* que possibilita la generació de mòduls complets (model-vista-controlador) a partir d'un diagrama UML i a partir d'ella ens iniciem en el disseny del model de dades d'OpenERP.
2. Ens endinsem en el disseny del model de dades d'OpenERP utilitzant el llenguatge Python.

1.1 Explotació i adequació

La gran diversitat de funcionaments de les empreses fa que sigui altament improbable que un ERP estigui ideat per donar sortida a totes les necessitats empresarials. Quan es detecta una funcionalitat no coberta per l'ERP, hi ha tres camins per trobar-hi una solució. Per una banda, es pot adequar l'ERP per donar resposta a les funcionalitats requerides. Per l'altra, es pot adequar el funcionament de l'empresa a les funcionalitats facilitades per l'ERP. O bé, un tercer camí basat en la combinació dels dos camins anteriors.

> La detecció de les funcionalitats de l'empresa no suportades per l'ERP que es vol implantar és una de les tasques principals en el procés de consultoria.

El fet de que l'empresa canviï el seu funcionament per adaptar-se a l'ERP, tot i semblar molt brusc, és una solució que s'adopta en moltes ocasions, sobretot quan els consultors són capaços de demostrar als responsables de l'organització que el nou funcionament té clars avantatges respecte als mètodes emprats en aquell moment per l'empresa. Cal tenir en compte que, en ocasions, l'empresa pot haver establert els seus funcionaments –en temps passats– coaccionada per les possibilitats del sistema informàtic vigent en aquell moment, i aquests funcionaments han esdevingut, amb el pas del temps, maneres de fer fonamentals i intocables en l'organització. En aquestes ocasions, la mà esquerra dels consultors i implantadors és fonamental per dur a terme la implantació de l'ERP amb èxit.

Tot i això, no sempre es pot adequar els mètodes de l'empresa a les funcionalitats facilitades per l'ERP i, en conseqüència, cal adequar l'ERP, fet que pot comportar

la necessitat de desenvolupar mòduls específics que es puguin acoblar a l'ERP i/o retocar mòduls (formularis i informes) de l'ERP.

En l'adequació d'ERP és fonamental garantir el correcte funcionament de les adequacions en les futures actualitzacions de l'ERP.

L'adequació d'un ERP a través del desenvolupament de mòduls específics es pot dur a terme, en principi, amb qualsevol llenguatge de programació i accedint directament a la base de dades de l'ERP. Fer-ho així, però, comporta inconvenients:

Hi ha ERPs que prohibeixen l'accés a la base de dades en mode escriptura i castiguen aquest fet amb la pèrdua del servei de suport.

- Accedir directament a la base de dades implica tenir un coneixement total de la base de dades i de les relacions existents entre els seus components. Aquestes relacions poden canviar en una actualització de l'ERP.
- Els mòduls desenvolupats amb un llenguatge de programació forà a l'ERP impliquen tenir peces fora de l'ERP, a no ser que siguem capaços d'emular la interfície de l'ERP i d'introduir les noves opcions a l'arbre de menús de l'ERP.

El retoc de mòduls (formularis i informes) de l'ERP també té inconvenients:

- Només és possible si disposem del codi font de l'ERP (opció vàlida en programaris de codi obert).
- Els retocs efectuats han de garantir-ne la continuïtat en properes actualitzacions de l'ERP sense necessitat de tornar-los a desenvolupar.

Els diversos inconvenients presentats ens porten a la necessitat de conèixer i utilitzar els mètodes de desenvolupament i/o retoc de mòduls de l'ERP facilitats pel fabricant. A més, cal efectuar els processos d'actualització de dades (altes, baixes i modificacions) a través de les regles de negoci de l'ERP, que tenen en compte totes les implicacions, i mai amb accessos directes a la base de dades.

Cal tenir en compte, també, que en ocasions pot ser convenient accedir a les dades en mode consulta per a extreure'n informació. En aquests casos, tenint coneixement de l'estructura de la base de dades, l'accés directe a la base de dades pot ser adequat, amb la precaució que l'estructura pot canviar en futures actualitzacions de l'ERP.

Quan l'organització necessita informes que l'ERP no facilita...

Si la necessitat dels informes és puntual, és perfectament lícit desenvolupar-los amb qualsevol eina, tot accedint directament a la base de dades, i executar-los per aconseguir el resultat esperat. No tenim cap necessitat d'incorporar-los com a opcions de menú de l'ERP ni de guardar-los per posteriors execucions. En cas de guardar-los, som conscients que després d'haver actualitzat l'ERP, pot ser necessari revisar el seu disseny davant possibles canvis en la base de dades.

Si la necessitat dels informes és periòdica, és lògic desenvolupar-los amb les eines que faciliti l'ERP, utilitzant les seves regles de negoci, i incorporant-los com a noves opcions de menú dins l'ERP. En aquest cas, davant una actualització de l'ERP, és molt possible que

les regles de negoci evocades continuïn existint (malgrat que hagi canviat l'estructura de la base de dades) i, en conseqüència, el nostre informe continuï funcionant.

A vegades, dins de les organitzacions existeixen persones que, possiblement per mancances de bones solucions BI vinculades a l'ERP, necessiten accedir a la base de dades per extreure'n informació (mode consulta) i, des de les eines ofimàtiques que dominen (bases de dades ofimàtiques i fulls de càlcul), desenvolupar informes i quadres de comandament (*dashboards*) adequats a les seves necessitats. En aquest cas, l'accés directe (mode consulta) a la base de dades també és lògic, tot i que l'usuari ha de ser conscient que els seus muntatges es poden veure afectats davant d'una actualització de l'ERP.

Aquests raonaments ens porten a les conclusions següents:

- Cal conèixer l'estructura de la base de dades de l'ERP i possibilitar-hi accessos en mode consulta, per extreure'n informació que pugui ser gestionada des d'eines externes.
- Cal conèixer la gestió de regles de negoci de l'ERP.
- Cal conèixer les eines recomanades pel fabricant de l'ERP per al desenvolupament i/o retoc de mòduls de l'ERP, fet que pot anar acompanyat del coneixement de llenguatges de programació.

1.2 La BD d'OpenERP

La gestió d'una empresa pot fer necessari, en un determinat moment, l'accés a la base de dades de l'ERP per tal d'obtenir informació en un format no facilitat per l'ERP. Per això, és important conèixer el disseny de la BD (base de dades) de l'ERP i, en el nostre cas, de l'OpenERP.

En l'OpenERP no hi ha un disseny explícit de la base de dades, sinó que la base de dades d'una empresa d'OpenERP és el resultat del mapatge del disseny de classes de l'ERP cap el SGBD PostgreSQL que és el que proporciona la persistència necessària per als objectes.

En OpenERP el model de dades es descriu i es manipula a través de les classes i els objectes definits amb el llenguatge Python.

Als annexos del web trobareu l'apartat "Eines per ajudar-nos a conèixer la BD d'OpenERP". Amb aquestes eines podreu obtenir diagrames parcials relacionals de la base de dades d'OpenERP.

En conseqüència, l'OpenERP no facilita cap disseny entitat-relació sobre la base de dades d'una empresa ni tampoc cap diagrama del model relacional. No obstant això, si ens interessa disposar d'un model relacional, es troben moltes eines que ens el poden construir a partir de la base de dades implementada en el SGBD PostgreSQL.

1.2.1 Com identificar les taules que contenen determinada informació?

Si sorgeix la necessitat de detectar la taula o les taules on resideix determinada informació, és perquè es coneix l'existència d'aquesta informació gestionada des de l'ERP i, per tant, es coneix algun formulari de l'ERP a través del qual s'introdueix la informació.

L'OpenERP possibilita, mitjançant els clients web i GTK, recuperar el nom de la classe Python que defineix la informació que s'introdueix a través d'un formulari i el nom de la dada membre de la classe corresponent a cada camp del formulari. Aquesta informació permet arribar a la taula i columna afectades, tenint en compte dues qüestions:

- El nom de les classes Python d'OpenERP sempre són en minúscula (s'utilitza el guió baix per fer llegible els mots compostos) i segueix la nomenclatura `nom_del_modul.nom1.nom2.nom3...` en la qual s'utilitza el punt per indicar un cert nivell de jerarquia. Cada classe Python d'OpenERP és mapada en una taula de PostgreSQL amb moltes possibilitats que el seu nom coincideixi amb el nom de la classe, tot substituint els punts per guions baixos.

- Els noms dels atributs d'una classe Python sempre són en minúscula (s'utilitza el guió baix per fer llegible els mots compostos). Cada dada membre d'una classe Python d'OpenERP que sigui persistent (una classe pot tenir dades membres calculades no persistents) és mapat com un atribut en la corresponent taula de PostgreSQL amb el mateix nom.

Noms de classes OpenERP i corresponents taules PostgreSQL

La classe Python `sale.order` d'OpenERP està pensada per descriure la capçalera de les comandes de venda i la corresponent taula a PostgreSQL és `sale_order`.

La classe Python `sale.order.line` d'OpenERP està pensada per descriure les línies de les comandes de venda i la corresponent taula a PostgreSQL és `sale_order_line`.

La classe Python `sale.order` d'OpenERP té una dada membre anomenada `data_order` i, en conseqüència, la taula `sala_order` de PostgreSQL conté l'atribut `data_order`.

D'aquesta manera, coneixent el nom de la classe i el nom de la dada membre, és molt possible conèixer el nom de la taula i de la columna corresponents. Es pot configurar els clients web i GTK per tal que informin del nom de la classe i de la dada membre en situar el ratolí damunt les etiquetes dels camps dels formularis. Aquesta opció es configura de la manera següent:

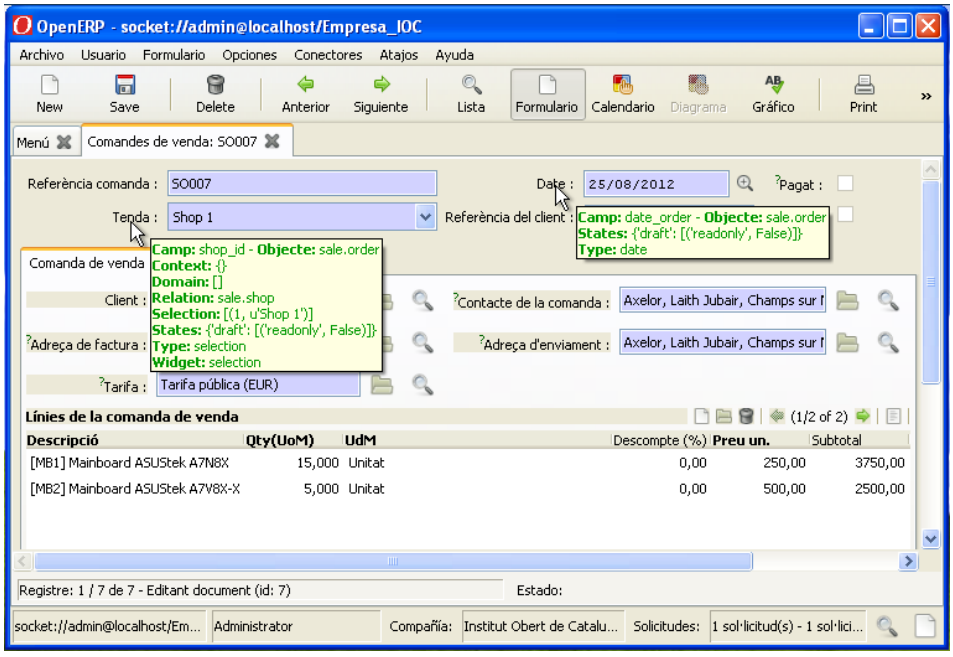
- En el client GTK cal activar l'opció *Ajuda | Activa les ajudes del mode de depuració*.
- En el client web de la versió 6.0 no calia activar res i sempre que l'usuari situava el punter del ratolí damunt les etiquetes del camp, apareixia la

informació d'ajuda, fet que era de gran ajuda per als desenvolupadors però que confonia als usuaris finals. Per això, la versió 6.1 del client web és similar al client GTK i cal activar la funcionalitat amb l'opció *Activar mode de desenvolupador* de la pantalla de diàleg que apareix en prémer el botó *Quant a* de la part superior dreta del client web.

La figura 1.1 mostra la informació que facilita el client GTK en posar el ratolí sobre l'etiqueta de dos camps del formulari *Comandes de Venda.* Podem observar-hi que:

- Per l'etiqueta *Data* obtenim que es tracta del camp `data_order` de l'objecte `sale.order` i, per tant, si necessitem efectuar una consulta sobre la base de dades accedint a la data de la comanda, sabem que haurem d'anar al camp `data_order` de la taula `sale_order` de la base de dades de PostgreSQL.
- Per l'etiqueta *Tenda* tenim molta més informació ja que es tracta del camp relacional `shop_id` de l'objecte `sale.order` que fa referència a la relació `sale.shop`. Amb aquesta informació, si necessitem efectuar una consulta a la base de dades per accedir a la descripció de la botiga corresponent a la comanda sabem que haurem d'anar al camp `shop_id` de la taula `sale_order` i a través d'aquest camp, establir un `join` amb la clau primària de la taula `sale_shop`.

Figura 1.1. Ajuda del mode de depuració que facilita el client GTK en situar el ratolí damunt les etiquetes dels camps en un formulari



Un client web amb el mode desenvolupador activat no només facilita informació en posar el ratolí sobre l'etiqueta dels camps del formulari, sinó que, a més, al costat del títol de qualsevol formulari obert apareix un desplegable, tal com mostra la figura 1.2, amb diverses opcions que poden ajudar-nos com a desenvolupadors de mòduls d'OpenERP.

Figura 1.2. Ajudes en el client web amb el mode desenvolupador activat

La figura 1.2 mostra com, per defecte, l'opció del desplegable que apareix en tenir el mode desenvolupador activat és *Debug View #...* amb un valor al costat del símbol coixinet que correspon al número identificador de la vista activa. La resta d'opcions són:

- *View Fields* que permet obtenir una llista dels camps de la vista actual, amb els paràmetres corresponents.
- *Fields View Get* que mostra l'XML generat per la vista actual. Cal tenir en compte que si la vista s'ha generat a partir de diverses vistes XML heretant les unes de les altres, aquí se'n mostra el resultat final.
- *Manage Views* que mostra un llistat de les vistes relacionades amb la vista actual. Des d'aquest punt es poden crear noves vistes, eliminar-les o editar-les, encara que és recomanable utilitzar aquesta funcionalitat només per consultar. Es recomana realitzar les modificacions creant nous mòduls, per no perdre les modificacions davant actualitzacions de l'ERP.
- *Edit TreeView*, *Edit SearchView*, *Edit Action* i *Edit Workflow* que serveixen per accedir a l'edició de les vistes relacionades amb la vista actual.
- Si estem editant un registre (mode formulari) o consultant-lo (mode pàgina), apareix una nova opció *View Log (perm_read)* que mostra informació relativa al registre actual.

1.2.2 Com facilitar l'accés de només lectura a la base de dades?

Les empreses acostumen a tenir, entre els seus responsables, usuaris finals que poden efectuar consultes no previstes a la base de dades i que, per aconseguir-

ho, poden utilitzar eines gràfiques per elaborar consultes o, fins i tot, si són prou espavilats, executar consultes SQL des d'una consola d'accés.

> L'accés a la base de dades per part d'usuaris finals ha de ser de només lectura.

En aquesta situació, cal facilitar als usuaris que calgui, un usuari per accedir a l'SGBD PostgreSQL amb els privilegis d'accés, en mode consulta, als objectes de la base de dades que correspongui. S'aconsella seguir dos passos:

1. Crear els usuaris d'SGBD amb les contrasenyes que corresponguin.

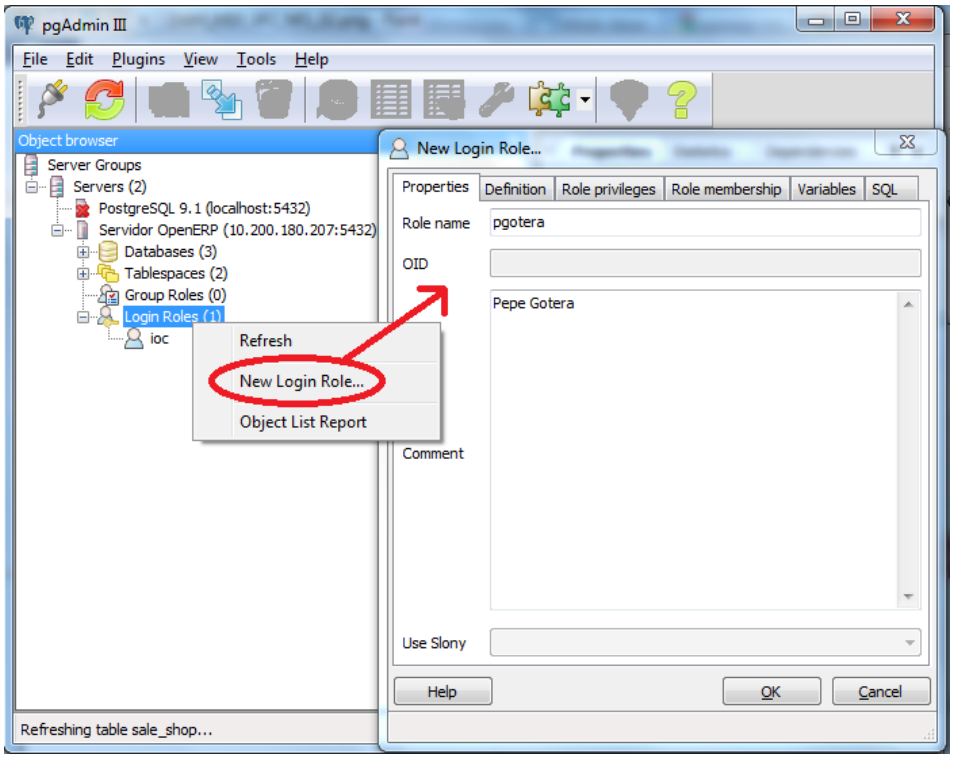
2. Donar els privilegis d'accés adequats als usuaris que corresponguin.

Crear els usuaris d'SGBD amb les contrasenyes que corresponguin

Per tal de crear els usuaris d'SGBD utilitzem una eina d'administració de PostgreSQL (pgAdmin o la pròpia consola textual psql si ens hi movem amb soltesa). Hem de connectar-nos al SGBD PostgreSQL amb un usuari amb privilegi de creació d'usuaris (rol `CREATEROLE`) de l'SGBD. Amb aquestes dues coses, podem crear els usuaris d'SGBD amb les contrasenyes que corresponguin.

En una implantació d'OpenERP en la qual cada usuari de l'empresa té un inici de sessió (*login*) específic, sembla adequat utilitzar el mateix nom d'usuari de l'OpenERP per crear un usuari d'accés a l'SGBD. La contrasenya pot ser inicialment la mateixa, tot i que l'usuari haurà de ser conscient que un canvi de contrasenya en l'ERP no afectaria a la contrasenya de PostgreSQL ni viceversa.

Figura 1.3. Pantalla de pgAdmin per crear un nou usuari (Login Role)



La figura 1.3 mostra la pantalla que facilita pgAdmin per crear un nou usuari (botó secundari del ratolí damunt del node *Login Roles* de l'*Object Browser* i seleccionar l'opció *New Login Role*).

Distinció entre usuaris, grups d'usuaris i rols en PostgreSQL

L'eina pgAdmin no mostra, per enlloc, els conceptes usuari i grups d'usuaris i en canvi mostra els termes *Login Role* i *Group Role*. Si fem una ullada a la documentació del llenguatge SQL de PostgreSQL veiem l'existència de les instruccions `create user` (per a la creació d'usuaris) i `create group` (per a la creació de grups d'usuaris).

PostgreSQL utilitzava els conceptes usuaris i grups d'usuaris en versions anteriors a la 8.1, versió en la qual va substituir aquests conceptes pel concepte de rol amb dues variants (rol que pot iniciar sessió, *Login Role*, i rol que no pot iniciar sessió *Group Role*) i va introduir la nova instrucció SQL `create role`. Per mantenir la compatibilitat amb el joc d'instruccions SQL de les versions anteriors a la 8.1 va mantenir l'existència de les instruccions `create user` i `create group`, de manera que `create user` equival a la creació d'un *Login Role* i `create group` equival a la creació d'un *Group Role*.

Donar els privilegis d'accés adequats als usuaris que correspongui

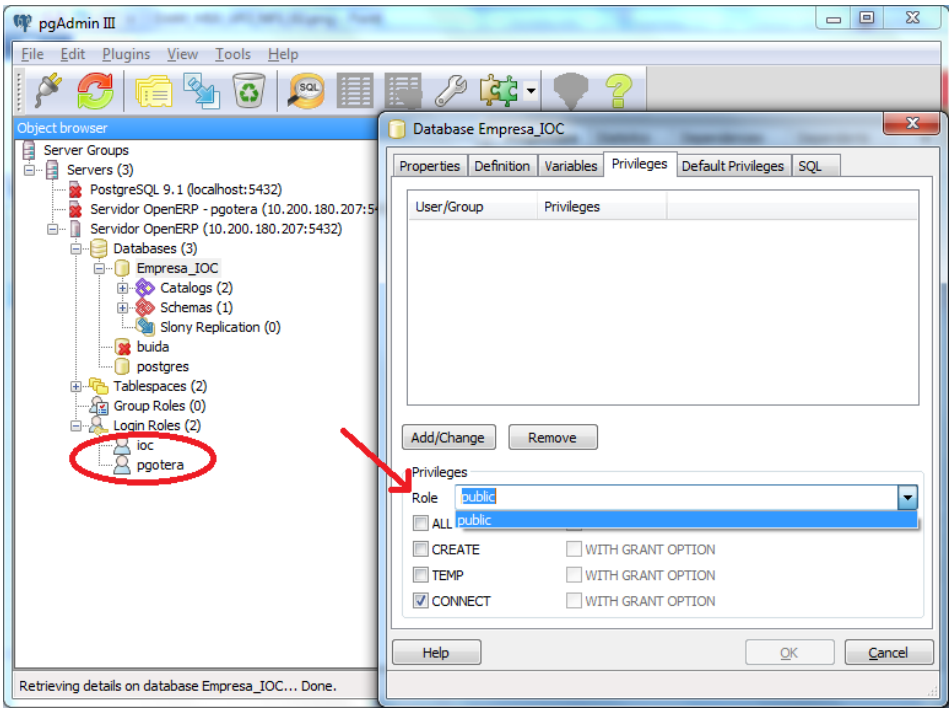
L'eina pgAdmin (la versió 1.8.4 de 8 de juny de 2008 instal·lada per OpenERP 6.1 i la versió 1.14.3 de 1 de juny del 2012) té certes limitacions a l'hora d'assignar privilegis, ja que no permet l'assignació a *Login Role* i, en conseqüència, si utilitzem aquesta versió ens veiem obligats a crear un *Group Role* al qual assignem els *Login Role* i concedim privilegis als *Group Role*. Un altre funcionament una mica defectuós en aquestes versions de pgAdmin es troba en el fet que una vegada creat un *Group Role* cal situar-se damunt el node de definició del servidor i refrescar les dades, per tal que els objectes de les diverses bases de dades del servidor puguin concedir privilegis al nou *Group Role*.

L'assignació de privilegis s'efectua sobre cada objecte i, a través de pgAdmin, per donar privilegis a un objecte determinat cal situar-se damunt del nom de l'objecte, dins l'*Object Browser*, editar les seves propietats (amb el botó secundari del ratolí) i anar a la pestanya *Privileges*. Els privilegis es poden concedir a nivell de base de dades (per permetre o no l'accés), a nivell d'esquema (per utilitzar-lo i/o crear-hi objectes), a nivell d'objectes de la base de dades (taules, vistes, funcions, disparadors...) i fins i tot a nivell de columnes de les taules i vistes. Per donar privilegis d'accés a columnes, cal situar-se damunt la columna i editar les seves propietats.

PostgreSQL va incorporar la possibilitat d'assignar privilegis a nivell de columna, en la versió 8.4. Cal tenir-ho present per si hem instal·lat el servidor PostgreSQL que incorpora la distribució d'OpenERP, ja que es tracta d'una versió 8.3. La versió 1.8.4 de l'eina pgAdmin no permet assignar privilegis a nivell de columna, ja que és coetània de PostgreSQL 8.3. Les versions de pgAdmin coetànies de servidors PostgreSQL 8.4 o posteriors mostren desactivades les opcions de concessió de privilegis a nivell de columna si el servidor PostgreSQL no facilita la funcionalitat. En cas d'haver de donar privilegis a nivell de columna en un servidor que no facilita la funcionalitat, es pot crear una vista que accedeixi a les columnes i donar privilegis d'accés a la vista definida.

La figura 1.4 mostra la limitació de pgAdmin per concedir privilegis als *Login Role*. Podem observar-hi la concessió de privilegis a nivell de la base de dades *Empresa_IOC* i veiem que hi ha dos *Login Role* creats (*ioc* i *pgotera*) i, en canvi, en el desplegable *Role* de la pestanya *Privileges* de la base de dades *Empresa_IOC* només apareix el rol *public* (que es refereix a tothom). Si tinguéssim *Group Role* definits, aquests sí que hi apareixerien.

Figura 1.4. Pantalla de pgAdmin per concedir privilegis d'accés a un base de dades



Per tant, si volem assignar privilegis directament a un usuari (*Login Role*) haurem d'utilitzar la instrucció SQL `grant` des de la consola textual psql.

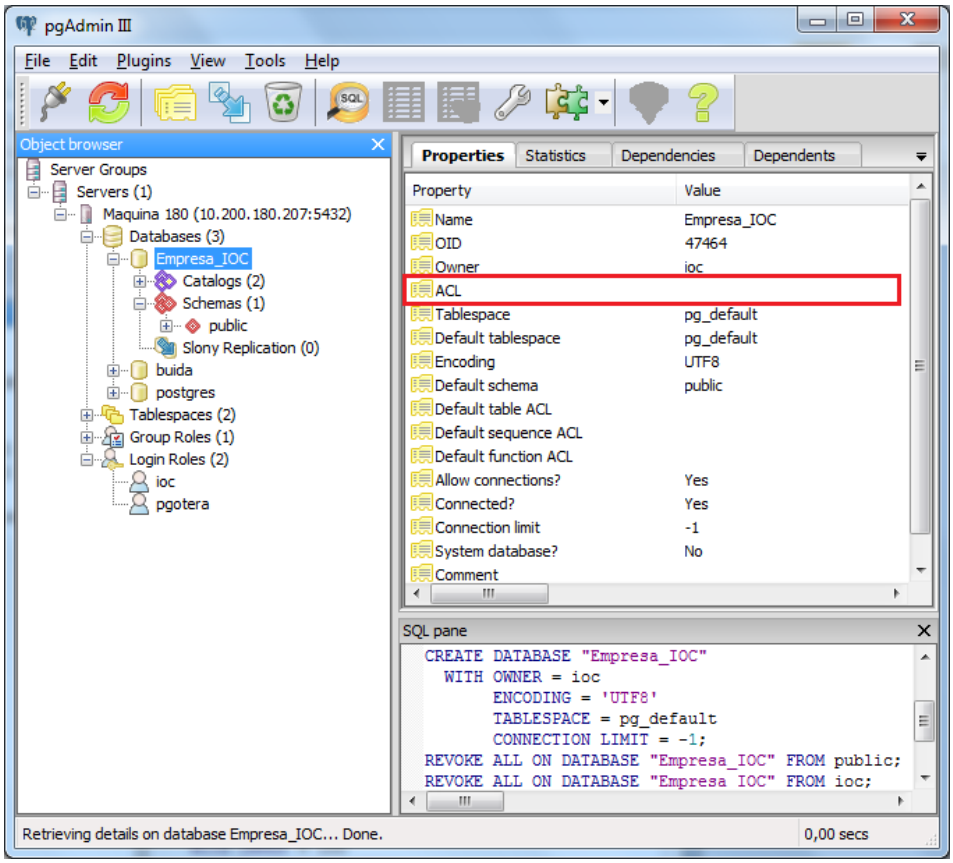
És altament recomanable fer una ullada a la part de la documentació de PostgreSQL referent a l'assignació de privilegis. La documentació de PostgreSQL es troba a www.postgresql.org/docs i per nosaltres té interès fer una ullada als capítols 20 (*Database Roles*) i 5.6 (*Privileges*), així com a les ordres SQL `grant`, `revoke` i `create role`.

Una manera ràpida de veure, des de pgAdmin, els privilegis concedits a un objecte, és situar-nos damunt de l'objecte i veure, a la part dreta de la pantalla, el contingut de la propietat ACL (*Access Control List*), tal com mostra la figura 1.5.

La figura 1.5 mostra el contingut de la llista ACL sobre la base de dades *Empresa_IOC*. Podem situar-nos damunt de qualsevol objecte de la base de dades (esquema, taula, vista, columna...) i tindrem accés a la llista ACL dels privilegis concedits sobre aquell objecte.

En el cas de la figura 1.5 observem com la base de dades *Empresa_IOC* no té definida la llista ACL i això és un problema, ja que en principi permet l'accés a qualsevol usuari. Per tal que no hi hagi cap privilegi d'accés, la llista ACL hauria de mostrar el valor `{}`. Podem aconseguir-ho si editem les propietats de la base de dades, naveguem fins la pestanya *Privileges* i afegim el rol `public` sense arribar-li a concedir cap privilegi, de manera que quan enregistrem el canvi, veurem que el valor d'ACL és `{}`.

Figura 1.5. Propietat ACL sobre una base de dades de PostgreSQL



Un punt molt important a tenir en compte en la gestió de privilegis de PostgreSQL és conèixer els privilegis existents, de forma automàtica, després de la creació d'una base de dades. Cal saber que:

- La base de dades es crea amb ACL no definida, fet que permet que qualsevol usuari del servidor PostgreSQL pugui obrir sessió en aquella base de dades.
- PostgreSQL facilita el rol `public` que engloba a tots els usuaris de forma automàtica.
- PostgreSQL facilita, a totes les bases de dades, l'esquema `public`, propietat de l'usuari que ha creat la base de dades, i amb privilegis d' utilització de l'esquema (`usage`) i creació d'objectes (`create`) al rol `public` (és a dir, a qualsevol usuari). Així, si observem la propietat ACL de l'esquema `public` d'una base de dades creada per l'usuari `ioc`, veiem el valor `{ioc=UC/ioc, =UC/ioc}` que hem de llegir com: l'usuari `ioc` té privilegis `UC` (`usage+create`) i el rol `public` (no apareix a l'esquerra del símbol =) té privilegis `UC` (`usage+create`) i que en ambdós casos han estat concedits per l'usuari `ioc` (valor que apareix després del símbol `/`).

Un usuari qualsevol, pel fet de pertànyer al rol `public`, té accés `UC` sobre l'esquema `public` de qualsevol base de dades. Això implica que pot veure la relació (noms) de tots els objectes existents a l'esquema (taules, vistes...), veure la descripció de qualsevol objecte (taules, vistes...) i crear nous objectes dins l'esquema, però no

pot accedir als continguts de les taules ni vistes, a no ser que el propietari d'aquests objectes li concedixi accés.

En cas que haguem de facilitar accés a la base de dades corresponent a una empresa de PostgreSQL a nous usuaris i no ens interessi mantenir aquesta situació, procedirem de la següent manera:

- Definirem el valor de la propietat ACL de la base de dades i indicarem els usuaris als quals es facilita el privilegi de connexió.
- Modificarem el valor de la propietat ACL de l'esquema `public`, eliminarem l'assignació de privilegis al rol `public` i assignarem la utilització (només `usage`) de l'esquema `public` als usuaris o rols corresponents. Aquesta acció executada mentre el servidor OpenERP està engegat pot provocar que OpenERP no pugui connectar amb la base de dades fins que es reinicïi el servidor OpenERP.
- Assignarem els privilegis (normalment de lectura) als usuaris o rols corresponents sobre els objectes (taules, vistes, columnes...) que interessi.

Exemple d'assignació de privilegis a través d'ordres SQL des de consola psql

Suposem que tenim l'usuari *pgotera* acabat de crear; per tant, encara no se li ha concedit cap privilegi. Suposem també que hem eliminat l'accés `public` a l'esquema `public`.

Mirem la seqüència d'instruccions SQL, des de la consola psql, per aconseguir que l'usuari *pgotera* pugui veure, per les comandes de venda, el número, la data de la comanda i el nom de la botiga que ha efectuat cada comanda. Suposem que es tracta d'un servidor PostgreSQL 9.1 resident a la màquina d'IP 10.200.190.207, que *ioc* és un superusuari i que la base de dades s'anomena *Empresa_IOC*.

```
C:\...>psql –Uioc –h10.200.190.207 –dEmpresa_IOC
Password for user ioc:
...
```

Ja hem establert connexió amb l'usuari *ioc*. Per donar permís de connexió a l'usuari *pgotera* cal fer:

```
=# grant connect on database "Empresa_IOC" to pgotera;
GRANT
```

Per donar permisos d'utilització de l'esquema `public` a l'usuari *pgotera* (això és necessari ja que s'ha eliminat l'accés `public` a l'esquema `public`):

```
=# grant usage on schema public to pgotera;
GRANT
```

Per donar permís de lectura a la taula `sale_shop` (botigues) a l'usuari *pgotera*:

```
=# grant select on table sale_shop to pgotera;
GRANT
```

Per donar permís de lectura a les columnes `id` (número de comanda), `date_order` (data de comanda) i `shop_id` (identificador de la botiga que ha efectuat la comanda), de la taula `sale_order` (comandes), a l'usuari *pgotera*:

```
=# grant select (id, date_order, shop_id) on
–# sale_order to pgotera;
GRANT
```

S'ha decidit donar únicament accés a les tres columnes per fer notar que en ocasions pot ser necessari donar permís a nivell de columnes perquè hi pot haver altres columnes amb informació confidencial a les quals no ha de tenir accés l'usuari *pgotera*. Recordem que això només és possible a partir de la versió 8.4 de PostgreSQL i, per tant, en versions anteriors ens veuríem obligats a donar permís d'accés a tota la taula `sale_order`.

```
=# \q

C:\...>
```

Per comprovar que l'usuari *pgotera* pot obtenir la informació desitjada:

```
C:\...>psql -Upgotera -h10.200.190.207 -dEmpresa_IOC
Password for user pgotera:
...
```

```
=# select so.id as "Comanda",
-#  so.date_order as "Data",
-#  ss.name "Botiga"
-#  from sale_order so, sale_shop ss
-# where so.shop_id = ss.id;
```

L'exemple ens ha mostrat com donar privilegis a nivell de columnes. En un servidor PostgreSQL anterior a la versió 8.4, en no ser possible l'assignació de privilegis a nivell de columnes, podríem crear una vista i donar accés a la vista. Vegem com fer-ho. Connectats com a superusuari per poder crear la vista i donar accés de consulta a l'usuari *pgotera*:

```
=# create view ioc_sale_order_pgotera as
-# select id as "comanda", date_order as "data",
-#  shop_id as "botiga"
-# from sale_order;
CREATE VIEW
=# grant select on ioc_sale_order_pgotera to pgotera;
GRANT
```

Connectats com a usuari *pgotera*:

```
=> select * from ioc_sale_order_pgotera;

=> select comanda, data, name
-> from ioc_sale_order_pgotera, sale_shop
-> where botiga = id;
```

Cal anar amb compte amb la definició de vistes dins l'esquema `public`, ja que cohabitaran amb vistes de mòduls de l'OpenERP. En cas de fer-ho és altament recomanable utilitzar una nomenclatura que ens permeti identificar les vistes afegides que no formen part de l'OpenERP; així, a l'exemple, fixem-nos que hem utilitzat el prefix `ioc_` per a la vista creada. També és possible crear esquemes diferenciats de l'esquema `public` que utilitza PostgreSQL i crear-hi les vistes necessàries, però això ja són conceptes avançats d'administració de PostgreSQL que no tractarem ara.

1.2.3 Com accedir a PostgreSQL des d'aplicacions clients?

En moltes ocasions serà precís configurar estacions de treball per tal que algunes aplicacions instal·lades puguin connectar amb l'SGBD PostgreSQL per accedir a les dades emmagatzemades. Aquest és el cas de les eines ofimàtiques actuals, que acostumen a facilitar la connectivitat amb els SGBD a través de connectors ODBC o JDBC que cal tenir instal·lats a la màquina des d'on es vol utilitzar l'eina ofimàtica.

Als annexos del web trobareu l'apartat "Accés a SGBD corporatives des d'eines ofimàtiques" en el qual es mostra com instal·lar i utilitzar connectors ODBC i JDBC.

1.3 Desenvolupament de mòduls en OpenERP

El desenvolupament de mòduls en OpenERP és indispensable per poder adequar l'OpenERP a les necessitats d'implantació en una organització. Per tal d'aprendre com desenvolupar mòduls en OpenERP en primer lloc és convenient conèixer l'eina de diagramació Dia que permet efectuar dissenys UML i per a la qual existeix un connector (*pluggin*) que permet generar mòduls d'OpenERP a partir dels dissenys efectuats amb Dia. Un cop fet això, s'ha d'aprofundir en el disseny de mòduls, generant-los directament amb la utilització dels llenguatges Python i XML.

Per dissenyar diagrames amb l'eina Dia, és convenient tenir uns coneixements bàsics del llenguatge UML, de nivell similar als adquirits en el mòdul de *Programació*. Al web podeu trobar molts documents introductoris al llenguatge UML.

Per desenvolupar mòduls d'OpenERP és necessari el coneixement del llenguatge Python. Als annexos de la pàgina web trobareu l'apartat "Llenguatge Python" amb recomanacions per a l'aprenentatge.

Als annexos del web trobareu l'apartat "Instal·lació de Dia amb Phyton (PyDia) a Windows per desenvolupar mòduls OpenERP".

Trobareu el fitxer uml_test.dia dins l'apartat "Diagrames Dia" dels annexos del web.

1.3.1 Disseny de mòduls amb Dia

Dia és una aplicació gràfica de propòsit general per a la creació de diagrames, desenvolupada com a part del projecte GNOME. Està construïda de forma modular, amb diferents paquets de formes per a diferents necessitats.

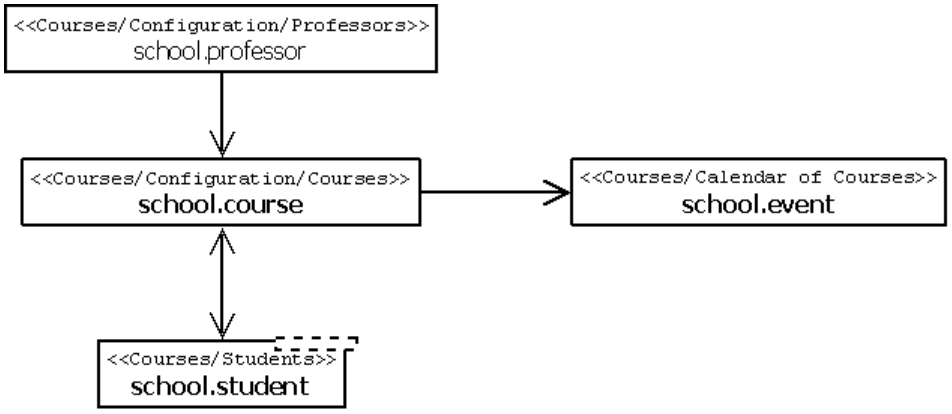
A nosaltres ens interessa aquesta eina per dissenyar els diagrames UML dels mòduls OpenERP que volem desenvolupar i generar-ne, posteriorment, el mòdul per a OpenERP. Per fer això, haurem d'instal·lar Dia amb Python amb l'opció de generació de mòduls OpenERP.

El procés d'instal·lació de Dia amb Python amb l'opció de generació de mòduls OpenERP incorpora un exemple de diagrama UML desenvolupat amb Dia (arxiu `uml_test.dia` facilitat per OpenERP) que correspon a un senzill mòdul de gestió escolar per a OpenERP. Utilitzarem aquest diagrama per iniciar-nos en el disseny de mòduls OpenERP mitjançant l'eina de diagramació Dia.

Disseny de classes amb Dia

Posem en marxa l'eina Dia i obrim el fitxer uml_test.dia. La figura 1.6 correspon al disseny UML del diagrama de classes emmagatzemat.

Figura 1.6. Diagrama UML confeccionat amb l'eina DIA



L'eina Dia permet desenvolupar diversos tipus de diagrama i, com que ens interessen els diagrames UML, caldrà tenir seleccionada l'opció UML a la caixa d'eines de la part esquerra de la pantalla de Dia.

L'exemple de la figura 1.6 correspon a un model per gestionar, en una escola que facilita cursos (en diverses edicions) i disposa de professors, els estudiants dels diversos cursos. El disseny d'aquest model, tal com veurem, pot no semblar molt "lògic" i l'hem de considerar com un exemple que ens permet introduir diversos conceptes bàsics per la generació de mòduls OpenERP amb Dia.

En una situació com la presentada, a l'hora de desenvolupar un diagrama UML, és molt lògic pensar en un disseny que contingui classes per modelar els conceptes curs, professor, edicions dels cursos i estudiants... de forma similar al diagrama `uml_test`, que contempla:

- Classe `school.course`, per modelar els cursos que facilita l'escola.
- Classe `school.event`, per modelar les edicions dels cursos.
- Classe abstracta `school.professor`, per modelar els professors.
- Plantilla de classe `school.student`, per modelar els estudiants.

Una primera ullada al diagrama permet introduir els següents conceptes per mòduls OpenERP:

- Tots els elements d'un diagrama cal escriure'ls en anglès (fins i tot les etiquetes i els textos informatius). Si ens interessa tenir-lo en català o castellà, utilitzarem posteriorment les eines de traducció que facilita OpenERP.
- Totes les classes s'anomenen en minúscules i es recomana incorporar, com a prefix, el nom del mòdul al qual pertanyen.
- Els noms de mòduls, classes i membres de les classes han de ser escrits en minúscula amb guions baixos per fer llegibles els noms compostos.

La nomenclatura indicada és la que marca OpenERP i, com possiblement sabreu, és diferent de les nomenclatures recomanades en altres entorns de POO, com és el cas del llenguatge Java.

Com que els noms de les classes de la figura 1.6 tenen el prefix `school`, deduïm que es tracta del diagrama d'un mòdul anomenat `school`. S'aconsella batejar el diagrama Dia amb el nom del corresponent mòdul, ja que en el procés de generació del mòdul a partir del diagrama, Dia proposa el nom del diagrama com a nom del mòdul, tot i que es pot canviar.

En aquests materials evolucionarem el diagrama `uml_test` amb l'objectiu d'obtenir un mòdul de nom `school` i, per coherència, utilitzarem noms `school##` per les successives versions de diagrames desenvolupats.

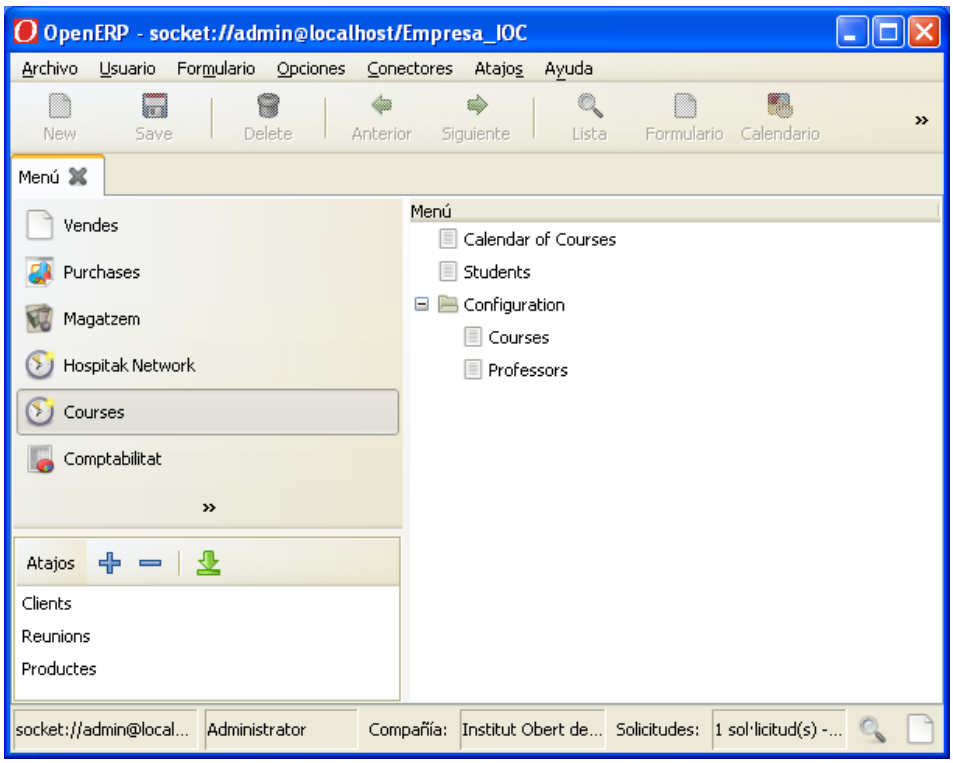
Observem el diagrama de la figura 1.6. Per a la classe `school.event` del mòdul `school`, en tractar-se del model per a les edicions dels cursos potser hagués estat més encertat el nom `school.course.event`, de forma similar a la nomenclatura utilitzada per OpenERP per a les comandes de venda i les seves línies (`sale.order` i `sale.order.line`).

A UML, l'estereotip és un mecanisme d'extensibilitat que permet indicar una distinció d'ús de l'objecte sobre el qual es defineix.

En el diagrama UML de la figura 1.6, `school.professor` és una classe abstracta i `school.student` és una plantilla de classe. A l'hora de generar els mòduls per OpenERP, això no té cap efecte i podrien estar dissenyades com a classes normals.

Fixem-nos que cada classe del diagrama de la figura 1.6 incorpora un estereotip (nom encerclat entre els símbols << >> per sobre del nom de la classe). El procés de generació de mòduls OpenERP a partir de diagrames Dia interpreta el contingut del camp estereotip com la jerarquia de menús, dins OpenERP, en la qual ubicar els formularis generats a partir del diagrama i utilitza el símbol / per separar els diversos nivells. És a dir, segons el diagrama de la figura 1.6, el formulari corresponent al manteniment dels cursos estarà en una opció de menú anomenada *Courses*, dins el submenú *Configuration* del menú principal *Courses* (ja que l'estereotip de la classe `school.course` és `<<Courses/Configuration/Courses>>`). La figura 1.7 mostra la jerarquia de menús esperada a partir del diagrama de la figura 1.6.

Figura 1.7. Jerarquia de menús pel mòdul desenvolupat a partir del diagrama



És aconsellable instal·lar el mòdul `uml_test` original facilitat per OpenERP, per poder comparar el disseny efectuat mitjançant l'eina Dia amb el resultat final dins OpenERP.

Instal·leu el mòdul `uml_test` original facilitat per OpenERP, seguint les indicacions finals de l'annex "Instal·lació de Dia amb Phyton (PyDia) a Windows per desenvolupar mòduls OpenERP".

En el diagrama de la figura 1.6 observem que entre les classes hi ha uns connectors (associacions entre classes) que són únicament de caire informatiu i no tenen cap incidència en la generació del mòdul OpenERP. Les relacions entre les classes es defineixen a través d'atributs relacionals dins de cada classe; la facilitat de diagramació d'associacions entre classes que facilita l'eina Dia no és aprofitada pel mòdul `uml_dia` quan genera els mòduls OpenERP; en conseqüència, la utilització dels connectors associatius de Dia serveixen, únicament, com a documentació gràfica.

Cal interpretar les associacions que observem a la figura 1.6 com:

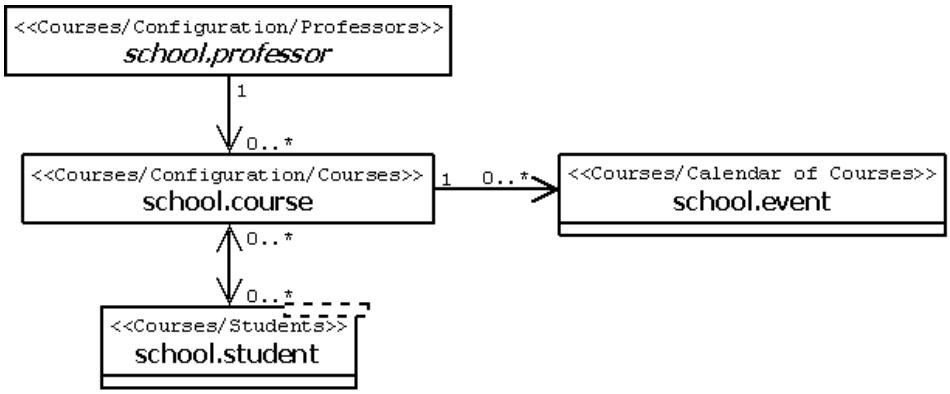
1. Un professor (fletxa de `school.professor` cap a `school.course`) pot impartir diversos cursos.

2. Un curs (fletxa de `school.course` cap a `school.event`) pot tenir diverses edicions.

3. Un curs pot tenir diversos estudiants i un estudiant pot cursar diversos cursos (fletxa amb doble sentit entre `school.course` i `school.student`).

Si seleccionem qualsevol de les associacions del diagrama de la figura 1.6 i n'editem les propietats, observem que l'eina Dia, per documentar encara més el diagrama, permet indicar els rols i la multiplicitat. Així, podríem retocar el diagrama de la figura 1.6 per aconseguir un diagrama més documentat com el de la figura 1.8.

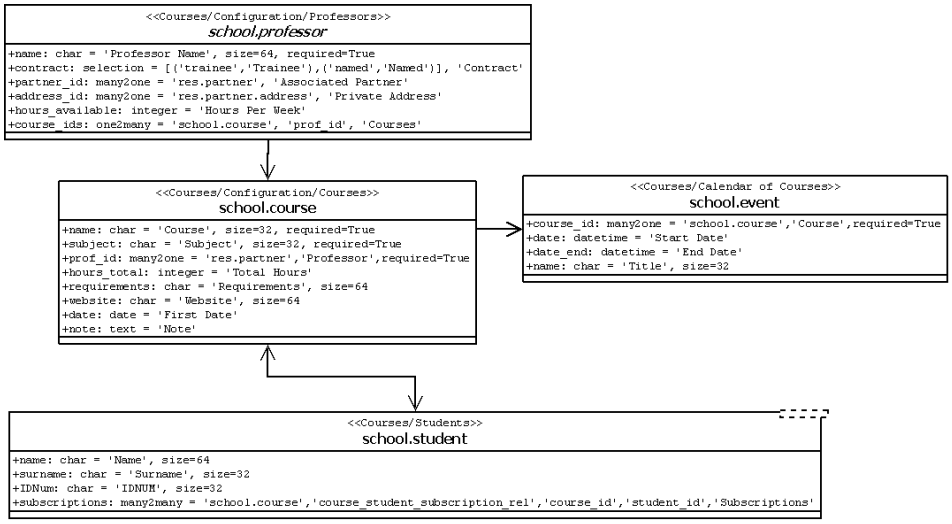
Figura 1.8. Diagrama UML amb multiplicitats a les associacions



El diagrama de classes de les figura 1.6 i figura 1.8 mostra únicament la capçalera de les classes (nom i estereotip) i aquestes classes poden contenir membres (dades i mètodes). Per visualitzar dades i/o mètodes (atributs i/o operacions, segons nomenclatura de l'eina Dia) d'una classe, cal seleccionar la classe i editar les seves propietats (botó secundari del ratolí), i activar les caselles de verificació *Atributs visibles* i *Operacions visibles*. En activar-les per a totes les classes, obtenim la visualització de la figura 1.9.

Una ullada ràpida al diagrama de la figura 1.9 ens permet afirmar que les seves classes no incorporen cap operació i únicament contenen atributs. Aprofitarem les classes d'aquest diagrama per exemplificar els diversos tipus d'atributs de les classes d'un mòdul OpenERP i com es defineixen en un diagrama dissenyat amb l'eina Dia.

Figura 1.9. Diagrama UML amb la visualització d'atributs i operacions activades



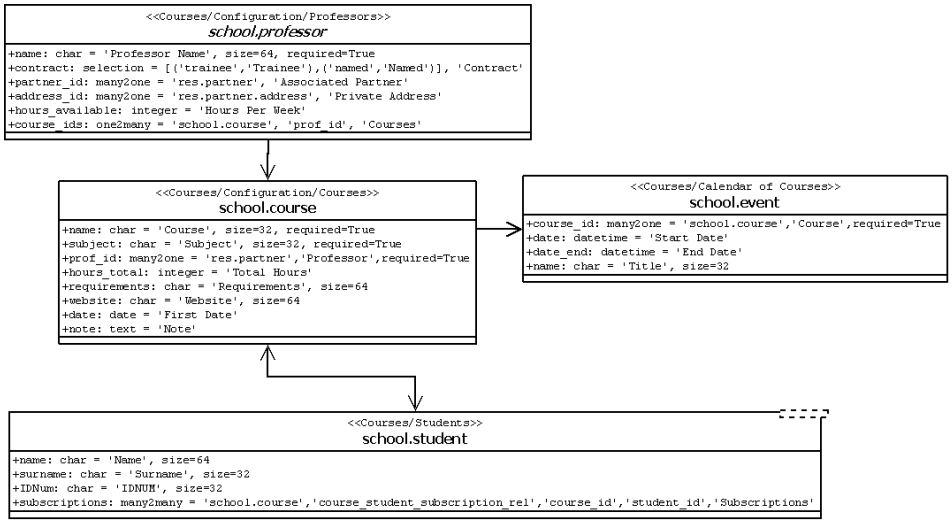
Disseny d'atributs de classe amb Dia

Els atributs de les classes d'un mòdul OpenERP són bàsicament de tres categories: **bàsicsosimples**, **relacionals** i **funcionals**.

> Els atributs **simples o bàsics** són aquells destinats a contenir un valor concret (enter, real, booleà, cadena...). Els atributs **relacionals** són aquells destinats a representar les relacions entre classes. Els atributs **funcionals** són uns camps especials perquè no s'emmagatzemen a la base de dades i es calculen en temps real a partir d'altres atributs.

Situem-nos en el mòdul del fitxer uml_test.dia i activem la visibilitat d'atributs per a totes les classes (figura 1.10). Fixem-nos, per exemple, en els vuit atributs de la classe `school.course`: set són simples o bàsics (`name`, `subject`, `hours_total`, `requirements`, `website`, `date` i `note`) i un és relacional (`prof_id`).

Figura 1.10. Diagrama uml_test.dia amb visibilitat de tots els atributs



Atributs simples o bàsics

Un diagrama Dia amb la visualització d'atributs activada mostra per cada atribut simple, la seqüència d'informació següent:

1. Atribut de visibilitat (+ per públic, - per privat i # per protegit).
2. Nom de l'atribut, seguit del símbol :.
3. Tipus d'atribut (`boolean`, `integer`, `float`, `char`, `text`, `date`, `datetime`, `binary`, `selection`, `reference`).
4. Símbol =.
5. Etiqueta pel camp, entre cometes simples, d'obligada existència.
6. Cap o més paràmetres, seguint la sintaxi `nom=valor` i separats per coma.

Així, la classe `school.course` del diagrama uml_test.dia té un atribut públic (símbol `+`), destinat a emmagatzemar el nom (`name`) del curs, amb etiqueta `Course`, de tipus caràcter (`char`) de 32 caràcters com a molt (paràmetre `size`) i amb obligatorietat de valor (`required='True'`). Fixem-nos que la definició del camp en el diagrama és:

```
+name: char = 'Course', size=32, required = 'True'
```

La taula 1.1 presenta, amb detall, els diversos tipus de dades possibles pels atributs simples o bàsics.

Taula 1.1. Tipus de dades pels atributs simples o bàsics en mòduls OpenERP

Nom	Què emmagatzema?	Observacions
`boolean`	Valors lògics	Valors vàlids: `True` i `False`
`integer`	Valors enters	
`float`	Valors reals	Pot anar acompanyat d'un paràmetre `digits` destinat a indicar la precisió i l'escala de l'atribut. L'escala és el nombre de dígits després del punt decimal i la precisió és el nombre total de dígits significatius (abans i després del punt decimal). Si el paràmetre `digits` no hi és, serà considerat com un nombre en punt flotant amb doble precisió.
`date`	Valor data	
`datetime`	Valor data i hora en un mateix camp	
`char`	Cadena de longitud limitada	És obligatori indicar el paràmetre `size=valor`, per indicar la màxima longitud permesa.
`text`	Cadena de longitud il·limitada	
`binary`	Cadena binària	
`selection`	Un valor d'una llista de valors possibles predefinits	En aquest cas, abans del valor de l'etiqueta del camp, cal indicar la llista de valors possibles, com a seqüència de parelles (valor, etiqueta): `[(valor1, etiqueta1), (valor2, etiqueta2)...]`.
`reference`	Punter cap a un objecte existent en OpenERP	Els objectes d'OpenERP que es poden seleccionar per una referència es defineixen en el menú *Settings|Personalització|Objectes de baix nivell|Sol·licituds|Tipus de referències en sol·licituds* d'OpenERP.

Com a exemple d'atribut `selection` podem observar l'atribut `contract` de la classe `school.professor` del diagrama uml_test.dia. Fixem-nos que la llista conté dos valors (`trainee` i `named`) pels quals la visualització des del programa serà `Trainee` i `Named`, a no ser que apliquem un procés de traducció de l'aplicació (però a nivell de la base de dades, els valors emmagatzemats sempre seran `trainee` i `named`, tot i que pugui semblar més lògic que a la base de dades s'hi emmagatzemi uns valors més simples com, per exemple, les incials `t` i `n`.

La taula 1.2 presenta, amb detall, els paràmetres que poden acompanyar als atributs, distingint si són obligatoris o optatius i si són comuns o específics d'algun tipus de dada.

Taula 1.2. Paràmetres que poden acompanyar els atributs bàsics

Nom	Obligatori	Atribut al qual acompanya	Significat
`size`	Sí	`char`	Llargada màxima de la cadena
`digits`	No	`float`	Indicar la precisió i escala en format `(p,s)`. Veure explicació del camp `float` a la taula 1.1.
`required`	No	Qualsevol	Indicar l'obligatorietat de l'atribut amb el valor `True`. Per defecte, el seu valor és `False`.
`readonly`	No	Qualsevol	Indicar que l'atribut és de només lectura, amb el valor `True`. Per defecte, el seu valor és `False`.
`select`	No	Qualsevol	Exigir que a nivell de la base de dades es generi un índex per aquest camp per tal d'optimitzar els processos de recerca, amb el valor `1`. Per defecte el seu valor és `0`.
`help`	No	Qualsevol	Indicar el missatge d'ajuda que OpenERP visualitza en situar el ratolí damunt l'etiqueta del camp. Provoca que l'etiqueta vagi acompanyada per un petit interrogant a l'extrem superior esquerre.
`translate`	No	`char` o `text`	Indicar que el valor del camp pot ser traduït pels usuaris, amb el valor `True`. Per defecte, el seu valor és `False`.

En el client web, els camps amb l'atribut `translate` a `True` presenten, en mode edició, una banderola al final de la caixa de text que conté el valor del camp, per indicar que el camp és traduïble. Prement el ratolí damunt la banderola, s'obre una pantalla que recull tots els camps traduïbles del formulari i ens permet traduir-los. Podeu comprovar-ho en el camp *Nom* del formulari de productes d'OpenERP. En el client GTK, també hi ha una banderola, però la funcionalitat de traducció no funciona en la versió 6.1.

La introducció dels diversos valors per definir un atribut simple des de l'eina Dia, s'efectua des de la pestanya *Atributs* de les propietats de la classe, seguint els criteris següents:

- La visibilitat de l'atribut (públic, privat o protegit) s'escull en el camp *Visibilitat*.

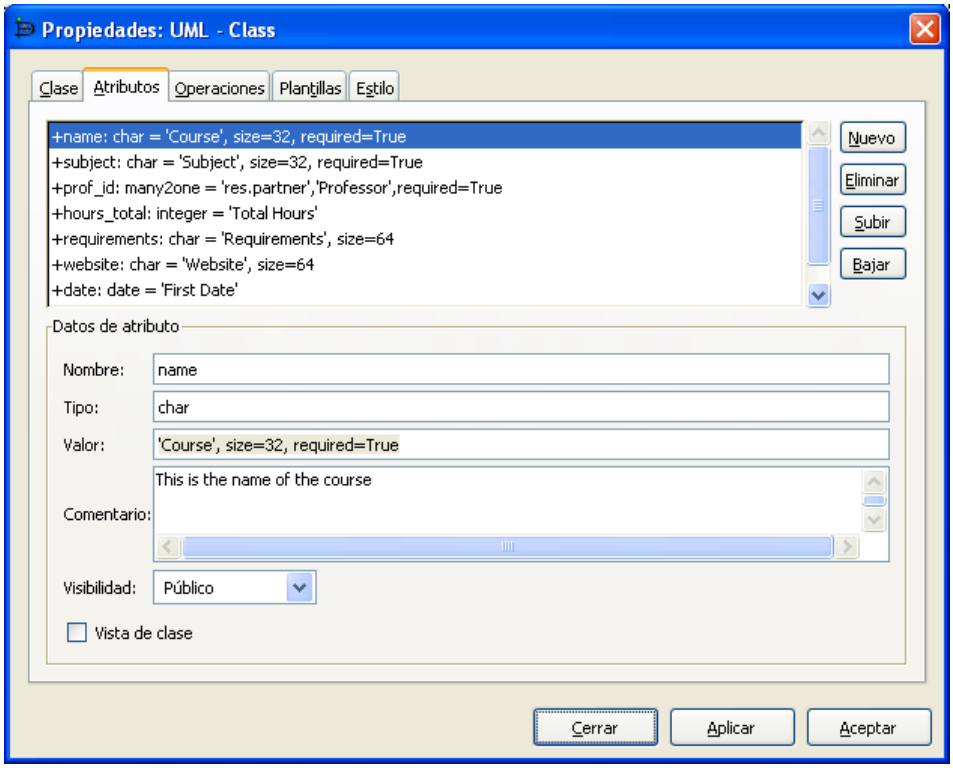
- El nom de l'atribut s'introdueix a l'apartat *Nom*.
- El tipus de l'atribut s'introdueix a l'apartat *Tipus*.
- El nom de l'etiqueta (obligatori i entre cometes simples), seguit de la seqüència de parelles `paràmetre=valor` que correspongui, separades per coma, s'introdueix a l'apartat *Valor*.
- L'ajuda que acompanya el camp (paràmetre `help`), s'introdueix a l'apartat *Comentari*.

Podeu comprovar l'aplicació dels criteris anteriors tot editant les propietats de qualsevol dels atributs bàsics de les classes del diagrama`uml_test`. La figura 1.11 mostra la definició de l'atribut `name` de la classe `school.course`.

Us heu adonat que, en els paràmetres presentats, no n'hi ha cap que faci referència a la definició d'un camp identificador de la classe?

OpenObject és el *framework* sobre el qual està desenvolupat OpenERP, de manera que en parlar de facilitats de desenvolupament, ens hem de referir a l'OpenObject. Es pot utilitzar OpenObject per desenvolupar altres aplicacions que no tinguin res a veure amb OpenERP.

Figura 1.11. Exemple de definició d'atribut bàsic de classe de mòdul OpenERP des de Dia



A OpenObject, cada classe té un atribut identificador, de nom `id`, generat automàticament per la capa *Object Relational Mapping* (ORM) d'OpenObject, de manera que no s'ha d'indicar en la definició dels atributs de la classe. El valor de l'atribut `id` per a cada objecte és generat automàticament per OpenObject i identificarà l'objecte (el registre a nivell de la base de dades) per sempre més. Aquest identificador també s'utilitza per relacionar objectes (integritat referencial a nivell de base de dades).

Llavors, si l'atribut identificador dels objectes d'una classe el genera OpenObject, com podem definir un camp que identifiqui un objecte tal com estem acostumats en la majoria d'aplicacions informàtiques? Per exemple, fixem-nos en l'atribut

`IDNum` de la classe `school.student`. L'ajuda introduïda diu, textualment: *This is the uniq ID of the student*. Com podem aconseguir que `IDNum` sigui veritablement identificador? Doncs per això haurem d'utilitzar la restricció d'unicitat que facilita OpenObject, però que no podem indicar a través de l'eina Dia i ho haurem de fer, més endavant, via el llenguatge Python.

L'atribut `IDNum` no segueix el conveni d'estar escrit en minúscules. Això provoca que no s'instal·li en servidors OpenERP en plataformes Linux.

L'atribut `id` és, doncs, un atribut especial però no és l'únic. La taula 1.3 presenta amb detall els atributs de classe bàsics especials per tenir en compte quan desenvolupem OpenObject.

Taula 1.3. Atributs de classe bàsics especials en OpenObject

Nom	Tipus	Explicació
`id`	`integer`	Atribut identificador dins la classe utilitzat per identificar cada objecte (registre a nivell de base de dades) per sempre més i per establir les relacions entre objectes de diverses classes (integritat referencial a nivell de base de dades). El crea l'eina ORM d'OpenObject i, per tant, no s'ha d'indicar en dissenyar una classe.
`create_date`	`datetime`	Emmagatzema el moment temporal en què es crea l'objecte. El crea l'eina ORM.
`write_date`	`datetime`	Emmagatzema el moment temporal de la darrera modificació de l'objecte. El crea l'eina ORM.
`create_uid`	`integer`	Emmagatzema l'identificador (`id`) de l'usuari que va crear l'objecte. Aquest usuari és un objecte de la classe `res.user` (o registre de la taula `res_user`). El crea l'eina ORM.
`write_uid`	`integer`	Emmagatzema l'identificador (`id`) de l'usuari que ha modificat per darrera vegada l'objecte. Aquest usuari és un objecte de la classe `res.user` (o registre de la taula `res_user`). El crea l'eina ORM.
`name`	`char`	És l'atribut que utilitza OpenObject quan ha de mostrar objectes en una llista desplegable. En conseqüència, aquest atribut és obligatori en qualsevol classe C susceptible de ser referenciada des d'altres classes, ja que en aquestes caldrà facilitar una llista desplegable per facilitar la selecció dels objectes de C.
`active`	`boolean`	És un atribut que afegirem a aquelles classes per les quals interessi poder amagar objectes en algun moment de la seva vida. Podria servir, per exemple, per amagar objectes històrics sense eliminar-los. Un objecte inactiu ja no és accessible ni apareix per enlloc, fins que es torni a activar.
`state`	`selection`	És un atribut que afegirem a aquelles classes per les quals interessi poder definir diversos estats del cicle de vida d'un objecte.
`parent_id`	`integer`	És un atribut que permet definir estructures jeràrquiques entre objectes de la mateixa classe. Acostuma a sorgir com a conseqüència d'una associació reflexiva en una classe.

Com a exemple de la utilització que fa OpenObject de l'atribut `name`, podem anar a consultar la llista de clients d'una empresa, on observarem la columna *País*, que

mostra el contingut de l'atribut `name` del país al qual pertany el client. Si anem a la base de dades a consultar el contingut de la taula `res_partner_address` (que conté les adreces dels clients), hi veurem el camp `country_id`, com a clau forana de l'atribut `id` de la taula `res.country`. A la llista de clients, però, no veiem el contingut de l'atribut `country_id` (que és el que relaciona el client amb el país) sinó el contingut de l'atribut `name` del país referenciat.

OpenERP té moltes classes que contenen l'atribut `active`, com per exemple la classe `res.users` (taula `res_users`). Així, per exemple, podeu anar a *Settings / Usuaris*, editar el registre *Demo User*, desmarcar la casella *Actiu* per deixar-lo no actiu i enregistrar el canvi. Podreu observar com el registre *Demo User* ha desaparegut de la llista d'usuaris i ja no sortirà mai més fins que el fem actiu. Per fer actiu un registre inactiu hem d'aconseguir accedir-hi per canviar el valor de la casella *Actiu*. Per aconseguir-ho ens hem de valer de la recerca avançada dels formularis. Per fer-ho cal seleccionar el camp *Actiu* i indicar que volem veure els registres que el tenen en valor fals.

Com a exemple d'utilització de l'atribut `state`, podem fixar-nos en la classe `sale.order` corresponent a les comandes de venda. OpenERP hi defineix l'atribut `state` de tipus `selection` amb els següents valors que defineixen els diferents estats pels quals pot passar una comanda de venda:

```
[('draft', 'Quotation'),
('waiting_date', 'Waiting Schedule'),
('manual', 'To Invoice'),
('progress', 'In Progress'),
('shipping_except', 'Shipping Exception'),
('invoice_except', 'Invoice Exception'),
('done', 'Done'),
('cancel', 'Cancelled')
]
```

Un exemple d'utilització de l'atribut `parent_id` el trobem, en OpenERP, en la definició dels tercers (clients i proveïdors). Si editem un client/proveïdor, veurem com a l'apartat *Informació General* de la pestanya *Vendes & Compres*, es pot indicar quina és l'empresa pare que, en cas de tenir valor, és un altre tercer.

Atributs relacionals

Entre les classes d'OpenObject es pot establir tres tipus de relacions, de manera similar a les relacions entre les entitats en un model entitat-relació per a bases de dades. Es defineixen via atributs relacionals dins les classes. Tenim, en conseqüència, tres tipus d'atributs relacionals: **molts a un**, **un a molts**, **molts a molts**.

Atribut relacional "molts a un" (`many2one`)

L'atribut relacional "molts a un" (`many2one`) s'utilitza per representar una relació cap a una classe pare, de molts a un, és a dir, molts objectes de la classe que conté l'atribut poden estar relacionats amb un mateix objecte de la classe pare.

En el diagrama `uml_test` observem molts atributs relacionals `many2one`. Un exemple el tenim a la classe `school.event`, en la qual l'atribut `course_id` està

destinat a contenir, per a cada objecte de la classe `school.event`, l'identificador de l'objecte pare de la classe `school.course`.

OpenERP mostra els camps `many2one` acompanyats per una llista per seleccionar l'objecte de la classe pare. És a dir, en el formulari per gestionar els cursos (objectes de la classe `school.event`), el camp `course_id`, amb etiqueta *Course*, estarà acompanyat d'un desplegable per seleccionar el curs que correspongui d'entre els objectes de la classe `school.course`). Això obliga que la classe referenciada pel camp `many2one` contingui el camp `name`.

A nivell de la base de dades, els atributs `many2one` es mapen en atributs dins la taula corresponent a la classe que els conté, amb restriccions d'integritat referencial cap a l'atribut identificador de la taula referenciada. Així, l'atribut `many2one` `course_id` de la classe `school.event` del diagrama uml_test.dia, queda mapat a la taula `school_event` en el camp `course_id` amb una restricció d'integritat referencial cap a l'atribut `id` de la taula `school_course` (automàticament creat per OpenERP).

Per definir un atribut `many2one` en l'eina Dia, cal crear l'atribut indicant que és del tipus `many2one` i a l'espai valor introduir:

```
'classeReferenciada','etiquetaDelCamp'...
```

Els punts suspensius del final indiquen que es pot utilitzar alguns paràmetres opcionals. Concretament:

- `ondelete`: per indicar com actuar quan l'objecte referenciat per l'atribut sigui eliminat. Valors possibles: `cascade`, `set null`. Valor per defecte: `set null`.
- `required`, `readonly`, `select`: amb funcionament idèntic als camps no relacionals.

Podem observar que l'atribut `course_id` de la classe `school.event` té com a valor:

```
'school.course','Course',required=True
```

Està acordat, entre els desenvolupadors en OpenERP, anomenar els atributs `many2one` amb el nom de la classe a la qual referencien (o darrer nivell) acompanyat de `_id`, com és el cas de l'atribut `course_id`.

Atribut relacional "un a molts" (`one2many`)

Un atribut relacional "un a molts" (`one2many`) s'utilitza per representar una relació cap a una classe filla, d'un a molts, és a dir, un objecte de la classe que conté l'atribut pot estar relacionat amb molts objectes de la classe filla.

Tot atribut one2many és complementari d'un atribut many2one que ha d'existir obligatòriament a la classe filla. Per definir un atribut one2many en l'eina Dia cal crear l'atribut indicant que és del tipus one2many i a l'espai valor introduir:

```
'classeReferenciada','nomAtributClasseReferenciada','etiquetaDelCamp'...
```

La classe filla (classeReferenciada) ha de contenir l'atribut nomAtributClasseReferenciada de tipus many2one apuntant a la classe en la qual hi ha l'atribut one2many.

Els punts suspensius del final indiquen que es pot utilitzar alguns paràmetres opcionals. Concretament:

- invisible: per impedir que el seu contingut sigui visible en els formularis corresponents a la classe a la qual pertany el camp. Valors possibles: True, False. Valor per defecte: False.

- readonly: amb funcionament idèntic als camps no relacionals.

Està acordat, entre els desenvolupadors en OpenERP, anomenar els atributs one2many, amb el nom de la classe a la qual referencien (o darrer nivell) acompanyat de _ids.

En el diagrama uml_dia observem l'atribut course_ids del tipus one2many a la classe school.professor, amb el següent contingut a l'espai valor:

```
'school.course', 'prof_id', 'Courses'
```

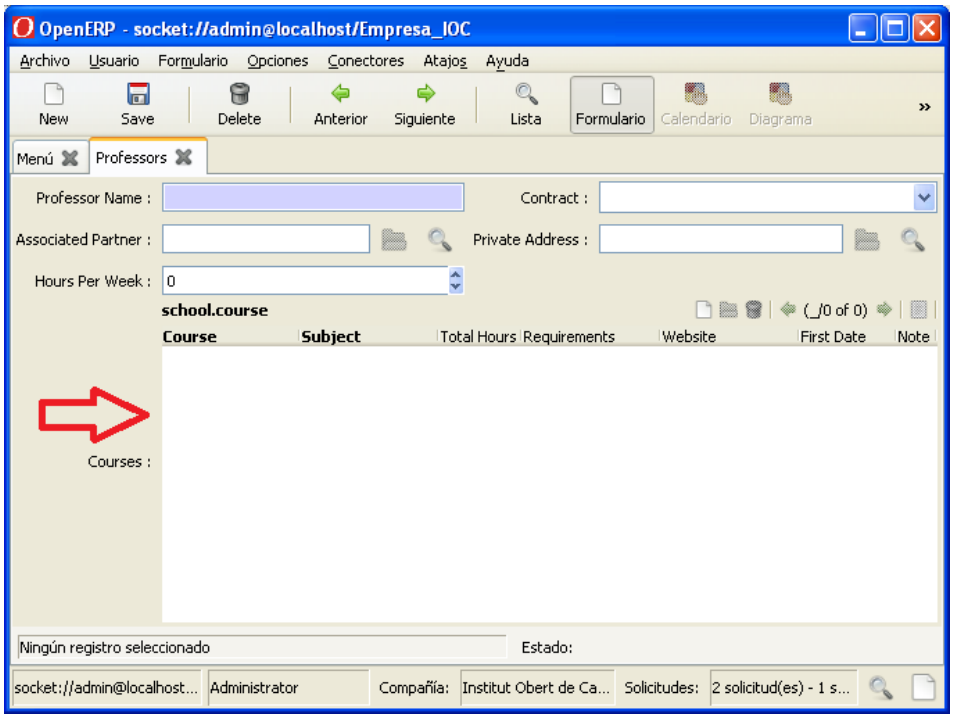
L'atribut course_ids de la classe school.professor relaciona un professor amb el conjunt de cursos que imparteix. En conseqüència, la classe school.course ha de contenir l'atribut prof_id de tipus many2one cap a la classe school.professor.

L'existència d'un atribut one2many implica l'existència d'un atribut many2one, però l'existència d'un atribut many2one no implica l'existència d'un atribut one2many. La definició dels atributs one2many té dos objectius:

- Aconseguir que en les vistes (formularis) que continguin el camp one2many aparegui una llista dels objectes referenciats (tal com mostra la zona apuntada per la fletxa de la figura 1.12).

- Facilitar l'accés, quan es desenvolupen mètodes (lògica de negoci) de la classe que conté el camp one2many, cap als objectes referenciats per l'objecte sobre el qual s'executi el mètode.

FIGURA 1.12. Formulari per al manteniment de professors amb un subformulari



El desenvolupament de mètodes (lògica de negoci) en OpenERP es troba a l'apartat "OpenERP: la vista i el controlador"

El paràmetre `invisible=True` en un atribut `one2many` provoca que cap formulari en el qual aparegui l'atribut incorpori la llista dels objectes referenciats. D'aquesta manera, l'atribut únicament pot servir per facilitar l'accés als objectes referenciats en els mètodes desenvolupats sobre la classe.

Si observem el diagrama `uml_dia` original facilitat per OpenERP, podem observar que l'atribut `prof_id` de la classe `school.course` conté, a l'espai valor:

```
1 'res.partner','Professor',required=True
```

Si heu instal·lat el mòdul `uml_test` proporcionat per OpenERP, podeu comprovar el mal funcionament si feu la creació d'un professor i intenteu assignar-li cursos i des d'un curs intenteu assignar-li el seu professor.

La definició de l'atribut `prof_id` a la classe `school.course` és errònia, ja que no apunta a la classe `school.professor`, sinó a la classe `res.partner`. El mòdul generat quan es manté aquesta definició té un mal funcionament.

Per solucionar la situació errònia i aconseguir que l'atribut `prof_id` de la classe `school.course` sigui complementari de l'atribut `course_ids` de la classe `school.professor`, haurem de canviar el contingut a l'espai valor de l'atribut `prof_id` de la classe `school.course`, per:

```
1 'school.professor','Professor',required=True
```

La figura 1.12 també ens permet observar l'efecte dels atributs `many2one`. Observem que els camps etiquetats per *Associated Partner* i *Private Address* van acompanyats per dues icones: una icona lupa, que permet accedir a la llista dels objectes de la classe referenciada per aquests camps, i una icona carpeta que permet accedir a la fitxa de l'objecte referenciat. Si observem els corresponents atributs dins la classe `school.professor` veurem que es tracta dels camps `partner_id` i `address_id`, de tipus `many2one`.

Atribut relacional "molts a molts" (many2many)

L'atribut relacional "molts a molts" (many2many) s'utilitza per representar una relació de molts a molts entre dos objectes, és a dir, quan cada objecte d'una classe A pot estar relacionat amb molts objectes d'una classe B i cada objecte d'una classe B pot estar relacionat amb molts objectes d'una classe A.

Un exemple de relació many2many el trobem entre les classes school.student i school.course, ja que un estudiant es pot inscriure a molts cursos i en un curs hi pot haver molts estudiants inscrits.

Per definir un atribut many2many en l'eina Dia cal crear l'atribut en alguna de les dues classes que relaciona, o en ambdues, indicant el tipus many2many i, a l'espai valor, introduir:

```
'classeReferenciada','nomTaulaRelacional','nomCampFKPropi','
    nomCampFKAltraClasse','etiquetaDelCamp'
```

Està acordat, entre els desenvolupadors en OpenERP, anomenar els atributs many2many, amb un nom (adequat a les dues classes que relaciona) acompanyat de _ids. Fixem-nos que la definició d'un atribut many2many implica la creació d'una taula relacional a la base de dades.

En el diagrama uml_dia observem l'atribut subscriptions del tipus many2many a la classe school.student, amb el següent contingut a l'espai valor:

```
'school.course','course_student_subscription_rel','course_id','student_id','
    Subscriptions'
```

L'atribut subscriptions relaciona la classe school.student (en la qual és definit) amb la classe school.course, fet que provocarà l'aparició d'una nova taula a la base de dades, de nom course_student_subscription_rel, que tindrà l'atribut course_id amb restricció d'integritat referencial cap a la classe school.student i l'atribut student_id amb restricció d'integritat referencial cap a la classe school.course.

L'atribut subscriptions presenta algunes anomalies de notació:

- El nom no segueix el conveni, ja que seria bo que el seu nom finalitzés en _ids.
- El nom de la taula relacional que es generarà a la base de dades seria bo que comencés amb el prefix school_, de la mateixa manera que la resta de taules vinculades a aquest mòdul.
- Els noms assignats als tercer i quart paràmetres del contingut de l'espai valor haurien d'estar a l'inrevés, de manera que course_id fos el nom per fer referència a la classe school.course i student_id fos el nom per fer referència a la classe school.student.

Així doncs, seria millor redefinir l'atribut subscriptions, canviant-li el nom per course_ids i amb el següent contingut a l'espai valor:

```
'school.course','school_student_course_rel','student_id',
'course_id','Subscriptions to courses'
```

Un atribut `many2many` provoca que l'eina Dia, en generar el formulari per al manteniment dels objectes de la classe, hi incrusti una zona de tipus *llista* per accedir als registres de la classe referenciada per l'atribut `many2many`. La figura 1.13 mostra com serà el formulari generat per l'eina Dia per al manteniment dels estudiants. En ella hi observem la zona apuntada per la fletxa, corresponent a la llista de cursos assignats a l'estudiant indicat a la capçalera del formulari.

Figura 1.13. Formulari per al manteniment d'estudiants amb els cursos on estan subscrits

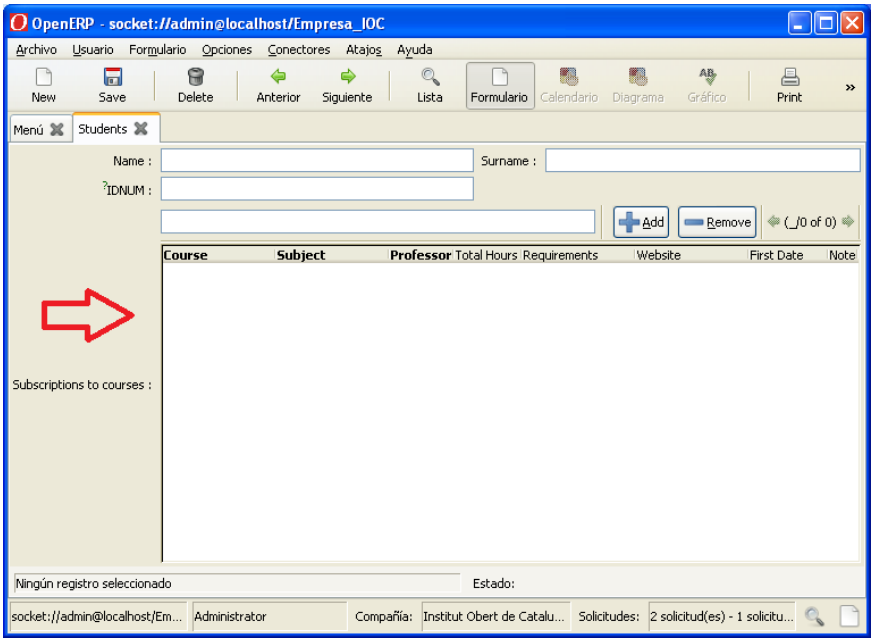
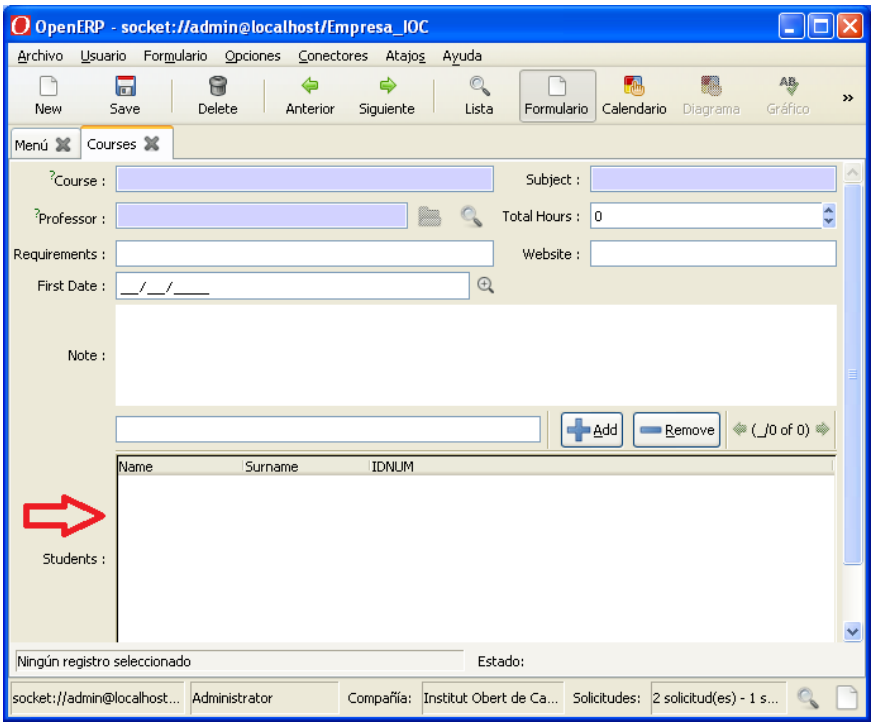


Figura 1.14. Formulari per al manteniment de cursos amb els estudiants que hi estan subscrits



En cas que interessi que en el formulari dels cursos hi aparegui una zona de tipus *llista* per accedir als estudiants subscrits al curs, caldrà definir un atribut `many2many` a la classe `school.course`, en el qual el nom de la taula relacional i els noms dels seus camps han de coincidir amb la definició de l'atribut `many2many` de la classe `school.student`. Així, per exemple, podem definir-hi l'atribut `student_ids` amb el següent contingut a l'espai valor:

```
'school.student','school_student_course_rel','course_id','student_id','Students'
```

La figura 1.14 en mostra el resultat en el formulari de manteniment dels cursos.

Generació de mòdul OpenERP des de Dia

L'eina Dia permet generar, a partir d'un diagrama dissenyat per a un mòdul OpenERP, el corresponent mòdul per ser instal·lat seguint el procediment habitual.

A l'annex "Instal·lació de Dia amb Phyton (PyDia) a Windows per desenvolupar mòduls OpenERP" trobareu les indicacions per aconseguir que Dia pugui generar mòduls OpenERP per a la seva instal·lació.

Per aconseguir el mòdul OpenERP, cal haver incorporat a l'eina Dia el guió `codegen_openerp.py` de Python i, en tal situació, l'opció *Fitxer* | *Exportar* de Dia mostra l'opció d'exportació *PyDia Code Generation (OpenERP) (*.zip)* que és la que hem d'utilitzar per aconseguir el mòdul per ser instal·lat en OpenERP.

En el moment d'exportar un diagrama Dia cap a mòdul OpenERP, l'eina proposa com a nom de mòdul el mateix nom del diagrama, però es pot decidir el nom que es desitgi i l'eina genera un fitxer ZIP amb el nom indicat, que no es podrà canviar de manera simple, ja que en el seu interior es generen continguts que utilitzen el nom indicat i caldria fer diversos retocs. En conseqüència, el nom indicat en el moment de la generació és el nom del mòdul que podrà ser instal·lat en OpenERP.

Errors de l'eina Dia o del guió codegen_openerp.py

Entre generació i generació de mòdul OpenERP des de l'eina Dia cal reiniciar l'eina, ja que sembla que, en algunes ocasions, la generació no parteix de zero, sinó que afegeix el codi generat al codi de la generació prèvia.

Cal generar el mòdul en el mateix directori en el qual es troba el diagrama Dia, ja que del contrari es genera un mòdul buit.

Cal tenir en compte que el guió `codegen_openerp.py`, facilitat en el mòdul `uml_dia` per la versió 5.0 d'OpenERP, no genera correctament l'arbre de menús adequat a la versió 6.1 d'OpenERP. Recordem que el menú en el qual havia d'estar ubicat el manteniment corresponent a una classe s'indicava en el camp *estereotip* de les propietats de la classe. En conseqüència, abans d'instal·lar un mòdul generat per Dia amb el guió `codegen_openerp.py` de la versió 5.0, en un servidor OpenERP 6.1, caldrà retocar la definició de menús en el mòdul generat.

Amb motiu de tenir accés a la història de les diverses millores incorporades al diagrama Dia, corresponents al mòdul `school`, afegirem un sufix numèric `_##` a les diverses versions que anem generant.

Suposant que hem incorporat els diversos retocs indicats fins ara al diagrama `uml_dia` a original, procedim a enregistrar-lo amb nom `school_01.dia` i procedim a generar el mòdul `school`.

Per a poder instal·lar el mòdul `school` aquí generat, cal retocar els menús tal com s'indica al final de l'annex "Instal·lació de Dia amb Phyton (PyDia), en Windows, per desenvolupar mòduls OpenERP", utilitzant els menús indicats en school_01_menus.txt, fitxer ubicat a l'apartat "Diagrames Dia" dels annexos del web. També trobareu el mòdul ja preparat sota el nom school_01.zip dins l'apartat "Mòduls OpenERP" dels annexos del web.

Abans d'instal·lar el nou mòdul `school`, si a l'empresa on pensem instal·lar-lo hi ha instal·lat el mòdul `uml_test` cal desinstal·lar-lo, ja que ambdós mòduls tenen taules coincidents i són incompatibles. Posteriorment, haurem d'establir una connexió amb la base de dades de PostgreSQL i eliminar les

taules generades pel mòdul `uml_test`: `school_professor`, `school_course`, `school_event`, `school_student` i `course_student_subscription_rel`. Una vegada instal·lat el nou mòdul `school`, la base de dades contindrà les taules: `school_professor`, `school_course`, `school_course_event`, `school_student` i `school_student_course_rel`.

1.3.2 Disseny del model amb Python

L'eina de diagramació Dia, amb l'extensió per generar mòduls per a OpenERP, facilita el disseny de mòduls per OpenERP. Les possibilitats que permet l'eina Dia són, però, limitades i, en conseqüència, ens cal aprendre com escriure directament els mòduls utilitzant els llenguatges Python i XML.

Un mòdul d'OpenERP incorpora, en el seu interior, tots els elements del patró model-vista-controlador en el qual es basa el desenvolupament d'OpenObject. L'eina Dia permet definir el model via diagrama UML i l'extensió per generar mòduls per a OpenERP ens facilita el mòdul que conté:

- El model, definit a través de llenguatge Python, corresponent al diagrama UML dissenyat.
- Una vista, definida a través del llenguatge XML, generada de forma automàtica.
- Un controlador buit, ja que l'eina Dia no permet dissenyar els mètodes.

En el Technical Memento d'OpenERP hi trobareu el recordatori sobre tot allò que cal saber respecte el disseny del model.

El nostre objectiu final és dissenyar mòduls utilitzant directament el llenguatge Python pel model i el controlador, i el llenguatge XML per la vista.

Estructura d'un mòdul

Els mòduls d'OpenERP s'ubiquen dins la carpeta `addons` ubicada en alguna subcarpeta d'on s'ha instal·lat OpenERP. No podem donar una ubicació concreta ja que ha anat canviant amb les versions d'OpenERP. Així, per exemple:

- En les versions 5.0 i 6.0 d'OpenERP per a Windows s'ubiquen a:

```
camíOnÉsUbicatOpenERP\Server\addons
```

- En la versió 6.1 d'OpenERP per a Windows, a:

```
camíOnÉsUbicatOpenERP\Server\server\openerp\addons
```

Els passos necessaris per a crear un nou mòdul són:

1. Crear la subcarpeta dins la carpeta `addons`.

2. Crear el fitxer __init__.py, necessari perquè Python consideri que el contingut de la carpeta correspon a un paquet.

3. Crear un arxiu de descripció del mòdul, que s'ha d'anomenar __openerp__.py. Fins la versió 5.0 s'anomenava __terp__.py i les versions posteriors continuen admetent aquest nom per qüestions de compatibilitat.

4. Crear els fitxers Python que han de contenir les classes, amb els atributs i els mètodes. En aquests fitxers es defineix el model i el controlador (arquitectura MVC).

5. Crear els fitxers XML per la gestió de les dades (vistes, accions, menús, dades d'exemple...). En aquests fitxers es defineix la vista (arquitectura MVC).

6. De forma opcional, crear informes, assistents i fluxos de treball, que poden estar en qualsevol lloc dins la carpeta, però es recomana ubicar-los en subcarpetes anomenades `report`, `wizard` i `workflow`.

7. De forma opcional, incorporar traduccions i regles de seguretat, que poden estar en qualsevol lloc dins la carpeta, però es recomana ubicar-les en subcarpetes anomenades `i18n` i `security`.

Nomenclatura d'OpenObject vers nomenclatura POO

OpenObject no utilitza la nomenclatura habitual de POO i no parla de classes i d'objectes. Així, les instàncies que gestiona (productes, clients, proveïdors...) s'anomenen recursos (en lloc d'objectes) i l'abstracció o definició de les instàncies s'anomena objecte (en lloc de classe).

Inicialització del mòdul: fitxer __init__.py

El fitxer __init__.py és executat, com en qualsevol mòdul Python, en el procés d'inicialització del programa.

En aquest fitxer cal situar tots els imports de fitxers Python que cal carregar per a l'execució del programa.

Així, si el mòdul conté un fitxer de nom module.py que conté la descripció de les classes, caldrà incorporar, dins el fitxer __init__.py la instrucció:

```
1 import module
```

Observeu el contingut del fitxer __init__.py de qualsevol dels mòduls que incorpora OpenERP o del mòdul `school` generat.

Descripció del mòdul: fitxer __openerp__.py

Tot mòdul d'OpenERP ha de contenir un fitxer anomenat __openerp__.py (fins la versió 5.0 s'anomenava __terp__.py) ubicat a l'arrel de la carpeta corresponent al mòdul. Aquest fitxer ha de tenir format d'un diccionari de Python i és el responsable de determinar, entre altres coses, el nom, la descripció, els autors, la llicència, la versió, la categoria, els fitxers XML que seran analitzats durant l'arrencada del servidor OpenERP i les dependències amb altres mòduls.

Aquest fitxer ha de contenir un diccionari Python amb els següents valors:

- `name`: el nom del mòdul, en anglès.
- `version`: la versió del mòdul.
- `description`: la descripció del mòdul (text).
- `author`: l'autor del mòdul.
- `website`: el lloc web de l'autor del mòdul.
- `license`: la llicència del mòdul (per defecte, la que correspon a la versió de servidor OpenERP on s'instal·larà el mòdul).
- `category`: categoria a la qual pertany el mòdul. Text separat per barres / amb la ruta jeràrquica de les categories.
- `depends`: llista Python de mòduls dels quals depèn aquest mòdul. El mòdul base s'acostuma a afegir perquè conté dades utilitzades en vistes, informes...
- `init_xml`: llista Python de noms de fitxers XML que es carregaran en iniciar el servidor OpenERP amb l'opció `-init=module`. Les rutes dels fitxers han de ser relatives a la carpeta on és el mòdul. En aquesta llista s'acostuma a posar els fitxers relatius a definició de *workflows* i les dades a carregar en el procés d'instal·lació del mòdul.
- `update_xml`: llista Python de noms de fitxers XML que es carregaran en iniciar el servidor OpenERP amb l'opció `-update=module`. Les rutes dels fitxers han de ser relatives al directori on és el mòdul. En aquesta llista s'inclouen els fitxers relatius a vistes, informes i assistents.
- `demo_xml`: llista Python de noms de fitxers XML que es carregaran si s'ha instal·lat el servidor OpenERP amb dades de demostració o exemple. Les rutes dels fitxers han de ser relatives al directori on és el mòdul.
- `installable`: els valors permesos són `True` o `False` i determinen si el mòdul és o no és instal·lable.
- `active`: els valors permesos són `True` o `False` i determinen si el mòdul serà instal·lat automàticament en el procés de creació de l'empresa. Per defecte, `False`.

En Python, els diccionaris es defineixen entre `{}` amb parelles `clau:valor` separades per comes; les llistes entre `[]` amb valors separats per comes.

Definició d'objectes OpenERP amb Python

Observeu el contingut del fitxer __openerp__.py de qualsevol dels mòduls que incorpora OpenERP o el fitxer __terp__.py del mòdul `school` generat per l'eina *Dia*.

El model en OpenERP es descriu a través de classes escrites en Python i OpenObject s'encarrega de fer-les persistents en el SGBD PostgreSQL.

Per definir un objecte (classe) d'OpenERP cal definir una classe de Python i posteriorment instanciar-la, fet que provoca la creació de l'objecte (classe) d'OpenERP. La classe ha d'heretar obligatòriament de la classe `osv` del mòdul `osv`.

L'esquelet de la classe Python que permet definir l'objecte OpenERP té la forma:

```
class name_of_the_object (osv.osv):
  _name = 'name.of.the.object'
  _columns = {...}
  ...
name_of_the_object()
```

Les classes Python que permeten definir objectes (classes) d'OpenERP tenen dos atributs obligatoris i altres opcionals. Els atributs obligatoris són:

- `_name`: nom de l'objecte (classe) d'OpenERP. Aquest nom s'utilitza de forma automàtica per generar el nom de la taula de PostgreSQL, i canvia els punts per guions baixos.
- `_columns`: diccionari Python amb la declaració de les dades de l'objecte (classe) d'OpenERP, que passaran a ser les columnes de la taula de PostgreSQL.

Els atributs opcionals són:

L'aplicació de l'atribut `_auto` queda palesa en el disseny d'informes i quadres de comandament.

- `_auto`: els valors possibles són `True` i `False`. Per defecte `True`. Determina si s'ha de generar la taula PostgreSQL a partir de la definició de l'objecte OpenERP. El valor `False` s'utilitza per objectes OpenERP no persistents que es basen en vistes definides en PostgreSQL.
- `_constraints`: definició de restriccions sobre l'objecte, definides amb codi Python.
- `_sql_constraints`: definició de restriccions SQL sobre l'objecte, definides a la corresponent taula de PostgreSQL.
- `_defaults`: diccionari Python que conté els valors per defecte per les dades (definides a `_columns`) de l'objecte (classe) d'OpenERP que ho precisin.
- `_inherit`: objecte (classe) d'OpenERP del qual hereta l'objecte (classe) que estem definint.
- `_inherits`: llista d'objectes (classes) d'OpenERP dels quals hereta l'objecte (classe) que estem definint. La llista ha de seguir la sintaxi d'un diccionari Python de la forma:

```
{'nom_objecte_pare':'nom_de_camp',...}
```

- `_log_access`: valors possibles `True` i `False`. Per defecte, `True`. Indica si els accessos d'escriptura als recursos (objectes) han de quedar enregistrats. En cas afirmatiu, de forma automàtica es creen a la corresponent taula de PostgreSQL les columnes `create_uid`, `create_date`, `write_uid` i `write_date` per enregistrar els accessos d'escriptura.

- `_order`: nom dels atributs utilitzats per ordenar els resultats dels mètodes de cerca i lectura. Per defecte, s'utilitza el camp `_id` generat automàticament en totes les taules de PostgreSQL corresponents a objectes (classes) d'OpenERP. Exemples:

```
_order = 'name'
```

```
_order = 'date_order desc'
```

- `_rec_name`: nom de l'atribut de l'objecte (classe) d'OpenERP que conté el nom informatiu de cada recurs (objecte) concret. Per defecte `name`.
- `_sequence`: nom de la seqüència SQL que gestiona els identificadors (contingut del camp `_id`) pels recursos d'aquest objecte (classe) d'OpenERP. Per defecte, `nomDeLaTaula_id_seq`.
- `_sql`: codi SQL a executar en la creació de l'objecte, en cas que `_auto` sigui `True`.
- `_table`: nom de la taula SQL si es vol que sigui diferent del nom automàtic a partir del nom de l'objecte substituint els punts per guions baixos.

Una vegada presentats els possibles atributs de les classes Python dissenyades per generar objectes (classes) d'OpenERP, ens cal endinsar-nos en els possibles continguts d'alguns d'aquests atributs.

Tipus de camps

L'atribut `_columns` de la classe Python que defineix l'objecte (classe) d'OpenERP ha de contenir un diccionari Python amb la definició dels atributs de l'objecte d'OpenERP, a partir dels camps (`fields`) predefinits a la classe `osv`.

La sintaxi general per definir una columna és la següent:

```
'nom_camp':fields.tipus_de_camp(paràmetres)
```

on `tipus_de_camp` ha de ser un dels tipus facilitats per la classe `osv` i el contingut de `paràmetres` depèn del tipus del camp.

Hi ha diverses categories de camps:

- Camps simples: `boolean`, `integer`, `float`, `char`, `text`, `date`, `datetime`, `binary` i `selection`.
- Camps relacionals: `many2one`, `one2many`, `many2many` i `related`.
- Camps `function`.
- Camps `property`.

Els tipus de camps simples i relacionals habituals (`many2one`, `one2many` i `many2many`) ja ens són familiars a causa de la seva utilització en el disseny de mòduls amb l'eina de diagramació Dia. Per a tots aquests camps, el contingut de `paràmetres` en la crida `fields.tipus_de_camp(paràmetres)` es correspon amb el contingut de l'espai valor en la definició de cada atribut de classe mitjançant l'eina Dia.

Exemples d'utilització dels tipus de camps simples i relacionals habituals (`many2one`, `one2many` i `many2many`) els podeu trobar a l'apartat "Disseny d'atributs de classe amb Dia" en aquest document.

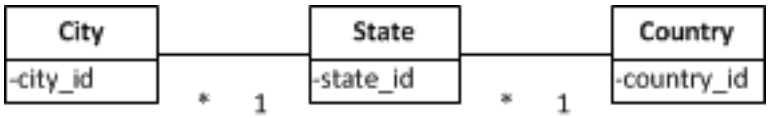
En aquest punt és convenient observar les classes del fitxer `school.py` generat a partir de l'eina Dia. Observem que les classes definides incorporen únicament els atributs obligatoris `_name` i `_columns`. En la classe Python no hi observem cap atribut opcional (valors per defecte, restriccions...) i en l'objecte (classe) d'OpenERP només apareixen camps simples i relacionals habituals (`many2one`, `one2may` i `many2many`). Centrem-nos ara en els tipus de camps encara desconeguts (`related`, `function` i `property`).

Tipus relacional `related`

Els camps `related` permeten accedir a un camp d'un altre objecte referenciat des del propi objecte; és a dir, com si es tractés de camps encadenats.

Com a exemple, suposem que en un mòdul d'OpenERP tenim les classes exemplificades en el diagrama UML de la figura 1.15. En un mòdul per a OpenERP, a la classe `City` hi haurà un camp relacional `many2one` cap a la classe `State`, i a la classe `State` hi haurà un camp relacional `many2one` cap a la classe `Country`. Un camp `related` ens permet accedir des de la classe `City` a la classe `Country` via la classe `State`.

Figura 1.15. Disseny UML amb dues relacions many2one



La definició d'una columna `related` ha de seguir la sintaxi:

```
'nom': fields.related('campPontPropi','campOnAccedir',
                      type='tipusDelCamp', string='etiquetaNouCamp'
                      [,store=True/False][,...]),
```

Els punts suspensius del final indiquen que hi pot haver més paràmetres segons el tipus del camp. Així, en els camps de tipus `many2one`, caldrà afegir un paràmetre `relation` per indicar el nom de la classe relacionada. El paràmetre `store` (per defecte `False`) permet indicar si el valor accedit ha de quedar enregistrat a la base de dades (cas clar de redundància) per tal de facilitar una major eficiència d'OpenERP en els processos de recerca i de lectura. En cas d'activar aquesta redundància, els valors sempre són mantinguts per OpenERP i mai pels usuaris.

Així, per aconseguir que la classe `City` de la figura 1.15 pugui accedir a l'identificador de la classe `Country` via la classe `State`, hauríem de tenir:

```
# Dins la classe module.state:
_columns: {
  ...
```

```
  'country_id': fields.many2one('module.country','Country'),
  ...
}

# Dins la classe module.city:
_columns: {
  ...
  'state_id': fields.many2one('module.state','State'),
  'country_id': fields.related('state_id,'country_id',
                               type='many2one',
                               relation='module.country',
                               string='Country',store=False),
  ...
}
```

Un cas real d'utilització el podem observar en el disseny de l'objecte (classe) `res.partner` d'OpenERP, on veiem les columnes `city`, `function`, `subname`, `phone`, `mobile`, `country` i `email` com a camps `related` a través del camp `address` de la mateixa classe:

```
_columns = {
  ...
  'address': fields.one2many('res.partner.address', 'partner_id','Contacts'),
  ...
  'city': fields.related('address', 'city', type='char', string='City'),
  'function': fields.related('address', 'function', type='char',
                             string='function'),
  'subname': fields.related('address', 'name', type='char',
                            string='Contact Name'),
  'phone': fields.related('address', 'phone', type='char', string='Phone'),
  'mobile': fields.related('address', 'mobile', type='char', string='Mobile'),
  'country': fields.related('address', 'country_id', type='many2one',
                            relation='res.country',string='Country'),
  'email': fields.related('address', 'email', type='char', size=240,
                          string='E-mail'),
  ...
}
```

Exemple d'utilització dels camps related

L'objecte (classe) `school.professor` del mòdul `school` que estem millorant incorpora l'atribut `address_id` que permet accedir a l'adreça del professor (objecte `res.partner.address`).

En executar el formulari de manteniment de professors, apareix el camp *Private Address* amb el contingut identificador de l'adreça. Si es vol tenir accés a tota la informació continguda a l'adreça (carrer, telèfon, correu electrònic...) el formulari permet accedir a la fitxa de l'adreça.

Imaginem-nos que és important, per a nosaltres, que el telèfon de cada professor es vegi directament a la fitxa del professor, sense haver d'accedir a la fitxa de l'adreça.

Per aconseguir-ho, com que el telèfon del professor resideix a `res.partner.address` i ja accedim a aquest objecte a través del camp `address_id`, podem definir el següent camp `_related`:

```
'phone':fields.related('address_id','phone',type='char',size=64,
                       string='Phone',store=False,readonly=True),
```

Podeu efectuar aquesta modificació directament en el programa Python school.py, però no podreu observar-ne l'efecte per què encara no sabem com afegir camps en els fitxers XML que proporcionen la vista dels programes. Si efectueu, però, la inserció del nou camp, a través de l'eina Dia i procediu a generar de nou el mòdul per a OpenERP i actualitzeu el mòdul instal·lat, podreu veure'n l'efecte.

Podeu aprofitar aquesta actualització per canviar l'arxiu __terp__.py per __openerp__.py i actualitzar el seu contingut (autor, versió...)

Per poder instal·lar el mòdul `school` generat cal retocar els menús tal com s'indica al final de l'annex "Instal·lació de Dia amb Python (PyDia) a Windows, per desenvolupar mòduls OpenERP", utilitzant els menús indicats a school_02_menus.txt, fitxer ubicat a l'apartat "Diagrames Dia" dels annexos del web. També trobareu el mòdul ja preparat sota el nom school_02.zip dins l'apartat "Mòduls OpenERP" dels annexos del web.

Tipus `function`

Els camps `function` simulen camps reals però es calculen mitjançant una funció Python enlloc d'emmagatzemar-se a la base de dades PostgreSQL. En casos especials, per augmentar la velocitat de consulta d'OpenERP i facilitar les consultes, es fan redundants a la base de dades, és a dir, s'emmagatzemen a la base de dades, però sempre són calculats i actualitzats a través de funcions i mai pels usuaris.

La definició d'una columna `function` ha de seguir la sintaxi:

```
'nom': fields.function(fnct, arg=None, fnct_inv=None, fnct_inv_arg=None,
                       type='Float', fnct_search=None,
                       string='etiquetaNouCamp', store=False,
                       multi=False [,...]),
```

en la qual:

- Els paràmetres que van acompanyats d'un símbol `=Valor` no són obligatoris i el valor indicat és el que es considera en cas de no explicitar el paràmetre.
- `type`: és el tipus de camp retornat per la funció. Pot ser qualsevol tipus excepte `function`. Els punts suspensius del final indiquen que hi pot haver més paràmetres segons el tipus del camp. Així, en els camps de tipus `'many2one'` caldrà afegir un paràmetre `obj` per indicar el nom de la classe relacionada.
- `store`: per indicar si el camp ha de residir a la taula de la base de dades (redundància).
- `multi`: per indicar que el càlcul de la funció s'efectuï per a diversos camps, a causa que la mateixa funció és invocada en diferents camps (veure, com exemple, el mòdul `auction` d'OpenERP).
- `fnct`: és el mètode que calcula el valor del camp. És un paràmetre obligatori i ha d'existir abans de declarar el camp funcional. Ha de retornar, obligatòriament, un diccionari de parelles de la forma `{id1:valor1, id2:valor2,...}` en el qual `id1`, `id2`... han de ser identificadors d'objectes i `valor1`, `valor2`... els corresponents valors que han de correspondre amb el tipus indicat a `type`. La sintaxi ha de ser:

```
def fnct(self, cr, uid, ids, field_name, arg, context)
```

- `fnct_inv`: és un mètode utilitzat per escriure un valor en lloc del camp. La sintaxi ha de ser:

```
def fnct_inv(obj, cr, uid, id, name, value, fnct_inv_arg, context)
```

- `fnct_search`: és el mètode utilitzat per fer recerques per aquest camp. Ha de retornar la llista de tuples que especifiquin el criteri de cerca a utilitzar pel mètode `search` facilitat per OpenObject. La sintaxi ha de ser:

```
def fnct_search(obj, cr, uid, obj, name, args)
```

Els paràmetres `arg` i `arg_inv` sembla que han de servir per poder passar paràmetres a les funcions o mètodes `fnct` i `fnct_inv`, però no hi ha documentació al respecte ni cap mòdul d'OpenERP (versió 6.1) que els utilitzi.

Els mòduls d'OpenERP estan plens d'exemples de camps funcionals, però no tots són senzills d'entendre en una primera aproximació al tema. Així, per iniciar-nos en els camps funcionals, podem centrar-nos en l'atribut `number_of_days` del mòdul `hr_holidays`. Seria interessant que, si no el teniu instal·lat, procedíssiu a instal·lar aquest mòdul, per poder comprovar el que comentarem a continuació.

El desenvolupament de mètodes (lògica de negoci) en OpenERP, necessari per als camps funcionals, es troba a l'apartat "OpenERP: la vista i el controlador"

Fixeu-vos en la part de la definició de la classe que ens interessa comentar:

```
class hr_holidays(osv.osv):
  ...
  def _compute_number_of_days(self, cr, uid, ids, name, args, context=None):
    result = {}
    for h in self.browse(cr, uid, ids, context=context):
    if h.type=='remove':
      result[h.id] = -h.number_of_days_temp
    else:
      result[h.id] = h.number_of_days_temp
    return result

  _columns = {
    ...
    'type': fields.selection([('remove','Leave Request'),
                              ('add','Allocation Request')],
                             'Request Type', required=True,readonly=True, ...),
    'number_of_days_temp': fields.float('Number of Days', readonly=True,
                                        states={'draft':[('readonly',False)]}),
    'number_of_days': fields.function(_compute_number_of_days,
                                      string='Number of Days',store=True),
    ...
  }
  ...
```

El mètode `_compute_number_of_days` crida el mètode `browse` facilitat per OpenObject. La llista dels mètodes facilitats per OpenObject es troba a l'apartat "OpenERP: la vista i el controlador".

Fixem-nos que `number_of_days` és un atribut funcional que, malgrat ser calculat mitjançant una funció, el seu resultat resta emmagatzemat a la base de dades, a causa de `store=True`. Podem comprovar, mirant la descripció de la taula `hr_holidays` a la taula de PostgreSQL, l'existència de la columna `number_of_days`.

Si utilitzeu el mòdul, observareu que permet gestionar, per cada empleat, els dies d'absència i els dies treballats fora del calendari laboral. Cada recurs (objecte) de l'objecte (classe) `hr.holidays` d'OpenERP correspon a un dels dos tipus possibles (`add` o `remove`) segons la definició de l'atribut `type`. L'atribut `number_of_days_temp` conté el nombre de dies afectats, sempre en positiu. En canvi, l'atribut `number_of_days` vol contenir el nombre de dies afectats, en positiu si es tracta de dies de treball addicional, i en negatiu si es tracta de dies d'absència.

En conseqüència, el camp `number_of_days` es pot calcular a partir del camp `number_of_days_temp` i per això es defineix com un camp funcional, que executa el mètode `_compute_number_of_days` definit prèviament.

Observem també que el mètode invocat retorna el diccionari anomenat `result` format per una única parella on la clau és l'identificador del recurs sobre el qual s'executa i el valor és el nombre de dies calculats.

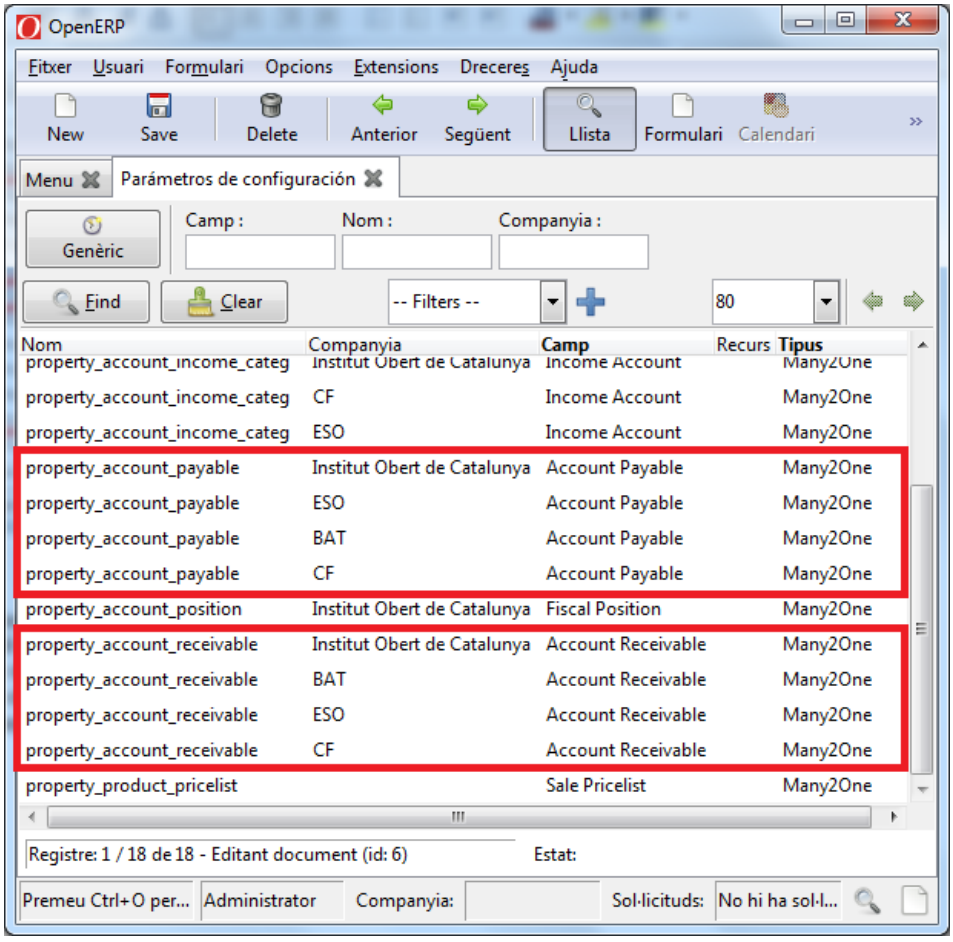
Tipus `property`

Els camps `property` són camps dinàmics amb drets d'accés específics, que permeten contenir diferents valors en funció de la companyia.

Un exemple de funcionament dels camps `property` el podem veure en els camps *Compte a cobrar* i *Compte a pagar* de l'apartat *Comptabilitat* de la fitxa dels tercers, en el qual els valors que proposa OpenERP depenen de la companyia a la qual pertany l'usuari que està gestionant el tercer. Per a comprovar-ho necessitem acomplir els següents requeriments:

- Cal tenir més d'una companyia a la nostra empresa. Per exemple, suposem que a l'empresa que hem anat utilitzant (Empresa_IOC) hi tenim creades les companyies ESO, BAT i CF dedicades als corresponents estudis.
- Cal tenir instal·lat el mòdul oficial de comptabilitat (*Accounting & Finance* – `account_accountant`) i tenir desplegat el pla comptable a les diverses companyies.

Figura 1.16. Llista de paràmetres (camps property) en una empresa d'OpenERP

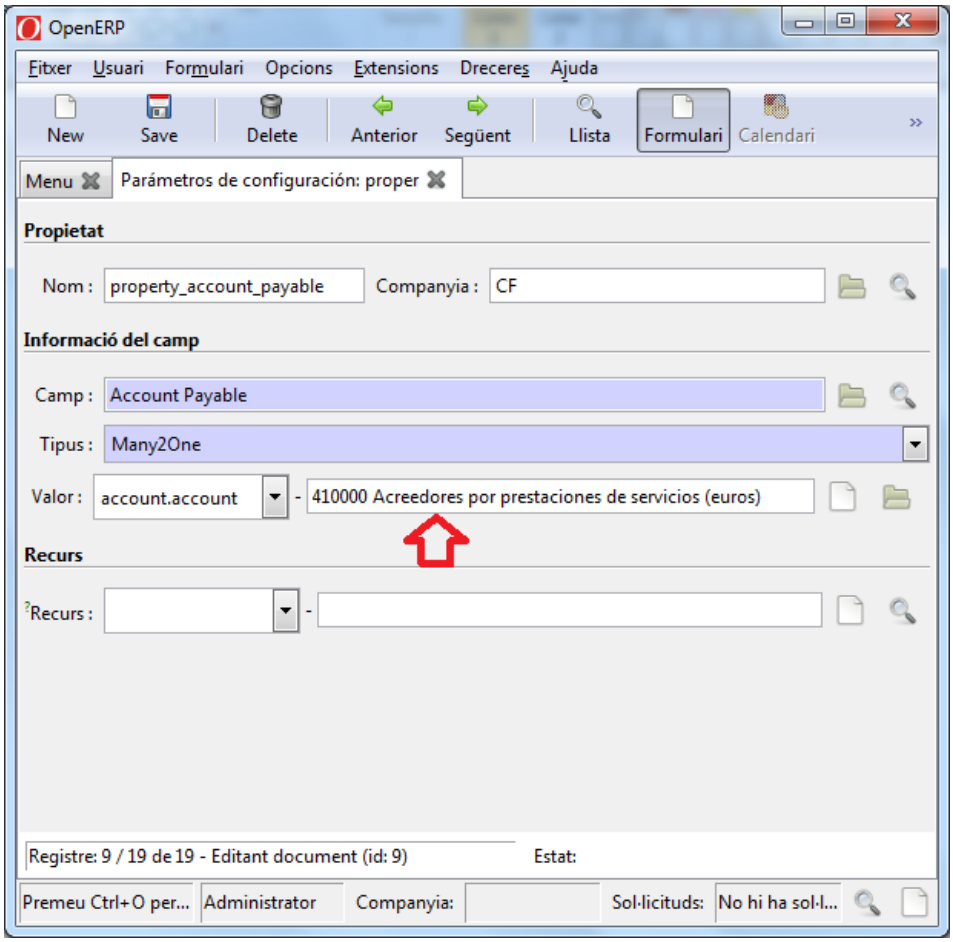


Amb els requeriments validats, naveguem fins l'apartat *Settings* | *Configuració* | *Paràmetres* | *Paràmetres de configuració*. Allà observem una llista de paràmetres (són els camps `property` definits en els mòduls actualment instal·lats). Si els ordeneu per nom, observareu fàcilment, tal com mostra la figura 1.16, que els

paràmetres `property_account_receivable` i `property_account_payable` es repeteixen quatre vegades, una per cadascuna de les companyies existents.

Si editem qualsevol dels vuit paràmetres observarem (figura 1.17) que el formulari conté el camp *Valor* amb un contingut que podem modificar. Podem observar que el contingut pel camp *Valor* és idèntic en les quatre companyies per a un mateix paràmetre, ja que és el valor que el procés de configuració de comptabilitat assigna (compte comptable 410000 per a proveïdors i compte comptable 430000 per a clients), però podem modificar-lo per a la companyia que interessi.

Figura 1.17. Assignació de valor a un camp property



Amb aquesta possibilitat de configuració personalitzada per a cada companyia, OpenERP proposarà, en els camps *Compte a cobrar* i *Compte a pagar* de l'apartat *Comptabilitat* de la fitxa dels tercers, el valor corresponent a la companyia que tingui assignat l'usuari que està gestionant el tercer.

A més, per cada tercer al qual, per un d'aquests camps `property`, s'assigni un valor diferent del que li pertoca segons la companyia, OpenERP afegeix un nou registre en el formulari de la figura 1.16, tot indicant a la columna *Recurs* el nom de l'objecte que té un valor diferent. Els registres que tenen buida la columna *Recurs* són els que contenen el valor per defecte a assignar al camp `property`.

Una vegada ja coneixem el funcionament dels camps `property` a nivell d'usuari, ens pertoca saber com definir-los en una classe d'OpenERP i ho veurem seguint l'exemple dels camps *Compte a Cobrar* i *Compte a Pagar* de la fitxa dels tercers.

Sembla clar que el lloc on hem de cercar aquests camps té a veure amb l'objecte (classe) `res.partner` d'OpenERP. Si fem una ullada a aquesta classe, definida dins el mòdul `base` d'OpenERP (fitxer `res_partner.py`) no hi trobarem cap dels dos camps `property` indicats. Els camps que cerquem es troben dins l'objecte `res.partner` definit dins el mòdul `account` (fitxer `partner.py` del mòdul `account`) que és una classe derivada de la classe `res.partner` principal (l'existent en el mòdul `base`). Vegem la part corresponent als dos camps `property` de la definició d'aquesta classe derivada:

```
class res_partner(osv.osv):
  _name = 'res.partner'
  _inherit = 'res.partner'
  _description = 'Partner'
  ...
  _columns = {
    ...
    'property_account_payable':
      fields.property('account.account', type='many2one',
                      relation='account.account', string="Account Payable",
                      view_load=True, domain="[('type', '=', 'payable')]",
                      help="This account will be used instead of the default
                           one as the payable account for the current partner",
                            required=True),
    'property_account_receivable':
      fields.property('account.account', type='many2one',
                      relation='account.account', string="Account Receivable",
                      view_load=True, domain="[('type', '=', 'receivable')]",
                      help="This account will be used instead of the default
                           one as the receivable account for the current
                           partner", required=True),
    ...
  }
```

Valors per defecte

La classe Python per definir els objectes d'OpenERP incorpora l'atribut opcional `_defaults` que permet la definició dels valors per defecte per un o diversos tipus de dades simples de l'objecte d'OpenERP.

Els valors per defecte dels camps d'un objecte d'OpenERP es defineixen en un diccionari de la forma:

```
_defaults = {
  'nom_del_camp1': funció1 o valor1,
  'nom_del_camp2': funció2 o valor2,
  ...
  }
```

Observem que els valors per defecte (sempre corresponents a tipus de dades simples) poden ser valors estàtics o valors dinàmics resultants de la crida de funcions que han d'estar declarades amb anterioritat a la definició del diccionari `_defaults`.

Les funcions per ser invocades en un valor per defecte dinàmic, han de contenir obligatòriament 4 paràmetres:

- `obj`: objecte `osv` corresponent al recurs que s'està creant.

- `cr`: cursor de base de dades.
- `uid`: ID de l'usuari que està donant d'alta el recurs.
- `context`: el context actual (facilitat pel client).

Els mòduls d'OpenERP estan plens d'exemples d'utilització de valors per defecte. Així, per exemple, observem els valors per defecte de l'objecte `res.users` (definit en el fitxer `res_users.py` dins el mòdul `base`) ideat per gestionar els usuaris de l'empresa gestionada per OpenERP:

```
_defaults = {
  'password' : '', 'context_lang': 'en_US',
  'active' : True, 'menu_id': _get_menu,
  'company_id': _get_company, 'company_ids': _get_companies,
  'groups_id': _get_group, 'menu_tips': False
}
```

En aquesta definició observem la coexistència de valors estàtics (`password`, `llenguatge`, `active` i `menu_tips`) i valors dinàmics (`menu_id`, `company_id`, `company_ids` i `groups_id`). Si observeu la definició completa de la classe `res.users` podreu comprovar com abans de la definició del diccionari `_defaults` es troba la definició de diverses funcions, entre les quals hi ha les quatre funcions invocades des de `_defaults` (`_get_menu`, `_get_company`, `_get_companies` i `_get_group`). Veureu, també, que les quatre funcions tenen els paràmetres `self`, `cr`, `uid` i `context`.

En algunes ocasions, la definició d'un valor per defecte estàtic s'efectua amb la crida a una funció `lambda` de Python sense paràmetres. La definició dels valors per defecte estàtics a l'anterior exemple s'hagués pogut efectuar com:

```
_defaults = {
  'password': lambda *a: '', 'context_lang': lambda *a: 'en_US',
  'active': lambda *a: True, 'menu_id': _get_menu,
  'company_id': _get_company, 'company_ids': _get_companies,
  'groups_id': _get_group, 'menu_tips': lambda *a: False
}
```

Per últim, per no veure's obligats a crear funcions que només siguin invocades en un valor `_defaults`, es pot utilitzar les funcions `lambda` de Python en el mateix moment d'efectuar la crida, estalviant-nos la definició de la funció amb nom. En el següent codi corresponent al diccionari `_defaults` de la classe `document.directory` (fitxer document_directory.py del mòdul `document`) observem la definició de dues funcions `lambda`:

```
_defaults = {
  'company_id': lambda s,cr,uid,c:
    s.pool.get('res.company')._company_default_get(cr, uid, 'document.directory
        ', context=c),
  'user_id': lambda self,cr,uid,ctx: uid,
  'domain': '[]',
  'type': 'directory',
  'ressource_id': 0,
  'storage_id': _get_def_storage,
  'resource_find_all': True,
}
```

Exemple d'utilització de l'atribut _defaults

L'objecte (classe) `school.professor` del mòdul `school` que estem millorant, incorpora l'atribut `contract` que consisteix en una selecció entre dos possibles valors. En el procés de creació d'un nou professor podem observar que el camp `contract` apareix sense valor per defecte. Volem definir que el valor per defecte d'aquest camp sigui `trainee`.

Així mateix, observem que l'objecte (classe) `school.professor` no incorpora el camp especial `active` que permet ocultar els recursos (objectes) de la classe quan ja no interessi, sense eliminar-los. Aprofitarem aquest exemple per afegir aquest camp i posar-li el valor 1 (cert) com a valor per defecte.

El codi de la classe `school.professor` modificada és:

```
class school_professor(osv.osv):
  """Professors"""
  _name = 'school.professor'
  _columns = {
    'name': fields.char('Professor Name', size=64,required=True),
    'contract': fields.selection([('trainee','Trainee'),
                                  ('named','Named')],'Contract'),
    'partner_id': fields.many2one('res.partner',
                                  'Associated Partner'),
    'address_id': fields.many2one('res.partner.address',
                                  'Private Address'),
    'phone': fields.related('address_id','phone', type='char',
                            string='Phone', store=False,
                            readonly=True),
    'hours_available': fields.integer('Hours Per Week'),
    'course_ids': fields.one2many('school.course', 'prof_id',
                                  'Courses'),
    'active': fields.boolean('Active'),
  }
  _defaults = {
    'contract': lambda *a: 'trainee',
    'active': lambda *a: 1,
  }
school_professor()
```

Observem que en la definició dels valors per defecte, hem utilitzat una funció `lambda` de Python, però no hagués estat necessari.

Una vegada actualitzat el mòdul `school` amb les noves millores, si obrim el formulari de manteniment de professors observarem que en donar d'alta un professor, en el camp `Contract` apareix per defecte el valor `Trainee`. En canvi, no apareix el nou camp `Active`, a causa que no hem modificat encara el formulari XML, però sí que podem veure la seva existència a la taula `school_professor` de la base de dades, amb valor `True` per tots els professors que ja existien a la taula i també pels nous professors.

Podeu trobar la versió millorada del mòdul `school` a l'arxiu school_03.zip dins l'apartat "Mòduls OpenERP" dels annexos del web.

Restriccions

Els objectes (classes) d'OpenERP poden incorporar, de forma opcional, restriccions d'integritat, addicionals a les que es puguin establir sobre la pròpia base de dades. OpenERP valida aquestes restriccions en les modificacions de dades i, en cas de violació, OpenERP mostra una pantalla d'error.

Les restriccions d'un objecte d'OpenERP es declaren a la classe Python que defineix l'objecte amb un atribut `_constraints` consistent en una llista de tuples, on cada tuple correspon a una restricció:

```
_constraints = [
  (nomMètode,'missatgeError','llistaNomsCamps')
  (nomMètode,'missatgeError','llistaNomsCamps'),
  ...
]
```

Els tuples corresponents a una restricció venen definits per tres camps:

- `nomMètode`: és el nom del mètode de l'objecte utilitzat per validar la restricció. El seu prototipus ha de tenir la forma `def _nomMètode (self, cr, uid, ids)` i ha de retornar un valor booleà.
- `missatgeError`: és el missatge d'error que es mostrarà a l'usuari si la comprovació falla.
- `llistaNomsCamps`: és la llista dels noms dels camps que s'afegeixen al missatge d'error amb l'objectiu d'ajudar a l'usuari a entendre el motiu pel qual ha fallat la validació de la restricció.

Els mòduls d'OpenERP incorporen validacions `_constraints` en nombrosos llocs. Un cas el trobem en la validació dels comptes bancaris segons la normativa IBAN.

Codis IBAN i BIC

El codi IBAN (*International Bank Account Number*) s'utilitza per a identificar a nivell internacional un compte bancari. Sempre comença per dos caràcters que identifiquen el país i va seguit d'una seqüència de dígits. En el cas dels comptes bancaris espanyols, comença per ES i va seguit de 22 dígits, dels quals, els 2 primers són dígits de control i els altres 20 corresponen al CCC (Codi Compte Client) utilitzat des de molt de temps a Espanya.

El codi BIC *(Bank Identifier Code*) serveix per identificar una entitat bancària.

A la xarxa trobareu gran quantitat de documentació relativa als codis IBAN i BIC.

Fixem-nos en el següent fragment de codi de la classe Python `res_partner_bank` definida en el mòdul `base_iban` (fitxer base_iban.py):

```
1 _constraints = [
2   (check_iban, _construct_constraint_msg, ["iban"]),
3   (_check_bank, '\nPlease define BIC/Swift code on bank for bank type IBAN
        Account to make valid payments', ['bic'])
4   ]
```

L'anterior atribut `_constraints` inclou dues restriccions:

- `check_iban`: aquest és el nom del mètode declarat a la mateixa classe, que s'encarrega de validar el codi IBAN introduït per l'usuari. En cas que sigui erroni, la restricció informa que l'error es produeix en el camp `iban` i afegeix el missatge que retorna la crida al mètode `_construct_constraint_msg` (mètode que construeix el missatge ja que és força complex perquè difereix en cada país).
- `_check_bank`: aquest és el nom del mètode declarat a la mateixa classe que s'encarrega de validar el codi BIC introduït per l'usuari (obligatori en cas que l'usuari hagi indicat que està introduint un codi IBAN). En cas que sigui erroni, la restricció informa que l'error es produeix en el camp `bic` i mostra el missatge d'error que indica la restricció.

Per comprovar el funcionament d'aquestes restriccions, executeu els següents passos:

1. Obriu el manteniment de clients i situeu-vos en un client qualsevol.

2. Editeu-lo i aneu a la pestanya *Comptabilitat* (cal tenir instal·lat el mòdul de comptabilitat). Allà hi veureu una zona de línies anomenada *Detalls del banc* destinada a incloure els comptes bancaris del client.

3. Procediu a donar d'alta un compte bancari amb els següents valors:

   Tipus de compte de banc: *Compte IBAN* (del contrari no fa cap verificació)

   Número de compte: *ES0012341234161234567890*

   Propietari compte bancari: seleccioneu el client al qui esteu assignant el compte

   Codi d'identificador bancari: introduïu el text *Qualsevol*

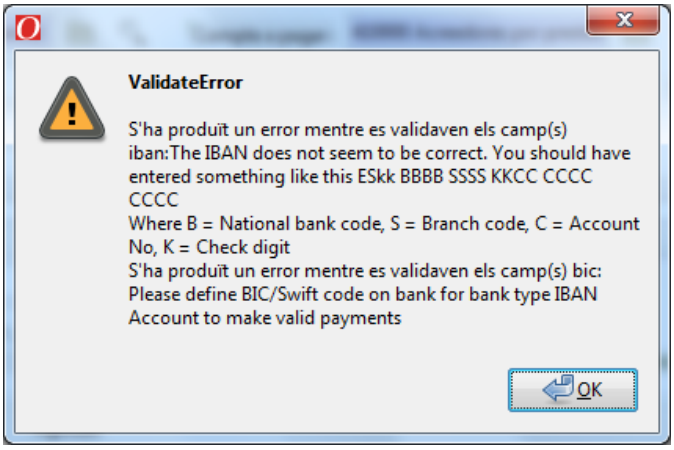
4. Procediu a *Desar i Tancar*, fet que us portarà de nou a la fitxa del client, on veureu el compte bancari introduït i procediu a intentar enregistrar el client. En aquest moment, OpenERP comprova les restriccions introduïdes i, en aquest cas, com que els codis IBAN i BIC són erronis, apareix la pantalla d'error de la figura 1.18. Hi observem dos errors: el primer que ens informa de l'error en validar el camp `iban` i ens explica com ha de ser el contingut d'aquest camp. El segon ens informa que s'ha produït un error en validar el camp `bic`.

5. Procediu a modificar el valor introduït en el camp `iban` amb el valor *ES7712341234161234567890,* que té un format IBAN correcte, tot i que gairebé segur que no existeix realment un CCC de codi *12341234161234567890*. Torneu a intentar enregistrar el client. Observareu que ja només apareix l'error relatiu al camp `bic`.

6. Per solucionar l'error del camp `bic` cal indicar el valor `bic` d'una de les entitats bancàries donades d'alta a l'empresa activa. Podeu donar d'alta les diverses entitats bancàries a través de l'opció *Vendes* | *Configuració* | *Llibreta d'adreces* | *Bancs* o, si ho preferiu, instal·leu el mòdul `l10n_es_partner` (*Adaptación de partner para Estado Español*) que, entre altres coses, facilita un assistent (*Vendes* | *Configuració* | *Llibreta d'adreces* | *Import Bank Data Wizard*) que afegeix dades de 191 bancs i caixes espanyoles a partir del registre oficial del Banc d'Espanya. Una vegada indiqueu un codi BIC existent a l'empresa OpenERP, podreu finalitzar l'enregistrament correcte del client.

Als annexos del web trobareu l'apartat "Recursos de programari" que inclou el fitxer `l10n_es_partner_20130328.zip` corresponent al mòdul `l10_es_partner` que potser voleu instal·lar.

Figura 1.18. Pantalla d'error d'OpenERP quan els codis IBAN i BIC són erronis



2. OpenERP: la vista i el controlador

OpenERP és un programari de gestió empresarial desenvolupat sobre el *framework* OpenObject de tipus RAD (*Rapid Application Development*) que facilita una arquitectura MVC (model-vista-controlador) per als desenvolupaments.

Una vegada es domina el disseny del model de dades d'una aplicació desenvolupada sobre OpenObject, com és el cas d'OpenERP, cal entrar en el coneixement del disseny de la vista i del controlador.

En les aplicacions desenvolupades sobre el *framework* OpenObject, el concepte *vista* engloba les pantalles que permeten exposar la informació a l'usuari (anomenades vistes o *views*) per a visualitzar-la i/o editar-la, i els menús, que faciliten un accés organitzat a les vistes. En conseqüència, ens cal aprendre a dissenyar les vistes (*views*) i els menús.

En el *framework* OpenObject el disseny del controlador s'efectua amb el llenguatge Python, tot ampliant les classes Python que defineixen el model de dades, amb la incorporació de mètodes.

Els menús i vistes dissenyats en arxius XML, en instal·lar-se en una empresa d'OpenERP, obtenen un identificador numèric dins l'empresa i que OpenERP utilitza per a la gestió de l'aplicació. Aquest identificador acostuma a ser diferent a cada empresa, ja que no totes les empreses tenen els mateixos mòduls instal·lats ni han estat instal·lats en el mateix ordre. A causa que és molt possible que en la fase de disseny el dissenyador hagi de fer referència a menús i vistes es fa necessari un identificador XML per a cada vista/menú que el dissenyador pugui utilitzar. Per això, en la definició XML de menús i vistes s'inclou un identificador (text) que ha de ser únic dins el mòdul i al qual es pot fer referència des del propi mòdul a través del seu nom o des de qualsevol altre mòdul `m` a través de la sintaxi `m.identificador`.

2.1 Menús

Les aplicacions desenvolupades amb el *framework* OpenObject contenen el model, la vista i el controlador seguint el patró de disseny MVC. Dins la vista s'inclouen les diverses pantalles que permeten gestionar la informació i els menús que permeten un accés organitzat a les vistes.

Els menús d'un mòdul OpenObject es dissenyen en arxius XML i han d'estar referenciats des de l'apartat `update_xml` del fitxer descriptor del mòdul __openerp__.py. Els arxius XML acostumen a incorporar les pantalles (*views*) i els menús que permeten accedir a les pantalles. Els dos tipus d'elements, però, podrien residir en fitxers XML específics (un per menús i un per *views*).

Com a exemple, podem analitzar el mòdul `school` generat per l'eina Dia, en el qual observem la presència del fitxer anomenat school_view.xml:

```
"update_xml" : ['school_view.xml'],
```

Els menús d'OpenObject i les seves opcions han de tenir una declaració similar a:

```
<menuitem id="menuitem_id"
  name="títol_del_menú"
  action="action_id"
  icon="nom_de_icona"
  web_icon="nom_icona_web"
  web_icon_hover="nom_icona_web_flotant"
  parent="menuitem_id"
  groups="grups_usuaris"
  sequence="<integer>"
/>
```

en la qual els valors específics de cada menú són:

- Atribut `id`: conté l'identificador XML del menú, únic dins el mòdul.
- Atribut `name`: conté el títol del menú. Aquest camp és opcional i, si no s'indica, el menú agafarà el nom de l'acció que executi el menú.
- Atribut `action`: especifica l'identificador de l'acció que executa el menú i que ha d'existir en el mòdul. Quan no s'especifica l'acció s'entén que es tracta de l'arrel d'un menú que conté altres menús i/o opcions. El disseny de la corresponent acció depèn del tipus d'execució associada (obrir una finestra, executar un informe, posar en marxa un assistent...).
- Atribut `icon`: especifica la icona que acompanya al menú en el client GTK. La icona per defecte és una carpeta. La llista d'icones possibles es pot consultar en el desplegable *Icona* del manteniment de menús accessible a través de *Settings* | *Personalització* | *Intefície d'usuari* | *Elements menú*.
- Atribut `web_icon`: especifica la icona que es mostra en el client web, a la pàgina inicial (*Home*) .
- Atribut `web_icon_hover`: especifica la icona que es mostra en el client web, a la pàgina inicial, en passar el ratolí pel damunt. Acostuma a ser la versió acolorida de la icona especificada a l'atribut `web_icon`.
- Atribut `parent`: especifica l'identificador del menú pare del qual depèn. Si no es defineix, indica que es tracta d'un menú arrel (menú principal).
- Atribut `groups`: especifica quin grup d'usuaris pot veure el menú. Si es desitja introduir més d'un grup, cal separar-lo per comes (per exemple, `groups="admin,user"`).
- Atribut `sequence`: és un enter que s'utilitza per ordenar el menú dins l'arbre de menús, de manera que a major valor, més avall apareix el menú. Aquest valor no és obligatori i per defecte pren el valor 10. Els menús que tinguin el mateix valor s'ordenen per moment temporal de creació.

Les icones dels atributs `web_icon` o `web_icon_hover` han de residir a la carpeta del mòdul i el seu nom ha d'anar acompanyat del camí que indiqui la subcarpeta (normalment `images` o `icons`) en la qual resideixen (o sense camí si no es troben a cap subcarpeta).

La versió del mòdul `school` de l'arxiu school_04.zip, que podeu trobar a l'apartat "Mòduls OpenERP" dels annexos del web, incorpora dues icones student.png i student-hover.png pels atributs `web_icon` i `web_icon_hover`.

Accions en OpenObject

Les accions defineixen el comportament del sistema en resposta als esdeveniments generats per l'usuari, ja sigui en seleccionar una opció de menú, en prémer un botó, en fer clic damunt un registre...

Hi ha diferents tipus d'accions:

- `Window`: per obrir una finestra.
- `Report`: per imprimir un informe.
- `Wizard`: per iniciar un assistent vinculat a un treball o procés.
- `Execute`: per executar un mètode en el servidor.
- `Group`: per reunir un conjunt d'accions en un grup.

En el Technical Memento d'OpenERP disponible a doc.openerp.com/memento hi trobareu el recordatori sobre el disseny de menús.

Exemple de menús en el mòdul school

Els menús originals del menú school generats per l'eina Dia per a la versió 5.0 d'OpenERP no són vàlids per a la versió 6.1 d'OpenERP, de manera que ens veiem obligats a refer-los segons les normes anteriors. Així, una possibilitat és la següent:

```
<menuitem name="Courses" id="menu_school"/>
<menuitem name="Calendar of Courses" id="menu_school_event"
  action="action_school_course_event" parent="menu_school"/>
<menuitem name="Students" id="menu_school_student"
  action="action_school_student" parent="menu_school"/>
<menuitem name="Configuration" id="menu_school_configuration"
  parent="menu_school"/>
<menuitem name="Courses" id="menu_school_course"
  action="action_school_course"
  parent="menu_school_configuration"/>
<menuitem name="Professors" id="menu_school_professor"
  action="action_school_professor"
  parent="menu_school_configuration"/>
```

En els elements `menuitem` anteriors detectem dues definicions de menús i quatre definicions d'opcions que executen accions. Cap dels dos menús incorpora l'atribut `icon` i, en conseqüència, en el client GTK la icona que els acompanya és una carpeta. El menú principal `menu_school` tampoc incorpora els atributs `web_icon` i/o `web_icon_hover` i, per tant, el mòdul `Courses` no va acompanyat de cap icona a la pàgina inicial del client web.

2.2 Vistes

Les vistes d'OpenERP són les pantalles que faciliten l'accés de l'usuari a la informació, tant per consultar-la com per modificar-la (altes, baixes i modificacions).

OpenObject facilita diversos tipus d'interfícies per facilitar l'accés de l'usuari a la informació (formularis, llistes, diagrames, gràfics, calendaris, arbres i targetes *kanban*) i totes elles són dinàmiques.

Les vistes d'OpenERP són dinàmiques i es construeixen en temps d'execució a partir de descripcions XML accessibles des del client, fet que en possibilita, fins i tot, la modificació en temps d'execució.

En cas que una vista es modifiqui en temps d'execució (des del client web activant el mode de desenvolupament, o des del client GTK, pel menú *Settings | Personalització | Interfície d'usuari | Vistes*), simplement cal tancar-la i reobrir-la per observar els canvis.

La descripció XML per les vistes associades a un objecte (classe) d'OpenERP, resideix en fitxers XML dins el mòdul corresponent a l'objecte. En cas de modificació en temps d'execució, el fitxer XML no es modifica i això cal tenir-ho molt present, ja que qualsevol procés d'actualització del mòdul provocarà la pèrdua dels canvis efectuats.

Per evitar la pèrdua de les modificacions efectuades a les vistes corresponents a mòduls susceptibles de ser actualitzats periòdicament (mòduls oficials i extres de la comunitat), el camí és crear un nou mòdul i definir-hi la nova vista com a herència de la vista a modificar, tot introduint els canvis en la vista heretada.

L'herència de vistes es troba a l'apartat "OpenERP: desenvolupament avançat".

La descripció de les vistes ha de residir en un fitxer XML que ha d'estar referenciat des de l'apartat `update_xml` del fitxer descriptor del mòdul __openerp__.py. Així, en el mòdul `school` hi observem la presència del fitxer anomenat school_view.xml i en el fitxer __openerp__.py hi trobem la línia:

```
"update_xml" : ['school_view.xml'],
```

Podeu trobar la versió vigent del mòdul `school` a l'arxiu school_04.zip dins l'apartat "Mòduls OpenERP" dels annexos del web.

El nom del fitxer XML en el qual resideixen les vistes pot ser qualsevol, però és altament aconsellable utilitzar alguna combinació en la qual aparegui el nom del mòdul i quelcom que indiqui el seu contingut (com per exemple el mot `view`).

La declaració d'un vista ha de ser similar a:

```
<record model="ir.ui.view" id="view_id">
  <field name="name">view.name</field>
  <field name="model">object_name</field>
  <field name="type">xxx</field>
  <!-Valors possibles: tree,form,calendar,search,graph,gantt,kanban-->
  <field name="priority" eval="16"/>
  <field name="arch" type="xml">
    <!-- view content: <form>, <tree>, <graph>, ... -->
  </field>
</record>
```

en el qual els valors específics de cada vista són:

- Element `record` que conté l'atribut `model` amb valor `ir.ui.view` obligatori per a les vistes, i l'atribut `id` amb l'identificador XML de la vista dins el mòdul.
- Element `field name="name"` amb el nom de la vista.

- Element `field name="model"` amb el nom de l'objecte (classe) d'OpenERP sobre el qual es defineix la vista.
- Element `field name="type"` amb el tipus de la vista, en el qual els valors possibles actuals són: `tree`, `form`, `calendar`, `search`, `graph`, `gantt` i `kanban`.
- Element `field name="priority"` amb la prioritat de la vista. Com més petita, més prioritat. Per defecte: 16.
- Element `field name="arch" type="xml"` amb l'arquitectura o estructura de la vista que es defineix mitjançant diverses etiquetes XML. Aquesta estructura és diferent segons el tipus de vista (`tree`, `form`, `calendar`, `search`, `graph`, `gantt`, `kanban`).

Així doncs, cal endinsar-nos en l'estructura dels diversos tipus de vista i en els mecanismes (accions) que possibiliten a l'usuari la seva execució.

En el Technical Memento d'OpenERP disponible a doc.openerp.com/memento hi trobareu el recordatori sobre el disseny de vistes i accions.

2.2.1 Accions window

Una vista d'OpenObject entra en execució a partir d'un esdeveniment d'usuari (seleccionar una opció de menú, prémer un botó...) que està vinculat a una acció `window`, responsable de la posada en marxa de la vista.

L'element `field name="type"` de la declaració d'una vista obliga a definir la vista amb un dels tipus següents: `tree`, `form`, `calendar`, `search`, `graph`, `gantt`, `kanban`. Cal tenir en compte que sota el tipus `tree` s'aixopluguen dos tipus diferents: llista jerarquitzada (normalment anomenada arbre -`tree`-) i llista no jerarquitzada (normalment anomenada llista -`list`-).

El fet d'utilitzar el mot `tree` per fer referència a dos tipus de vista provoca maldecaps que OpenObject hagués pogut evitar.

De la mateixa manera que els menús i les vistes, les accions (no només les `window`) es declaren en arxius XML que han d'estar referenciats des del fitxer descriptor del mòdul. A causa que les vistes es posen en marxa a través d'accions i que hi ha menús que porten associades aquestes accions, és molt comú (però no obligatori) introduir en el fitxer XML les declaracions en el següent ordre:

- Vistes vinculades a un objecte (acostuma a haver-n'hi diverses).
- Accions que executen vistes (pot ser única i que accioni diverses vistes).
- Menús per facilitar a l'usuari l'execució d'accions.

Exemple de vistes, accions i menús relacionats

Fixem-nos en els següents fragments de vistes, accions i menús, en el mòdul `school` (versió *school_04*) corresponents a l'objecte `professor`:

```
1 <record model="ir.ui.view" id="view_school_professor_form">
2   <field name="name">school.professor.form</field>
3   <field name="model">school.professor</field>
4   <field name="type">form</field>
```

```
  <field name="arch" type="xml">
  ...

<record model="ir.ui.view" id="view_school_professor_tree">
  <field name="name">school.professor.tree</field>
  <field name="model">school.professor</field>
  <field name="type">tree</field>
  <field name="arch" type="xml">
  ...

<record model="ir.actions.act_window"
        id="action_school_professor">
  <field name="name">Professors</field>
  <field name="res_model">school.professor</field>
  <field name="view_type">form</field>
  <field name="view_mode">tree,form</field>
</record>

<menuitem name="Professors"
  id="menu_school_professor" action="action_school_professor"
  parent="menu_school_configuration"/>
```

Hi observem:

- Dues vistes: `view_school_professor_form` (de tipus `form`) i `view_school_professor_tree` (de tipus `tree`).
- Una acció: `action_school_professor`, que no està associada a cap de les dues vistes. Quina s'executa, doncs, en activar l'acció?
- Un menú: `menu_school_professor`, associat a l'acció anterior.

En executar, des d'OpenERP, l'opció de menú `Professors`, observem com apareixen els professors en format llista (`tree`) i que la vista formulari (`form`) també es pot seleccionar.

Els elements `field name="view_type"` i `field name="view_mode"` defineixen, quan no s'explicita la vista a executar (com és el cas), el comportament d'OpenObject per escollir la vista.

L'exemple anterior serveix per il·lustrar que en OpenObject hi ha certs comportaments per defecte que s'activen quan el programador no els ha explicitat, com és el cas de l'elecció de la vista a executar quan el programador no la indica.

Abans de presentar com és la declaració d'una acció `window` és molt convenient entendre el comportament d'OpenObject en l'elecció i visualització de les possibles vistes sobre un objecte.

OpenObject categoritza totes les vistes (`tree`, `form`, `calendar`, `search`, `graph`, `gantt` i `kanban`) en dos grans grups:

- `tree` per les vistes jeràrquiques, que permeten la visualització de dos tipus de vistes:
  - `tree`: visualització arbre
  - `form`: visualització formulari
- `form` per les vistes no jeràrquiques, que permeten la visualització de diversos tipus de vistes:
  - `tree`: visualització llista

  - `form`: visualització formulari
  - `calendar`: visualització calendari
  - `graph`: visualització gràfic
  - `gantt`: visualització diagrama de gantt
  - `search`: visualització d'una zona de filtres
  - `kanban`: visualització dels recursos com a targetes agrupades sota un criteri, que poden ser editables i/o arrossegables

En la definició d'una acció `window` s'ha d'indicar si correspon a una vista jeràrquica (categoria `tree`) o una vista no jeràrquica (categoria `form`) i, una vegada definida la categoria, cal indicar –en l'ordre que interessi– els tipus de vista (adequats a la categoria) que permetrà visualitzar. Això s'aconsegueix amb dos elements que formen part de la definició XML d'una acció `window`:

- Element `field name="view_type"` amb una de les dues categories possibles: `form` o `tree`.
- Element `field name="view_mode"` amb una seqüència ordenada dels tipus de vista que facilita l'acció.

Els valors de `view_mode`, separats per comes, han de ser coherents amb el valor de `view_type` i, a més, el mòdul ha d'incorporar alguna vista per cadascun dels tipus indicats a la seqüència de `view_mode`. En cas que per algun tipus de vista dels indicats a `view_mode`, hi hagi diverses vistes definides en el mòdul, OpenObject escollirà la vista més prioritària (element `priority` de la definició de les vistes).

Exemple de "view_type" i "view_mode" en la definició d'accions window

En el mòdul `school` (versió *school_04*) observem el següent fragment d'acció `window`:

```
<record model="ir.actions.act_window"
        id="action_school_professor">
  <field name="name">Professors</field>
  <field name="res_model">school.professor</field>
  <field name="view_type">form</field>
  <field name="view_mode">tree,form</field>
</record>
```

En aquest cas, a causa que `view_type` té el valor `form`, l'element `view_mode` podria contenir, en principi, qualsevol seqüència ordenada dels valors `tree`, `form`, `calendar`, `search`, `graph`, `gantt` i `kanban`. Fixem-nos que la seqüència és `"tree,form"` i, en conseqüència, en executar l'acció (manteniment de professors), les dues vistes possibles són llista i formulari i el manteniment s'obre en vista llista. Si el contingut de `view_mode` hagués estat `"form, tree"` les vistes possibles haguessin estat les mateixes, però el manteniment s'obriria en mode formulari buit. Per tal que tot funcioni, el mòdul facilita per a l'objecte `school.professor`, com a mínim, una vista de tipus `form` i una vista de tipus `tree` no jeràrquica:

- `view_school_professor_form` per a la vista `form`
- `view_school_professor_tree` per a la vista `tree` no jeràrquica

En el mòdul `school` (versió *school_04*) observem el següent fragment d'acció `window`:

```
<record model="ir.actions.act_window"
        id="action_school_course">
  <field name="name">Courses</field>
  <field name="res_model">school.course</field>
  <field name="view_type">form</field>
  <field name="view_mode">tree,form,calendar</field>
</record>
```

Aquest és un altre cas en el qual `view_type` té el valor `form`, però, en canvi, l'element `view_mode` conté la seqüència `"tree,form,calendar"` i, en conseqüència, en executar l'acció (manteniment de cursos), tenim tres vistes possibles (llista, formulari i calendari) i el manteniment s'obre en vista llista. Ja que l'acció indica tres vistes possibles, el mòdul hauria de contenir, com a mínim, una vista de cada tipus, i en aquest cas, l'eina Dia, que ha generat l'element `view_mode` amb els tres tipus de vista, només facilita dues vistes:

- `view_school_course_form` per a la vista `form`
- `view_school_course_tree` per a la vista `tree` no jeràrquica

Dia ha incorporat el tipus de vista `calendar` a `view_mode` perquè l'objecte `school.course` té un camp `date` i, en conseqüència, aquest objecte és susceptible de tenir una vista `calendar`. Però Dia no ha definit cap vista `calendar`. En procedir a l'execució des d'un client (web o GTK), OpenObject intenta generar una vista calendar automàtica en el mateix instant, però no pot perquè l'objecte `school.course` no té atributs adequats i mostra l'error de la figura 2.1.

Figura 2.1. Pantalla d'OpenERP en intentar mostrar una vista Calendar errònia

Dia també ha generat l'acció `window action_school_course_event` en el mòdul `school` (versió *school_04*) amb valor `"tree,form,calendar"` per a l'element `view_mode` sense proporcionar cap vista de tipus `calendar`, però en aquest cas OpenObject pot proporcionar una vista `calendar` per defecte a partir dels dos camps `date` existents (`date` i `date_end`). El funcionament, però, no és el desitjable:

- En el client GTK, l'interval de dies d'un recurs `school.course.event` es visualitza sempre en color negre i no hi ha cap text que identifiqui el recurs.
- En el client web, l'interval de dies d'un recurs `school.course.event` només és visible en passar el ratolí per damunt els dies afectats.

Caldrà, doncs, dissenyar vistes `calendar` per aconseguir el funcionament desitjat o eliminar el valor `calendar` de l'element `view_mode`.

La declaració d'un acció `window` ha de ser similar a:

```
<record model="ir.actions.act_window" id="action_id">
```

```
  <field name="name">action.name</field>
  <field name="view_type">form|tree</field>
  <field name="view_mode">...</field>
    <!-- combinació ordenada dels tipus de vista
      possible, coherent amb el contingut de
      l'element view_type -->
  <field name="view_id" ref="nomVista"/>
  <field name="search_view_id" ref="nomVistaDeCerca"/>
  <field name="domain">
    ["list of 3-tuples (max 250 characters)"]
  </field>
  <field name="context">
    {"context dictionary (max 250 characters)"}
  </field>
  <field name="res_model">Open.object</field>
  <field name="target">new</field>
</record>
```

en la qual els valors específics per a cada acció són:

- Element `record` que conté l'atribut `model` amb valor `ir.actions.act_window` obligatori per a les accions `window` i l'atribut `id` amb l'identificador XML de l'acció dins el mòdul.
- Element `field name="name"` amb el nom de l'acció. És obligatori. Els menús sense `name` que invoquin l'acció, prenen el `name` de l'acció com a `name` propi.
- Elements `view_type` i `view_mode`, estudiats prèviament.
- Element `field name="view_id"` amb l'atribut `ref` que ha de contenir el nom de la vista per mostrar quan s'activa l'acció. En cas que aquest camp no estigui definit, OpenObject decideix en funció del contingut de l'element `view_mode`.
- Element `field name="search_view_id"` amb l'atribut `ref` que ha de contenir el nom de la vista de cerca (capçalera de les pantalles que mostren una llista de registres, per facilitar-ne el filtrat) per mostrar quan s'activa l'acció.
- Element `res_model` amb el nom de l'objecte sobre el qual opera l'acció.
- Element `domain` amb una llista Python de condicions utilitzada per refinar els resultats d'una selecció i, en conseqüència, mostrar menys recursos a la vista. Les condicions de la llista s'enllacen amb una clàusula AND i són tuples Python de tres valors (`'camp','condició','valor'`).
- Element `context` amb un diccionari contextual que s'utilitzarà a la vista que s'obri en executar l'acció. Els diccionaris contextuals es declaren com un diccionari Python. Aquests diccionaris contenen informació de context a utilitzar, com per exemple l'idioma, la moneda, la tarifa, el magatzem...
- Element `target` per indicar on s'ha d'obrir la finestra. Valors possibles: `new` (en una nova finestra), `current` (substituint la finestra actual) i `inline` (dins la finestra actual).

2.2.2 Vistes form

La vista formulari per a un objecte d'OpenObject consisteix en la correcta distribució en la pantalla dels camps de l'objecte amb l'objectiu de facilitar-ne la visualització i/o l'edició d'un recurs.

Les vistes formulari es distingeixen perquè en la seva declaració incorporen:

```
1 <field name="type">form</field>
```

i l'element arrel de l'XML que defineix l'arquitectura de la vista és `form`.

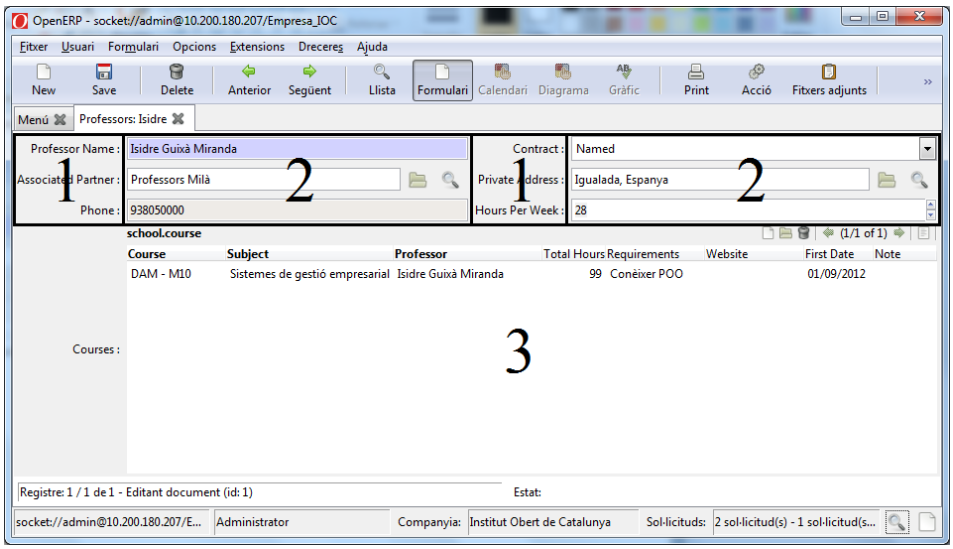
Els camps en una vista formulari d'OpenObject sempre estan distribuïts en la pantalla segons les següents normes:

- Per defecte cada camp va precedit d'una etiqueta que és el seu nom.
- Els camps se situen a la pantalla d'esquerra a dreta i de dalt a baix, segons l'ordre en què estan declarats en el fitxer XML que descriu la vista.
- Cada pantalla es troba dividida en 4 columnes, cadascuna de les quals pot contenir una etiqueta o un camp. Ja que cada camp és precedit (per defecte) per una etiqueta amb el seu nom hi haurà dos camps amb les seves respectives etiquetes a cada línia de la pantalla.

Com a exemple, considerem el formulari de manteniment de professors del mòdul `school` (versió *school_04*) de la figura 2.2, generat per l'eina Dia. Observem-hi que:

- Les zones 1 i 2 corresponen a les 4 columnes en les quals es troba dividida la pantalla.
- Les zones 1 contenen les etiquetes dels camps ubicats a les zones 2.

Figura 2.2. Exemple de distribució dels camps en un formulari d'OpenERP



OpenObject facilita opcions d'ubicació més avançades. Així, per exemple, la zona 3 de la figura 2.2 correspon a un camp `one2many` que ocupa les 4 darreres columnes de la pantalla (1 per l'etiqueta *Courses* i les altres 3 per al contingut del camp). A més, el camp `one2many` conté un `form` de tipus llista que visualitza 8 columnes. És molt possible que interessi que el contingut del camp `one2many` ocupi les 4 columnes de la pantalla i que l'etiqueta *Courses* passi a substituir el nom de la classe `school.course`. OpenObject ens facilita opcions avançades de visualització per aconseguir-ho.

OpenObject permet:

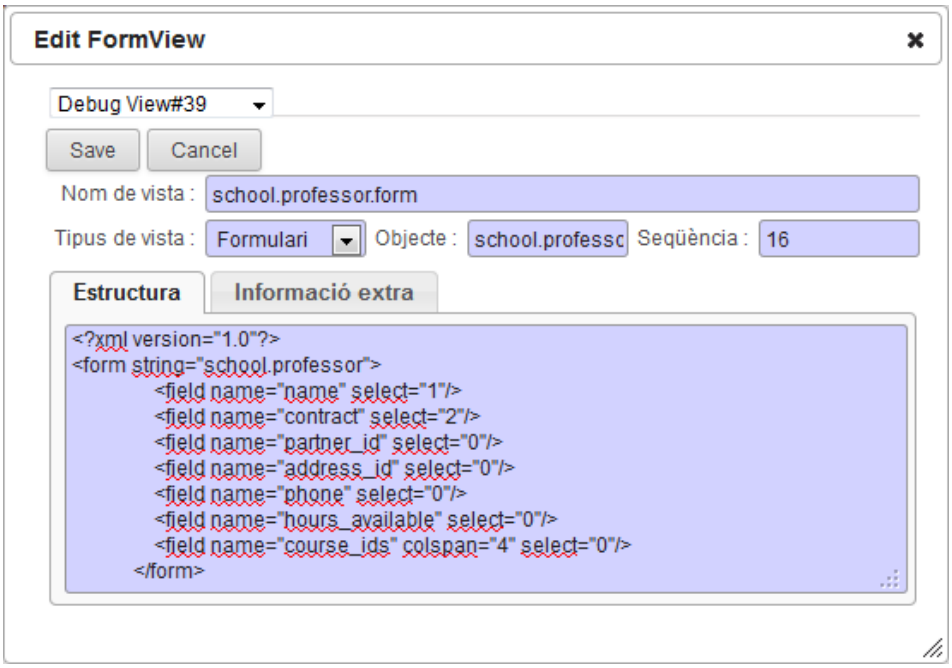
- Que un camp pugui ocupar més d'una columna, tot utilitzant l'atribut `colspan`.
- Agafar un grup de columnes i dividir-les en les columnes que es desitgi, tot utilitzant l'etiqueta `group` i els atributs `colspan` i `col`.
- Distribuir els camps d'un objecte en diverses pestanyes, tot utilitzant les etiquetes `notebook` i `page`.

El contingut del fitxer school_view.xml corresponent al formulari de la figura 2.2 (manteniment de professors) és:

```
<record model="ir.ui.view" id="view_school_professor_form">
  <field name="name">school.professor.form</field>
  <field name="model">school.professor</field>
  <field name="type">form</field>
  <field name="arch" type="xml">
    <form string="school.professor">
      <field name="name" select="1"/>
      <field name="contract" select="2"/>
      <field name="partner_id" select="0"/>
      <field name="address_id" select="0"/>
      <field name="phone" select="0"/>
      <field name="hours_available" select="0"/>
      <field name="course_ids" colspan="4" select="0"/>
    </form>
  </field>
</record>
```

En aquest cas, detectar dins el fitxer school_view.xml el registre corresponent a la vista no ha estat difícil, però en altres casos en els quals hi ha moltes vistes pot ser més dificultós. En canvi, des del client web, tenint obert el formulari i el mode de desenvolupament activat, podem demanar d'editar la vista, obtenint el formulari de la figura 2.3, en la qual observem el nom de la vista i, a més, podem modificar-la i enregistrar els canvis en el servidor OpenERP. Recordem que els canvis no queden enregistrats en el fitxer XML del mòdul.

Figura 2.3. Edició d'una vista d'OpenERP des del client web



Fixem-nos que el contingut XML de l'element `arch` no és altre que el conjunt de camps a visualitzar, de l'objecte (classe) d'OpenERP en el qual es basa la vista. Recordem la definició de l'objecte `school.professor` (versió *school_04*):

```
1  class school_professor(osv.osv):
2    _name = 'school.professor'
3    _columns = {
4      'name': fields.char(...),
5      'contract': fields.selection(...),
6      'partner_id': fields.many2one(...),
7      'address_id': fields.many2one(...),
8      'phone': fields.related(...),
9      'hours_available': fields.integer(...),
10     'course_ids': fields.one2many(...),
11     'active': fields.boolean('Active'),
12   }
13 ...
```

La definició de la classe Python `school_professor` incorpora el camp `active` que no apareix a la vista `school.professor.form`. Això és a causa que la vista `school.professor.form` que estem analitzant va ser generada per l'eina Dia i el camp `active` el vam incorporar posteriorment en el codi Python sense modificar la corresponent vista.

Suposem que volem situar el camp `active`, que correspon a una casella de verificació, a la 4a fila del formulari. Simplement l'haurem de situar en 7è lloc dins l'XML de l'element `arch`, entre els camps `hours_available` i `course_ids`. Si efectuem la modificació en el fitxer school_view.xml caldrà actualitzar el mòdul; en canvi, si efectuem la modificació des dels clients web o GTK només cal recarregar el formulari. Sigui quin sigui el camí, observarem el camp `active` amb etiqueta *Active* a la 4a línia de la pantalla.

En el formulari `school.professor.form` anterior també podem observar com el camp `one2many` de nom `course_ids` va acompanyat de l'atribut `colspan=4`

que indica que el camp (conjuntament amb la seva etiqueta *Courses*) ha d'ocupar les 4 columnes de la pantalla.

Dins l'estructura del formulari (contingut de l'element `field name="arch"`) hi pot haver diversos tipus d'elements i aquests elements poden tenir diversos atributs. La taula 2.1 mostra els atributs comuns als diversos tipus d'elements possibles i la taula 2.2 mostra els diversos tipus d'elements possibles acompanyats dels atributs específics de cada tipus d'element.

Taula 2.1. Relació d'atributs comuns pels elements de la definició d'una vista form

Atribut	Utilització
`string`	Etiqueta de l'element que substitueix l'etiqueta definida a la classe. També s'utilitza en els processos de recerca.
`nolabel`	Permet amagar (valor="1") l'etiqueta del camp.
`colspan`	Permet indicar el nombre de columnes que ha de tenir el camp.
`rowspan`	Permet indicar el nombre de files que ha de tenir el camp.
`col`	Permet indicar el nombre de columnes en què es divideix l'espai assignat al camp.
`invisible`	Permet ocultar (valor="1") el camp i la seva etiqueta.
`eval`	Cadena avaluada com a codi Python per obtenir el valor del camp.
`attrs`	Permet indicar sota quines condicions dinàmiques, basades en els valors d'altres camps del formulari, l'element ha de ser `readonly` i/o `invisible` i/o `required`. Ha de seguir el format: `{'atribut':[('nomCamp','operador','valor'), ('nomCamp','operador','valor')...],...}` on `atribut` ha de ser `readonly` o `invisible` o `required`.

Exemples d'utilització de l'atribut attrs en la ubicació dels camps en vistes

En els mòduls d'OpenERP es pot veure la utilització de l'atribut `attrs` en múltiples vistes. Per exemple, en la vista *Leave Request* del mòdul `hr_holidays` (fitxer hr_holidays_view.xml):

```
<field name="name"
  attrs="{'readonly':[('state','!=','draft'),
                      ('state','!=','confirm')]}"/>
```

El camp `name` serà de només lectura quan el camp `state` no tingui els valors `draft` ni `confirm`.

```
<field name="category_id"
  attrs="{'required':[('holiday_type','=','category')],
          'readonly':[('state','!=','draft')]}"/>
```

El camp `category_id` serà obligatori quan el camp `holiday_type` valgui `category` i de només lectura quan el camp `state` no valgui `draft`.

Taula 2.2. Relació dels elements possibles en la definició d'una vista form

Element	Utilització
`field`	Camp per visualitzar, d'entre les columnes definides a l'objecte (classe) OpenERP en la qual es basa la vista. Cada camp porta un giny (*widget*) associat segons el tipus de dada del camp. Així, si el camp és de tipus `booleà`, el giny és una casella de verificació; si el camp és de tipus `date` el giny és un camp formatat per a data acompanyat d'un calendari per poder seleccionar la data amb comoditat, com mostra la figura 2.4. La taula 2.3 mostra els atributs específics d'aquest element.
`button`	Giny (*widget*) en forma de botó per ser premut i que està associat a determinades accions. La taula 2.4 mostra els atributs específics d'aquest element.
`separator`	Línia de separació horitzontal per estructurar vistes, amb etiqueta opcional.
`newline`	Permet afegir espai en blanc per completar la línia actual de la vista i saltar a la següent.
`label`	Títol de text lliure en el formulari.
`group`	Permet organitzar els camps en grups amb etiquetes opcionals. Aporta un requadre al voltant del grup.
`notebook` `page`	Un element `notebook` és un contenidor de pestanyes, definides cadascuna per un element `page`. Els atributs específics d'aquest element són: * `name`: etiqueta per la pestanya * `position`: posició de la pestanya dins el `notebook`, amb valors possibles: `inside`, `top`, `bottom`, `left` i `right`.

Taula 2.3. Relació d'atributs específics per l'element field en la definició d'un form

Atribut	Significat
`select`	1 per mostrar el camp en cerques normals i 2 per mostrar el camp només en cerques avançades. A partir de la versió 6 d'OpenERP, les cerques avançades han deixat d'existir i, en canvi, els clients faciliten l'opció de filtres avançats en els quals es pot escollir qualsevol dels camps del model. Per tant, el valor 2 és obsolet a partir de la versió 6 d'OpenERP. La utilització d'aquest atribut queda obsoleta en el moment en què la versió 6 d'OpenERP facilita les vistes `search`.
`required`	Substitueix l'atribut `required` definit en el model. 1 per tal que el camp sigui obligatori.
`readonly`	Substitueix l'atribut `readonly` definit en el model. 1 per tal que el camp sigui de només lectura.
`default_focus`	Valor 1 per indicar que aquest camp ha de tenir el focus en obrir el formulari. Només hi pot haver un element amb aquest atribut a `1`.
`password`	`True` per ocultar els caràcters que s'introdueixen en el camp.
`context`	Codi Python per definir un diccionari contextual.
`domain`	Codi Python per definir una llista de tuples amb condicions per restringir valors, en camps relacionals.

Taula 2.3 (continuació)

Atribut	Significat
`on_change`	Mètode Python que s'executa en canviar el valor del camp. S'utilitza per calcular automàticament el valor d'altres camps quan el camp actual canvia.
`groups`	Llista Python d'identificadors de grups d'usuaris autoritzats a visualitzar el camp.
`widget`	Giny alternatiu al proporcionat de forma automàtica segons el tipus del camp. Valors possibles: * `date` (figura 2.4) * `float_time` (figura 2.5) * `datetime` (figura 2.6) * `selection` (figura 2.7) * `number` (figura 2.8) * `many2one_list` (figura 2.9) * `one2many_list` (figura 2.10) * `many2many` (figura 2.11) * `url` (figura 2.12) * `email` (figura 2.13) * `image` (figura 2.14) * `reference` (figura 2.15) * `text_wiki` * `text_html` * `progressbar` (figura 2.16)

Taula 2.4. Relació d'atributs específics per l'element button en la definició d'un form

Atribut	Significat
`type`	Tipus de botó. Valors possibles: * `workflow` (flux de treball, valor per defecte): envia el senyal (indicat a l'atribut `name`) al flux de treball del mòdul. * `object`: crida la funció o mètode indicada a l'atribut `name`. * `action`: executa l'acció indicada a l'atribut `name`.
`name`	Segons el valor de l'atribut `type`: * Senyal del flux de treball * Nom de la funció o mètode a executar * Nom de l'acció a executar
`special`	Únicament és operatiu en una finestra emergent (*pop-up*) i en cas d'utilitzar-lo en una finestra no emergent, no és operatiu. Només té un possible valor: `cancel`. És incompatible amb l'atribut `type`.
`confirm`	Text de missatge de confirmació per quan es prem el botó.
`states`	Llista d'estats, separats per comes, en els quals el botó es mostra. Si aquest atribut no apareix, el botó és sempre visible.
`icon`	Nom d'icona. Per defecte, el botó és textual. Es pot utilitzar qualsevol de les icones existents a la subcarpeta `web/static/src/img/icons` de la carpeta `addons` on es troben els mòduls o qualsevol altra icona indicant la seva ubicació (normalment en una subcarpeta `images` o `icons` dins el mòdul).
`default_focus`	Valor a 1 per indicar que aquest botó ha de tenir el focus en obrir el formulari. Només hi pot haver un element amb aquest atribut a 1.

Figura 2.4. Giny date

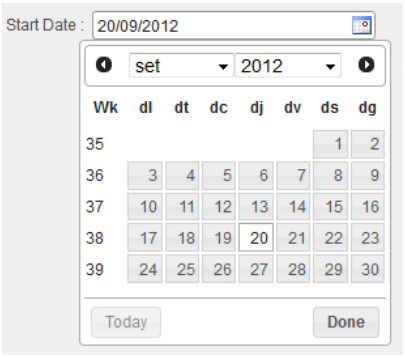


Figura 2.5. Giny float_time

Figura 2.6. Giny datetime

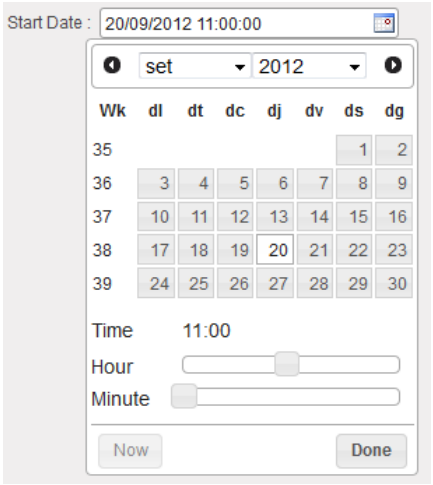


Figura 2.7. Giny selection

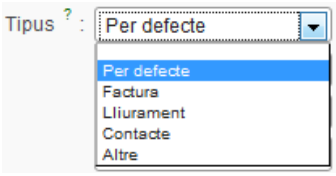


Figura 2.8. Giny number en el client GTK

Figura 2.9. Giny many2one_list

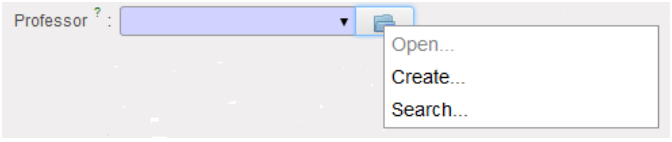


Figura 2.10. Giny one2many_list

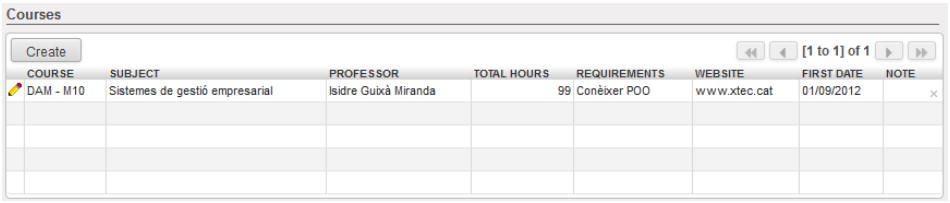


Figura 2.11. Giny many2many

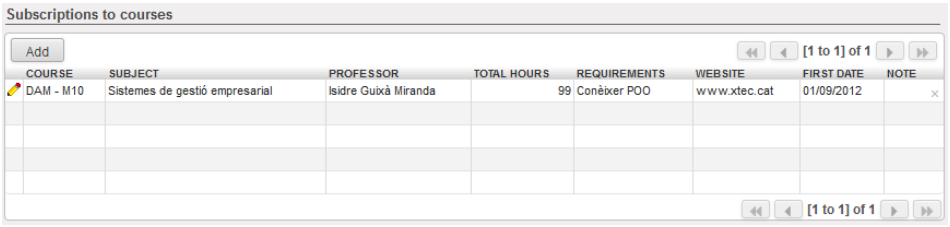


Figura 2.12. Giny url

Figura 2.13. Giny email

Figura 2.14. Giny image

Figura 2.15. Giny reference

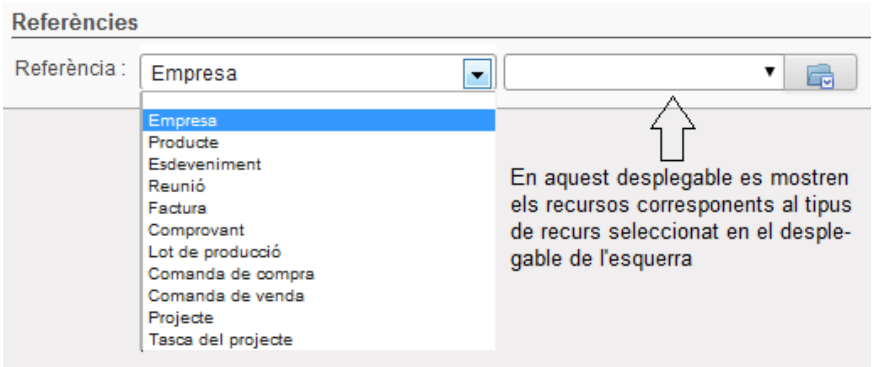


Figura 2.16. Giny progressbar

Millora de vistes form en el mòdul school

Ens proposem millorar les vistes form del mòdul `school` vigent (versió *school_04*).

En totes les vistes canviem el contingut de l'atribut `string` de l'element arrel de l'arquitectura de la vista, generat de forma automàtica per l'eina Dia, amb un nom entenedor –en llengua anglesa–, ja que allà on OpenERP visualitzi la vista i no s'expliciti un títol, OpenERP utilitzarà el contingut de l'atribut `string`.

En tots els camps de les vistes `form` eliminem –per obsolets– els atributs select amb valor 0 o 2. Hi mantenim els atributs `select="1"` mentre no incorporem cap vista `search`.

En el formulari de manteniment dels professors:

- Fem visible el camp `active` en la tercera fila del formulari, a la dreta del camp `hours_avalilable`, de manera que els dos camps (`active` i `hours_available`) ocupin les columnes 3 i 4 del formulari.

- Fem desaparèixer l'etiqueta *Courses* del sotsformulari i afegim un separador, amb etiqueta *Courses*, just abans del sotsformulari.

En el formulari de manteniment dels estudiants, fem desaparèixer l'etiqueta *Subscriptions to courses* del sotsformulari i afegim un separador, amb etiqueta *Subscriptions to courses*, just abans del sotsformulari.

En el formulari de manteniment dels cursos:

- Millorem el camp `website` de manera que es pugui navegar fins l'adreça que conté.

Podeu trobar la versió millorada del mòdul `school` a l'arxiu school_05.zip dins l'apartat "Mòduls OpenERP" dels annexos del web.

- El sotsformulari amb etiqueta *Students* el situem en una pestanya *Students* i afegim una altra pestanya *Calendar* amb accés a les diverses edicions del curs (cal modificar el model de dades de l'objecte `school.course` afegint el camp `event_ids`).

2.2.3 Vistes tree (arbre/llista)

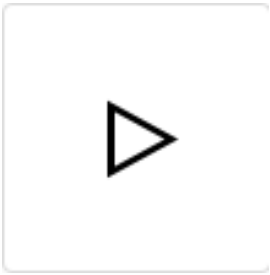
Les vistes arbre/llista per a un objecte d'OpenObject consisteixen en la distribució de la pantalla en línies amb l'objectiu de facilitar la visualització i/o l'edició d'un conjunt de recursos de l'objecte. La majoria de les vistes arbre/llista són de només visualització, però OpenObject permet fer-les editables.

La vista arbre mostra els recursos de l'objecte seguint una estructura jeràrquica, mentre que la vista llista mostra els recursos en seqüència, sense cap jerarquia.

En OpenERP tenim una gran quantitat de vista llista com, per exemple, en accedir a l'accés directe *Clients*, en el qual se'ns presenta els clients en vista llista. OpenERP també inclou exemples de vista arbre; un exemple el trobem en accedir a l'opció de menú *Magatzem* | *Productes* | *Productes per categoria*, com ens mostra la figura 2.17.

Figura 2.17. Exemple de vista en arbre, amb desplegable per escollir el recurs de primer nivell de la jerarquia

La vista arbre de la figura 2.17 presenta un desplegable que ens permet escollir entre les categories de primer nivell (*Tots els productes, Productes negociables, Altres productes*). La figura 2.17 mostra la jerarquia de categories per sota de *Tots els productes*. En executar l'opció de menú, per a *Tots els productes* únicament se'ns presenten les dues categories filles: *Privat* i *Vendible*. La categoria *Vendible*, com que té categories filles, va precedida d'un símbol en forma de triangle amb vèrtex cap a la dreta que, en prémer-lo, desplega les categories filles: *Complements d'ordinador* i *Serveis*. I així successivament. L'elecció de qualsevol de les categories, situant-nos al damunt, provoca l'aparició de la llista dels productes de la categoria seleccionada.



Símbol utilitzat en les vistes arbre

Les vistes arbre/llista es distingeixen perquè en la seva declaració incorporen:

```
1 <field name="type">tree</field>
```

i l'element arrel de l'XML que defineix l'arquitectura de la vista és `tree`.

La diferència entre les vistes arbre i vistes llista en la declaració radica en l'existència, en les vistes arbre, del següent element per indicar el camp `one2many` que determina la jerarquia:

```
1 <field name="field_parent">camp_one2many</field>
```

Les vistes arbre/llista són més simples que les vistes formulari i tenen menys opcions. De manera similar a les vistes formulari, incorporen en la seva estructura elements `field`. La taula 2.5 recull els atributs que poden acompanyar l'element arrel `tree`.

Taula 2.5. Relació d'atributs per l'element "tree" en la definició d'un "tree"

Atribut	Significat
`colors`	Llistes de colors definides en un diccionari Python per tal que les línies es puguin acolorir segons diferents condicions.
`editable`	En cas d'estar definit, els valors possibles són `top` i `bottom` i permeten editar els registres directament a la llista, sense necessitat de passar a la vista formulari. Si el valor és `top`, els nous registres s'afegeixen al capdamunt de la llista i si el valor és `bottom`, els nous registres s'afegeixen al final.
`toolbar`	Només per a les vistes arbre. Valor a `True` per mostrar el nivell superior de la jerarquia d'objectes amb una barra d'eines lateral.

Exemple d'anàlisi del codi de la vista arbre Productes per categoria

El codi de la vista arbre Productes per categoria (figura 2.17) és:

```
<record id="product_category_tree_view" model="ir.ui.view">
  <field name="name">product.category.tree</field>
  <field name="model">product.category</field>
  <field name="type">tree</field>
  <field name="field_parent">child_id</field>
  <field name="arch" type="xml">
    <tree toolbar="True" string="Product Categories">
      <field name="name"/>
    </tree>
  </field>
</record>
```

Veiem que l'element `field_parent` informa que l'arbre es munta a partir de la jerarquia indicada pel camp `child_id`. Fixem-nos com és la definició de la classe `product.category`:

```
class product_category(osv.osv):
  ...
  _name = "product.category"
  _description = "Product Category"
  _columns = {
    ...
    'parent_id':
      fields.many2one('product.category', 'Parent Category',
                      select=True, ondelete='cascade'),
    'child_id':
      fields.one2many('product.category', 'parent_id',
                      string='Child Categories'),
    ...
```

Si modifiquem la vista `product_category_tree_view` eliminant l'atribut `toolbar`, cosa que podem aconseguir ràpidament des del client web, i tornem a demanar l'opció *Productes per categoria*, veurem que el desplegable per escollir el recurs de primer nivell de la jerarquia ha desaparegut (figura 2.18) i directament ens facilita la llista de tots els recursos de primer nivell.

Figura 2.18. Exemple de vista arbre sense l'atribut "toolbar"

Exemple d'anàlisi de la vista llista Productes

Quan accedim a *Productes* (ja sigui amb la drecera o des de l'opció de menú), apareixen els productes en vista llista. La majoria dels productes apareixen de color negre, però n'hi ha alguns en vermell i alguns en blau. Això és a causa que s'està utilitzant l'atribut `colors` en la definició de l'element `tree`.

El codi de la vista llista és:

```
<record id="product_product_tree_view"
    model="ir.ui.view">
  <field name="name">product.product.tree</field>
  <field name="model">product.product</field>
  <field name="type">tree</field>
  <field eval="7" name="priority"/>
  <field name="arch" type="xml">
    <tree
      colors="red:virtual_available&lt;0;
              blue:virtual_available&gt;=0 and
                   state in ('draft','end','obsolete');
              black:virtual_available&gt;=0 and
                    state not in ('draft','end','obsolete')"
      string="Products">
    <field name="default_code"/>
    <field name="name"/>
    <field name="categ_id" invisible="1"/>
    <field name="variants"
           groups="product.group_product_variant"/>
    <field name="uom_id" string="UoM"/>
    <field name="type"/>
    <field name="qty_available"/>
    <field name="virtual_available"/>
    <field name="lst_price"/>
    <field name="price"
           invisible="not context.get('pricelist',False)"/>
    <field name="standard_price" groups="base.group_extended"/>
    <field name="state" groups="base.group_extended"/>
    <field name="company_id"
           groups="base.group_multi_company"
           invisible="1"/>
    </tree>
  </field>
</record>
```

Observem-hi el contingut de l'atribut colors a l'element `tree`: es defineix el color que ha de mostrar el registre en funció dels valors dels camps `virtual_available` i `state`.

Si voleu que la llista sigui editable només heu d'afegir l'atribut `editable` a l'element `tree`, amb el valor `top` o `bottom` segons vulgueu que els nous registres s'afegeixin per dalt o per baix. Podeu comprovar-ho, de manera ràpida, modificant la vista des del client web.

Els ginys `one2many` i `many2many` emprats en les vistes `form` mostren la informació de la vista llista associada als objectes referenciats pels camps `one2many` i `many2many`.

Relació entre els ginys one2many/many2many i les vistes tree

Situem-nos en el mòdul `school` vigent (*school_05*) i considerem la vista llista del calendari de cursos (`school.course.event.tree`). Aquesta vista mostra quatre columnes: `course_id`, `date`, `date_end` i `name` i té l'atribut `string="school.course.event"`.

Fixem-nos en la pestanya *Calendar* del formulari manteniment de *Courses*. En aquesta pestanya s'hi visualitza el camp `event_ids` que és un camp `one2many` que referencia la classe `school.course.event`. Observem que mostra les quatre columnes i l'etiqueta poc adequada de la vista `school.course.event.tree`.

Si modifiquem la vista llista `school.course.event.tree` (columnes, etiqueta...) els canvis repercutiran en la visualització de la pestanya *Calendar* del manteniment de *Courses*.

Però, i si volem mantenir la vista llista sota una determinada estructura i, en canvi, ens interessa que en un camp `one2many` o `many2many` la visualització sigui diferent?

L'element `tree` també s'utilitza per modificar l'estructura dels ginys `one2many` i `many2many` en una vista `form`. Per aconseguir-ho, a l'element `field` corresponent al camp `one2many` o `many2many` se li ha d'afegir un element fill `tree`, que haurà de contenir els elements `fields` que es vulguin visualitzar en el giny. És a dir, mentre que el funcionament automàtic dels ginys `one2many` i `many2many` mostra la vista llista associada a l'objecte referit en el camp `one2many` i `many2many`, amb l'element `tree` podem alterar-la i mostrar aquells camps que interessi i amb les característiques adequades (ordre, color...).

Cal tenir en compte, també, que si en un element `tree` explicita una columna `one2many` o `many2many`, OpenERP hi visualitzarà el nombre de recursos (registres) que en aquell moment estan referenciats.

Exemple de millora de vistes tree en el mòdul school

Ens proposem millorar les vistes `tree` del mòdul `school` vigent (versió *school_05*).

En la vista llista dels professors fem que els professors amb `contract 'named'` es visualitzin de color blau i els professors amb `contract 'trainee'` es visualitzin de color verd.

En el formulari de manteniment dels professors, modifiquem les columnes que es visualitzen a l'apartat *Courses* de manera que desaparegui el nom del professor, ja que és redundant amb la informació de la capçalera del formulari i la columna *Note*. Ens interessa, però, que per cada curs aparegui el nombre d'estudiants i el nombre d'edicions.

En el formulari de manteniment dels estudiants modifiquem les columnes que es visualitzen a l'apartat *Subscriptions to courses*, de manera que apareguin: *Course*, *Subject*, *Professor* i *Total hours*. Ho fem perquè ens sembla que hi ha massa columnes i únicament ens interessa mostrar-ne les indicades.

En el formulari de manteniment dels cursos, a la pestanya *Calendar* hi eliminem la columna *Course*, ja que és redundant amb el contingut de la capçalera, eliminem també la inadequada etiqueta que hi apareix i fem que sigui editable. Després afegim els nous registres al final.

Podeu trobar la versió millorada del mòdul `school` a l'arxiu school_06.zip dins l'apartat "Mòduls OpenERP" dels annexos del web.

De la mateixa manera que en un camp `one2many` o `many2many` d'una vista `form`, la utilització de l'element `tree` permet explicitar les columnes a visualitzar (exemples anteriors), ja que si no s'utilitza OpenERP mostra tots els camps dels objectes referenciats, pot interessar que l'edició en vista `form` de qualsevol dels recursos mostrats en un giny `one2many` rebi algun camp ja emplenat i no sigui editable.

Com a exemple, fixem-nos que en la situació actual del mòdul `school` (versió 06) en editar un professor, l'apartat de detall *Courses* no és editable a la pròpia graella i, per tant, quan editem un dels cursos existents o creem un nou curs, se'ns obre la vista `form` de manteniment dels cursos, en la qual el camp *Professor* és editable. Això suposa un problema greu, ja que si hem arribat a aquest manteniment des de l'edició d'un professor, és clar que el camp *Professor* no ha de ser modificable i, en crear un curs, el camp *Professor* ja ha d'aparèixer emplenat amb el valor del professor pel que estem creant els cursos.

La solució es troba en introduir, després de l'element `tree` que facilita el contingut de la graella detall del formulari principal, un element `form` per facilitar el formulari d'edició corresponent als objectes de la graella. Aquest darrer element `form` no ha de contenir el camp que referencia l'objecte del formulari principal i OpenERP és prou intel·ligent per emplenar aquest camp amb l'identificador que correspon. El següent esquema visualitza la solució:

```
<record model="ir.ui.view" id=...>
  ...
  <field name="type">form</field>
  <field name="arch" type="xml">
    <form string="Títol">
    <!-- Camps del formulari principal -->
      <field name="xxx" colspan="4" nolabel="1">
      <!-- xxx és un camp one2many -->
        <tree>
          <!-- Indiquem els camps de la graella -->
        </tree>
        <form>
          <!-- Indiquem els camps que es desitja que
          apareguin en editar/crear un recurs de
          la graella corresponent al camp xxx -->
        </form>
      </field>
    </form>
  </field>
</record>
```

Proveu a retocar el manteniment de professors del mòdul `school` vigent (versió *school_06*) de manera que en editar un professor, l'acció de crear un curs o editar un curs ja assignat mostri un formulari en el qual no aparegui el camp professor i, en canvi, l'assignació del professor sigui l'adequada.

Podeu trobar la versió millorada del mòdul `school` a l'arxiu school_07.zip dins l'apartat "Mòduls OpenERP" dels annexos del web.

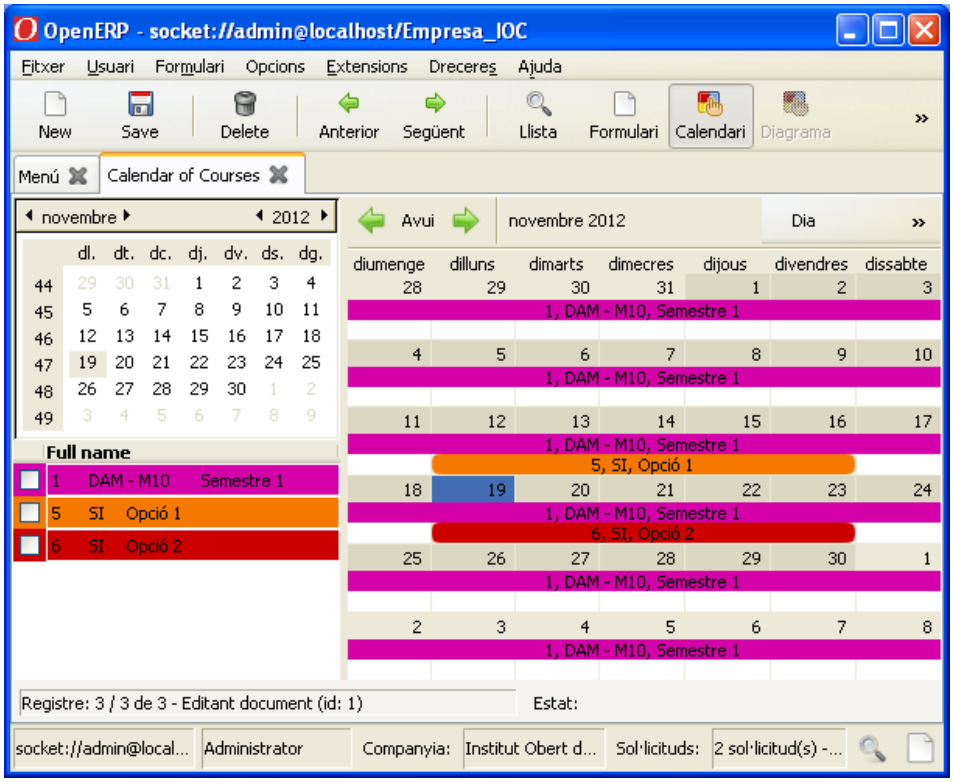
2.2.4 Vistes calendar

Les vistes calendari per a un objecte d'OpenObject són possibles quan l'objecte conté camps `date` o `datetime` que permeten situar cada recurs en un interval de

temps (un dia, un dia-hora o un interval de dies) i faciliten la visualització i/o l'edició dels recursos dins el seu interval de temps.

La figura 2.19 mostra un exemple de vista *calendar* visualitzada des del client GTK corresponent al calendari de cursos (objecte `school.course.event`) del mòdul `school`. Hi podem veure, a la zona esquerra, un giny corresponent al mes actual (novembre del 2012) amb el dia actual marcat i, a sota, la llista de tots els recursos de l'objecte `school.course.event` situats en aquest mes. Fixem-nos que hi ha tres recursos, amb diferent color i amb una llegenda explicativa. A la part dreta podem observar com el recurs *1 DAM-M10 Semestre 1* apareix en tots els dies del mes (a causa que es desenvolupa des del 20 de setembre del 2012 fins el 19 de gener del 2013) mentre que el recurs *5 SI Opcio 1* s'ubica des del 12 fins el 16 de novembre i el recurs *6 SI Opció 2* des del 19 fins el 23 de novembre. La vista calendari permet editar qualsevol recurs fent doble clic damunt i també permet l'arrossegament per canviar-los d'ubicació. Així mateix, si el calendari mostra molts recursos i la visibilitat és difícil, sempre podem seleccionar els recursos que volem visualitzar a la llista de recursos de la part esquerra.

Figura 2.19. Exemple de vista calendari des del client GTK

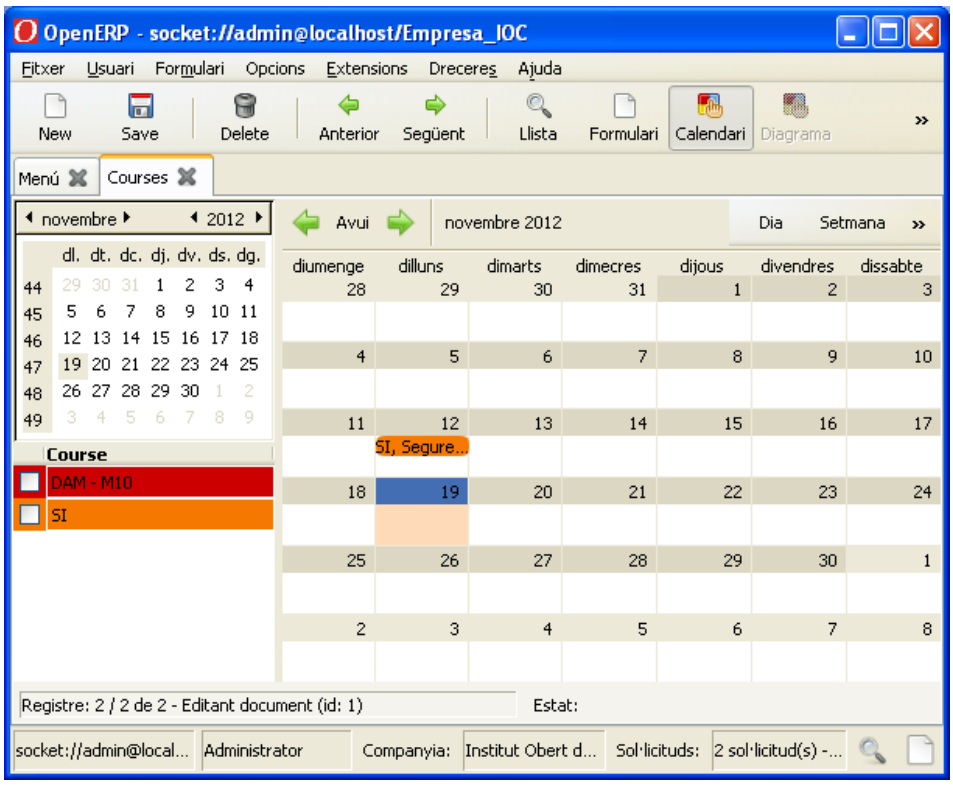


La visualització de les vistes *calendar* des del client web de la versió 6.1 d'OpenERP presenta alguna anomalia, com ara no mostrar, en un mes, els recursos que s'han iniciat en el mes anterior. Així, per exemple, la visualització de la vista *calendar* de la figura **??** des del client web, no mostra el recurs *1 DAM-M10 Semestre 1* ja que s'ha iniciat en un mes anterior. Esperem que aquest *bug* se solucioni en posteriors versions.

La figura 2.20 mostra un altre exemple de vista *calendar*. En aquest cas es tracta de la vista *calendar* per a l'objecte `school.course` que conté una data del curs i,

la vista, situa cada recurs en la seva data. Observem com el recurs *SI* està ubicat en el dia 12 de novembre. El recurs *DAM-M10*, que té com a data el 20 de setembre, lògicament no apareix en el mes de novembre.

Figura 2.20. Exemple de vista calendari des del client GTK



Les vistes calendari es distingeixen perquè en la seva declaració incorporen:

```
1 <field name="type">calendar</field>
```

L'element arrel de l'XML que defineix l'arquitectura de la vista és `calendar` i pot contenir els següents atributs:

- `string`: per al títol de la vista.
- `date_start`: que ha de contenir el nom d'un camp `datetime` o `date` del model.
- `date_delay`: que ha de contenir la llargada en hores de l'interval. Aquest atribut té preferència sobre l'atribut `date_stop`.
- `date_stop`: que ha de contenir el nom d'un camp `datetime` o `date` del model. Aquest atribut és ignorat si existeix l'atribut `date_delay`.
- `day_length`: per indicar la durada en hores d'un dia. OpenObject utilitza aquest valor per calcular la data final a partir del valor de `date_delay`. Per defecte, el seu valor és 8 hores.
- `color`: per indicar el camp del model utilitzat per distingir, amb colors, els recursos mostrats a la vista.

- `mode`: per mostrar l'enfoc (dia/setmana/mes) amb el qual s'obre la vista. Valors possibles: `day`, `week`, `month`. Per defecte, `month`.

Els elements fills de l'element arrel `calendar` s'utilitzen per mostrar el seu contingut dins la barra acolorida que ubica cada recurs dins el calendari.

Exemple de vista calendar en el mòdul school

Ens proposem definir dues vistes `calendar` en el mòdul school vigent (versió *school_07*).

A l'objecte `school.course.event` la vista calendar següent:

```
<record model="ir.ui.view"
        id="view_school_course_event_calendar">
  <field name="name">school.course.event.calendar
  </field>
  <field name="model">school.course.event</field>
  <field name="type">calendar</field>
  <field name="arch" type="xml">
    <calendar color="full_name" date_start="date"
              date_stop="date_end">
      <field name="id"/>
      <field name="course_id"/>
      <field name="name"/>
    </calendar>
  </field>
</record>
```

Fixem-nos que els elements fills de l'element calendar són els camps `id`, `course_id`, `name`, i són els camps que es visualitzen, separats per comes, a l'interior de les barres acolorides que mostren cada recurs dins el calendari. L'atribut `color` de l'element `calendar` conté el camp `full_name` que pretén mostrar la concatenació dels camps `id`, nom corresponent a `course_id` i `name`. Per aconseguir-ho, hem hagut d'ampliar el model de l'objecte `school.course.event` amb el camp funcional `full_name` que es basa en el mètode `_full_name` definit a la pròpia classe.

```
class school_course_event(osv.osv):
  def _full_name(self, cr, uid, ids, field_name, arg, context):
  result = {}
  for r in self.browse(cr, uid, ids, context=context):
    result[r.id]=str(r.id)+'\t'+str(r.course_id.name) \
                          +'\t'+r.name
  return result
  ...
  _columns = {
    ...
    'full_name': fields.function(_full_name,type='char',
                                 size=60,readonly='True',
                                 string='Full name')
  }
```

A l'objecte `school.course`, la vista `calendar` següent:

```
<record model="ir.ui.view"
        id="view_school_course_calendar">
  <field name="name">school.course.calendar</field>
  <field name="model">school.course</field>
  <field name="type">calendar</field>
  <field name="arch" type="xml">
    <calendar color='name' date_start='date'>
      <field name="name"/>
      <field name="subject"/>
    </calendar>
  </field>
</record>
```

El disseny de mètodes es troba al subapartat "Els mètodes".

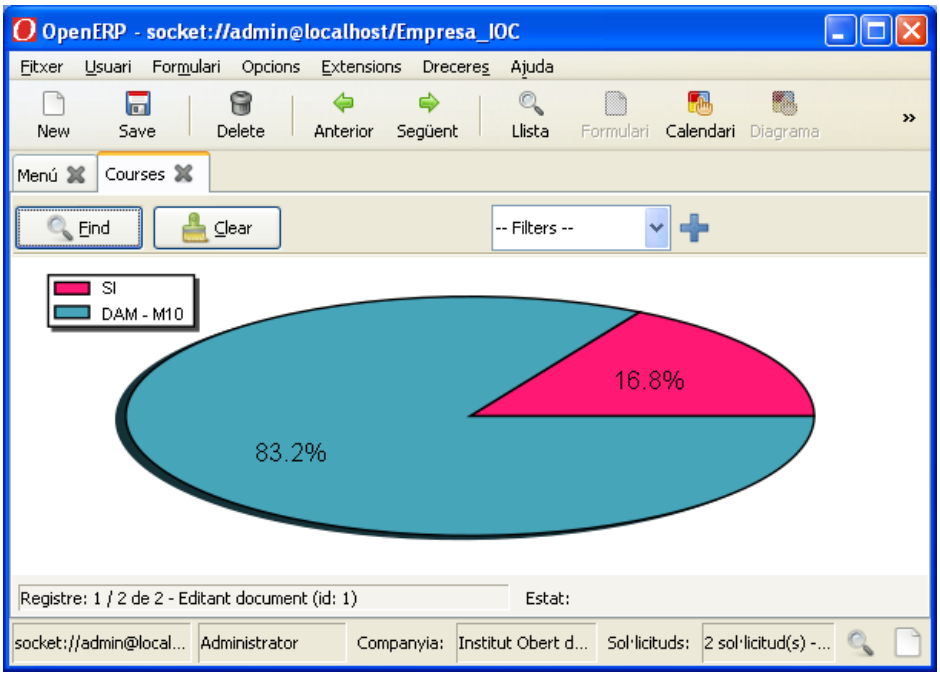
Podeu trobar la versió millorada del mòdul `school` a l'arxiu school_08.zip dins l'apartat "Mòduls OpenERP" dels annexos del web.

2.2.5 Vistes graph

Les vistes gràfic són un tipus de vista que permeten la visualització de gràfics construïts a partir de les dades contingudes en els recursos.

La figura 2.21 mostra un gràfic de sectors corresponent a les hores de cadascun dels cursos respecte el total d'hores dels cursos existents. La figura 2.22 correspon al mateix càlcul però amb un gràfic de barres.

Figura 2.21. Exemple de gràfic de sectors en el client GTK



Les vistes gràfiques es distingeixen perquè en la seva declaració incorporen:

```
1 <field name="type">graph</field>
```

L'element arrel de l'XML que defineix l'arquitectura de la vista és `graph` i pot contenir els següents atributs:

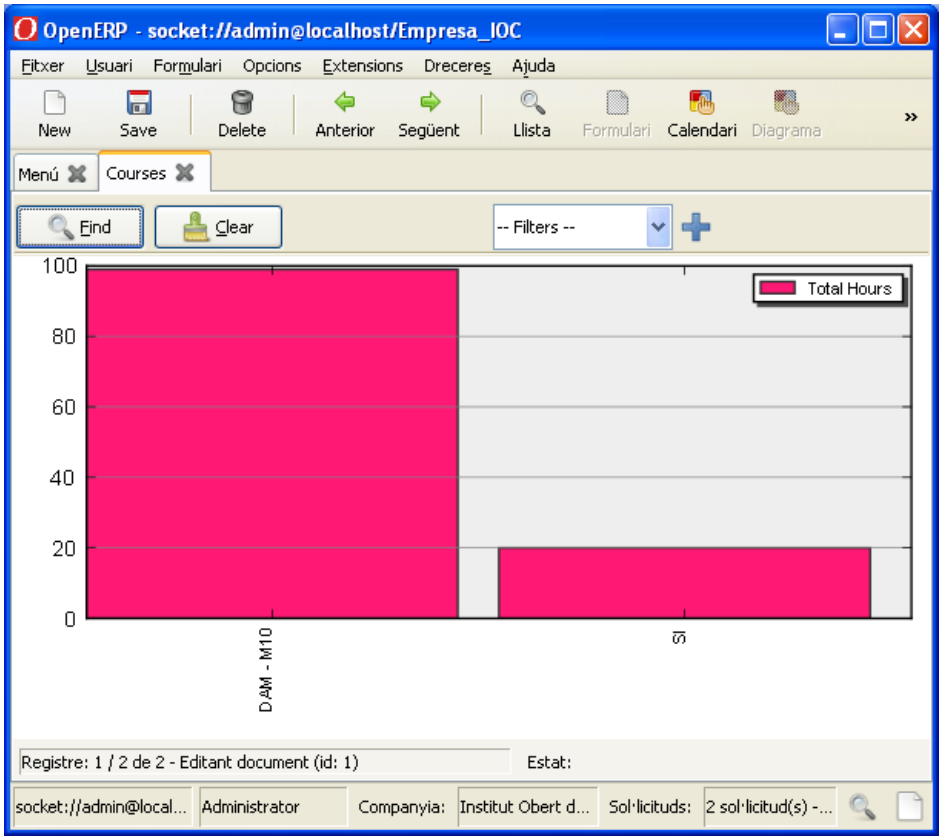
- `string`: per al títol de la vista.
- `type`: per al tipus de gràfic. Per defecte és un gràfic de sectors (figura 2.21). Cal indicar el valor `bar` per un gràfic de barres verticals (figura 2.22).
- `orientation`: per indicar el valor `horizontal` quan `type` sigui `bar` i es desitgi un gràfic de barres horitzontal.

La definició dels elements fills de l'element arrel `graph` determina el contingut del gràfic:

- El primer camp indica el contingut de l'eix X (horitzontal). És obligatori.

- El segon camp indica el contingut de l'eix Y (vertical). És obligatori.
- El tercer camp indica el contingut de l'eix Z en gràfics tridimensionals. És optatiu.

Figura 2.22. Exemple de gràfic de barres en el client GTK



A cadascun dels camps que determinen els eixos se'ls pot aplicar els atributs següents:

- `group="True"`, per utilitzar en el segon o tercer camp per tal d'indicar que cal agrupar els recursos d'igual valor que hi ha en aquest camp. El primer camp sempre actua amb `group="True"`.
- `operator`, per indicar l'operador que cal utilitzar per calcular el valor en la resta dels camps, com a conseqüència de l'agrupació indicada a l'atribut `group`. Els operadors permesos són: `+` (suma), `*`(producte), `**`(exponent), `min` i `max` (menor i major valors de la llista de valors de l'agrupació). Per defecte, actua l'operador `+`.

La visualització de les vistes `graph` que utilitzen l'atribut `group`, des del client web de la versió 6.1 d'OpenERP, presenta alguna anomalia. Esperem que aquest *bug* se solucioni en posteriors versions.

Exemple de vista graph en el mòdul school

Ens proposem definir una vista `graph` en el mòdul `school` vigent (versió `school_08`).

A l'objecte `school.course`, introduïm la vista següent:

```
<record model="ir.ui.view"
        id="view_school_course_graph">
  <field name="name">school.course.graph</field>
  <field name="model">school.course</field>
  <field name="type">graph</field>
  <field name="arch" type="xml">
    <graph string="Percentage of hours">
      <field name="name"/>
      <field name="hours_total"/>
    </graph>
  </field>
</record>
```

Es tracta d'un gràfic de sectors (bidimensional) en el qual el camp que facilita la llegenda (eix horitzontal en cas d'un gràfic de barres) és `name` de l'objecte `school.course` i el camp que facilita el valor per calcular els sectors (eix vertical en un gràfic de barres) és `hours_total`.

Cal tenir present que l'actual mòdul `school` permet tenir diversos cursos amb el mateix nom i que el gràfic anterior agrupa els cursos amb igual `name` (agrupació automàtica pel primer `field` de l'element `graph`). Si interessa no poder tenir diferents cursos amb igual nom, cal definir una restricció d'unicitat (`_sql_constraint`) sobre el camp `name` de l'objecte `school.course`. En canvi, si interessa permetre diferents cursos amb igual nom però en el gràfic anterior no volem que s'agrupin, caldrà crear un camp `function` a l'objecte `school.course`, que serà la concatenació dels camps `id` i `name`, i utilitzar aquest nou camp en el gràfic en lloc del camp `name`.

Podeu trobar la versió millorada del mòdul `school` a l'arxiu school_09.zip dins l'apartat "Mòduls OpenERP" dels annexos del web.

La manera més eficient per generar gràfics estadístics sobre objectes d'OpenERP és:

1. Crear en OpenERP un objecte estadístic basat en una vista que OpenERP crea en la base de dades de PostgreSQL.

2. Crear una vista llista i una vista gràfic sobre l'objecte estadístic.

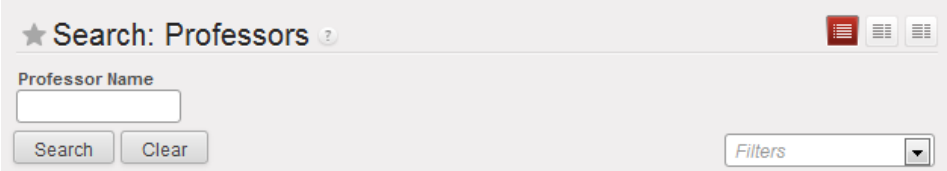
El disseny d'objectes estadístics es troba a l'apartat "Reporting & BI"

2.2.6 Vistes search

Les vistes *search* són una nova funcionalitat d'OpenObject des de la versió 6.0 que consisteixen en la possibilitat de personalitzar el panell de cerca en les vistes no formulari.

Si observem qualsevol de les vistes llista del mòdul `school` que estem millorant, veurem que apareix un panell de cerca similar al de la figura 2.23. Hi veiem un desplegable *Filters* que ens permet afegir filtres per acotar els registres per visualitzar i els botons *Search* per executar la cerca i *Clear* per netejar els filtres. Aquesta vista *search* és la que incorpora OpenObject quan no se n'indica cap, i incorpora el camp de filtre *Professor Name* perquè la vista `form` de la classe `school.professor` té el camp *name* amb l'atribut `select="1"`. OpenERP a partir de la versió 6.0 té en compte l'atribut `select` quan el model no incorpora cap vista *search*.

Figura 2.23. Vista search per defecte en vistes llista



Un exemple de vista *search* sofisticada el tenim en el manteniment dels tercers (*partners*), tal com mostra la figura 2.24. En cas que s'hagi instal·lat el mòdul `crm`, la vista *search* es veu ampliada amb més filtres, tal com mostra la figura 2.25.

Figura 2.24. Vista search en la llista de tercers (sense instal·lar el mòdul crm)

La vista *search* de la figura 2.24 conté:

- Els botons *Clients* i *Proveïdors* per facilitar el filtre segons el tipus de tercer. L'usuari els ha de prémer (un o ambdós o cap) segons el filtrat que li interessi.
- Els camps de filtre *Nom*, *Contactes*, *País* i *Venedor*. L'usuari ha d'introduir el valor de filtre.
- Una zona expandible *Agrupa per...* que mostra el botó *Venedor* per facilitar veure els registres resultants dels filtres agrupats per venedor.
- Els elements de la vista *search* per defecte: *Search, Clear* i *Filters*.

Figura 2.25. Vista search en la llista de tercers (amb el mòdul crm instal·lat)

Les vistes *search* es distingeixen perquè en la seva declaració incorporen:

```
1 <field name="type">search</field>
```

L'element arrel de l'XML que defineix l'arquitectura de la vista és `search` i pot tenir els següents elements fills:

- `group`: per agrupar un conjunt de filtres i/o condicions d'agrupament sota un mateix títol, que s'indica a l'atribut `string`. No és obligatòria

l'existència del títol. També admet l'atribut `expand` (valors possibles: 0/1 – per defecte: 1) per indicar si el grup ha d'aparèixer expandit (1) o no (0).

- `separator`: per situar un separador, amb atribut `orientation`, que per defecte és `vertical`. En una vista *search* no té sentit el separador horitzontal. El separador no es visualitza en el client web d'OpenERP 6.1.
- `label`: per situar una etiqueta.
- `field`: per situar un camp textual de filtre en el qual l'usuari ha d'escriure un valor.
- `filter`: per situar un botó que permet filtrar i/o agrupar sota unes determinades condicions definides en el botó.
- `newline`: per provocar un salt de línia.

Pel que fa a l'element `field`, cal saber que:

- En els camps `many2one`, quan l'usuari comença a escriure un valor, OpenObject mostra els valors possibles que comencen per aquell valor; si es vol que el camp incorpori un desplegable, cal utilitzar l'atribut `widget="selection"`.
- Pot anar acompanyat de l'atribut `context`.
- OpenObject realment construeix un domini `[('camp','=','valor')]` amb el valor introduït en el camp. Es pot alterar l'operador = de la condició utilitzant l'atribut `operator`. Així mateix, es pot alterar el domini anterior per un domini personalitzat, amb l'atribut `filter_domain` i, en tal cas, indicar el domini com una llista Python de condicions, en la qual les condicions són tuples Python de tres elements: `('camp','condició','valor')`.

La sintaxi de l'element `filter` és:

```
<filter string="Etiqueta" icon="nomIcona"
        domain="llistaCondicions"
        help="missatgeAjuda"
        context="{'group_by':'llistaCamps'}/>
```

en la qual:

- L'atribut `icon` ha de contenir una de les icones d'OpenERP, de manera anàloga a l'atribut `icon` dels elements `menuitem`.
- L'atribut `domain` serveix per indicar el filtre i és, com en les accions, una llista Python de condicions que s'enllacen amb una clàusula AND; les condicions són tuples Python de tres elements: `('camp','condició','valor')`.
- L'atribut `context` serveix per indicar un context i, amb el valor clau `group_by`, per aconseguir agrupar els registres pels camps indicats a `llistaCamps`, separats per comes.

L'element `filter` es pot ubicar com a fill d'un element `field` per indicar que el seu filtre s'aplica específicament sobre el camp. En aquest cas, el botó és més petit i s'adequa a l'altura d'un camp `field`.

Per finalitzar amb les vistes *search* recordeu que el nom de la vista *search* que es vol utilitzar s'ha d'indicar a l'element `search_view_id` de l'acció *window* que invoca la vista.

Exemple d'anàlisi del codi de la vista search

El codi XML corresponent a la vista *search* de la figura 2.24 és:

```
<record id="view_res_partner_filter" model="ir.ui.view">
  <field name="name">res.partner.select</field>
  <field name="model">res.partner</field>
  <field name="type">search</field>
  <field name="arch" type="xml">
    <search string="Search Partner">
      <group col='10' colspan='4'>
        <filter string="Customers" name="customer"
          icon="terp-personal"
          domain="[('customer','=',1)]"
          help="Customer Partners"/>
        <filter string="Suppliers" name="supplier"
          icon="terp-personal"
          domain="[('supplier','=',1)]"
          help="Supplier Partners"/>
        <separator orientation="vertical"/>
        <field name="name" select="1"/>
        <field name="address" select="1"/>
        <field name="country" select="1"/>
        <field name="user_id" select="1">
          <filter help="My Partners"
            icon="terp-personal+"
            domain="[('user_id','=',uid)]"/>
        </field>
      </group>
      <newline />
      <group expand="0" string="Group By...">
        <filter string="Salesman" icon="terp-personal"
          domain="[]"
          context="{'group_by' : 'user_id'}" />
      </group>
    </search>
  </field>
</record>
```

Fixem-nos que aquesta vista té dos grups, separats per un salt de línia (`newline`). A la figura 2.24 observem perfectament el primer grup, format pels botons *Clients* i *Proveïdors* i els camps *Nom*, *Contactes*, *País*, *Venedor* i un darrer botó enganxat al camp *Venedor*. Del segon grup només se'n veu el títol *Agrupat per...* ja que no està expandit (`expand="0"`).

Situem-nos en el primer grup. Observem que:

- Els dos primers botons es corresponen a dos elements `filter`. Observem que la condició del botó *Clients* és `[('customer','=',1)]` i que la condició pel botó *Proveïdors* és `[('supplier','=',1)]`.
- Els quatre camps que vénen a continuació es corresponen a elements `field`. El camp `user_id`, que és de tipus `many2one`, podria portar l'atribut `widget="selection"` i llavors l'usuari visualitzaria una llista desplegable.
- El darrer element, el botó petit, correspon a un element `filter`, fill del darrer element `field`, ja que es tracta d'un filtre sobre el mateix camp `user_id`.

Situem-nos en el darrer grup. Si l'expandim, veurem que conté un únic botó, que es correspon amb l'element `filter string "Salesman"` del codi anterior. Aquest element no incorpora cap filtre (`domain=[]`) però en canvi incorpora l'atribut context amb la clau `group_by` acompanyada del valor `user_id` que provoca que, en cas que l'usuari premi el botó, els socis de negocis apareixeran agrupats per venedor (`user_id`).

Exemple d'incorporació de vista search en el mòdul school

Ens proposem incorporar una vista *search* en el mòdul `school` vigent (versió *school_09*).

En la vista llista dels professors cal:

- Afegir dos botons de filtre *Named* i *Trainee* per seleccionar els professors de cada tipus de contracte.
- Afegir un camp de filtre *Partner* per permetre seleccionar els professors associats a un determinat soci de negocis.
- Afegir un botó de filtre, associat al camp anterior, per permetre seleccionar els professors associats a la companyia principal d'OpenERP.
- Afegir un camp de filtre per permetre seleccionar els professors la dedicació horària setmanal (en hores) dels quals estigui per sobre del valor a introduir per l'usuari.

La vista és:

```
<record model="ir.ui.view"
        id="view_school_professor_search">
  <field name="name">school.professor.search</field>
  <field name="model">school.professor</field>
  <field name="type">search</field>
  <field name="arch" type="xml">
    <search>
      <group>
        <filter string="Trainees" name="contract"
                icon="terp-personal"
                domain="[('contract','=','trainee')]"/>
        <filter string="Nameds" name="contract"
                icon="terp-personal"
                domain="[('contract','=','named')]"/>
        <separator orientation="vertical"/>
        <field name="partner_id">
          <filter help="Principal Company Professors's"
                  icon="terp-personal+"
                  domain="[('partner_id','=',1)]"/>
        </field>
        <field name="hours_available"
               string="Hours per Week >="
               operator=">="/>
      </group>
    </search>
  </field>
</record>
```

Podeu trobar la versió millorada del mòdul `school` a l'arxiu school_10.zip dins l'apartat "Mòduls OpenERP" dels annexos del web.

Fixeu-vos en el domini `[('partner_id','=',1)]` utilitzat per filtrar els professors associats a la companyia principal d'OpenERP. Aquest filtre funciona perquè la companyia principal sempre és la primera inserció a `res_partner` i, en conseqüència, sempre té `id=1`.

2.3 Mètodes

Per encarar amb garanties el disseny de mètodes en OpenObject es pressuposa uns coneixements mínims de disseny de mètodes en Python.

La capa ORM d'OpenObject facilita un seguit de mètodes que s'encarreguen del mapatge entre els objectes Python i les taules de PostgreSQL. Així, disposem de mètodes per crear, modificar, eliminar i cercar registres a la base de dades. Aquests mètodes són utilitzats de manera automàtica per OpenObject en l'execució dels diversos tipus de vista que OpenObject ens permet dissenyar.

En ocasions, però, pot ser necessari alterar l'acció automàtica de cerca – creació – modificació – eliminació facilitada per OpenObject i, llavors, haurem de sobreescriure els corresponents mètodes en les nostres classes.

Com exemple d'aquesta necessitat, podem considerar el cas de la gestió de comandes de venda (classe `sale_order`) dins el fitxer `sale.py` del mòdul `sale` d'OpenERP. Si hi fem una ullada, al final de la classe hi trobem el disseny dels mètodes `unlink` (eliminar), `create` (crear) i `write`(modificar). Cadascun d'ells executa un seguit de comprovacions i/o accions i, si tot és correcte, invoca la crida dels corresponents mètodes `unlink`, `create` i `write` de la capa ORM. Així, el mètode `unlink` (eliminació de comandes) comprova si les comandes a eliminar tenen l'estat *draft* o *cancel* (estats en els quals l'eliminació és permesa, segons la lògica de negoci) i si alguna de les comandes no és eliminable genera una excepció tot avortant l'eliminació; en canvi, si totes són eliminables, procedeix a l'eliminació tot invocant el mètode `unlink` subministrat per la capa ORM.

Els programadors en el *framework* OpenObject hem de conèixer els mètodes subministrats per la capa ORM i hem de dominar el disseny de mètodes per:

- Poder definir camps funcionals en el disseny del model.
- Poder definir l'acció que cal executar en modificar el contingut d'un *field* d'una vista *form* (atribut `on_change` del *field*)
- Poder alterar les accions automàtiques de cerca, creació, modificació i eliminació de recursos.

Als annexos del web hi trobareu l'annex "Depuració de codi Python en OpenERP", imprescindible a l'hora de desenvolupar mètodes.

Una darrera consideració a tenir en compte en l'escriptura de mètodes i funcions en OpenObject és que els textos de missatges inclosos en mètodes i funcions, per poder ser traduïbles, han de ser introduïts amb la sintaxi `_('text')` i el fitxer `.py` ha de contenir `from tools.translate import _` a la capçalera.

2.3.1 Mètodes ORM

La majoria de mètodes ORM que proporciona OpenObject tenen els paràmetres següents:

- `cr`: cursor de la base de dades
- `uid`: identificador de l'usuari que executa el mètode
- `ids`: llista d'enters amb els identificadors dels recursos als quals s'aplica el mètode
- `context`: diccionari Python amb un seguit de paràmetres que poden ser necessaris en l'execució del mètode, com per exemple: idioma, zona horària, companyia...

Recordem que OpenObject té, per cada classe Python, un únic objecte en memòria per gestionar tots els recursos del model associat a la classe. Per això, un mètode s'invoca sempre sobre l'únic objecte en memòria i rep, a través del paràmetre `ids`, la llista dels identificadors dels recursos sobre els quals actuar.

El per què del paràmetre ids en la definició d'un mètode d'OpenObject

Per ajudar a entendre la necessitat del paràmetre `ids`, obrim la vista `tree` de professors. Suposem que tenim diversos professors i n'efectuem una selecció (un o més d'un) per eliminar-los. En el moment que premem el botó *Delete* i confirmem l'eliminació, OpenObject invoca el mètode `unlink` sobre l'únic objecte `school_professor` existent en memòria i li passa la llista dels identificadors dels professors seleccionats, de manera que el mètode `unlink` pot eliminar-los.

El paràmetre `context` és un calaix de sastre ideat per OpenObject per ficar-hi tots aquells paràmetres que el mètode pot necessitar i que en canvi no s'ha previst en el prototipus del mètode.

Exemple de contingut dels paràmetres uid, ids i context

Per veure el contingut dels paràmetres `uid`, `ids` i `context` en algun mètode, sobreescrivim el mètode `unlink` (eliminació) a la classe `school.professor`:

```
def unlink(self, cr, uid, ids, context=None):
  print "Contingut de uid : " + str(uid)
  print "Llista ids amb '%d' valors: " % len(ids)
  for id in ids:
    print "\t'%d'" % id
  print "Diccionari context amb '%d' tuples: " % \
    len(context)
  for val in context.items():
    print "\t'%s'" % (val,)
  return osv.osv.unlink(self, cr, uid, ids, context=context)
```

Aquest mètode `unlink` mostra per la consola del sistema el contingut dels paràmetres `uid`, `ids` i `context` i invoca el mètode `unlink` de la capa ORM per procedir a l'eliminació. Observem el contingut en cas que s'hagi seleccionat l'eliminació de 3 professors:

```
Contingut de uid : 1
Llista ids amb '3' valors:
```

```
  '16'
  '17'
  '18'
Diccionari context amb '4' tuples:
  '('lang', u'ca_ES')'
  '('tz', False)'
  '('uid', 1)'
  '('section_id', False)'
```

L' `uid` 1 és l'identificador de l'usuari que està eliminant els professors; els `uids` dels professors per eliminar són `16`, `17` i `18`; el `context` conté 4 tuples: `lang` actiu, `timezone` activa, `uid` i `section_id`.

Alguns dels mètodes ORM més utilitzats són:

- Mètode `read`
- Mètode `name_get`
- Mètode `browse`
- Mètode `self.pool.get`
- Mètode `search`
- Mètode `create`
- Mètode `write`
- Mètode `unlink`

En el Technical Memento d'OpenERP disponible a doc.openerp.com/memento hi trobareu la relació completa dels mètodes ORM amb la seva sintaxi i algun exemple.

Mètode read

```
read (self, cr, uid, ids, fields=None, context=None)
```

`fields` : llista de camps dels quals es vol obtenir el contingut. Per defecte, tots els camps.

Per cada recurs d' `ids`, obté un diccionari de parelles `camp:valor` corresponent als camps indicats en el paràmetre `fields` i retorna una llista amb tots els diccionaris. Cada diccionari incorpora, encara que no s'hagi explicitat a `fields`, la parella `id:valor`.

Mètode name_get

```
name_get (self, cr, uid, ids, context=None)
```

Retorna una llista de tuples `(id, valor)` amb la representació textual dels recursos d' `ids`. Recordem que la representació textual d'un recurs és, en principi, el contingut del camp `name` (camp especial de l'atribut `_columns` de la classe Python) o, si està definit, el camp indicat a l'atribut `_rec_name` de la classe Python. Quan OpenObject necessita la representació textual d'un recurs, invoca automàticament el mètode `name_get`.

En ocasions, pot interessar que la representació textual d'un recurs sigui el resultat d'un determinat càlcul i llavors caldrà sobreescriure aquest mètode.

Exemple d'utilització dels mètodes read i name_get en el mòdul school

Situem-nos en la versió vigent del mòdul `school` (versió *school_10*).

La classe `school.student` té les columnes `idnum`, `name` i `surname`. Per tant, la representació textual d'un alumne és el seu nom i això és força adequat, però preferim que sigui la concatenació de cognom i nom, separats per una coma i un espai. Per aconseguir-ho cal sobreescriure el mètode `name_get` a la classe `school_student`:

```
def name_get (self, cr, uid, ids, context=None):
  res = []
  records = self.read(cr, uid, ids, ['name','surname'])
  for r in records:
    res.append((r['id'], r['surname']+", "+r['name']))
  return res
```

Fixem-nos que per poder generar la concatenació indicada ens cal per a cada recurs conèixer el cognom i el nom, i per aconseguir-ho hem utilitzat el mètode `read`. Observem, també, que `name_get` ha de retornar una llista de tuples i, per això, en invocar la funció `append` de Python, en el seu interior construïm el tuple `(r['id'],...)`.

La modificació efectuada a la classe `school_student` no és visible ja que no hi ha cap vista en la qual hagi d'aparèixer la representació textual d'un estudiant. Per forçar la utilització del nou mètode `name_get` ens plantegem que a la vista `tree` de l'objecte `school.student`, en comptes d'aparèixer les columnes *Name* i *Surname* per separat, aparegui una única columna *Full Name*. Per aconseguir-ho, cal afegir al model `school.student` el camp funcional `full_name`. Segons la definició dels camps funcionals, la funció que ha d'invocar un camp funcional ha de tenir un determinat nombre d'arguments, que no són els del mètode `name_get`. En conseqüència, cal crear una nova funció `_name_get_fnc` per utilitzar en la definició del camp funcional.

```
def _name_get_fnc(self, cr, uid, ids, prop, unknow_none,
                  context):
  res = self.name_get(cr, uid, ids, context)
  return dict(res)
```

A la zona `_columns` de `school.student` hi afegim el camp:

```
'full_name': fields.function(_name_get_fnc, type='char', size
    ='66',
                             string='Full Name',readonly='True'),
```

A causa que la funció invocada per un camp funcional ha de retornar un diccionari de parelles `id:valor` i que la funció `name_get` retorna una llista de tuples `(id, valor)`, ens va molt bé utilitzar la funció Python `dict` que, donada una llista de tuples de parelles, ens retorna el corresponent diccionari.

Per finalitzar l'exemple, comentar que sembla recomanable afegir l'atribut `_order` a la classe `school_student` per visualitzar els estudiants ordenats per cognom.

Per comprovar la visualització de la columna *Full Name* en la vista `tree` de `school.student`, cal retocar la vista.

Podeu trobar la versió millorada del mòdul `school` a l'arxiu school_11.zip dins l'apartat "Mòduls OpenERP" dels annexos del web.

Mètode browse

`browse (self, cr, uid, ids, context = None)`

Recupera els recursos de la llista `ids` com a llista d'objectes sobre els quals és possible utilitzar la notació de punt per accedir a qualsevol dels seus camps i, en el cas de camps relacionals, navegar cap als objectes apuntats. Si `ids` és un únic recurs (no llista), facilita el corresponent objecte (no llista d'objectes).

Aquest és un mètode fantàstic, a causa de la potència que facilita per poder navegar entre els objectes mitjançant els camps relacionals.

Com a exemple d'utilització del mètode `browse` podem revisar el mètode `_full_name` de la classe `school_course_event` del mòdul `school`.

Mètode self.pool.get

```
self.pool.get ('nom_objecte')
```

Mètode especial que permet obtenir l'objecte Python que gestiona els recursos del model associat a qualsevol classe.

Aquest mètode és imprescindible quan en el disseny d'un mètode d'una classe, cal invocar un mètode d'una altra classe. Recordem que OpenObject té un únic objecte que gestiona tots els recursos del model associat a cada classe. Per tant, en el contingut d'un mètode s'utilitza `self` per fer referència a l'objecte que gestiona els recursos de la classe, però és molt possible que calgui accedir a l'objecte que gestioni els recursos d'una altra classe i, en tal situació, el mètode `self.pool.get` hi facilita l'accés.

Mètode search

```
search (self, cr, uid, args, offset=0, limit=None, order=None,
context=None, count=False)
```

- `args`: llista de tuples especificant criteris de cerca.
- `offset`: nombre de registres a saltar (opcional).
- `limit`: màxim nombre de registres a recuperar (opcional).
- `order`: columnes sobre les quals ordenar el resultat. Per defecte, utilitza les columnes indicades a l'atribut `_order` de la definició de la classe Python i, en el seu defecte, la columna `id` de tota classe Python.
- `count`: si té valor `True` indica que el mètode només ha de retornar el nombre de registres que coincideixen amb els criteris indicats, en comptes de recuperar els seus identificadors.

Retorna una llista dels identificadors dels recursos que verifiquen els criteris de cerca o, si `count=True`, el nombre de registres.

Les condicions que cal introduir en els criteris de cerca han de verificar:

- Cada condició és un tuple del tipus `('nom_camp', 'op', valor)` en el qual `op` és qualsevol dels operadors binaris següents: `=`, `!=`, `>`, `>=`, `<`, `<=`, `like`, `ilike`, `in`, `not in`, `child_of`. L'operador `like` diferencia majúscules de minúscules, mentre que `ilike` no ho fa.
- La negació d'una condició s'efectua amb l'operador `!` davant el tuple.

- La combinació de condicions amb els operadors & (conjunció) i | (disjunció) s'efectua utilitzant la notació polonesa, també coneguda per notació prefix.

Exemple d'utilització de mètode self.pool.get i mètode search amb condició complexa

Suposem que en algun lloc (fora de cap mètode de la classe `school.professor`) necessitem efectuar una cerca de professors el nom dels quals conté el text `Rodríguez` i que tenen contracte `named`. Escriuríem:

```
prof = self.pool.get('school.professor')
ids = prof.search(cr,uid,['&',('contract','=','named'),
                              ('name','ilike','Rodriguez')])
```

Si la cerca s'hagués d'efectuar dins un mètode de la classe `school_professor` no caldria la crida a `self.pool.get` i escriuríem directament:

```
ids = self.search(cr,uid,['&',('contract','=','named'),
                              ('name','ilike','Rodriguez')])
```

Per obtenir els professors el nom dels quals no contingui el text Rodríguez i que tenen contracte `named`, escriuríem:

```
ids = self.search(cr,uid,['&',('contract','=','named'),
                          '!',('name','ilike','Rodriguez')])
```

Mètode create

`create (self, cr, uid, values, context=None)`

Crea un nou registre amb els valors especificats en el paràmetre `values`, de tipus diccionari. Retorna l'identificador del registre creat.

Mètode write

`write (self, cr, uid, ids, values, context=None)`

Modifica els registres especificats a la llista `ids`, amb els valors indicats en el paràmetre `values`, de tipus diccionari. Retorna `True`.

Mètode unlink

`unlink (self, cr, uid, ids, context=None)`

Elimina els registres especificats a la llista `ids`. Retorna `True`.

Els mètodes `create`, `write` i `unlink` són candidats a ser sobreescrits quan interessi alterar el seu funcionament habitual. En tal situació, la seva sobreescriptura ha d'incloure, en algun punt, la crida als mètodes `create`, `write` i `unlink` de la classe `osv.osv`. És a dir, la sobreescriptura de qualsevol d'aquests mètodes seria quelcom similar a:

```
def metode (self, cr, uid, ...):
  ... accions/comprovacions que corresponguin ...
  ... crida al mateix mètode de la classe osv.osv ...
  ... accions/comprovacions que corresponguin ...
  return ...
```

Hi ha més mètodes ORM interessants de conèixer i que trobareu en el Technical Memento d'OpenERP disponible a doc.openerp.com/memento.

2.3.2 Mètodes on_change

Les vistes `form` faciliten, en els camps editables, l'atribut `on_change` per indicar una acció a executar quan l'usuari canvia el valor del camp. El contingut d'aquest atribut ha de ser la crida a un mètode de la classe, amb els paràmetres que corresponguin (normalment el nou valor que ha pres el camp i, eventualment, valors d'altres camps).

El mètode invocat per l'atribut `on_change` té obligatòriament la sintaxi:

```
def metode (self, cr, uid, ids, paràmetres...)
```

i ha de retornar, sempre:

```
return {'value': resultat}
```

en què `resultat` ha de ser un diccionari de parelles `'camp':valor` amb els nous valors dels camps que s'han de veure afectats per l'execució del mètode.

Exemple d'utilització de mètodes on_change en el mòdul school

En el formulari de professors de la versió vigent del mòdul `school` (versió *school_11*), en canviar de professor, el telèfon –camp `related`– s'actualitza automàticament, però si editem un professor i li canviem l'adreça, mentre no s'enregistra el canvi, el formulari presenta la nova adreça però manté el telèfon de l'adreça antiga i això no és lògic. Cal, doncs, que en canviar el contingut de l'adreça es modifiqui automàticament el contingut del telèfon i això s'aconsegueix introduint, a la vista `form` l'atribut `on_change` en el camp `address_id`:

```
<field name="address_id"
  on_change="address_id_change(address_id)"/>
```

L'atribut `on_change` invoca el mètode que hem de tenir a la classe `school_professor`:

```
def address_id_change (self, cr, uid, ids, address_id):
  result = {}
  if address_id:
    obj = self.pool.get("res.partner.address")
    address = obj.browse(cr, uid, address_id)
    result['phone'] = address.phone
  else:
    result['phone'] = ""
  return {'value': result}
```

Així, en canviar l'adreça del professor, el telèfon canvia automàticament, sense haver d'esperar a enregistrar el canvi. En cas de deixar el professor sense adreça, el telèfon hauria de passar a estar en blanc. Aquest és el funcionament correcte des del client GTK, però en el client web d'OpenERP v.6.1 no funciona.

Podeu trobar la versió millorada del mòdul `school` a l'arxiu school_12.zip dins l'apartat "Mòduls OpenERP" dels annexos del web.

Per finalitzar amb els mètodes `on_change`, considerem la següent situació complexa. Suposem una vista `form A` des de la qual s'invoca una altra vista `form B` en què un atribut `on_change` invoca un mètode al qual s'ha de passar algun(s) valor(s) del formulari B (cap problema) i l'identificador del recurs actiu en el formulari A. Per solucionar aquesta situació, OpenObject ens facilita la sintaxi `parent.id`. En els mòduls OpenERP de la versió 6.1, aquesta situació únicament es presenta en dos mòduls:

- Vista `view_partner_form_inherit` del mòdul `base_contact` (fitxer base_contact_view.xml)
- Vista `view_inventory_form` del mòdul `stock` (arxiu stock_view.xml)