EPS
Escola Politècnica Superior
UdG

Projecte/Treball Fi de Carrera

Estudi:

Títol: Programari creador i consultor de genealogia familiar

Document: Memòria

Alumne: Ruben Fernández Torres

Director/Tutor: Joan Surrell
Departament:
Àrea: Llenguatges i Sistemes Informatics

Convocatòria (mes/any): Septembre/2008

Index

1.- Justificació i motivació del projecte

1.1. Introducció

Escollir el projecte va ser una tasca força costosa. No se'm va ocòrrer cap tema interessant i finalment vaig decidir realitzar un programari de gestió d'una empresa, el típic software de gestió. Un familiar pròxim és el propietari d'un petit negoci i al no trobar cap alternativa millor, vaig decidir començar-m'ho a plantejar.

Al poc temps em van donar la idea de relaitzar un projecte basat en l'ascendència i descència de les persones, dins una família, un treball que permetés a l'usuari crear la seva pròpia genealogia. Aquesta temàtica em va resultar força interessant i ràpidament vaig acceptar. Una de les raons que em va atraure, era que sabia que hi havia tasques molt diverses a desenvolupar; no s'acabaria convertint en un projecte monòton i, el més important, era un tema que m'interessava força. Per una part hi ha els familiars, per altra els arxius, per altra la utilització de les dades de la base de dades... la feina és molt variada.

D'altra banda creia, erròniament, que no hi hauria gaire software d'aquest tipus. Ràpidament em vaig donar compte que anava mal encaminat, milions de referències a la xarxa i moltíssims programes, em van fer adonar que molta altra gent, ja havia tractat aquest tema; per tant el programari ja no seria “novedós”, pero almenys intentaria que fos diferent i, potser, més intuïtiu que tots els que vaig tenir la ocasió de probar.

Suposo que una de les raons més importants perque la idea m'agradés ràpidament va ser la meva família. La meva, és formada per molts membres, però molts familiars viuen lluny i, vaig pensar, que aquesta era una molt bona idea, no per estar comunicats, pero si perque els llaços s'apropessin i també per compartir la informació d'una forma diferent; que d'aquesta forma tots poguessim compartir alhora esdeveniments de la vida personal, com també les fotos dels mateixos; així com molt contingut que podria quedar guardat i ser consultat en qualsevol moment.

En aquest projecte les dades més importants a guardar són els familiars. Aquests, es relacionen entre sí per mitjà dels lligams familiars que tenen, i que quedaran explícitament reflexats en l'arbre genealògic que es crearà quan l'usuari ho desitji i sobre el familiar que esculli. A més, els familiars són els propietaris, a més de molta informació que se'ls pot agregar, d'arxius i events (informació dels aconteixaments de la seva vida). Per tot això els familiars rebran a l'aplicació un tracte especial tant en la visualització de les seves dades, com en les diferents cerques que permetran, a l'usuari, seleccionar inequivocament i amb pocs passos el familiar desitjat.

El programari que volia desenvolupar haurà de ser senzill i còmode d'utilitzar. Amb multitud de cerques predeterminades per aconseguir una major interactivitat amb l'aplicació possible. Amb menús i accessos intuïtius, perquè qualsevol nivell d'usuari s'interrelacioni amb l'aplicació ràpidament. A més, el programari permetrà introduir arxius propis dels familiars, que alhora podran compartir amb tota la seva família. L'aplicació permetrà cercar els arxius, per mitjà de diferents cerques.

1.2 Objectius del projecte

Els objectius que espero aconseguir amb el projecte són els següents:

- Aconseguir crear una plataforma que permeti guardar dades de familiars agrupats en famílies.
- Agregar events de la vida personal a cada un dels membres de la família.
- Afegir la possibilitat d'associar arxius multimèdia tant a familiars com als events creats.
- Poder crear l'arbre genealògic complet, amb les dades introduïdes per l'usuari.
- Crear moltes vistes predeterminades, perque l'usuari pugui explorar i visualitzar, de diverses formes, les dades.
- Realitzar un treball de síntesi dels coneixements assolits durant el transcurs de la carrera.

1.3 Descripció de la memòria

En aquest document es trobarà tota la informació relacionada amb el projecte de final de carrera realitzat. S'explicaran detalladament tots els passos seguits per aconseguir un programar creador i consultor de genealogia familiar, així com també els resultats, millores i conclusions obtingudes.

En aquest primer punt s'esmenten les motivacions personals per a crear un programari de genealogia familiar, com també els objectius bàsics que es volen assolir finalment. S'ha intentat que en la memòria quedi pal.lès fidelment els passos, les dificultats i les impresions que s'han anat tenint durant l'el.laboració del programari.

En els següents punts s'explicaran els requisits funcionals del sistema, la recollida d'informació prèvia i els diagrames necessaris creats. En altres punts quedarà justificada la decisió sobre els programes necessaris escollits per l'elaboració d'aquest projecte final de carrera, el disseny final de la bdd, com també una extensa explicació sobre el codi generat. S'han adjuntat amb el codi moltes imatges del resultat final de l'aplicació, per ajudar a l'enteniment, així com també no fer tan farragosa la lectura del document.

En els últims punts s'analitzaran els resultats obtinguts, es mostraran una sèrie de probes d'execució, millores possibles que es podrien agregar al programari, un manual d'usuari i la bibliografia. Finalment hi ha un punt que engloba les conclusions del treball realitzat.

2.- Recollida inicial d'informació

Després d'estar segur que aquest era el projecte que havia de fer, per les raons anteriorment comentades, primer, vaig començar a recopil.lar la informació necessària, per realitzar un programari amb les característiques adients i, per això, havia de comparar els diferents programaris que hi havia al mercat.

Buscant per la xarxa hi ha moltes referències per crear arbres genealògics o bé per buscar familiars desconeguts, com també altre tipus d'informació molt relacionada. Del software més valorat a internet he fet un petit anàlisis dels següents...

2.1.Family tree builder

Apte per realitzar un arbre “profesional”, com per ser utilitzat per qualsevol tipus d'usuari, que vulgui formar un arbre genealògic senzill de la seva família. Permet buscar parentescs familiars per internet, així com publicar qualsevol arbre realitzat. Compta amb diferents opcions de vista, tant sobre arbres, com sobre fitxes personals. Finalment destacar que té la possibilitat d'agregar fotos personals als familiars, així com la possibilitat de veure-les incloses a l'arbre.

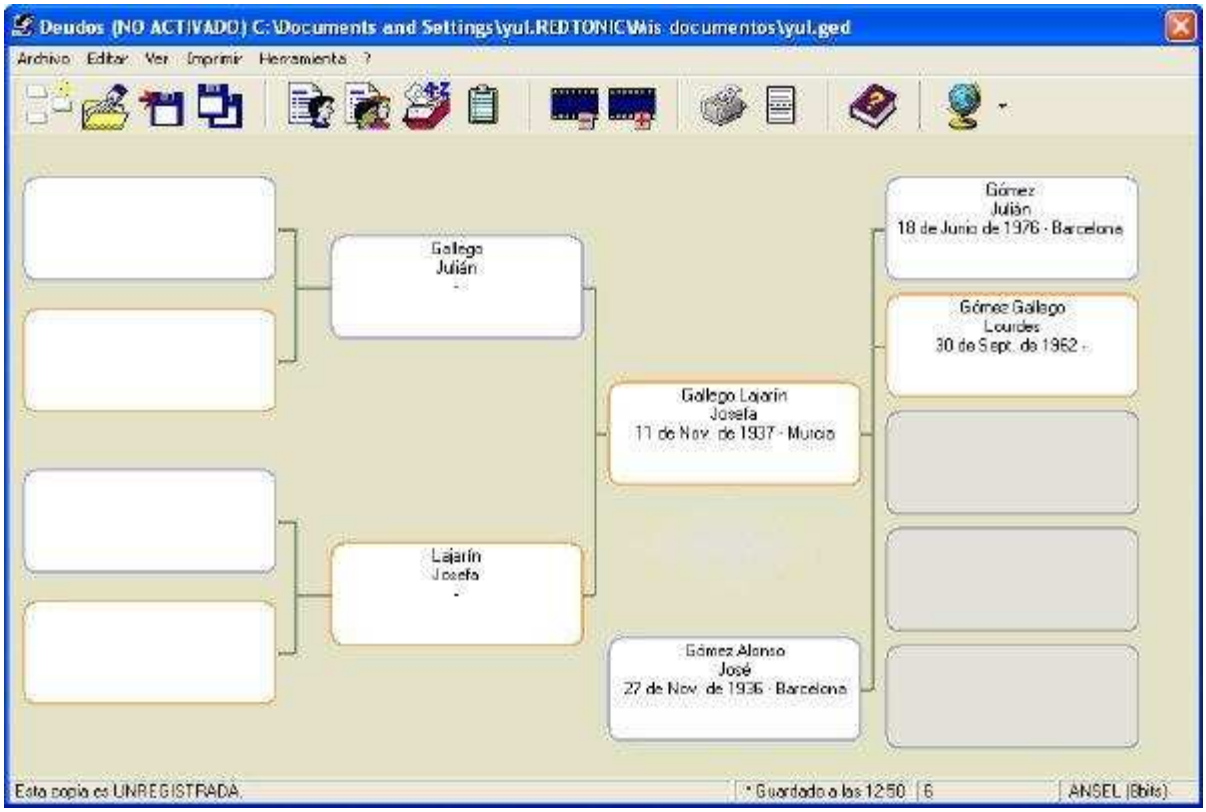
En definitiva un gran programari que a més es pot trobar gratuïtament per la xarxa. Els aspectes positius són molts, com ja he comentat, i el seu funcionament és molt intuïtiu i adaptable per diferents nivells d'usuari. Per contra tant els menús com els tutorials que es troben a la xarxa únicament es troben en anglès, fet que dificulta la seva utilització en cas de desconeixament de la llengua. Per altra part els passos que s'utilitzen per algunes accions, no són directes i a vegades és difícil trobar allò que es vol fer (un exemple és alhora d'intentar afegir fotografíes als familiars). La seva pàgna web és: http://www.myheritage.es/family-tree-builder

Per tant alhora d'implementar el projecte he d'intentar que sigui una aplicació que pugui utilitzar qualsevol usuari, tingui el nivell que tingui. Intentar que es puguin realitzar les operacions que aquesta aplicació genera, per amb un altre estil. I que alhora de gestionar els arxius quedi totalment clar e intuïtiu la forma de fer-ho.

2.2. Deudos 5

A més de les caracterísitiques de Family tree builder, la seva principal virtud és que disposa d'un arbre interactiu de molt fàcil navegació. Un altre dels aspectes importants és la interacció dels usuaris amb el sistema, sigui quin sigui el seu nivell, i la facilitat de trobar la opció desitjada en qualsevol moment.

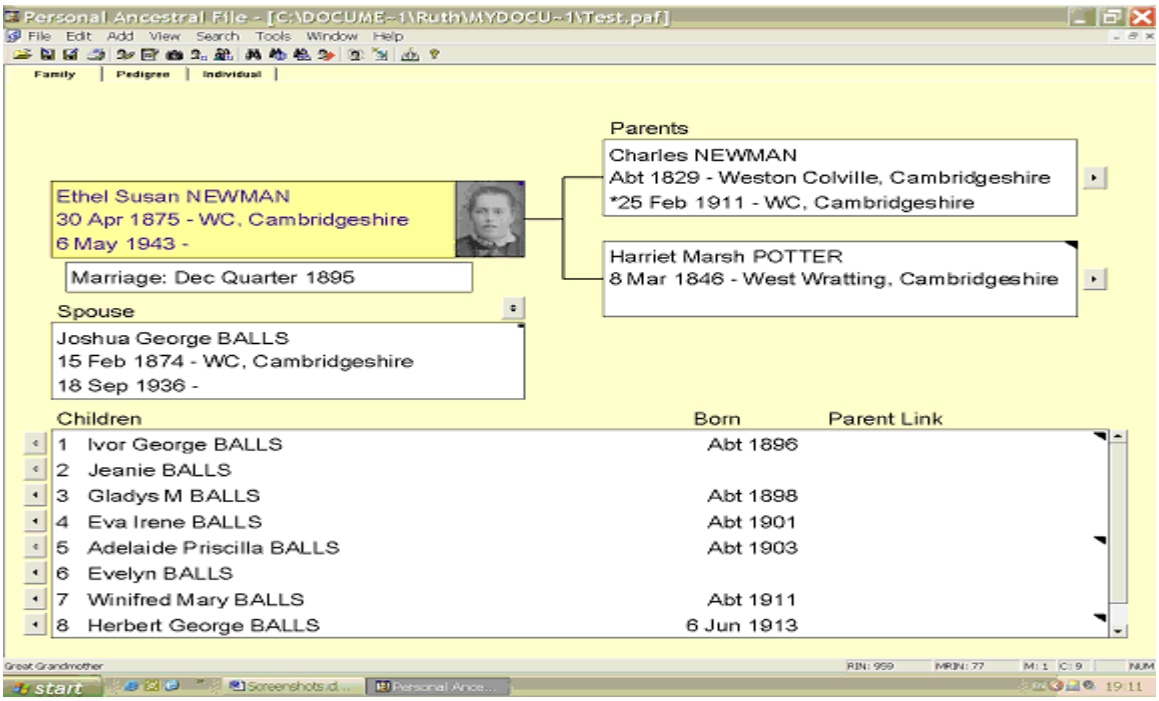
Tot i que la interacció amb el programa és molt destacable, l'entorn de treball a vegades queda sobrecarregat i no permet diferenciar la informació important de la resta. Les cerques estan molt limitades, l'apartat web és inexistent així com certes opcions, alhora d'insertar o buscar arxius és inexistent. Per tant, crec que pel desenvolupament del projecte em fixaré, pel que fa a aquest programari, en la interacció (tot i que seguiré un altre estil), i intentaré desenvolupar parts, que aquí s'han deixat de banda. La direcció electrònica és: http://www.deudos.com/



2.3 Personal Ancestral File

Disposa de moltíssims camps editables pels membres de la família. Aquest contingut es forma de diverses formes a l'arbre i segons les preferències de cada usuari. Les cerques de que disposa son moltes e interessants.

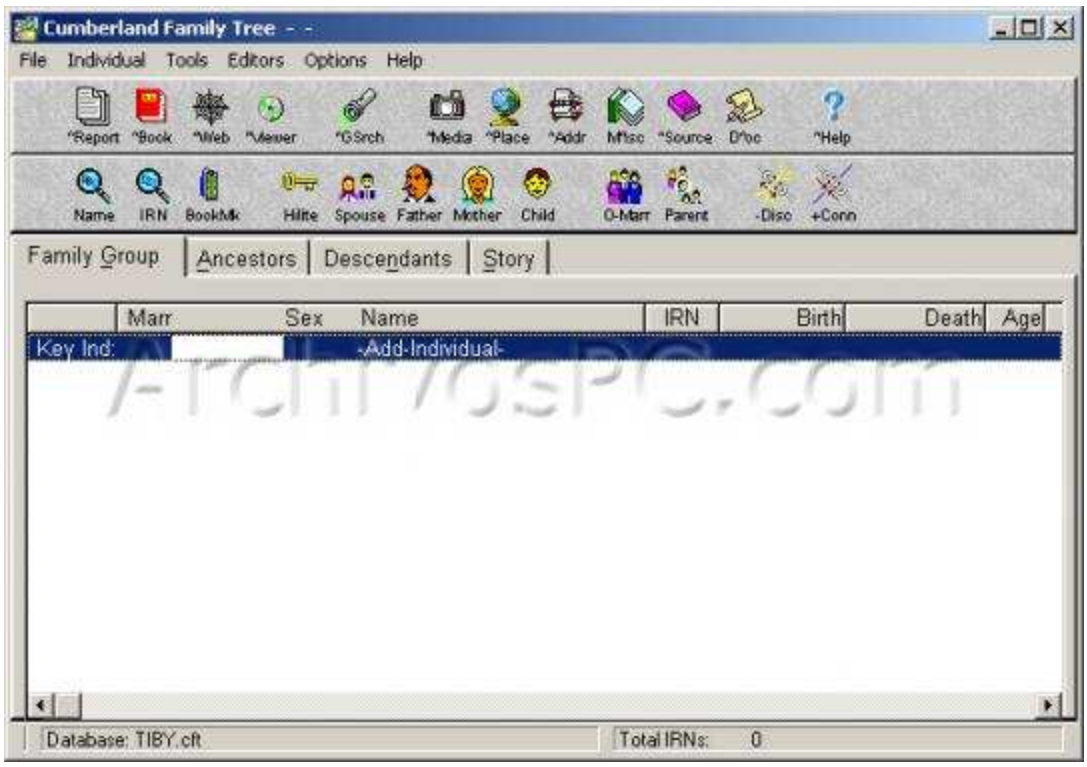
Funciona correctament tot i que no hi ha cap aspecte que destaqui sobre els altres. Un dels problemes que presenta és que l'idioma de l'última versió és l'anglès. Un altre dels errors és alhora de fer un llibre de família. La interacció en aquest punt, és una mica fluixa; a més, a cada familiar li apareixen tots els descendents entre cometes. Això amb pocs registres no comporta problemes, pero quan hi ha moltes generacions les dades queden una mica confuses. La direcció electrònica del programari és http://www.familysearch.org/eng/paf/



2.4.Cumberland Family Tree

Permet introduir infinitat d'events en la vida de les persones així com la possibilitat d'afegir a cada membre, múltiples i repetits esdeveniments; diversos casaments, canvis de residència.

Tot i que permet entrar amb facilitat a familiars, així com molta altra informació, no disposa de moltes de les opcions que altres programaris ofereixen. A més la informació dels familiars no queda ben reflexada per pantalla i moltes vegades, es denota una falta de dades relacionades amb els familiars. Finalment comentar que la interacció, en general, amb l'aplicació deixa molt que desitjar, i cap aspecte innovador o destacable respecte als anteriors.



2.5.Legacy family tree

És un dels que més he probat i examinat per realitzar aquest projecte. Fàcil d'utilitzar, un dels més intuitius; amb diversitat d'opcions diferents i pràctiques. Multitud de llistats, presentacions i vistes de l'arbre. És, sens dubte, el millor programari sobre genealogia que he probat. A més de multitud d'opcions, tant les dades que es mostren com els menús, com també la facilitat d'obtenir el que es desitja, fan d'aquesta aplicació la millor, segons el meu punt de vista. Tot i això intentaré que el projecte obtingut assoleixi un nivell superior, en quant a les prestacions dels arxius i la visualització de l'arbre, dos dels punts millorables.

La seva pàgina web és: http://www.legacyfamilytree.com/tipsNGSJulAug99.asp

2.6 Conclusions

Per concloure, cal dir, que gairebé tots els programes que he probat ,o bé he vist analitzats a la xarxa, realitzen les mateixes accions, encara que alguns d'una forma més clara o estètica que els altres. Tots permeten introduir dades a familiars: data de naixament, comunió, mort,... I relacionar els familiars entre si. Alguns permeten afegir informació multimèdia i altres permeten una interacció amb la xarxa; ja sigui amb la opció de buscar familiars, o bé de publicar el contingut de cada membre, o en global. Tot i que molts són per usuaris de qualsevol nivell a vegades era complicat seleccionar la opció desitjada, o bé el camí per accedir-hi era més llarg del desitjable; aquest és un dels punts en què he posat més ènfasis alhora de crear la meva aplicació.

Per tant el programari que he de realitzar ha de tenir bona part de les especificacions que els altres ja tenen; intentant agafar el millor de cadascun, com també afegint noves idees i maneres de fer, que finalment acabin sent una opció, si més no diferent a l'actual.

3.- Requisitis funcionals

3.1 Requisits funcionals de l'aplicació

Per tant, la meva aplicació hauria de desenvolupar, una plataforma que permeti guardar dades familiars de cara a poder obtenir un arbre genealògic detallat, que podrà ser visualitzat i modificat via web.

3.2 Requisits funcionals del sistema

S'han identificat les següents funcions que hauria de realitzar el sistema:

Generals:

- Gestionar les dades (altes, baixes, modificacions) via web o a través de l'aplicació i llavors actualitzar-ho via web.
- Actualitzar dades: si es fan modificacions en el programa pero no s'actualitzen a la web, les dades només les pot veure l'usuari que les ha modificat fins que efectua aquesta operacio.
- A partir de les dades entrades generar de forma automàtica un arbre.
- Crear diferents vistes segons diferents criteris (definits prèviament).
- Cercar (ja sigui familiars, arxius,...) que hi hagi a la base de dades.
- Crear copia seguretat.
- Possibilitat de gestionar multitutud de famílies, dins el mateix sistema.
- Possibilitat de crear diferents usuaris per cada una de les famílies.
- Gestió de dades públiques i privades.

3.3 Decisions sobre llenguatge i programari

Els aspectes que es tractaran en aquesta apartat quedaran explicats detalladament en punts posteriors, però és important que, després d'explicar el que es vol obtenir és explicar com ho vull fer. El primer és el llenguatge escollit. Entre les diferents opcions que hi havia, vaig decidir ràpidament amb el java, degut a la facilitat que dóna per realitzar aplicacions amb interfícies gràfiques estètiques i més fàcils d'utilitzar per usuaris de baix nivell. En aquesta aplicació la rapidesa alhora d'executar codi, no serà tan important com l'apartat estètic i, per tant, la millor opció era java.

Per desenvolupar aquest llenguatge vaig decidir instal.lar la plataforma de desenvolupament Eclipse, que és sens dubte la millor opció, dins el programari lliure. Vaig actualitzar bona part dels components i a més vaig instal.lar algunes opcions extres per aconseguir un entorn més còmode i pràctic, a més d'alguns extres imprescindibles. Alguns d'aquests van ser: SqlExplorer, MySqlConnector, Swt i Visual Editor Project(que perment un entorn gràfic molt més interactiu).

La metodologia usada serà l'**extreme programming** (XP) , també programació extrema, que es diferencia de les metodologies tradicionals, principalment, per que es posa més enfasis en l'adaptabilitat que en la previsibilitat. Els defensors d'aquesta tècnica creuen que ser capaós d'adaptar-se als canvis que apareixen durant la vida del projecte, en comptes de tenir-los previstos des del començament, és una aproximació millor i molt més realista.

Tot i que aquesta metodologia és la que més s'assembla a la utilitzada durant l'el.laboració del projecte, hi han hagut certes variacions que no permeten assegurar que sigui un XP tradicional. Per una banda, només una persona ha generat el programari. Tot i que per una altra banda, els passos que s'han anat donant durant la creació de l'apliació han sigut curts i probats. Per tant quan s'aconseguia generar el codi per donar d'alta un nou familiar, es probava el resultat i en cas que el resultat no fos correcte es modificava el codi fins a aconseguir-ho. Tot i que un cop obtingut es modificava si es trobava una opció millor o més eficient. Aquest procés s'ha realitzat amb totes les parts del projecte i és, precisament, una de les bases més importants de la programació extrema.

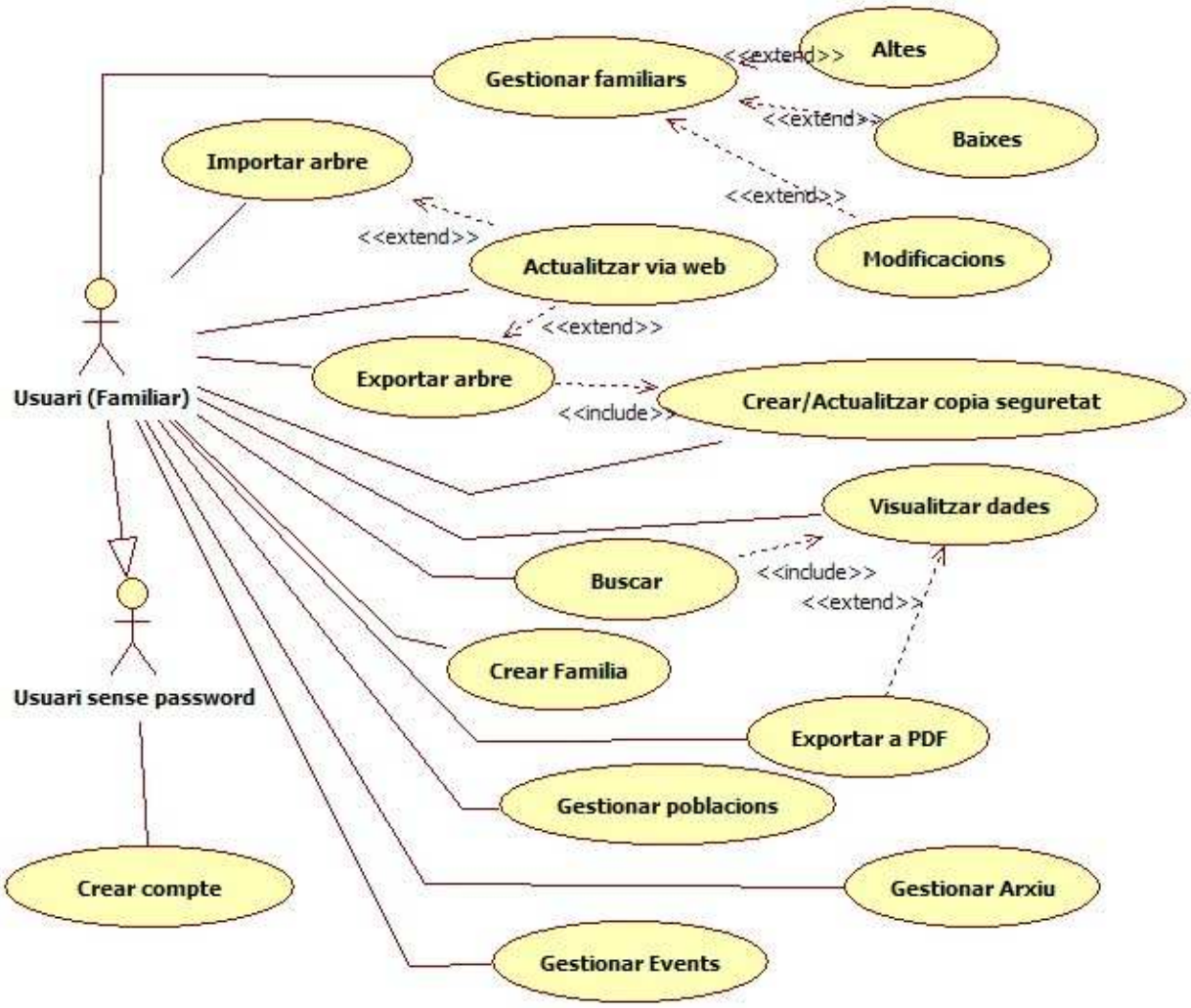
Pel que fa el codi alhora d'escollir entre diferents opcions sempre escolliré el programari lliure. Per tant, per exemple, entre MSOffice i OpenOffice, sense valorar un i altre, l'elecció serà OpenOffice, basant-me en la postura de seleccionar el programari lliure.

Per acabar amb aquest punt s'hauria de parlar de la seguretat i privacitat de les dades que els usuaris poden introduir en l'aplicació. Tot i que el programari, permet la possibilitat d'introduir contrasenyes a cada una de les famílies que es generin, així com la base de dades també està protegida sota contrasenya, aquest és un software per ús personal i sense finalitat comercial. Es tracta d'un projecte que pugui ser utilitzat per usuaris individuals, no col.lectius ni empreses, i sense ànim lucratiu. Per tant com es redacta a la llei de protecció de dades, no hi ha la obligatorietat de complir aquesta norma i queda sota responsabilitat de l'usuari la utilització i generació de dades, que puguin acabar sent visualitzades per tercers.

4. Model d'anàlisis del sistema

Abans de començar a implementar el codi, és important realitzar el disseny uml, per així tenia una idea dels requeriments que haurà de complir l'aplicació; com també dels possibles problemes i solucions que hem pugui trobar.

4.1.Diagrama de casos d'ús



Cada un dels casos d'ús anteriors quedaran reflexats en les interfícies gràfiques creades.

Després d'analitzar els requisits potencials del nou sistema he creat el següent

digarama de casos d'ús. Com es pot observar un usuari pot o no tenir contrasenya. En cas que en tingui significa que la família, a la que pertany, està protegida i, per tant, per poder-hi accedir la necessita. Si no en té, significa que la família no té accés restringit i que qualsevol persona, familiar o no, hi té accés.

A destacar que l'aplicació permet la creació de múltiples famílies en un mateix sistema; la creació d'infinitats de familiars agrupats en alguna de les famílies; la creació de poblacions, no definides inicialment, com també l'associació de familiars amb axius multimèdia, que poden ser visualitzats per qualsevol dels familiars d'una mateixa família. Finalment, també contempla l'opció d'associar events a familiars, com de crear-ne de nous, per afegir, així, tota la informació que es vulgui d'un familiar.

4.2. Fitxes de casos d'ús

Després d'examinar les necessitats de l'aplicació i realitzat el diagrama de casos d'ús he desenvolupat les següents fitxes de casos d'ús.

Cas d'ús : Gestionar familiars	
Descripció	Permet donar d'alta, modificar o eliminar algun familiar al sistema
Actor	Usuari
Precondició	L'usuari té nom i contrasenya
Flux principal	Depenent de la operació
Flux alternatiu	---
Postcondició	S'ha realitzat una alta, baixa o modificació amb èxit
Comentaris	Mirar el refinament en fitxes alta, baixa i modifcació familiar. El seu password també serà la seva clau de la base de dades

Cas d'ús : Importar arbre	
Descripció	Permet carregar al sistema un arbre.
Actor	Usuari
Precondició	L'usuari té nom i contrasenya
Flux principal	1- Anotar adreça correcta on es troba el fitxer 2- Carregar al programa l'arbre
Flux alternatiu	Si el tipus d'arbre no és compatible amb GEDCOM, l'arbre no serà carregat amb èxit.
Postcondició	L'arbre s'ha carregat i és utilitzable

Cas d'ús : Alta familiar	
Descripció	Permet donar d'alta algun familiar al sistema
Actor	Usuari
Precondició	L'usuari té nom i contrasenya
Flux principal	1- Introduir DNI 2- Introduir dades personals 3- Introduir relacions familiars/ descendència 4- Afegir arxius multimèdia 5- Formalitzar alta 6- Modificar arbre
Flux alternatiu	Si el dni és incorrecte o reptetit es tornaria a demanar. Si alguna de les dades verificables és incorrecte es demanaria que fos modificada. Si el tipus d'arxiu no és compatible es mostraria un missatge per pantalla.
Postcondició	El familiar s'ha donat d'alta
Comentaris	---

Cas d'ús : Modificar familiar	
Descripció	Permet modificar algun familiar del sistema
Actor	Usuari
Precondició	L'usuari té nom i contrasenya
Flux principal	1- Introduir DNI 2- Modificar dades personals 3- Modificar relacions familiars/ descendència 4- Afegir arxius multimèdia 5- Formalitzar modificacio 6- Modificar arbre
Flux alternatiu	Si el dni és incorrecte o reptetit es tornaria a demanar. Si alguna de les dades verificables és incorrecte es demanaria que fos modificada. Si el tipus d'arxiu no és compatible es mostraria un missatge per pantalla.
Postcondició	El familiar s'ha modificat
Comentaris	---

Cas d’ús : Baixa familiar	
Descripció	Permet eliminar algun familiar del sistema
Actor	Usuari
Precondició	L'usuari té nom i contrasenya
Flux principal	1- Introduir DNI 2- Confirmació eliminació 3- Formalitzar eliminació 4- Borrar les depèndencies del familiar
Flux alternatiu	Si el dni és incorrecte o reptetit es tornaria a demanar.
Postcondició	El familiar, arxius i relacions del mateix s'han eliminat.
Comentaris	---

Cas d'ús : Crea Familia	
Descripció	Permet crear una nova familia
Actor	Usuari
Precondició	L'usuari té nom i password
Flux principal	1- Introduir dades de la familia 2- S'actualitza la base de dades
Flux alternatiu	Si el nom de la familia, que correspon al cognom dominant, ja existeix s'hauran d'introduir ambdos cognoms
Postcondició	A la base de dades s'haurà creat una nova familia
Comentaris	Crear families serveix en cas que es vulguin crear diversos arbres genealògics amb la mateixa interfície.

Cas d'ús : Exportar arbre	
Descripció	Permet guardar un arbre en el HDD de l'usuari.
Actor	Usuari
Precondició	L'usuari té nom i contrasenya
Flux principal	1- Introduir l'@ del HDD on es vol guardar l'arbre 2- Es guarda l'arbre 3- S'actualitza la còpia de seguretat
Flux alternatiu	---
Postcondició	L'arbre queda guardat al HDD i la còpia de seguretat actualitzada.
Comentaris	---

Cas d'ús : Exportar a PDF	
Descripció	S'exporta l'arbre a format PDF perquè sigui fàcilment imprimible
Actor	Usuari
Precondició	L'usuari té nom i contrasenya
Flux principal	S'exporta l'arbre a format PDF perquè sigui fàcilment imprimible
Flux alternatiu	---
Postcondició	L'arbre es guarda en un arxiu pdf
Comentaris	---

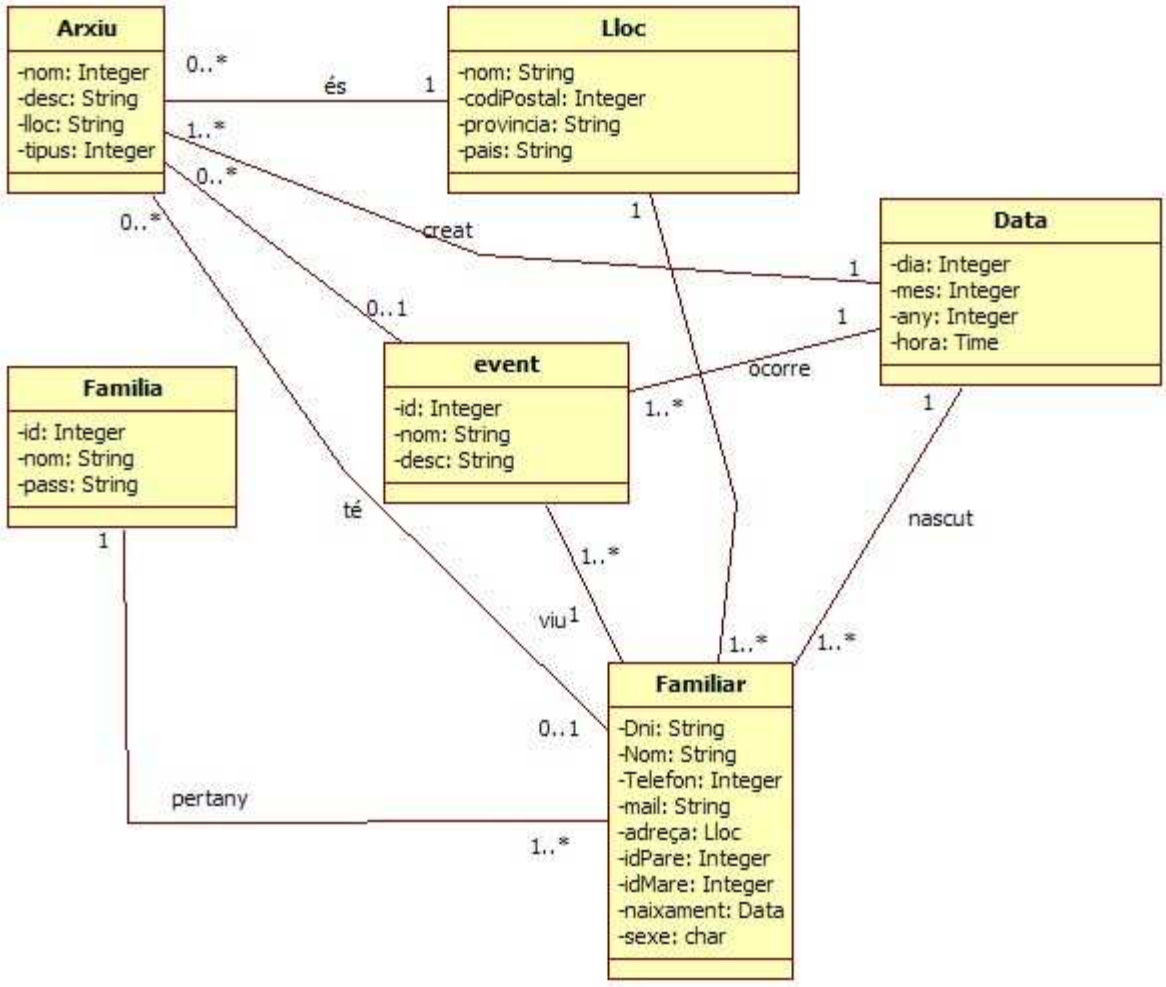
Cas d'ús : Crear/Actualitzar còpia de seguretat	
Descripció	Es crea o s'actualitza una còpia de seguretat en el sistema
Actor	Usuari
Precondició	L'usuari té nom i contrasenya
Flux principal	Crear: 1- Es crea un nou fitxer del tipus arbre GEDCOM 2- Es guarda l'arbre actual 3- S'exporta al fitxer predefinit pel programa Actualizar: 1- Es guarda l'arbre actual 2- S'actualitza el fitxer predefinit pel programa
Flux alternatiu	---

Cas d'ús : Crear compte d'usuari	
Descripció	S'exporta l'arbre a format PDF perquè sigui fàcilment imprimible
Actor	Usuari sense password
Precondició	---
Flux principal	1- Introduir DNI 2- Introduir dades personals 3- Introduir relacions familiars/ descendència 4- Afegir arxius multimèdia 5- Formalitzar alta 6- Assignar clau 7- Crear fitxa personal 8- Afegir a l'arbre
Flux alternatiu	Si el dni és incorrecte o reptetit es tornaria a demanar. Si alguna de les dades verificables és incorrecte es demanaria que fos modificada. Si el tipus d'arxiu no és compatible es mostraria un missatge per pantalla.
Postcondició	L'usuari és registrat i per tant podrà utilitzar totes les opcions del programa i a més s'haurà introduït la seva fitxa de familiar.
Comentaris	---

Cas d'ús : Buscar	
Descripció	Permet buscar diversa informació de l'arbre
Actor	Usuari
Precondició	L'usuari té nom i contrasenya
Flux principal	1- Introduir el que es vol buscar 2- Introduir tipus (familiar, fotos, so,...) 3- Introduir condició 4- Realitzar la búsqueda a la bdd 5- Mostrar Informació
Flux alternatiu	Si el que es busca no existeix a l'arbre, es torna a la pagina de cerca

Cas d'ús : Actualitzar via web	
Descripció	Permet carrgar/descarregar l'arbre que hi ha penjat a la web
Actor	Usuari
Precondició	L'usuari té nom i contrasenya
Flux principal	Carregar: 1- Introduir l'@ del HDD on es troba l'arbre que es vol actualitzar 2- Es comparen ambdos arbres 2,1- Si l'arbre és diferent: 2,1,1- es guarda l'actual arbre com una còpia (per si els canvis realitzats no son correctes) 2,1,2- es carrega la nova informació i s'elimina aquella que s'ha eliminat i es guarda com l'arbre actual de la web. 2,2- Si no és diferent no es fa res Descarregar: 1- S'introdueix l'@ del HDD on es vol descarregar l'arbre
Flux alternatiu	---
Postcondició	Carregar: l'arbre es carrega a la web i es guarda l'arbre anterior com a còpia per si s'hi vol tornar. Descarregar: l'arbre es decarrega al disc dur de l'usuari
Comentaris	---

4.3.Diagrama de classes



En total hi ha sis classes. La classe *familiar* guardarà tota la informació que hi pugi estar relacionada (dni, telèfon,...) a més d'una nova variable id ja que totes les que apareixen en el diagrama, son variables que poden estar repetides. Aquesta classe és la més important, no només pel volum de dades que conté, sinó perque relaciona totes les altres classes.

Una altra classe és *familia*, realcionada únicament amb familiar, permet agrupar els familiars a partir de dues variables nom (nom de la familia) e id (identificador). Finalment aquesta classe està formada per un altre atribut pass; que perment assignar una clau a cada família, tot i que és opcional.

La classe *arxiu*, també relacionada amb Familiar, permet guardar diversos arxius, de diferents tipus. Cal dir que un arxiu, com per exemple una fotografia, pot ser assignada a diferents familiars, pero que a la bdd quedarà guardada com a arxius de diferents familiars; això permet que la descripció d'un mateix arxiu físic, pugui ser interpretada, segons diversos familiar, de forma diferent. Aquest fet s'ha tingut en compte i durant la documentació s'explicarà amb deteniment.

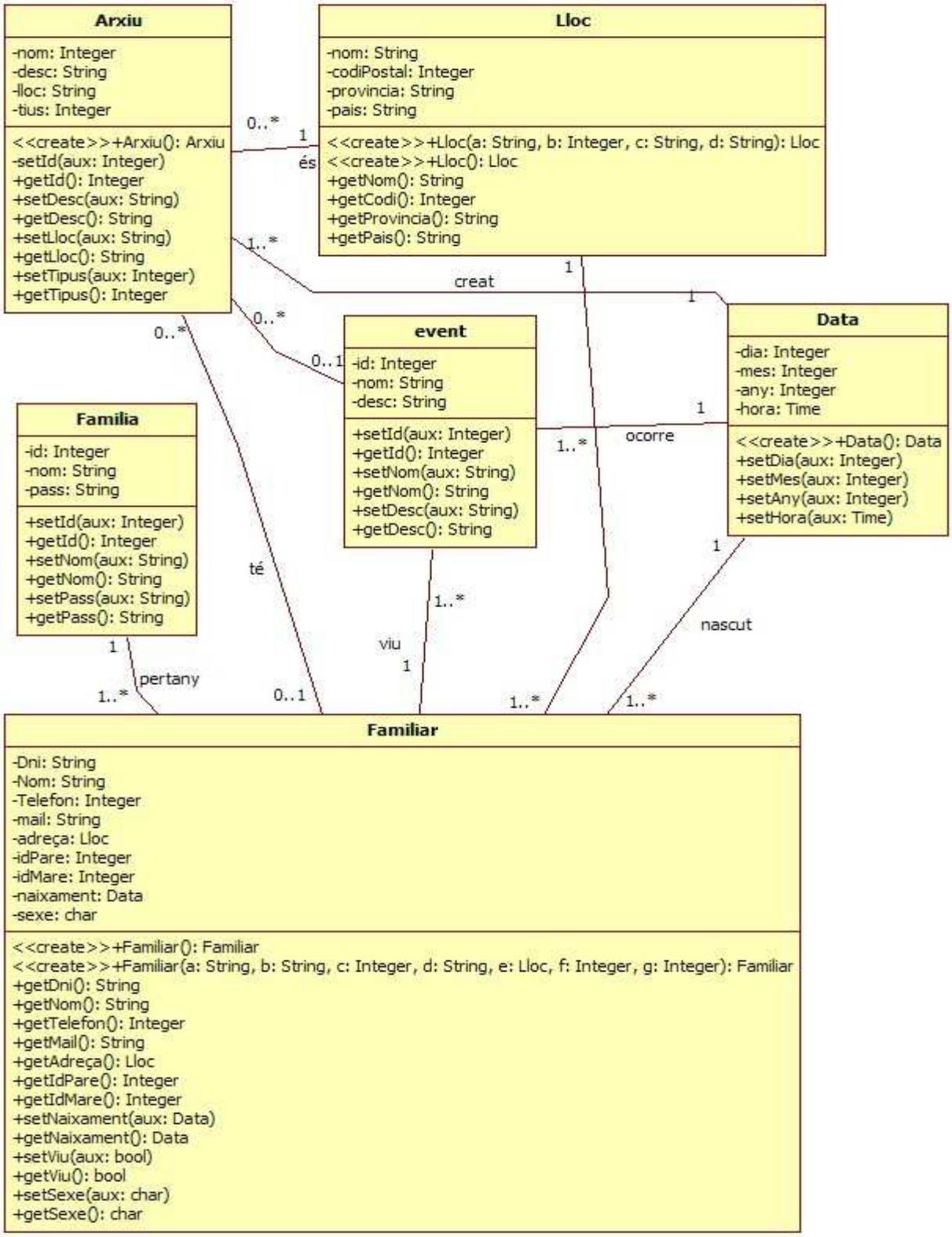
Cal dir que, a part que l'arxiu tindrà una data i una ubicació, podrà ser d'un familiar o bé d'un event (en aquest últim cas també acabaria pertanyent a un familiar, ja que un event és d'un familiar). Per tant l'aplicació es permetrà associar fotos a familiars: una foto personal, que serà mostrada en totes les interfícies del membre; a més de moltes altres que podran ser visualitzades com un àlbum. Les fotografíes associades als events també tindran aquesta opció.

La classe *lloc* igual que la classe *data*, són classes que hem permetran realitzar operacions de forma més senzilla i automatitzada. Totes les altres les relacionen, ja que és important que totes les dades puguin ser ubicades física i temporalment. Alhora de realitzar cerques i també alhora de mostrar, s'ha tingut en compte, permetent multitud de vistes diferents i de seleccionar els llocs i les dates dessitjades.

Finalment la classe *event*, relacionada amb Familiar i event, permet afegir molta més informació de cada un dels familiars. A un familiar se li podran assignar diferents events; és a dir, es possible guardar: el naixament, el casament, el divorci,... i a més, també es dóna la possibilitat de crear nou **tipus d'event**, com els citats casament, divorci,.... i d'aquesta forma, fer encara més interactiva l'aplicació.

4.4 Refinament dels diagrames de classe

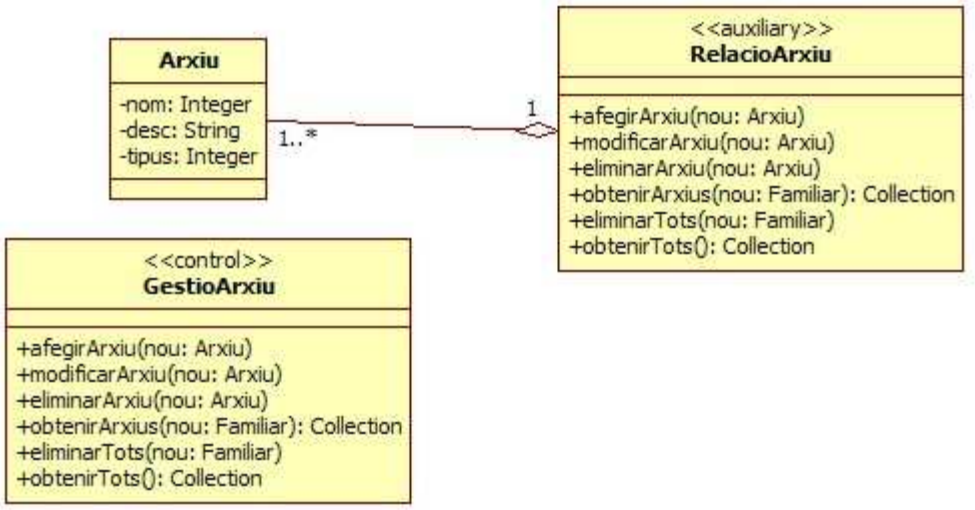
4.4.1 Refinament diagrama de classes



Primer he presentat el diagrama de classes de forma més senzilla (sense els mètodes), perque fos més clar a l'inici, i ara mostro el diagrama amb tots els mètodes que implementaré en el

projecte. El diagrama, va quedar explicat anteriorment, i els mètodes ho seran més endevant.

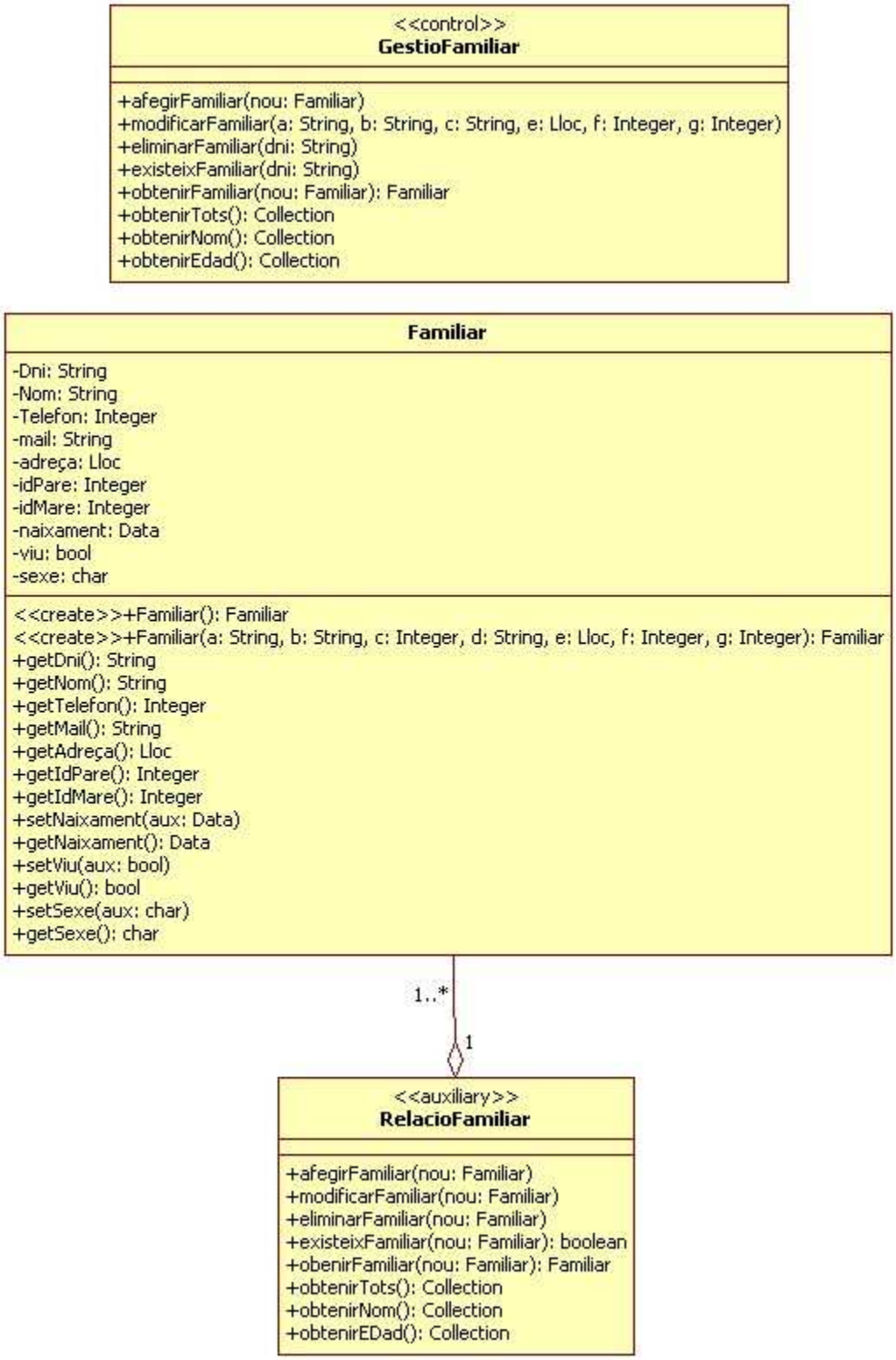
4.4.2 Refinament de la classe Arxiu



Aquesta classe és una de les més importants del projecte, per això és important explicar-la amb més deteniment. Té relacions amb pràcticament la totalitat de les altres classes que coexisteixen en el sistema i, per tant, a més dels atributs naturals que la defineixen (nom, descripció de l'arxiu i tipus), acabarà adoptant atributs de moltes de les altres. El fet que sigui una classe important, deguta a la multiplicitat de relacions, acaba comportant un control més exhaustiu, que les altres.

Els mètodes d'aquesta classe permeten afegir, modificar, eliminar i obtenir la informació dels arxius guardats a la bdd. A més per poder gestionar les consultes, com també alguns mètodes d'eliminació massius, és necessari la implementació dels mètodes **eliminarTots**, que elimina tots els arxius d'un familir, **obtenirArxius** (que obté tots els arxius que fan referència a un mateix familiar) o bé **obtenirTots**, que obté tots els arxius de la bdd.

4.4.3. Refinament de la classe Familiar

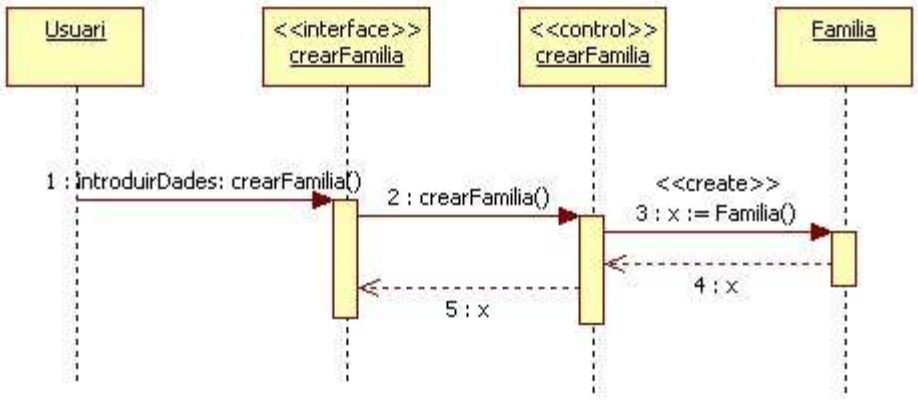


Aquesta classe és la més important del sistema, ja que principalment el que es desitja amb l'aplicació és la creació, modificació i obtenció de dades dels familiars que pertanyen a una mateixa família. També és la classe amb més atributs. Aquest fet és degut a que he intentat que es pugui guardar tota la informació possible, per aconseguir, a més de la mateixa informació, diversitat de cerques que permetin a l'usuari interactuar de forma més activa amb el programari.

Els mètodes que he implementat permeten afegir nous familiars, modficiar-los o bé eliminar-los. Els altres mètodes existents ajuden a alguna de les cerques que l'usuari pot utilitzar. Per a aquesta classe han sigut necessaris molts controls, degut a la multitud de paràmetres que engloba, així com també la possibilitat de no emplenar-ne molts.

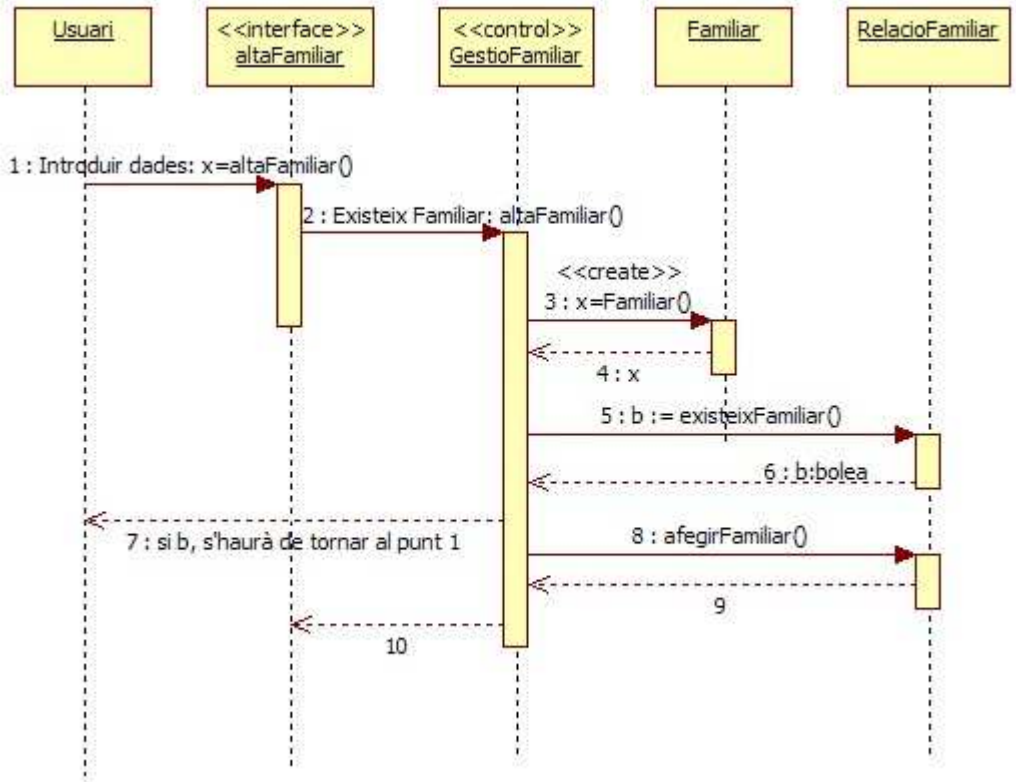
4.5. Diagrames d'activitat

4.5.1. Crear família



Serà el pas obligatori alhora de poder iniciar l'aplicació. L'usuari escollirà aquesta opció e introduirà el nom de la família així com una clau, en cas que es vulgui. Com s'ha comentat anteriorment, si no es desitja la família pot quedar desprotegida, poguent ser visualitzada per qualsevol usuari. El nom de la família no serà el camp clau, per tant dins la bdd de dades podrà haver-hi múltiples famílies amb el mateix, perquè quan es crea aquesta classe se li associa un identificador automàticament.

4.5.2. Alta familiar



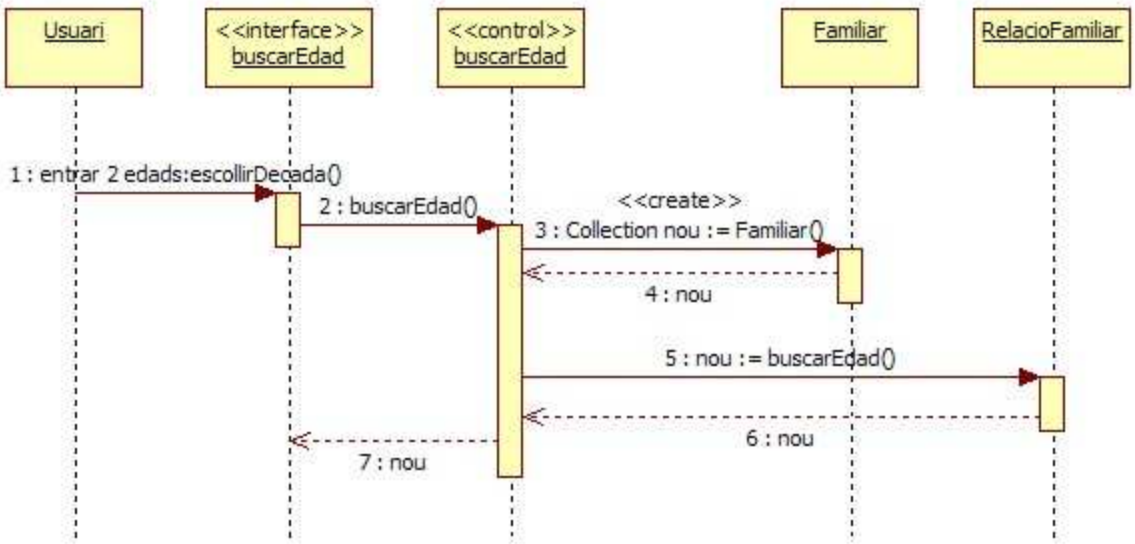
He realitzat una programació per capes, que a més d'avantatges alhora de modificacions del codi, permeten que l'usuari no hi tingui un accés directe. Per tant l'usuari, per mitjà de la pantalla (interfície) escull i emplena les dades del familiar, a donar d'alta, aquestes dades arriben a la classe GestioFamiliar, que crea un familiar i comproba, per mitjà de la classe RealacioFamiliar que no existeixi cap amb les dades introduïdes.

El nom no es pot comprobar que sigui repetit, ja que en molts casos ho serà (el primer fill normalment rep el mateix nom que el pare); el dni en ocasions també és repetible, encara que és molt improbable; ni les dates ni els llocs,.... És a dir, gairebé totes les dades són fàcilment repetibles; per tant, a la pràctica, es valida que les dades introduïdes siguin correctes i s'assigna un identificador únic al familiar.

Destacar que la majoria de camps no han de ser emplenats de forma obligatòria, tot i que en

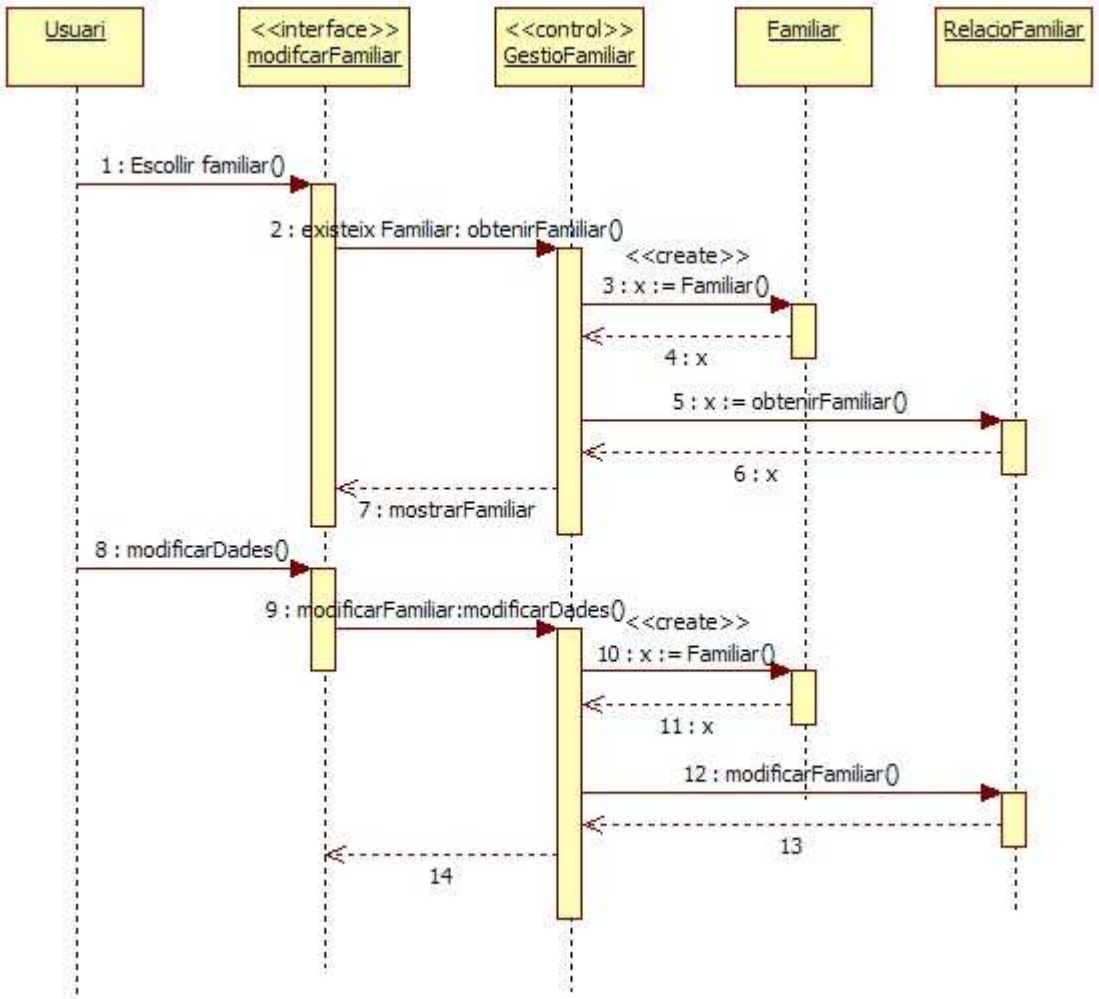
cas afirmatiu es comproba que sigui correcte. Ho he preferit d'aquesta forma ja que molta informació a vegades no és recordada i tampoc no repercuteix alhora d'analitzar els resultats.

4.5.3. Buscar familiars entre dates



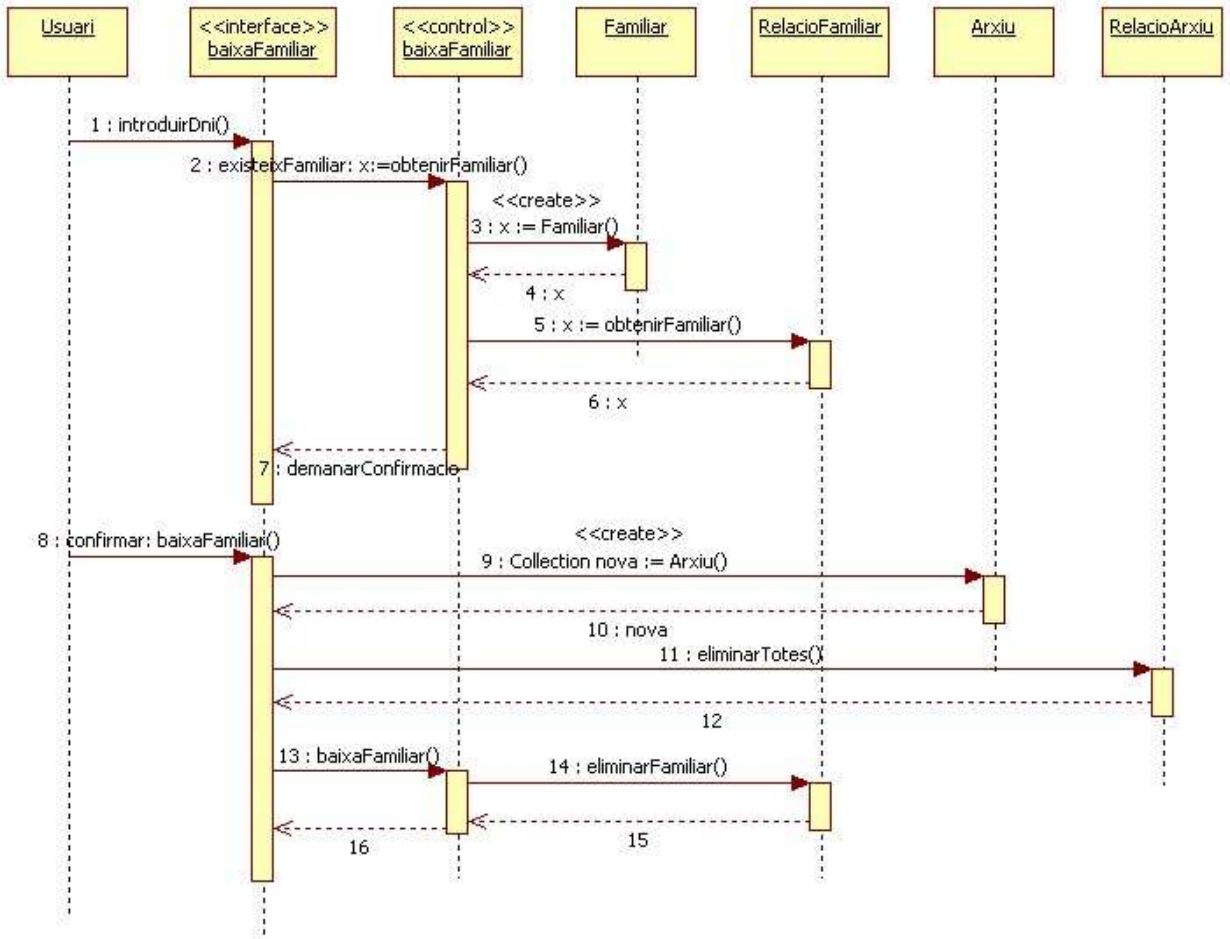
Una de les possibles cerques és la de busar familiars d'una dècada. Aquesta interfície permet a l'usuari veure un llistat de tots els familiars nascuts entre dues dates (1900-1920, 1920-1940,....). Un cop llistats els familiars, cap la possibilitat mostrar detalladament cada un d'ells.

4.5.4. Modificar familiar



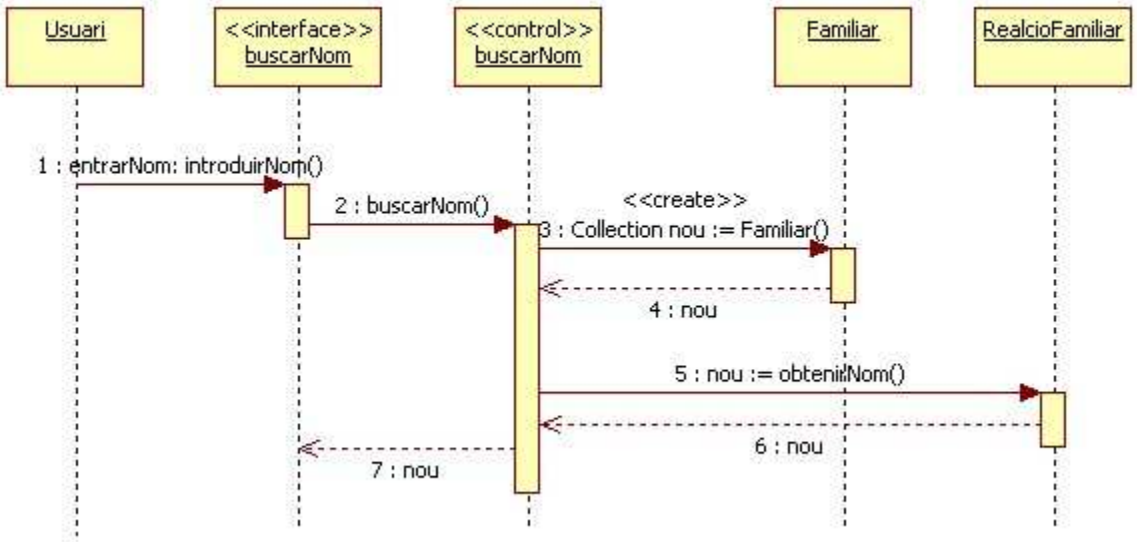
Per modificar un familiar primer es selecciona d'una llista (a l'aplicació hi haurà diferents possibilitats de veure-la: ja sigui buscant per nom; mostrant tots els familiars de la bdd; per edad;...) i un cop escollit es mostrarà per pantalla les seves dades i, si es desitja, podran ser modificades. Com la clau primària serà, com he comentat abans, creada de forma automàtica totes les dades son modificables (nom,dni,pare,mare,...).

4.5.5. Baixa familiar



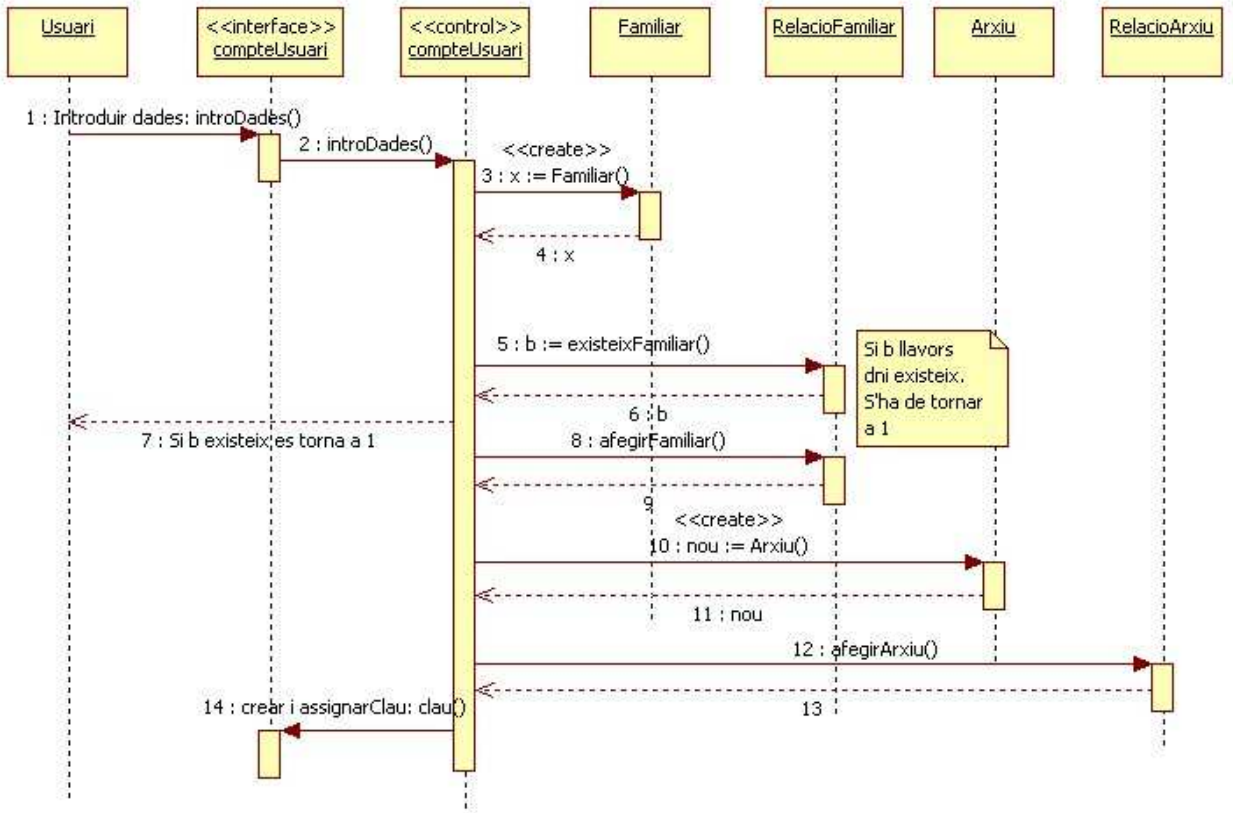
Com alhora de modificar, primer es selecciona un familiar d'una llista i posteriorment es mostra per pantalla. En cas que es vulgui eliminar es mostra una missatge per pantalla demanant la confirmació a l'usuari. És important recalcar, l'aplicació també ho fa, que si es continua amb l'eliminació del familiar es borrarà tant ell/a com tota la informació que el relaciona. És a dir tant els arxius, com els events com els familiars que tinguin la seva referència quedaran eliminats. En cas que un fill/a tingui la referència al familiar, aquesta quedarà eliminada, evidentment no tot el familiar.

4.5.6. Buscar per nom



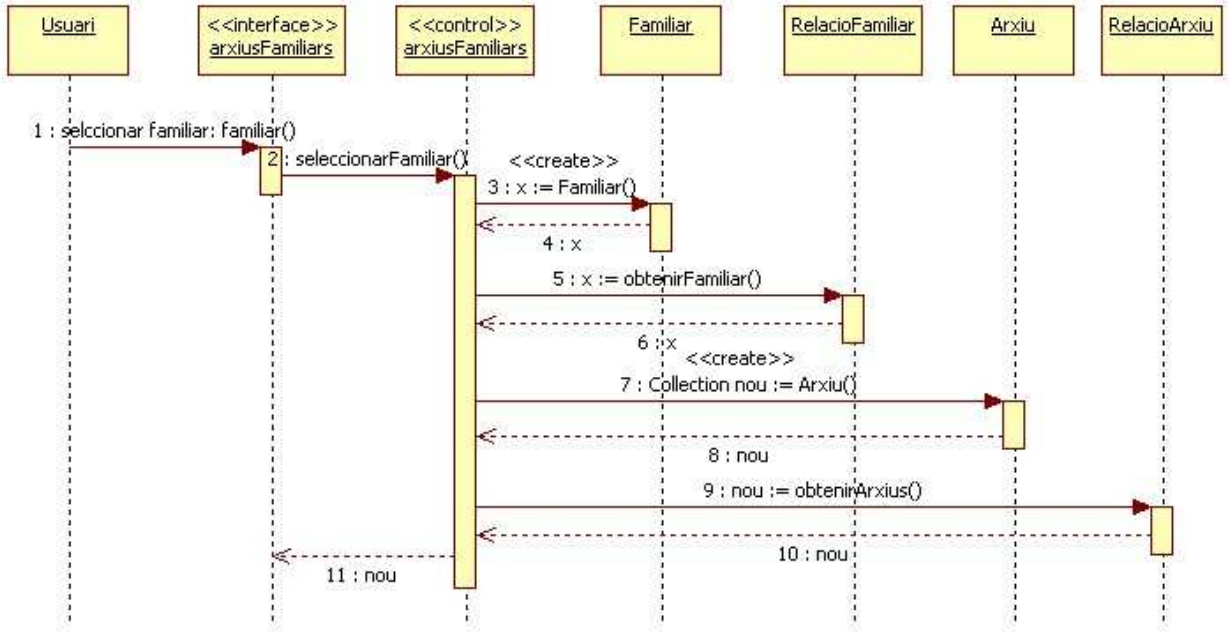
Aquesta cerca permet trobar tots els familiars amb el nom introduït per l'usuari. No només es busca el nom complet sinó que es busquen tots els que l'inici del nom coincideixi exactament amb el que l'usuari introdueixi. Per expemple si hi ha un familiar amb nom Maria Hernandez Jimeno, en cas que es busqui Maria, es llistaran totes les Maria que hi hagi a la bdd; pero si es busca Mari es llistaran les Maria pero també les Maricarmen. Aquest mètode també permet introduir els cognoms (buscar Maria Hern), amb el resultat Maria Hernandez.

4.5.7. Compte d'usuari



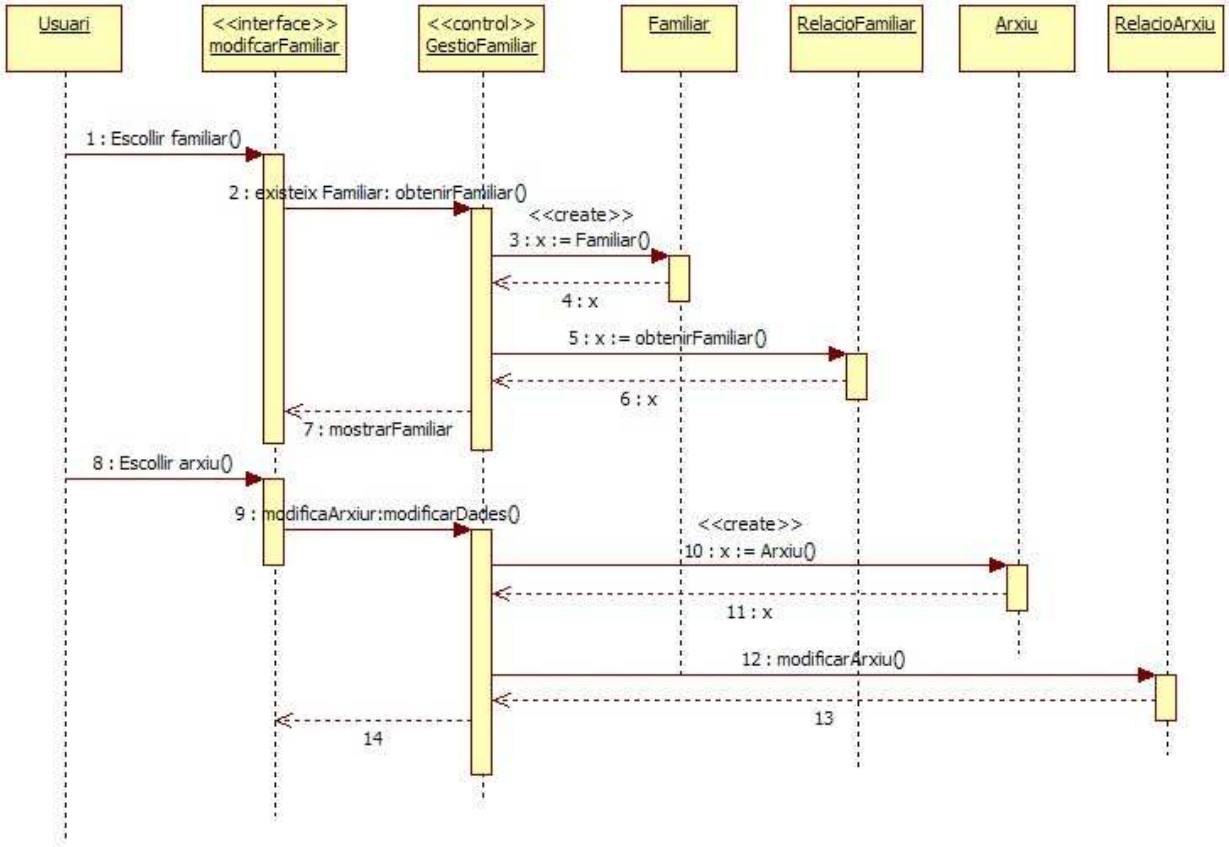
En realitat crear un compte d'usuari contempla afegir-se com a familiar, d'una familia, i obtenir la clau de la família. Primer es dóna d'alta a la base de dades, llavors si l'usuari ho desitja es poden afegir els arxius (especialment la foto personal, que anirà apareixent a totes les pantalles) i finalment, després d'actualitzar la bdd, se li assignarà al familiar la clau, sempre que la família en tingui; ja que la clau que se li dóna al familiar és la seva família.

4.5.8. Afegir arxius a familiar



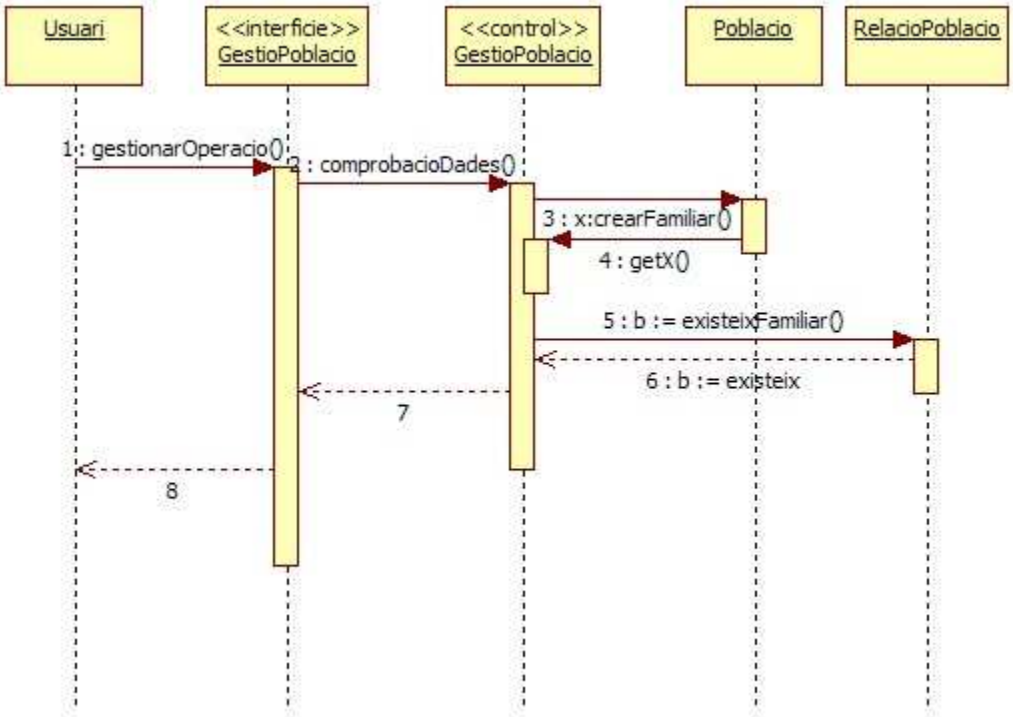
Per afegir un arxiu a un familiar o un event, primer cal escollir-lo d'una llista. Un cop triat, es mostra per pantalla i llavors l'usuari ha d'escollir quin tipus d'arxiu vol afegir i insertar la ruta. Quan s'afegeix un arxiu es pot afegir una data i un lloc a més de poder-li associar un familiar o un event. Hi ha moltes possibilitats, i es pot arribar a la mateixa interfície de moltes maneres, fet que ha comportat algunes comprobacions i també una estructura de dades diferent a la inicialment pensada. (Aquesta última part, quedarà ben explicada més endevant, quan documenti les parts més destacables del codi).

4.5.9. Modificar arxiu...



Alhora de modificar un arxiu, els passos a seguir son pràcticament els mateixos que quan se n'afegeix un. Primer es selecciona el menu arxiu, o bé es selecciona un familiar o un arxiu i, un cop seleccionat, llavors es selecciona l'arxiu que es vol modificar i es modifiquen els valors (descripció, lloc, data, event i familiar). Aquest diagrama mostra com es selecciona el familiar, tot i que també existeix altres vies, que son a partir de seleccionar un event, o directament desde el menu arxiu.

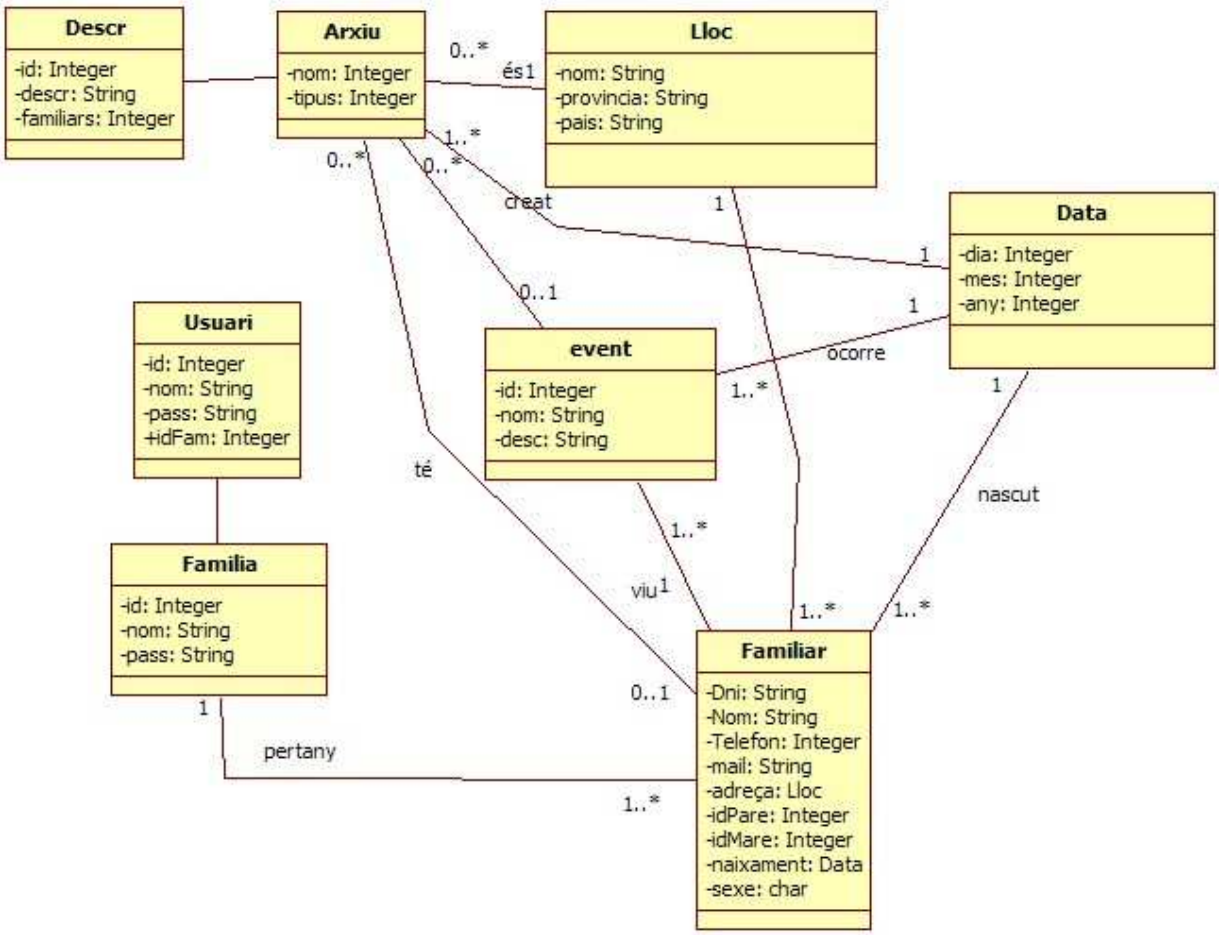
4.5.10 GestionarPoblacions



El cas d'ús gestionarPoblacions engloba tres accions bàsiques (alta, modificació i baixa de les poblacions). El diagrama que es mostra únicament engloba els primers passos que es realitza alguna d'aquestes accions. El següent pas és procedir amb l'alta, la baixa o bé la modificació. Cada procediment que s'executa en el dibuix anterior és obligatori sigui quina sigui la decesió de l'usuari. Fins i tot quan es realitza una alta és necessari comprobar que existeix una població amb les mateixes dades, que la que l'usuari vol introduir a la bdd.

5. Disseny final de la bdd

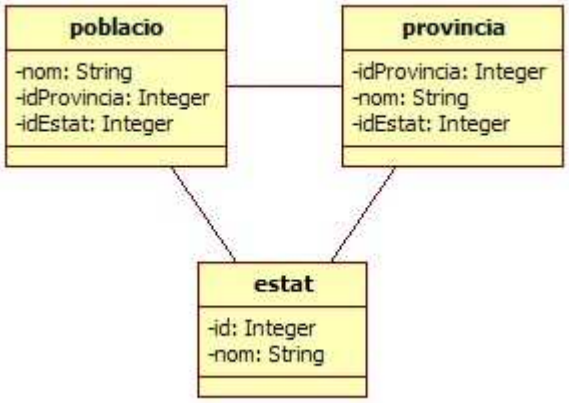
5.1 Model entitat relació



Aquest diagrama s'assembla molt al diagrama de classes anteriorment explicat, tot i que hi ha un parell de diferències. S'ha creat una nova taula **descr** que conté la descripció dels arxius. D'aquesta forma la bdd no es sobrecarrega amb informació innecessària, ja que la descripció pot ser extensa i guardar-la per a cada un dels arxius seria una pèrdua d'eficiència important. A més els arxius poden ser d'un o de diversos familiars, per això s'ha introduït una variable familiars. El segon cas és exclusiu per a arxius pertanyents a events que són de diferents familiars. Un exemple seria la fotografia d'un casament; en aquest cas l'arxiu en qüestió pertanyeria a dos familiars i per tant alhora de codificar-ho és important conèixer aquesta informació.

Per una altra banda també s'ha afegit una taula usuaris. Aquesta taula és exclusiva per l'apartat web i guarda les dades necessàries per a validar l'accés a la pàgina dels usuaris. En aquesta taula s'ha declarat l'atribut idFam, ja que els usuaris s'han de poder associar a alguna famñilia, sino no tindria cap raó d'existir. Tot i això l'usuari, dins la pàgina web, dispondrà de la possibilitat de canviar de família, permetent així una de les característiques de l'apliació, poder crear, modificar i consultar diferents famílies dins un mateix sistema.

Finalment la classe lloc s'ha dividit en tres taules principalment per la optimització del volum de dades de la bdd, així com també una millor implementació del sistema final. El refinament d'aquesta classe es pot observar en el següent diagrama.



Com es pot observar en el diagrama tant la poblacio com la provincia estan relacionades amb la taula estat i d'aquesta quan l'usuari desitgi modificar alguna dada, hi ha moltes possibilitats de canvis realitzables.

5.2. Taula familiar

Field	Type	Null	Key	NULL
ID	int(11)	NO	PRI	
DNI	varchar(9)	YES		NULL
ID_FAMILIA	int(11)	YES		NULL
NAIX	date	YES		NULL
MORT	date	YES		NULL
NOM	varchar(40)	NO		
TEL	int(11)	YES		NULL
MAIL	varchar(40)	YES		NULL
DIR	varchar(50)	YES		NULL
LLOC	varchar(20)	YES		NULL
LLOC_MORT	varchar(20)	YES		NULL
ID_PARE	int(11)	YES		NULL
ID_MARE	int(11)	YES		NULL
SEXE	char(1)	NO		

La meva primera idea va ser que el dni fos el camp clau, pero degut a que la gent d'avançada edad en moltes ocasions no en disposa, vaig acabar optant per crear un nou camp id que era una millor opció per ser el camp clau. Aquest identificador individual és creat automàticament i no és visible per l'usuari de l'aplicació. Destacar camps com naixament (data de naixament), mort (data de mort, si s'ha produït), lloc (lloc de naixament) i lloc_mort (lloc de mort, com abans, si s'ha produït). Molts dels camps poden ser nulls, és a dir, no cal que siguin emplenats ja que, com abans, moltes dades no son conegudes, oblidades o bé l'usuari no té obligació ni intenció d'emplenar.

5.3. Taula família

Field	Type	Null	Key	NULL
ID	int(11)	NO	PRI	
NOM	varchar(20)	NO		
PASS	varchar(8)	YES		NULL

Com l'aplicació permetrà crear diverses famílies, he acabat decidint que el millor era, com abans, crear un camp id que serà camp clau i no utilitzar el nom de la família. Aquesta decisió també permet que en un mateix sistema apareguin diverses famílies amb el mateix nom. A més existeix la possibilitat de crear una contrasenya i, d'aquesta forma, les dades d'alguna de les famílies, si es desitja, quedin almenys protegides. Aquesta és opcional i si no es desitja no cal crear-ne una.

5.4. Taula arxiu

Field	Type	Null	Key	NULL
ID	int(11)	NO	PRI	
UBICACIO	varchar(80)	YES		NULL
FAMILIAR	int(11)	YES		NULL
EVENT	int(11)	YES		NULL
LLOC	varchar(30)	YES		NULL
DATA	date	YES		NULL
DESCR	varchar(200)	YES		NULL
TIPUS	int(11)	YES		NULL

Un arxiu pot estar relacionat a un familiar o a un event, tot i que en aquest darrer cas acaba estant relacionat a un event. També he cregut interessant, que els arxiu guardin tant la data (en cas que es correspongui a un event serà la data en que es produeix l'event), el lloc i una descripció. Amb totes aquestes dades els arxius donen molta informació addicional. Hi ha un camp tipus, que permetrà coneixer si es tracta d'una foto, un vídeo o bé un arxiu de so.

5.5 Event

Primer crec que és important definir que és un event; un event és un esdeveniment en la vida d'una persona o diverses. Introduïts a la bdd n'hi haurà molts significatius, però a més de poder associar-los a diferents familiars, també existirà la opció de crear nous tipus, i d'aquesta forma, l'aplicació serà molt més interactiva. Alguns exemples d'events són: casament, canvi de residència,...

Per poder realitzar-ho amb claretat i que no hi hagi moltes dades repetides, he decidit separar aquesta taula en dues. La taula event guarda totes les dades d'events excepte la descripció que resta guardada en una altra taula descr.

Field	Type	Null	Key	NULL
ID	int(11)	NO	PRI	
ID_DESC	int(11)	NO		
FAM1	int(11)	NO		
FAM2	int(11)	NO		
POBLACIO	varchar(30)	YES		NULL
DATA	date	YES		NULL

Field	Type	Null	Key	NULL
ID	int(11)	NO	PRI	
DESCR	varchar(50)	NO		
FAMILIARS	int(11)	NO		

Un event pot ser d'un familiar o bé de dos. Per exemple, l'event casament engloba dues persones, mentre que el bateig només un. Com abans, crec que és necessari guardar la població i la data de l'event.

5.6. Taula lloc

Finalment per poder guardar el lloc he decidit crear tres taules. Cada una té una referència a una altra i d'aquesta forma hem queden totes relacionades a partir de la primera. Aquestes tres son: població, província i estat. Les dades que aquestes guarden són les següents....

Field	Type	Null	Key	NULL
NOM	varchar(30)	NO	PRI	
ID_PROVINCIA	int(11)	YES		NULL
ID_PAIS	int(11)	YES		NULL

Field	Type	Null	Key	NULL
ID	int(11)	NO	PRI	
NOM	varchar(20)	YES		NULL
ID_ESTAT	int(11)	NO		

Field	Type	Null	Key	NULL
ID	int(11)	NO	PRI	
NOM	varchar(20)	YES		NULL

6. Codificació

Abans de començar a explicar els programes utilitzats o bé el codi, m'atradaria parlar de la metodologia de treball utilitzada per aconseguir els resultats obtinguts. El que he utilitzat majoritàriament és l'**extreme programming** (XP) , també programació extrema, que es diferencia de les metodologies tradicionals, principalment, per que es posa més enfasis en l'adaptabilitat que en la previsibilitat. Els defensors d'aquesta tècnica creuen que ser capaós d'adaptar-se als canvis que apareixen durant la vida del projecte, en comptes de tenir-los previstos des del començament, és una aproximació millor i molt més realista

Aplicant aquesta metodologia he anat realitzant petites millores una rere l'altre, tornant enere i modificant algunes parts del codi i, per tant, adptant-me als canvis del projecte. Aquesta forma de treballar també obliga a realitzar moltes proves, sovint repetitives, degut a que els canvis que es van realitzant són petits i no es pot avançar fins que el pas anterior no està confirmat que funciona.

El avançar amb “pasos” curts obliga, a més de fer moltes proves, a que després d'implementar una part important, s'hagi de comprobar que no hi hagi errors, ja que aquests repercutirien en processos posteriors, o bé que els canvis a posteriori fossin molt més importants, que no en aquest punt. També he aplicat la divisió de codi, com recomana la tècnica, pero no com a divisió de feina per personal sinó en classes. D'aquesta forma, al partir el codi en capes, quan haig de modificar alguna part del codi, fet molt normal amb aquest tipus de tècnica, no he de modificar gaires classes diferents, sino que només he de canviar el codi de la classe que acaba realitzant l'acció (no totes les línies de codi, que criden el mètode; això és el que passaria si no es dividís en capes).

Finalment també he codificat, d'entrada, un codi senzill, ja que aquesta és la millor manera que els mètodes funcionin. Un cop tot funcioni correctament, s'afegeixen les funcionalitats que siguin necessaries. La programació extrema aposta per un codi senzill, encara que s'acabi tenint treball extre per modificar-lo, que un codi complicat que, potser mai, s'utilitzi. En resum millor realitzar un codi senzill i pràctic, corretgir els possibles errors que es generin; avançar comprobant que fins la data no hi ha problemes; i el més important adaptar-se a les possibles dificultats que es trobin durant la vida de creació del projecte.

6.1. Software utilitzat per l'el.laboració del sistema

Abans de començar a explicar el codi, és necessari explicar quines eines han estat utilitzades per poder-lo crear. D'aquesta forma quan es procedeixi a explicar-lo, ja es coneixeran i resultarà més fàcil d'entendre.

Cal dir que tot el software utilitzat és programari lliure i per tant totalment legal i utilitzable amb tranquil.litat. El primer que vaig haver d'escollir va ser el processador de textos. Entre les opcions possibles hi havia **Microsoft Word** una bona opció si es té un bon ordinador (gasta molts recursos, especialment la última versió), pero amb el problema que no és programari lliure. Dins aquest apartat hi ha dos possibles solucions: la primera **Abi Word** interfície senzilla pero amb recursos per sota de **OpenOffice** que a més disposa d'un paquet amb altres aplicacions com **OOBase** (un programa de bdd) o **OOCalc** (és com l'Excel).

Al final vaig optar pel paquet OpenOffice per crear tota la documentació, així com també alguna que altra fulla de càlcul. Per crear els diagrames, mostrats anteriorment, he utilitzat el StarUml, que permet desenvolupar qualsevol tipus de diagrama.

La meva primera idea per crear la bdd va ser OOBase, eina integrada al paquet OpenOffice, pero després de crear la bdd, hem va resultar impossible crear una bona connexió JDBC (tot i que vaig intentar documentar-me, no vaig trobar gaire informació sobre la possible connectivitat i vaig decidir-me per una altra opció). La solució escollida finalment va ser el MySql, on la configuració em va ser molt senzilla i ràpida. Per realitzar aquesta connexió vaig crear un nou origent de dades ODBC.

Al escollir alguns components com swt (la última versió), també era obligatòria l'actualització del Java a la versió 6. SWT (Standard Widget Toolkit) és un conjunt de components per construir interficies gràfiques en Java, widgets desenvolupats pel projecte Eclipse. Recupera la idea original de la biblioteca AWT d'utilitzar comenents natius, amb el que apadrina un estil més consistent en totes les plataformes, pero evita caure en les limitacions d'aquesta.

La biblioteca Swing, per altra banda, està configurada totalment en Java i sovint rep crítiques per no oferir una experiencia idèntica a la d'una apllicació nativa. El preu a pagar per aquesta millora es la dependencia (a nivell d'aspecte visual i no d'intericie de programació) de l'aplicació resultant del sistema operatiu amb la que s'executa. L'interificie del workbench de l'eclipse també depèn d'una capa intermitja *d'interfície* gràfica d'usuari (GUI) anomenada Jface, que simplifica la construcciñó d'aplicacions basades en SWT.

6.2. Llenguatge de programació utilitzat

El llenguatge utilitzat és el java i la versió mínima per aconseguir que el programari s'executi és Java versió 6.0 .

6.3. Entorn de programació

Com he comentat anteriorment he utilitzat el projecte Eclipse per crear el codi de l'aplicació. He descarregat diversos extres per millorar tant el disseny com la funcionalitat de l'entorn.

6.4 Justificació i explicació del codi

Abans de crear el codi va ser necessari crear un nou projecte, amb l'anteriorment citat Eclipse; que va rebre el nom de **projecteFinal**. També vaig haver d'importar tots els jars, que l'Eclipse no porta per defecte i afegir-los-hi**.** Els *.jar importats foren: el *mySqlConnector* utilitzat per poder connectar l'aplicació amb l'origen de base de dades, creat a l'inici; i el *Jface* que ajuda a una edició gràfica del codi molt més senzilla i directa.

Si s'obre el projecte es poden veure, apart dels citats *.jar anteriors, una sèrie de carpetes (packages) que explicaré a continuació:

- Imatges: es guarden les imatges predefinides. Imatges utilitzades pels menús i també per a alguns exemples que es donen d'inici. És la carpeta per defecte on l'usuari ha de desar les imatges, perquè puguin ser utilitzades.
- MySqlBDD: és la bdd utilitzada pel projecte. En aquesta hi ha guardades les taules i algunes dades que, com abans, serveixen d'exemple a l'inici. El seu nom és *ibdata1* nom per defete que MySql utilitza.
- Bdd: son les classes relacionades amb la base dades. Fan referència a totes aquelles operacions que modifiquen o hi accedeixen directa o indirectament.
- Interficies: son aquelles clases “pantalla”. Contenen el contingut que se li mostrarà a l'usuari i que aquest podrà modificar o accedir, a partir de les mateixes a la informació continguda a la bdd.
- Classes: fan referencia als objectes més importants i absolutament imprescindibles per poder realitzar les operacions a la bdd i per a un correcte funcionament de l'aplicació.

6.4.1. Definició de les classes

6.4.1.1 Classe Arxiu

Classe que s'encarrega de guardar fotos, videos i so; de familiars i/o arxius. Té els següents atributs:

int id: identificado únic.

`String ubicacio: ruta fins l'arxiu.`

`int familiar: (clau foràne a).`

int event: `(clau foràne a).`

String lloc: `(clau foràne a).`

String data: data en que es va produir l'event.

String descr: descripció de l'arxiu.

int tipus: tipus d'arxiu (foto,...).

Destacar que es guarda la ruta a l'arxiu com un String; no es guarda tot l'arxiu. Per seleccionar l'arxiu dessitjat s'ha d'introduir per pantalla la ruta sencera (desde la carpeta imatges, abmans comentada). Pel bon funcionament del programari s'ha afegit una carpeta imatges, al projecteFinal, i un accés directe a l'escritori. Amb aquest dos afegits, primer s'ha de copiar la foto a la carpeta abans esmentada, i llavors la ruta a introduir és molt més curta. Exemple: si es crea dins la carpeta, una altra amb el nom Hernández; al afegir la foto *Maria.jpg*, la ruta s'hauria d'introduir seria la següent */Hernandez/Maria.jpg*. Un cop fet, la ruta de la fotografia queda guardada a l'atribut **ubicacio**.

Per altra banda es guarda la clau de les classes event, lloc i familiar; com també el tipus d'arxiu. Per exemple si es tracta d'una fotografia, el tipus és 1.

He implementat un constructor que assigna 0 a tots els enters i una cadena buida a tots els String. A part de tots els getters/setters típics, he codificat un mètode assingar .

```
public void assignar(Arxiu aux){
        id=aux.getId(); ubicacio=aux.getUbicacio();
        familiar=aux.getFamiliar(); event=aux.getEvent();
        lloc=aux.getLloc(); data=aux.getData();
        descr=aux.getDescr(); tipus=aux.getTipus();
}
```

Com es pot observar copia en un arxiu totes les dades que hi ha a l'Arxiu aux passat per referència. Aquest mètode s'utilitza normalment quan es volen obtenir les dades que ja es troben guardades a la bdd.

6.4.1.2 Classe Evento

No he utilitzat event, ja que aquesta classe ja està implementada per Java i comportava certs problemes de compatibilitat.

Els atributs de la classe son:

private int id: identificador únic.

private String desc: descripció de l'event.

public int fam1: familiar.

public int fam2: segon familiar, en cas que existeixi.

public String data: data quan es va produir l'event.

public String lloc: lloc on es va produir l'event.

public int numFam: número de familiars.

Com s'ha dit abans, els events poden ser de un o de dos familiars. Un casament n'és, evidentment, de dos; una comunió, en canvi, només en fa referència a un. Per tant és necessari guardar els identificadors del/s familiar/s de l'event. A més he afegit un altre atribut numFam, per no tenir que accedir a la bdd i saber quants familiars el formen, d'aquesta forma intento no sobrecarregar els accesos.

Un cas una mica especial és el del canvi de residència. Moltes vegades un canvi de residència afecta a més d'un familiar (pares, fills,...) tot i això en realitat aquest tipus d'event en realitat és d'1 familiar, i per tant, se n'hauria de crear un per a cada un dels membres. A la meva aplicació quan es realitza un canvi de residència només afecta al familiar escollit; tot i que una de les possibles millores a realitzar podria ser, la de realitzar el canvi de forma múltiple; demanant a l'usuari si vol que l'event s'apliqui a altres familiars. El resultat acabaria sent el mateix, un nou event per cada un dels membres, però per l'usuari seria molt més ràpid aconseguir-ho.

Pel que fa als mètodes no hi ha res destacable. Els típics getters/setters, un constructor que inicialitza les variables i un assignar que copia les dades d'un altre event.

Per finalitzar comentar que aquesta classe guarda les dades a la bdd en dues classes *arxiu* i *descr.* A la taula arxiu hi ha totes les dades excepte la descripció que es guarda a la taula descr. Aquesta multiplicitat la vaig fer perquè sino hi havia molt volum de dades repetides dins la taula arxiu i a més d'aquesta forma l'agrupació d'events, i per tant la seva eficiència alhora de crear vistes per tipus d'arxiu, millora en gran mesura.

6.4.1.3 Classe Familia

Els atributs de la classe son:

private int id: identificador únic de la família.

private String nom: nom de la família.

public String pass: contrasenya (opcional).

Pel que fa als mètodes, igual que abans, he implementat els getters/setters i un assignar (copia les dades d'una família). Finalment en aquesta classe destacar que hi ha diferents constructors...

```
public Familia(){
        id=0; nom=null; pass=null;
}
public Familia(String aux){
        id=0; nom=aux; pass=null;
}
public Familia(int a,String b, String c){
        id=a; nom=b; pass=c;
}
public Familia(String b, String c){
        nom=b; pass=c;
}
```

Una de les raons principals perque hi hagi aquesta diversitat de constructors és que el password no és un camp obligatori.

6.4.1.4 Classe Lloc

Els atributs de la classe son:

private String nom: nom de la població.

private String provincia: nom de la província.

private String pais: nom de l'estat.

Ha sigut necessari implementar tres constructors. En un únicament se li passa el nom del lloc a crear, en un altre es passen els valors de la provincia i del pais, mentre que en l'últim es passen el nom del lloc a crear, l'identificador de la familia i el del pais. Aquesta multiplicitat de constructors és deguda a que l'usuari, pot donar crear regions (provincies), poblacions o bé estat de forma no lineal, i això tot i que dóna moltes possiblitats al usuari final, però acaba repercutint en el codi.

```
public Lloc(String aux){
        nom=aux;
}
public Lloc(String nom, String provincia, String pais){
        this.nom=nom; this.provincia=provincia; this.pais=pais;
}
public Lloc(String provincia, String pais){
        this.provincia=provincia; this.pais=pais;
}
```

Els mètodes que hi ha implementats són els getters/setters, un constructor que iniciailiza les variables a cadenes buides, un mostrar (utilitzat alhora de fer el debug) i un mètode assignar (copia un lloc passat per referencia). Finalment pel que fa a la bdd; aquesta classe queda dividida en tres taules. D'aquesta forma la redundància de dades és molt menor, a més de facilitzar-ne molt l'agrupació alhora de visualitzar les possibles vistes.

6.4.1.5 Classe main

Classe que únicament, tot i que de forma indispensable, inicialitza l'aplicació...

```
package classes;
import interficies.MenuInici;
public class Main {
        public static void main(String[] args) throws Exception {
                MenuInici nou=new MenuInici();
        }
}
```

6.4.1.6 Classe Familiar

Aquesta és la classe amb un volum de dades major; els atributs de classse son...

private int bdd: identificador únic guardat a la bdd.

```
private String id: dni.
```

private String nom: nom del familiar.

private int telefon: telèfon del familiar.

```
private String mail: mail.
```

private String adreça: adreça.

private int idPare: identificador del pare (clau forànea).

private int idMare: **iden**tificador **de la pare (**clau forànea).

```
private char sexe: sexe.
```

public int idFam: identificador de la família a la que pertany.

public String llocNaix: lloc de naixament.

public String llocMort: lloc de defunció, si s'ha donat el cas.

private String dNaix: data de naixament.

```
private String dMort: data de mort, si s'ha produït.
```

Com ja he comentat no tots els camps han de ser emplenats; en realitat només un el nom. Tots els altres, en cas que l'usuari no els introdueixi, s'inicialitzaran de forma automàtica. Quan es dóna d'alta un nou membre de la família, se li assigna un valor únic, aquest és representat per l'atribut bdd. L'atribut id, son les restes de la primera idea que vaig implementar. Creia que assignant la clau primària al dni ja n'hi hauria prou, pero em vaig donar compte que hi havia molts inconvenients: la gent de major edad podria ser que no el tingués, generacions passades fàcilment no el tindrien, possibilitat, encara que remota, que hi hagués repetits,.... per tot això al final, vaig preferir canviar el dni, per una variable inicialitzada de forma programada.

Constructors...

```
public Familiar() {
        nom="";llocNaix="";llocMort=""; bdd=0;
}
public Familiar(String aux) {
        nom=aux;
}
public Familiar(String aux, int aux2,String aux3,String aux4,Date aux5,char aux6, int aux7,int aux8,String aux9,String aux10){
        nom=aux; telefon=aux2;adreça=aux3; mail=aux4;
        dNaix=String.valueOf(aux5);
        sexe=aux6;
        idPare=aux7; idMare=aux8;
        id=aux9;
        llocNaix=aux10;
}
```

Pel que fa referència als mètodes s'han implementat els habituals (getters/setters, mostrat i un mètode assignar).

6.4.1.7 Classe Dades

He deixat aquesta classe pel final, ja que va ser la última que vaig crear com també la menys intuitiva. Totes les altres quedaven representades al diagrama de classes, a més de ser totalment lògica la creació de les mateixes. La utilització d'aquesta classe, que fa referència a les interfícies, serà explicada amb deteniment més endevant; tot i que ara en faré un resum.

Al realitzar el projecte tenia la intenció de que fos dinàmic i amb la possibilitat d'una navegació ràpida e intuitiva, per qualsevol nivell d'usuari. Per fer-ho vaig crear una barra, al estil d'un navegador d'internet, que permetés anar directament a les opcions principals i amb la opció de tornar a les pantalles anteriorment visitades. Com quedarà exposat després, el problema que hi ha és que al poder tornar és necessari guardar moltes dades; ja que les que s'hi havien introduït també era necessari conservar-les. Per aquest motiu vaig crear aquesta classe, que acaba sent un contenidor de totes les dades que es poden utilitzar durant l'execució del programari i que totes les interficies necessiten pel seu bon funcionament, especialment quan es fa ús del navegador.

Els atributs de classe són els següents...

`private String nom:` Nom de la familia.

`Private int idFam: Id de la familia.`

`private Familiar nou: Familiar`

private Familiar alta: Familiar per donar d'alta.

private Lloc lloc: Lloc.

private Arxiu arxiu: Arxiu.

private Evento event: Event.

private Vector donVinc: Vector de pàgines anteriors.

És important comentar el perquè hi ha guardat el nom de la Família com també el seu id. En la meva primera versió sempre que executava algun mètode utilitzaba el nom de la família; pero hem vaig donar compte que al poder tenir dues famílies amb el mateix cognom havia d'utilitzar l'id i no el nom. Alhora de cridar els mètodes era important fer servir l'id, ja que sino es podrien anar barrejant les famílies. Vaig decidir, per tant, guardar únicament l'id en aquesta classe. Però tot i que no es notava gaire, cada cop que es canviava de pantalla o bé utilitzava alguna de les opcions, es repetia molt l'accés a la bdd per aconseguir el nom de la familia. Finalment per millorar l'eficiència i no sobrecarregar la bdd, vaig optar per guardar ambdues variables (tant l'id com el nom).

El constructor és igual que els anteriors comentats. Pel que fa al mètodes (getters/setters) típics i el que s'hauria de destacar és el següent...

```
public void assignar(Familiar nou){
        this.nou.assignar(nou);
}
public void assignarFamAlta(Familiar nou){
        this.alta.assignar(nou);
}
public void assignarEvent(Evento event){
        this.event.assignar(event);
}
public void assignarArxiu(Arxiu arxiu){
        this.arxiu.assignar(arxiu);
}
public void assignarLloc(Lloc lloc){
        this.lloc.assignar(lloc);
}
```

Ara cal un mètode assignar per cada un dels objectes que formen la classe, ja que aquests no poden modificar el valor com els setters. Aquests mètodes criden al mètode que tenen implementats implícitament els objectes. L'*assignar* actualitza el valor del familiar, el *assignarEvent* actualitza l'event, *assignarArxiu* actualitza l'arxiu i l'*assignarLloc* actualitza el lloc. *AssignarFamAlta* actualitza alta, que també és un familiar. He tingut que afegir aquest atribut, ja que quan es dóna d'alta és necessari un altre familiar (més endevant s'explicarà amb detall).

Finalment aquesta classe té implementats dos mètodes que ajuden a la interactivitat, abans comentada...

```
public void afegirPagina(String actual){
        donVinc.add(actual);
}
public void eliminarPagina(){
        int posicio=donVinc.size()-1;
        donVinc.removeElementAt(posicio);
}
```

Els necessito per afegir i treure pàgines al vector donVinc. Aquest vector, com veurem més endevant, guarda totes les pàgines visitades per l'usuari. En cas que l'usuari canviï a una altra, es crida a *afegirPagina* que, com una pila, col.loca a la última posició la darrera pàgina visitada. En cas que es torni cap enrere el procés és el contrari, i s'elimina la última pàgina visitada. D'aquesta forma és possible anar recorrent totes les interfícies visitades per l'usuari.

6.5 Definició de les classes referents a la bdd

He realitzat un disseny per capes. Les interfícies es relacionen amb les classes *Gestio...*, aquestes amb *Rel...* i finalment aquestes amb *Bd...* Explicaré únicament les *Bd...* ja que són les que tenen particularitats en el codi, les altres són únicament classes pont per arribar a aquestes, tot i que son indispensables per poder realitzar una implementació per capes. Abans, pero, comentaré la classe **AccesBD** que és la que efectua les connexions i desconnexions entre l'aplicacio i la bdd, per mitjà de la connexió ODBC creada inicialment.

Per últim comentar, que sempre que sigui possible afegiré una fotografía de la interficie a la que correspongui el mètode. Tot i això, aquesta només serà per ajudar a millorar la comprensió del codi, ja que les interfícies s'explicaran a la part final.

6.5.1 AccesBD

```
/* Classe que conecta directament a la base de dades*/
public class AccesBD {
    Connection nova;
    /**
     * Constructor sense paràmetres. Crea una conexió a la base de dades
     * mySqlBdd, passant-li el nom de la conexió i el nom de l'origen de dades.
     */
    public AccesBD() {
        try {
            Class.forName("org.gjt.mm.mysql.Driver");
            nova = DriverManager.getConnection("jdbc:mysql://localhost:3306/projecte","root","projecte");

        }
        catch (Exception err) {
```

```
            }
        }
        /**
         * Tanca la conexió amb la base de dades.
         * @exception SQLException es llança una excepció quan hi ha algun probleam
         * de tipus sql.
         */
        public void desconnectar() throws SQLException {
            nova.close();
        }
}
```

6.5.2 BdArxiu

Aquesta classe modifica les taules arxiu i descr, ubicades a la bdd. Primer cal comentar que totes les classes *Bd...* extenen el contingut de *AccesBD* i per tant el constructor, de cada una d'elles, queda implementat de la següent forma:

```
public BdArxiu() {
    super();
} //END CONSTRUCTOR
```

Els mètodes implementats són els següents...
obtenirArxiu(String)
obtenirArxiu(int)
obtenirFotoPersonal(int)
insertarFotoPersonal(String, int)
afegirFotoFamiliar(String, int, int, String, String, String)
modificarFotoPersonal(String, int)
obtenirTotesFotos(String)

obtenirFotosFamiliar(Familiar)

obtenirFotosEvent(Evento)

obtenirDescripcioFoto(String)

modificarDescripcio(String, String)

borrarArxiu(String)

eliminarArxiusEvent(Evento)

obtenirFamiliarsAmbFotos(String)

eliminarTotsArxius(Familiar)

En els següents punts es comentaran amb deteniment els aspectes generals i caracterísitcs de cada un dels mètodes d'aquesta classe.

6.5.2.1 Mètode obtenirArxiu(int)

Mètode per obtenir tota la informació relacionada amb un arxiu a partir del seu identificador únic. Les variables utilitzades són les següents...

```
//Declaració e inicialització de variables.
Statement s;
ResultSet rs;
Arxiu resultat=new Arxiu();
```

La variable Statement és un objete utilitzat per executar una declaració SQL estàtica i retornar els resultats que produeix. La inicialització que se li efectua és `s = (Statement) nova.createStatement();` i d'aquesta forma queda sincronitzada amb la base de dades. Volia recalcar la declaració i la inicialització, ja que en tots els mètodes, de qualsevol de les classes bdd, és la mateixa.

Les crides per obtenir les daddes destitjades i ser usades són les següents:

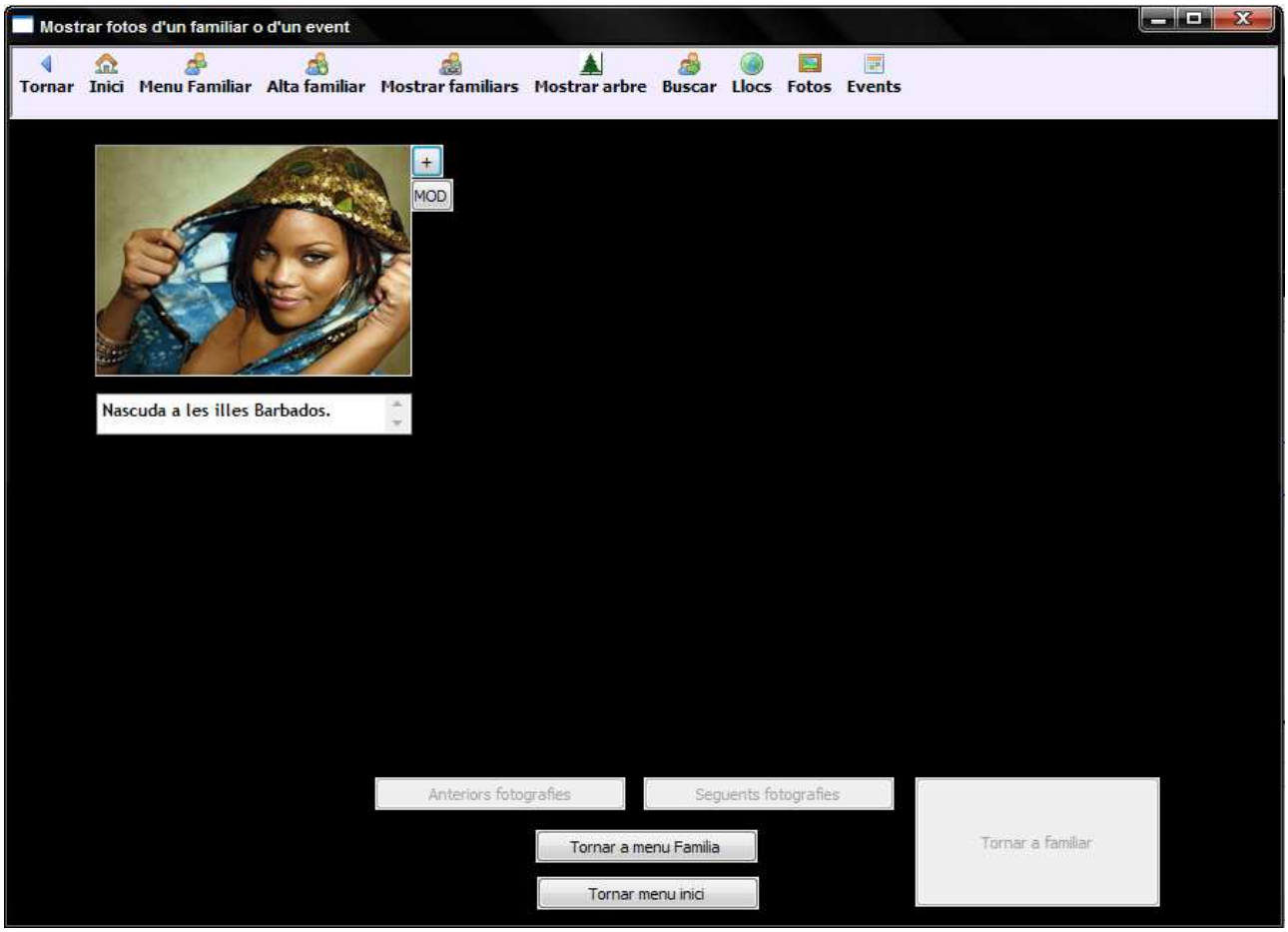
```
s.executeQuery("SELECT * FROM ARXIU "+"WHERE ID='"+id+"';");
rs = (ResultSet) s.getResultSet();
if (rs.next()) {
        resultat.setId(id);
        resultat.setUbicacio(rs.getString(2));
        resultat.setFamiliar(rs.getInt(3));
        resultat.setEvent(rs.getInt(4));
        resultat.setLloc(rs.getString(5));
        resultat.setData(rs.getString(6));
        resultat.setDescr(rs.getString(7));
        resultat.setTipus(rs.getInt(8));
}
```

En cas que es produeixi algún tipus d'error amb la connexió amb la bdd, i per tant no s'hagi pogut actualitzar el valor de l'Arxiu *resultat*, s'executa el catch(SqlException) que té el seu codi:

```
try{
/**TOT EL CODI...**/
}catch (SQLException e) {
        throw new Exception(e.getErrorCode() + " - " + e.getSQLState() + " - " +
                                        e.getMessage());
}catch(Exception e){
        throw new Exception(e.getMessage());
}
```

Per finalitzar es tanca la connexió amb la base de dades i es retorna *resultat*, que és l'arxiu amb l'identificador igual a la variable passada per referència. Aquest mètode s'ha explicat amb deteniment perquè era el primer; a partir d'ara, seran explicats més breument, això si, destacant el que mereixi alguna menció especial.

Aquest mètode s'utlitza especialment quan es mostren diversos arxius per pantalla; com es pot observar a continuació:



Amb aquesta inferfície *mostrarFotosFamiliar* (serveix per familiar i events) , com es pot observar es mostren fotos, tot i que en aquest cas només una, i a més una descripció de la mateixa. També hi ha una sèrie de botons (com explicaré més endevant a l'apartat interfícies) per poder modificar la foto, borrar-la o veure-la a pantalla completa.

6.5.2.2 Mètode obtenirFotoPersonal(int)

Aquest mètode retorna, en cas que existeixi, la ubicació de la fotografia personal d'un familiar; mentre que si no existeix es retorna una cadena buida. L'enter passat per referència és l'identificador únic del familiar. La select utilizada en aquest mètode és:

```
s.executeQuery("SELECT UBICACIO FROM ARXIU " +"WHERE FAMILIAR='"+aux+"' AND DESCR='Personal'");
```

Cal remarcar que per saber si la fotografia és la personal (la que es mostra en moltes interfícies com a foto de referència) el que procés que es fa és el de comprobar que la descripció de l'arxiu sigui igual a 'Personal'.

6.5.2.3 Mètode insertarFotoPersonal(String, int)

Aquest mètode permet assignar la fotografía personal d'un familiar. L'enter passat per referència, és l'identificador del fmailiar metnre que l'String és la ruta completa a la fotografía personal a assignar. El primer que es fa en aquest mètode és assignar l'identificador a l'arxiu, per fer-ho es busca el primer id lliure...

```
s1.executeQuery("SELECT MAX(ID) FROM ARXIU");
if (rs1.next()) {
    maxim=rs1.getInt(1);
}
```

La variable màxim és inicialitzada amb 6000 (l'id dels arxius comencen per aquest valor), en cas que no hi hagi cap arxiu inserit, el valor que se li assignarà a la foto desitjada serà aquest, mentre que si n'existeixen diversos, se li assigna l'útlim lliure.

Un cop assignat l'id, només queda efectuar la inserció a la bdd

```
s.executeUpdate("INSERT INTO ARXIU VALUES("+(maxim+1)+",'"+ruta+"','"+fam+"',0,
NULL,NULL,'Personal',1);");
```

Destacar a que tant lloc com la data s'inicialitzen a null, perquè son valors no necessaris. La descripció de la foto ha de ser, com he comentat anteriorment, amb 'Personal' i el tipus d'arxiu és 1.

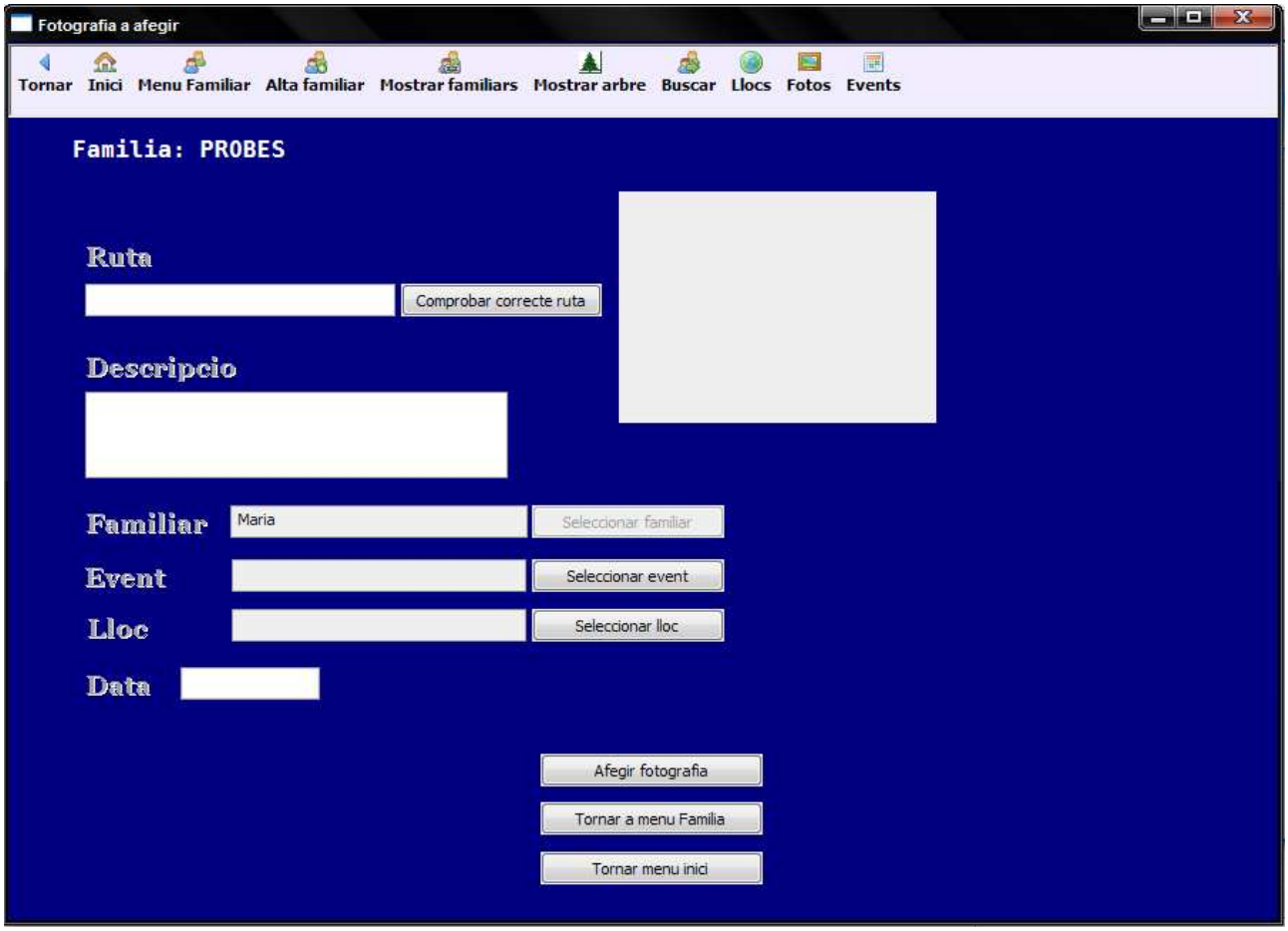
En el projecteFinal la interfície que utilitza aquest mètode és la següent:

L'únic que és necessari és afegir la ruta de l'arxiu. La única informació adicional que es necessita és l'id del familiar, pero ja es té guardada, perquè l'única forma d'arribar a aquesta pantalla és per mitjà de la pàgina personal del familiar.

6.5.2.4 Mètode afegirFotoFamiliar(String, int, int, String, String, String)

Aquest mètode afegeix una fotografia a un familiar. El primer String és la ruta (desde la carpeta imatges) de l'arxiu a inserir; la següent variable és un enter que correspon a l'identificador del familiar; el següent l'identificador de l'event; la pròxima és la descripció de l'arxiu; després hi ha el lloc i finalment la data.

Abans d'explicar amb deteniment el mètode mostraré la interfície que utilitza aquest mètode, ja que d'aquesta forma, segurament quedarà més clar.



Bona part de les variables poden ser nul.les ja que alhora d'insertar arxius, no tots els camps de les taules són obligatoris; tot i que familiar i descripció són camps obligatoris. Que el familiar sigui obligatori és lògic, però la descripció també perquè crec que és important aportar alguna informació extra, que a més serà mostrada en l'àlbum, juntament amb les fotos.

Com abans primer selecciono el valor de l'id que li correspon a l'arxiu que es vol inserir (com he comentat abans, el primer id que reben els arxiu és 6000). Llavors es realitza l'inserció a la bdd de l'arxiu. El codi s'ha implementat de la següent forma:

```
if(data.equals("aaaa-mm-dd")){
        s.executeUpdate("INSERT INTO ARXIU VALUES("+(maxim)+
        ",'"+("/imatges/"+ruta)+"','"+fam+"',"+event+",'"+lloc+"',NULL,'"
        +descr+"',1);");
}else{
```

```
        s.executeUpdate("INSERT INTO ARXIU VALUES("+(maxim)+
        ",'"+("/imatges/"+ruta)+"','"+fam+"','"+event+"','"+lloc+"','"+data+
        "','"+descr+"',1);");
}
```

Com es pot observar en el codi hi ha dues possibilitats: la primera que no s'hagi introduït la data en que es va realitzar la foto; i l'altre que si s'ha fet. El codi és pràcticament idèntic, pero si no s'ha entrat la data s'hi afegeix un String amb valor *NULL*.

6.5.2.5 Mètode modificarFotoPersonal(String, int)

Mètode que modifica la fotografía personal d'un familiar. Se li passen dues variables: un String corresponent a la nova ruta; i un enter que correspon a l'id del familiar. No cal l'identificador de l'arxiu ni cap altra variable, ja que cada familiar només pot tenir una foto Personal. Quan es crida aquest mètode significa que el familiar ja en té una i, per tant, no cal fer cap tipus de comprobació; únicament amb la nova ruta i el seu id, es selecciona primer el familiar i llavors se li modifica la ruta. D'aquesta forma sense fer cap comprobació ni validació, el familiar no tindrà mai dos arxius amb descripció 'Personal' a la bdd.

El codi de la modificació de la foto personal és força senzill i és el següent:

```
s.executeUpdate("UPDATE ARXIU SET UBICACIO='" +ruta+"' WHERE FAMILIAR='"+fam+"' AND DESCR='Personal';");
```

6.5.2.5 Mètode obtenirTotesFotos(int)

Aquest mètode permet obtenir totes les fotografíes, d'una mateixa família, que hi ha guardades a la base de dades. L'enter que es passa, és precissament l'identificador únic de la família. No és necessari res més, només coneguent la família, s'obtenen tots els arxius que li corresponen. Aquest mètode s'utilitza únicament en una de les interfícies i serveix, com s'explicarà en el moment adient, per mostrar tots els arxius en forma d'àlbum.

```
s.executeQuery("SELECT ARXIU.ID FROM ARXIU,FAMILIAR "+"where
familiar.id_familia ="+id+" and " + "familiar.id=arxiu.familiar and " +
"tipus=1;");
```

Aquesta és la crida que s'ha implementat per accedir a la bdd i obtenir totes les fotografies d'un familiar. Un cop obtinguda la llista d'arxius del familiar escollit, només queda guardar-los i retornar-los.

```
while (rs.next()) {
    Arxiu arxiu=new Arxiu();
    arxiu.setId(rs.getInt(1));
    resultat.addElement(arxiu);
}
```

D'aquesta forma totes les adreces a les fotografies que hi ha a la bdd, es guarden en el vector resultat, que és el que es retorna a la interfície, perque les pugui mostrar totes per pantalla. Finalment comentar que la taula arxiu no guarda l'id de la família, sino que guarda l'id dels familiars als que correspon l'arxiu. Per tant per poder saber si l'arxiu és de la família, el que s'ha de fer (i així queda pal.lès a la crida a la bdd) és comparar l'id de la família (passat per referència) amb l'id de la família del familiar (familiar si té com a atribut id_Familia).

6.5.2.6 Mètode obtenirFotosFamiliar(Familiar)

Aquest mètode és força semblant a l'anterior, però aquest només retorna les fotos d'un sol familiar, no de tots com l'anterior. La sentència sql és:

```
s.executeQuery("SELECT ID,UBICACIO,DESCR FROM ARXIU WHERE FAMILIAR='"+
fam.getBdd()+"';");
```

Finalment només queda emplenar el vector que es retornarà, amb totes les adreces de memòria dels arxius. El codi és el següent...

```
while (rs.next()) {
    Arxiu arxiu=new Arxiu();
    arxiu.setId(rs.getInt(1));
    arxiu.setUbicacio(rs.getString(2));
    arxiu.setDescr(rs.getString(3));
    resultat.addElement(arxiu);
}
```

6.5.2.7 Mètode obtenirFotosEvent(Evento)

Mètode que obté totes les fotos relacionades a un event. Es passa per referència l'event ja que conté l'id, que hem permetrà obtenir tots els seus arxius. El codi és pràcticament calcat als anteriors exposats, per tant l'únic que mostraré és la sentència sql...

```
s.executeQuery("SELECT ID FROM ARXIU WHERE EVENT="+event.getId()+";");
```

En cas que no hi hagi cap arxiu s'inicialitza el vector, perque és important que la interfície coneixi el resultat abans, ja que a vegades quan el resultat és buit, és millor no canviar de pantalla i mostrar un missatge per fer conèixer a l'usuari que és millor no canviar.

```
if (resultat.isEmpty()) {
        resultat.add("Error");
}
```

6.5.2.8 Mètode obtenirDescripcioFoto(String, int)

Aquest mètode retorna la descripció d'una foto; encara que en molts altres mètodes anteriors, s'inicialitza la descripció i es retorna de forma explícita. La forma més sencilla, probablement, seria buscar la descripció amb l'ajut de l'identificador de l'arxiu; pero en algun cas concret no era possible saber-lo i per això vaig crear aquest mètode. Cap la possibilitat, com es va comentar anteriorment, que hi hagi un mateix arxiu utilitzat per diferents familiars; en aquest cas si es busqué només per la ubicació de l'arxiu, el resultat podria no ser el desitjat. Per això la solució que hi vaig trobar, va ser la d'afegir l'identificador del familiar; ja que no veig viable que un mateix familiar guardi amb el mateix nom dues fotos amb descripcions diferents (no seria lògic). Per això amb la ruta i l'id del familiar:

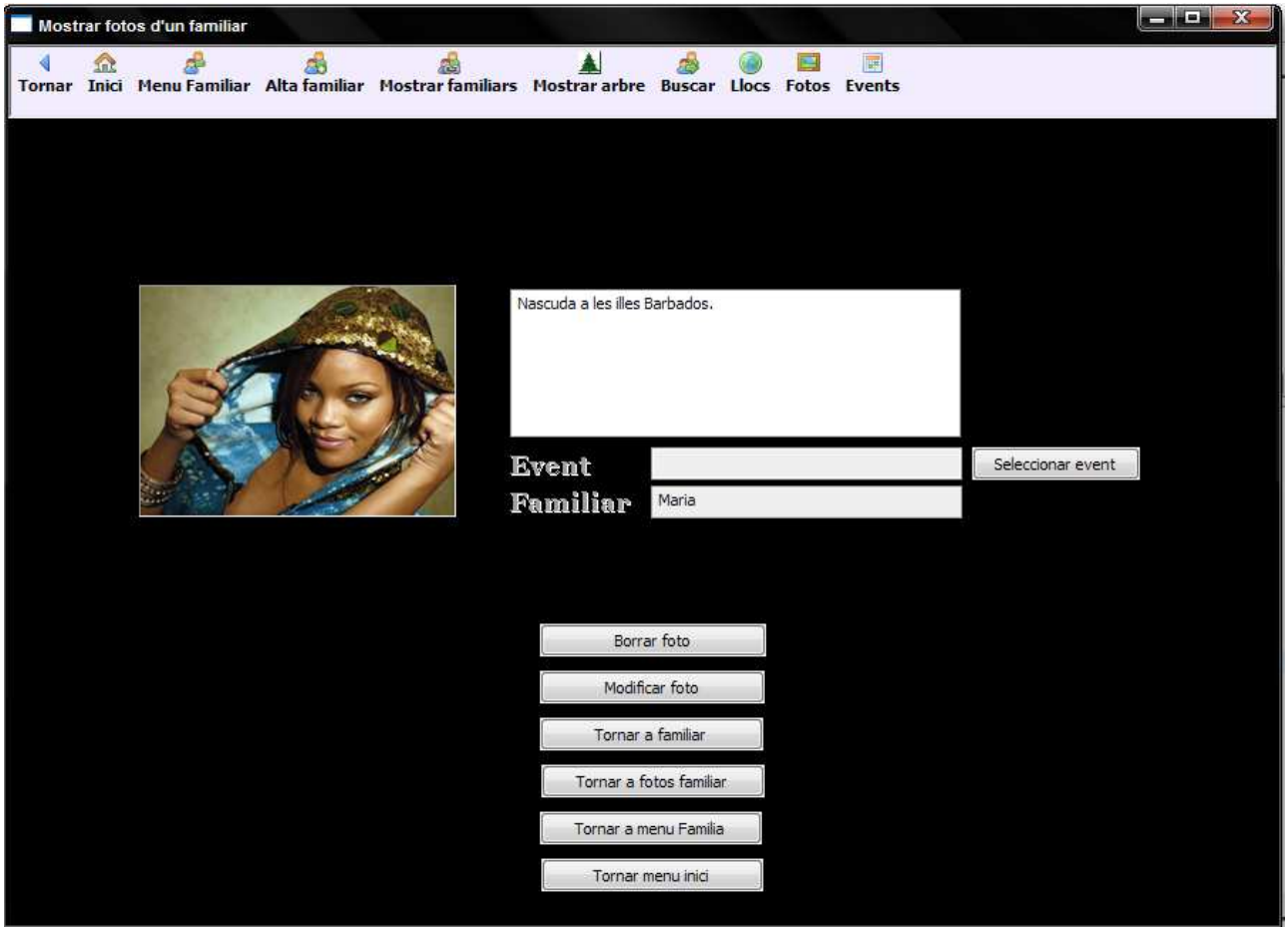
```
s.executeQuery("SELECT DESCR FROM ARXIU WHERE UBICACIO='"+ruta+"' AND FAMILIAR= '"+familiar+"';");
```

6.5.2.9 Mètode modificarDescripcio(String, int, String)

Mètode que s'assembla molt a l'anterior. Les dues primeres variables són les mateixen que en l'últim mètode explicat (nova ruta a l'arxiu i identificador del familiar). La darrera variable equival a la nova descripció. L'únic que cal recalcar és la sentència sql...

```
s.executeUpdate("UPDATE ARXIU SET DESCR= '"+descr+"' WHERE UBICACIO= '"+ubicacio+"' AND FAMILIAR="+familiar+";");
```

La interfície que permet modificar la descripció dúna fotografía és la següent...



Com es pot observar, a més aquesta interficie també permet modificar l'event al que pertany, borrar-la, així com evidentment modificar la descripció.

6.5.2.10 Mètode borrarArxiu(String, int)

Aquest mètode elimina una arxiu de la bdd. Té un codi molt semblant a l'anterior i només comentaré la sentència utilitzada:

```
s.executeUpdate("DELETE FROM ARXIU WHERE UBICACIO='"+ubicacio+"' AND FAMILIAR="+
familiar+";");
```

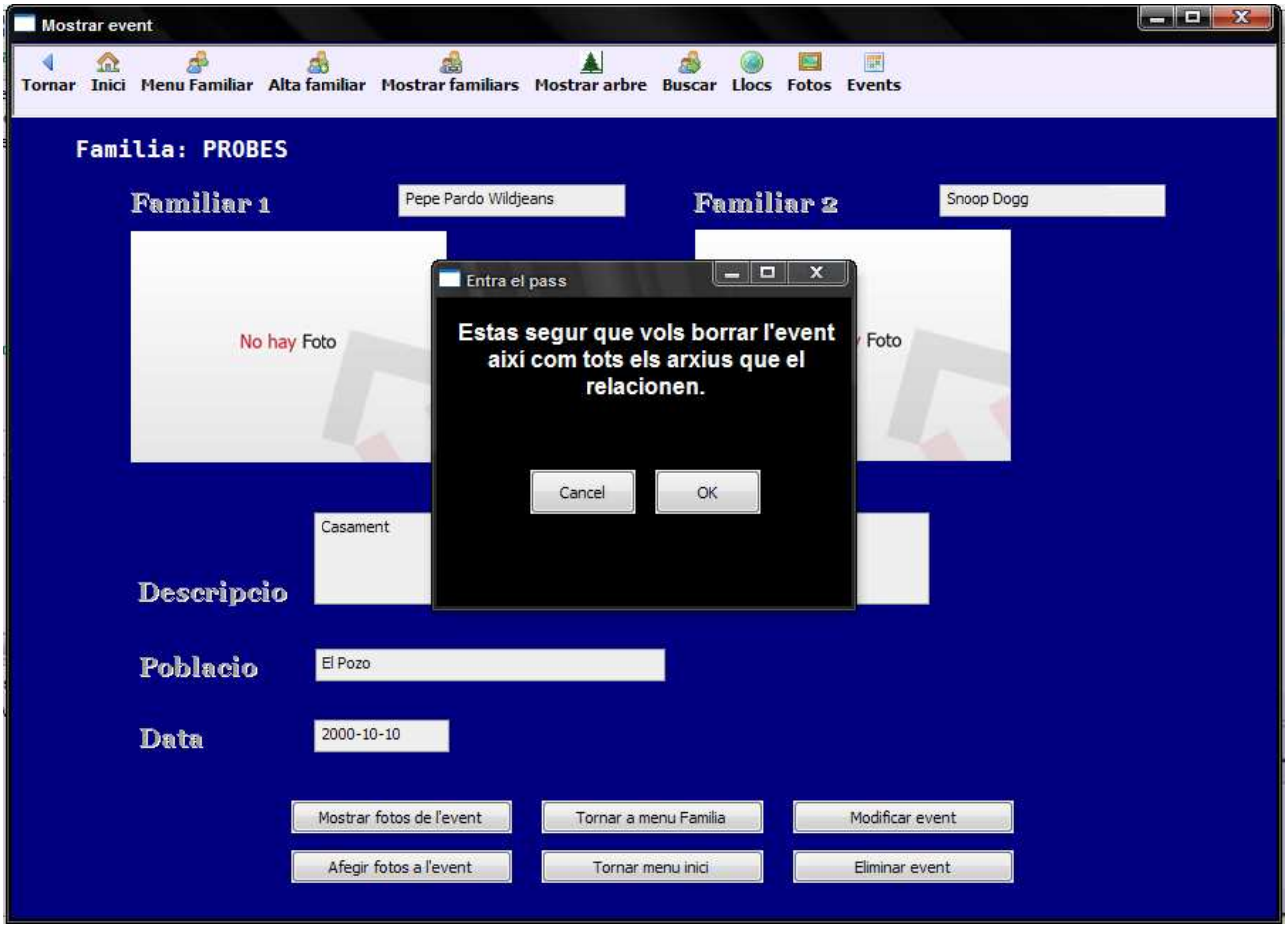
S'elimina l'arxiu que té com a ubicació i l'id del familiar iguals als passats per referència. Com abans, si no s'afegís la clau del familiar podría ser que s'eliminés un arxiu no dessitjat, degut a la multiplicitat d'accessos possibles a un mateix arxiu.

6.5.2.11 Mètode eliminarArxiusEvent(Evento)

Mètode que elimina tots els arxius que corresponen a un event. Únicament és necessari l'id de l'event. La sentència sql és la següent:

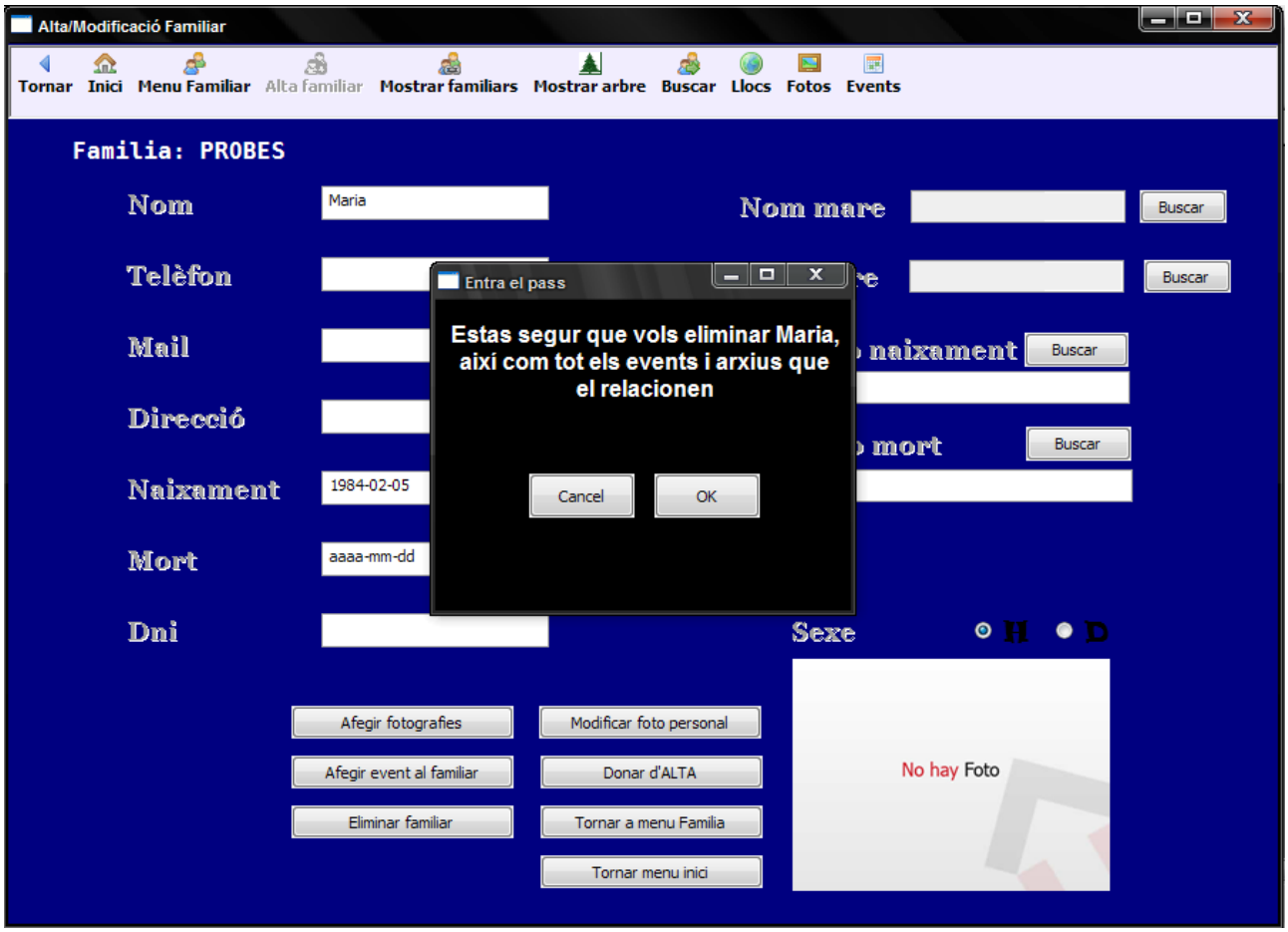
```
s.executeUpdate("DELETE FROM ARXIU WHERE EVENT="+event.getId()+";");
```

Aquest mètode s'utilitza quan desde la interfície s'intenta eliminar un event. Quan a la resposta la contestació és OK, el que s'ha de fer és eliminar tots els arxius de l'event.



6.5.2.12 Mètode eliminarTotsArxius(Familiar)

Aquest mètode, s'assembla molt a l'anterior pero en aquest cas elimina tots els arxius de la bdd pertanyents a un familiar. És cridat, quan s'esborra un familiar; degut a que, en aquest, cas tots els seus arxius han de ser eliminats



La sentència sql és la següent:

s.executeUpdate("DELETE FROM ARXIU WHERE FAMILIAR='"+fam.getBdd()+"';");

6.5.3 BdEvent

Aquesta classe s'utilitza per accedir, modificar, afegir o eliminar la informació ubicada a la taula *event* i *descr*. Els mètodes implementats són:

assignarEvent(Evento, ResultSet)
obtenirEvent(int)
obtenirTipusEvent(int)
estaCasat(String)
obtenirTotsEvents(String)
obtenirEventsLloc(String, String)
obtenirEventsFamiliar(Familiar)
obtenirTotsTipusEvents()
eliminarTotsEvents(Familiar)
eliminarEvent(Evento)
altaNouTipusEvent(String, int)
altaEvent(Evento)

Tots i cada un d'ells són mètodes públics, excepte assignarEvent que és privat ja que l'únic que fa es assignar tots els valors obtinguts en una sentència sql a un event, passat per referència. Procedeixo a explicar el més important de cada un dels mètodes...

6.5.3.1 Mètode *assignarEvent(Evento, ResultSet)*

Aquest mètode únicament assigna els valors que hi ha al ResultSet (que són la solució d'una select) a l'event que es passa. El codi és ben simple:

```
aux.setId(rs.getInt(1));
aux.setDesc(rs.getString(2));
aux.setFam1(rs.getInt(3));
aux.setFam2(rs.getInt(4));
aux.setData(rs.getString(5));
aux.setLloc(rs.getString(6));
```

6.5.3.2 Mètode *obtenirEvent(int)*

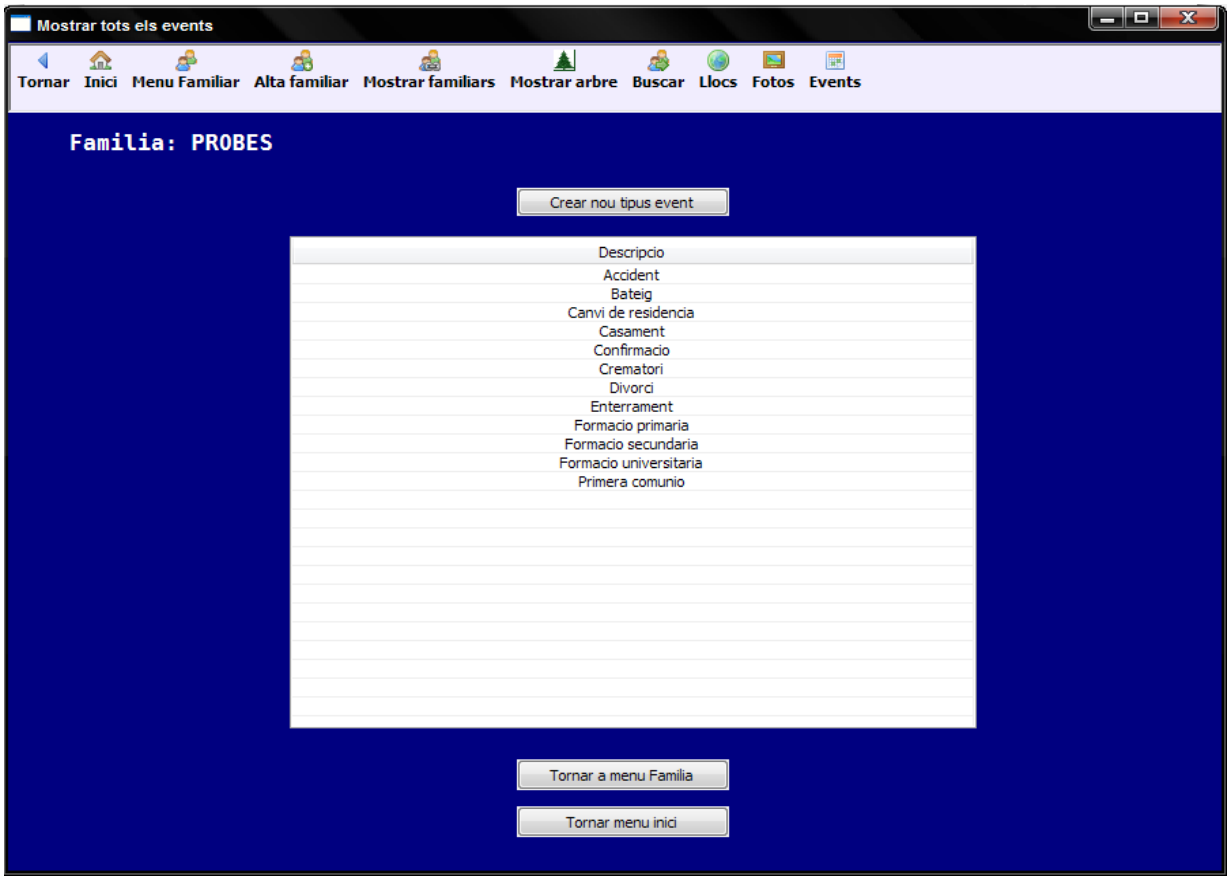
Aquest mètode retorna l'event amb identificador igual a l'enter passat. Del codi només destacar l'accés a la bd:

```
s.executeQuery("SELECT * FROM EVENT,DESCR "+"WHERE EVENT.ID="+ id +" AND EVENT.ID_DESC =DESCR.ID;");
```

6.5.3.3 Mètode *obtenirTipusEvent(int)*

Com s'ha comentat a l'inici hi ha events i tipus d'events. Un tipus d'event és, per exemple: casament, divorci... mentre que un event és un tipus d'event associat a un familiar o event. El mètode anterior permet obtenir un event, mentre que aquest mètode permet obtenir un tipus d'event, segurament per crear una vista o com a filtratge especial. La divisió entre event i tipus, és important especialment per a la creació de diverses vistes com també per l'eficiència que acaba comportant.

A la interfície resultant es mostren tots els tipus d'events; a més de la possibilitat de crear-ne un de nou, com es pot veure a la imatge següent.



Del codi es pot destacar:

s.executeQuery("SELECT * FROM DESCR WHERE ID="+id+";");

6.5.3.4 Mètode *estaCasat(int)*

Aquest mètode comprova si el familiar amb identificador igual al passat per referència està casat, en l'actualitat o no. Per comprobar-ho vaig crear el següent codi:

```
s.executeQuery("SELECT ID_DESC,FAM1,FAM2,DATA FROM EVENT " + "WHERE ID_DESC=5000 OR ID_DESC=5001 AND " + "FAM1='"+aux+"' OR FAM2='"+aux+"' " + "ORDER BY DATA DESC;");
```

És important destacar què fa aquesta sentència. El que es fa és seleccionar els events que tenen com a primero o segon familiar el corresponent amb la variable passada. Llavors selecciono els que tenen com identificador de la descripció el valor 5000(casat) o el valor 5001 que equival a un divorci; ordenat per data descendent.

```
if (rs.next()) {
        if(rs.getInt(1)==5000){
        if(aux==(rs.getInt(2))){
                resultat=rs.getInt(3);
        }else if(aux==(rs.getInt(3))){
                resultat=rs.getInt(2);
        }
}
```

Amb el que primer es comproba que hi hagi algun resultat. En aquest cas, si el primer valor (és a dir l'últim en el temps) equival a casament, significa que actualment està casat (potser ja havia estat casat diverses vegades, pero de la forma que ho he implementat aconsegueixo de forma ràpida conèixer el resultat desitjat.

Com explicaré posteriorment, té relacions diferents a les que hi havia en el passat. En aquest projecte no ha estat tractat de forma explicita totes les unions possibles (per cuestió únicament de temps) pero de forma ímplicita si que és possible qualsevol tipus d'unió. Per tant tot i que el familiar que es passi sigui una dona (familiar.sexe='D'), s'ha de comprobar quin dels dos familiars (fam1 o fam2) es correspon amb la variable passada. Per tant el que es fa es comprobar si es correspon amb fam1 i en aquest cas el resultat és fam2, mentre que si passa el contrari el resultat és fam1.

6.5.3.5 Mètode *obtenirTotsEvents(int)*

Aquest mètode retorna tots els events d'una família. El resultat que es retorna és un vector amb tots els events associats a una família. Per saber-ho es comproba a quina família pertanyen els familiars i així es coneix si l'event es pot afegir al vector resultant.

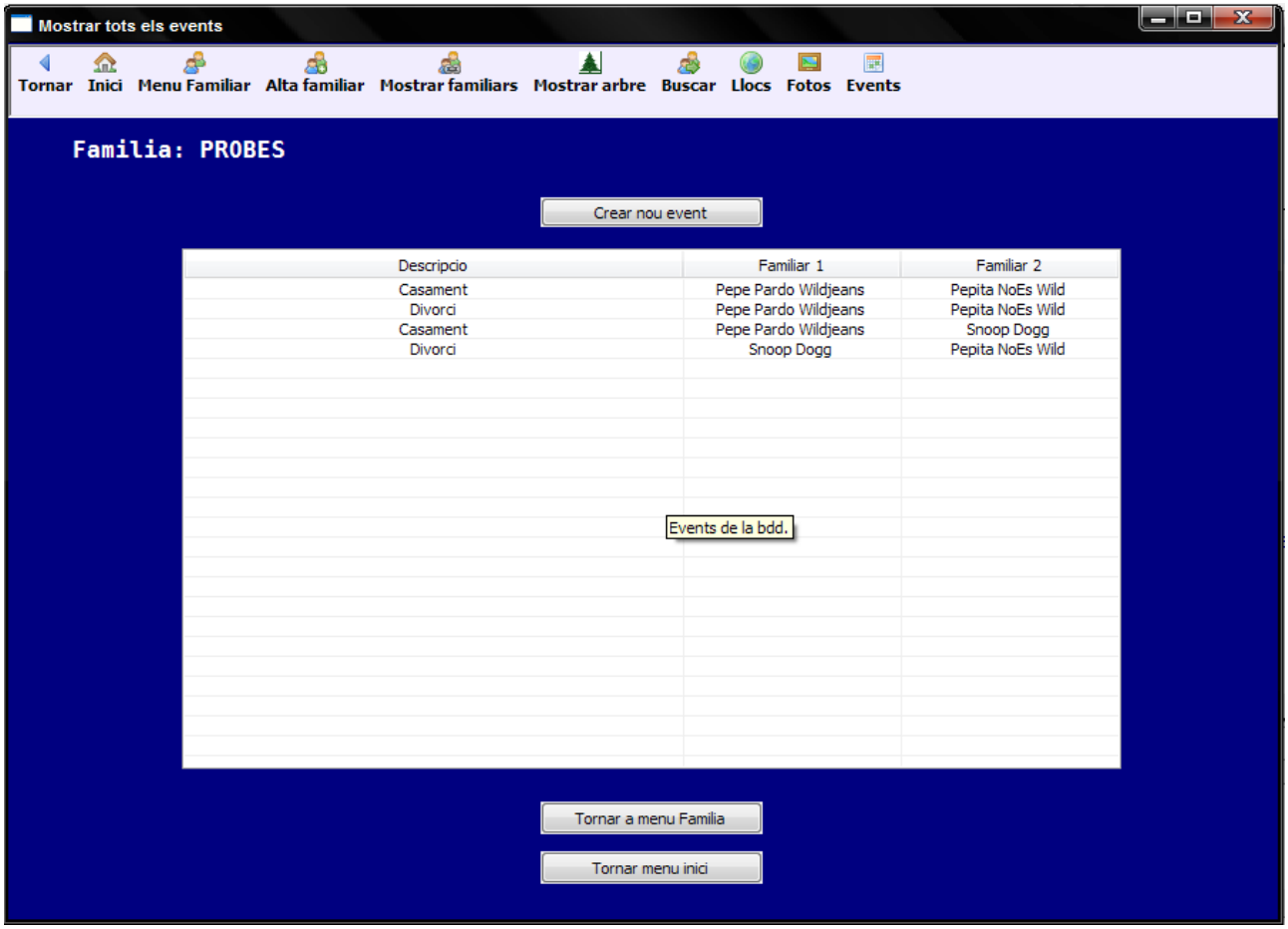
```
s.executeQuery("SELECT DISTINCT EVENT.ID, DESCR, FAM1, FAM2, DATA, POBLACIO, ID_FAMILIA FROM EVENT,DESCR,FAMILIAR WHERE EVENT.ID_DESC=DESCR.ID " +
```

```
"AND FAMILIAR.ID_FAMILIA="+idFam+" AND (FAMILIAR.ID=FAM1 OR FAMILIAR.ID=FAM2)
" +"ORDER BY DATA;");
```

La select és una mica més complicada que els anterior ja que en aquest cas es necessiten 3 taules per obtenir el resultat correcte; tot i que les comprobacions que es fan són únicament per sincronitzar les 3 taules. Finalment l'únicament que es fa es anar afegint els events al vector, per ser usats en la interfície...

```
while (rs.next()) {
    Evento event=new Evento();
    assignarEvent(event,rs);
    tots.addElement((Evento)event);
}
```

La interfície relacionada és la següent:



On es mostren tots els events de la família. La informació que es dona son els dos familiar i la descripció. A més s'ha afegit un botó per permetre afegir un nou event, i no obligar a l'usuari a tenir que navegar fins a trobar una opció, que possiblent hagi d'utilitzar.

6.5.3.6 Mètode *obtenirEventsLloc(int, String)*

Amb aquest mètode s'obtenen tots els events d'una família succeits en un lloc concret. Únicament es crida desde el menú buscar, com veurem després, i la crida a la bdd és la següent:

```
s.executeQuery("select event.id, descr, fam1, fam2, poblacio from event, descr, familiar
where id_desc=descr.id and poblacio='"+lloc+"' and fam1=familiar.id " +
"AND FAMILIAR.ID_FAMILIA="+idFam+" order by data;");
```

Com es pot veure es seleccionen moltes dades, que s'utilitzaran en la interfície, per mostrar de forma detallada els events. La sentència queda filtrada perquè la població i el identificador de la familía, es corresponguin amb els passats per referència. Ja l'últim pas interessant que queda per descriure és l'actualització del vector que es retornarà (vector que per cert conté tots els events que compleixin amb amdues condicions inicials).

```
while (rs.next()) {
        Evento event=new Evento();
        event.setId(rs.getInt(1));
        event.setDesc(rs.getString(2));
        event.setFam1(rs.getInt(3));
        event.setFam2(rs.getInt(4));
        event.setLloc(rs.getString(5));
        tots.addElement((Evento)event);
}
```

6.5.3.7 Mètode *obtenirEventsFamiliar(Familiar)*

Mètode per obtenir tots els events d'un familiar. Crec que és un mètode que s'utilitzarà molt, ja que entenc que és força interessant tant pel mateix familiar (que pot anar afegint events i fotos als mateixos), com pels altres membres; que podran conèixer més detalls i veure fotos.

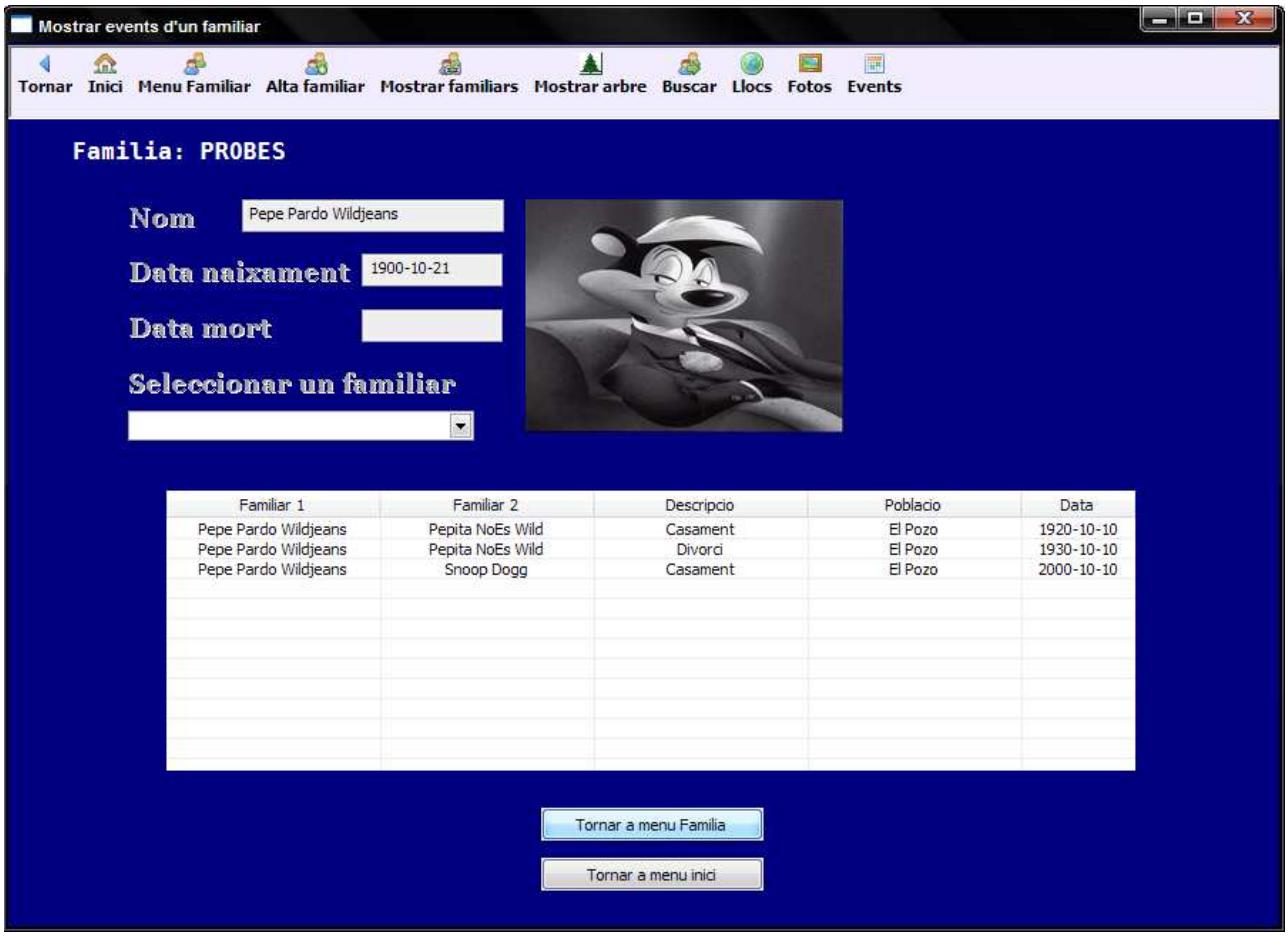
La sentència sql ha quedat implementada de la següent forma:

```
s.executeQuery("SELECT EVENT.ID, DESCR, FAM1, FAM2, DATA, POBLACIO FROM EVENT, DESCR WHERE EVENT.ID_DESC=DESCR.ID AND (FAM1= "+nou.getBdd()+" OR FAM2="+nou.getBdd()+") ORDER BY DATA;");
```

Com en l'anterior es seleccionen pràcticament totes les columnes de les taules i es filtra per l'identificador del familiar. Si els events fossin d'un sol familiar no hi hauria problema, pero com també n'hi ha de dos, he d'anar comprobant que els events del familiar desitjat tinguin el mateix id, en el fam1 o bé en el segon.

Cal recalcar que cabria la opció de col.locar sempre les dones com a familiar 1 o bé els homes com a familiar2 (sempre i quan l'event englobés dues persones). Pel que he comentat abans he preferit no fer-ho ja que, he decidit que la meva aplicació no delimitaria ni matrimonis, ni formes de casament, ni res... per tant com tot això ho deixo a decisió personal, he preferit no fer distincions, ja que ha estat la meva decisió i també, cal dir-ho, no s'ha complicat gairebe gens el codi.

Abans d'analitzar el codi mostraré una de les interfícies que utilitzen aquest mètode, que a part de mostrar les dades del familiar, fotografia personal incluïda, mostra tots els events en el que és protagonista. Afegir finalment, que tot el que apareix a la taula es pot mostrar detalladament.



Per tant l'últim pas després de realitzar la select és afegir els events al vector que es vol retornar i, en cas que no se n'hagi trobat cap, emplenar la primera posició del vector amb la cadena de caracters “Error”, perquè la interfície ho interpreti.

```
while (rs.next()) {
        Evento event=new Evento();
        assignarEvent(event,rs);
        tots.addElement((Evento)event);
        }if (tots.isEmpty()) {
                String e = "Error";
```

```
            tots.add(0, e);
        }
}
```

6.5.3.8 Mètode *obtenirTotsTipusEvents()*

Aquest mètode retorna tots els tipus d'events que hi ha introduïts a la base de dades. Des de l'inici hi haurà guardats una sèrie, tot i que l'usuari (com ha quedat explicat en un dels mètodes anteriors, podrà introduir-ne de nous, per facilitar la interacció amb l'aplicació). Els dos únics detalls a comentar són: la crida a la bdd:

```
s.executeQuery("SELECT * FROM DESCR ORDER BY DESCR;");
```

La crida fa referència a la taula descr i no arxiu. Com s'ha detallat abans, d'aquesta forma es millora tant el volum de dades de les taules com també l'eficiència i senzillesa alhora de realitzar vistes, ja que un dels camps, pels que es poden realitzar-ne diverses ja està dividit de l'altre taula. Per exemple alhora de buscar tots els tipus d'events, és molt més fàcil i eficient, obteir tots els tipus d'event, que no haver de recòrrer tots els arxius i anar mirant que la descripció del mateix no hagi estat ja introduida a la solució.

I l'últim detall a comentar és l'actualització del vector de sortida:

```
while (rs.next()) {
        Evento event=new Evento();
        event.setId(rs.getInt(1));
        event.setDesc(rs.getString(2));
        tots.addElement((Evento)event);
}
```

6.5.3.9 Mètode *eliminarTotsEvents(Familiar)*

En aquest cas s'eliminen tots els events relacionats amb un familiar. Un dels casos en què s'utilitza és quan s'elimina un familiar; ja que en aquest cas s'han de borrar tots els arxius que el relacionen.

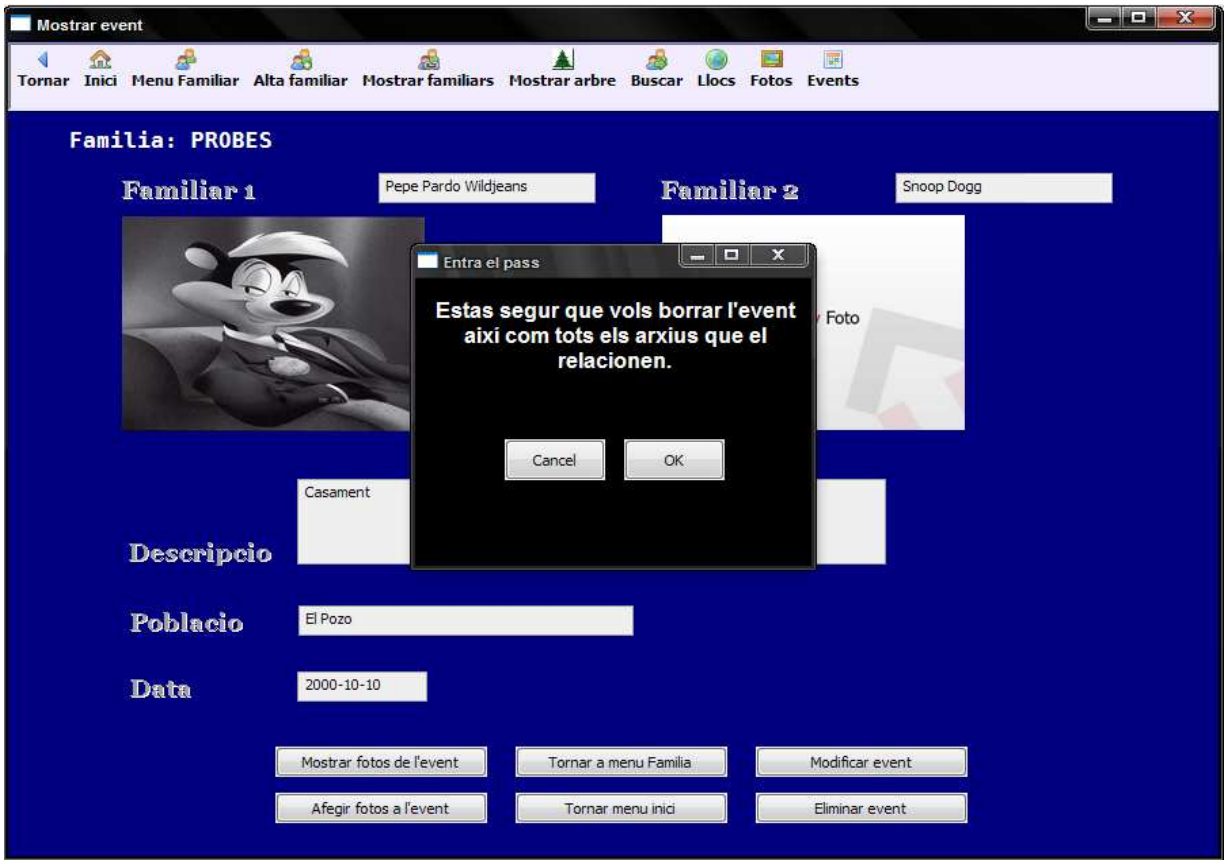
Només cal mostrar la sentència sql, ja que no hi ha cap altra part del codi remarcable:

```
s.executeUpdate("DELETE FROM EVENT WHERE FAM1='"+fam.getBdd()+ "' OR " + "FAM2='" +fam.getBdd()+"';");
```

Com s'ha comentat amb anterioritat s'han de comprobar ambdos camps del familiar.

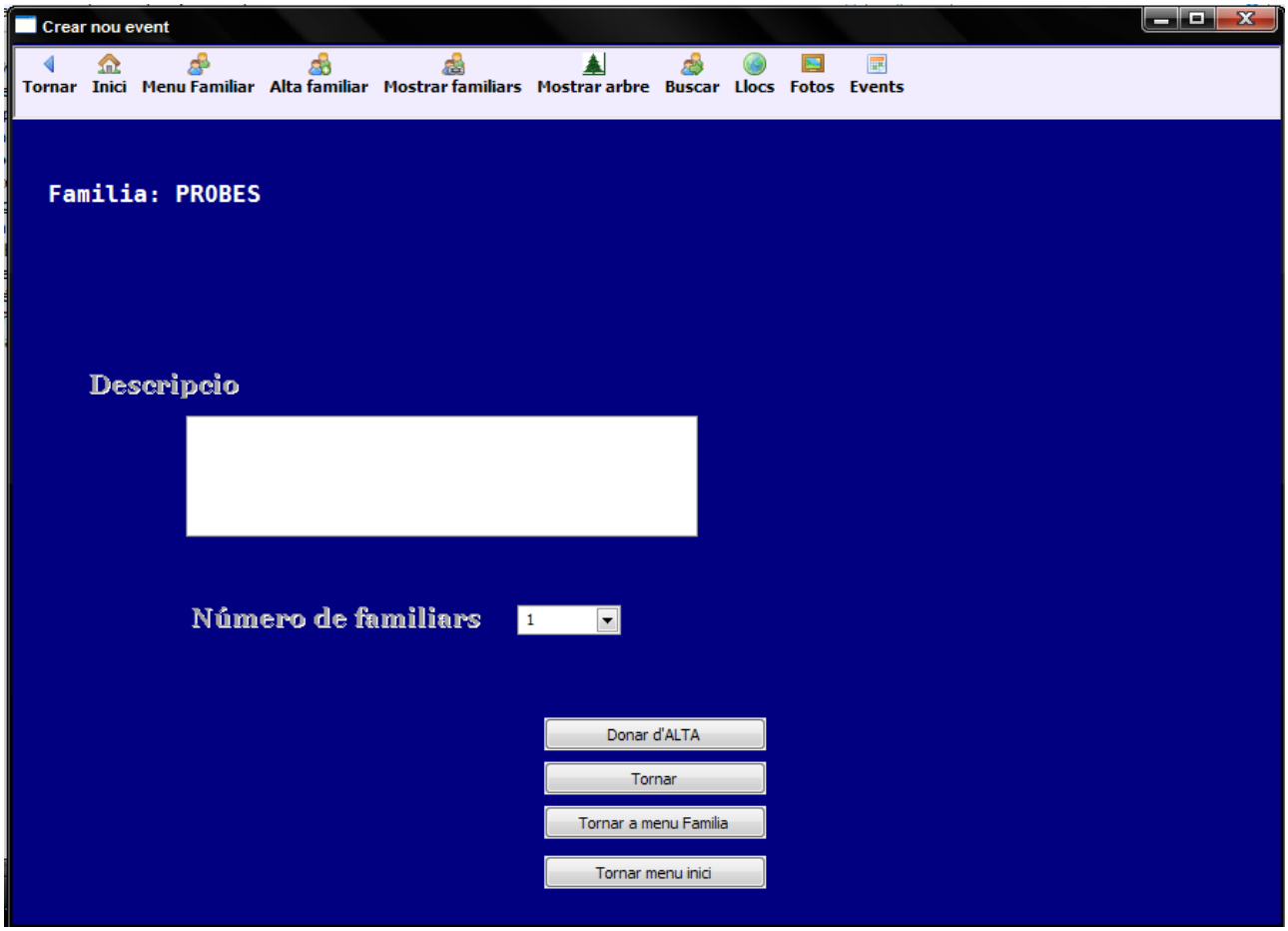
6.5.3.10 Mètode *eliminarEvent(Evento)*

En la interfície que relaciona aquest mètode, un event pot ser eliminat, tot i que abans es mostra un missatge per confirmar-ho. En cas que es confirmi s'executa aquest mètode, del que únicament es pot destacar...



s.executeUpdate("DELETE FROM EVENT WHERE ID="+event.getId()+";");

6.5.3.11 Mètode *AltaNouTipusEvent(String,int)*



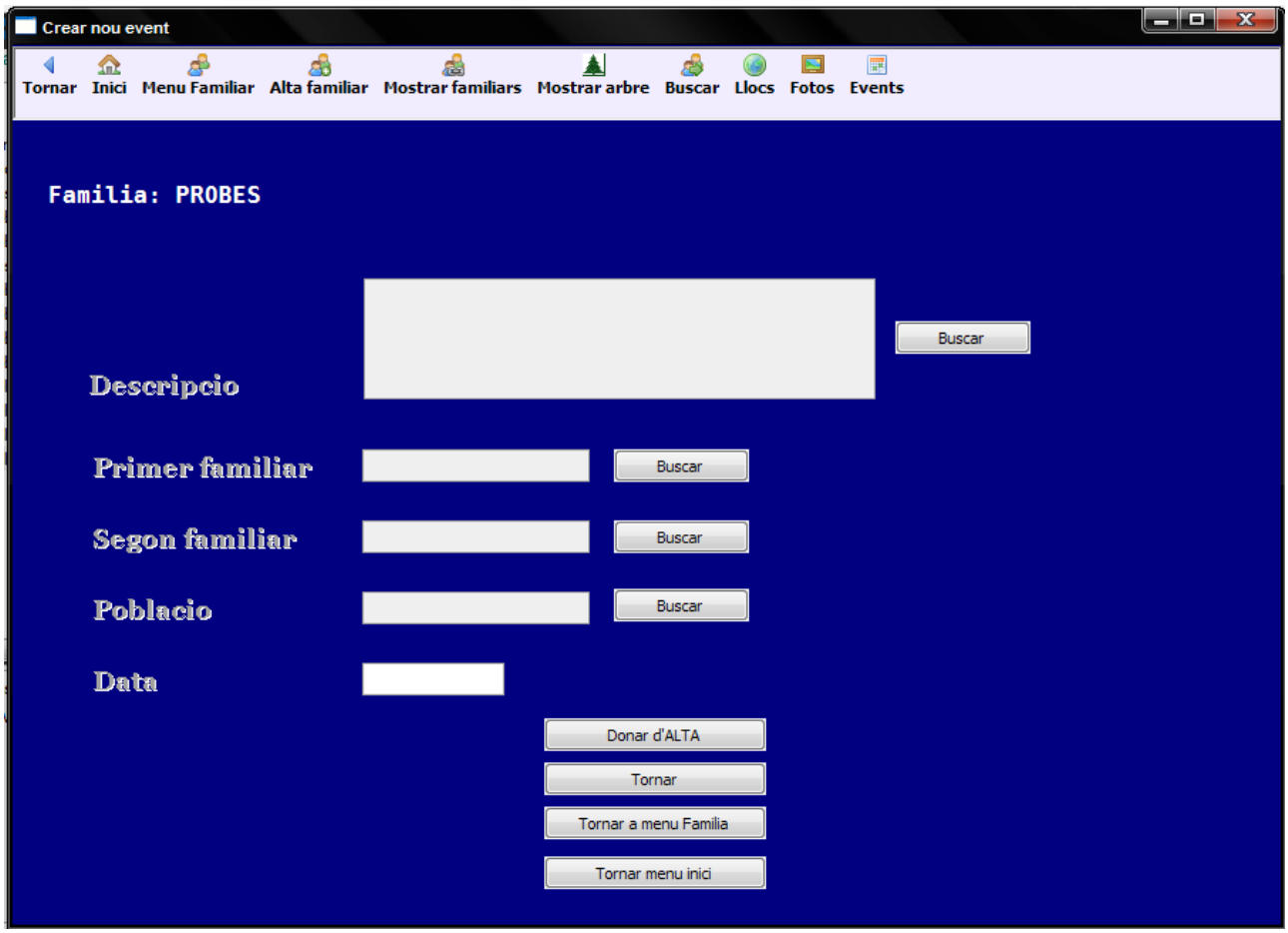
La pantalla pantalla que es pot veure a la fotografia ésforça senzilla. Ambdos camps són modificables (igual que el nombre de variables el mètode). Cada una d'elles correspon a: la descripció del nou tipus d'event (evidentment diferent a les ja introduides a la bdd); i el número de familiars del nou tipus d'event.

Com l'identificador del nou tipus d'event es crea de forma automàtica, el primer que fa aquest mètode és assignar-li, el primer valor lliure. Un cop escollit el valor únicament queda realitzar l'alta del nou tipus d'event a la bdd...

s.executeUpdate("INSERT INTO DESCR VALUES("+maxId+", '" +descripcio+ "'," +fam+ ");");

Tot i tractar-se d'un event, ja havia comentat que havia dividit aquesta classe en dues taules, ja que d'aquesta forma la bdd quedava molt millor estructurada i amb més facilitat per crear vistes diverses.

6.5.3.12 Mètode *altaEvent(Evento)*



Aquesta interfície permet assignar a tots els camps els valors desijtats. Hi ha camps obligatoris com la descripció i un familiar; a més d'una comprobació de tots els camps, fins i tot els no obligatoris com la data. Com en molts casos anteriors, l'identificador es crea de forma automàtica, per això el primer que fa aquesta funció es buscar el primer id lliure i assignar-lo al nou event a afegir. De la forma en la que s'ha implementat el codi, aquest mètode també realitza una modificació d'un event:

```
if(event.getId()>0){
```

```
        s = (Statement) nova.createStatement();
        s.executeUpdate("DELETE FROM EVENT WHERE ID="+event.getId()+";");
        maxId=event.getId();
}
```

Com es pot observar en cas que es passi un event sense dades inicialitzades, el que es fa es crear un nou identificador, mentre que si ja en té significa que el que s'ha de fer és modificar el ja existent; i per fer-ho, s'inicialitza l'identificador amb la clau de l'event passat per referència. Per finalitzar comentar el codi per donar d'alta l'event...

```
if(familiars==1){
        s.executeUpdate("INSERT INTO EVENT VALUES("+maxId+", '"+idDesc+ "',"
        +event.getFam1()+ ","+event.getFam2()+ ",'"+event.getLloc()+ "','"
        +event.getData()+"');");
}else{
        s.executeUpdate("INSERT INTO EVENT VALUES("+maxId+", '"+idDesc+ "',"
        +event.getFam1()+ ","+0+",'"+event.getLloc()+"','"+event.getData()+"');");
}
```

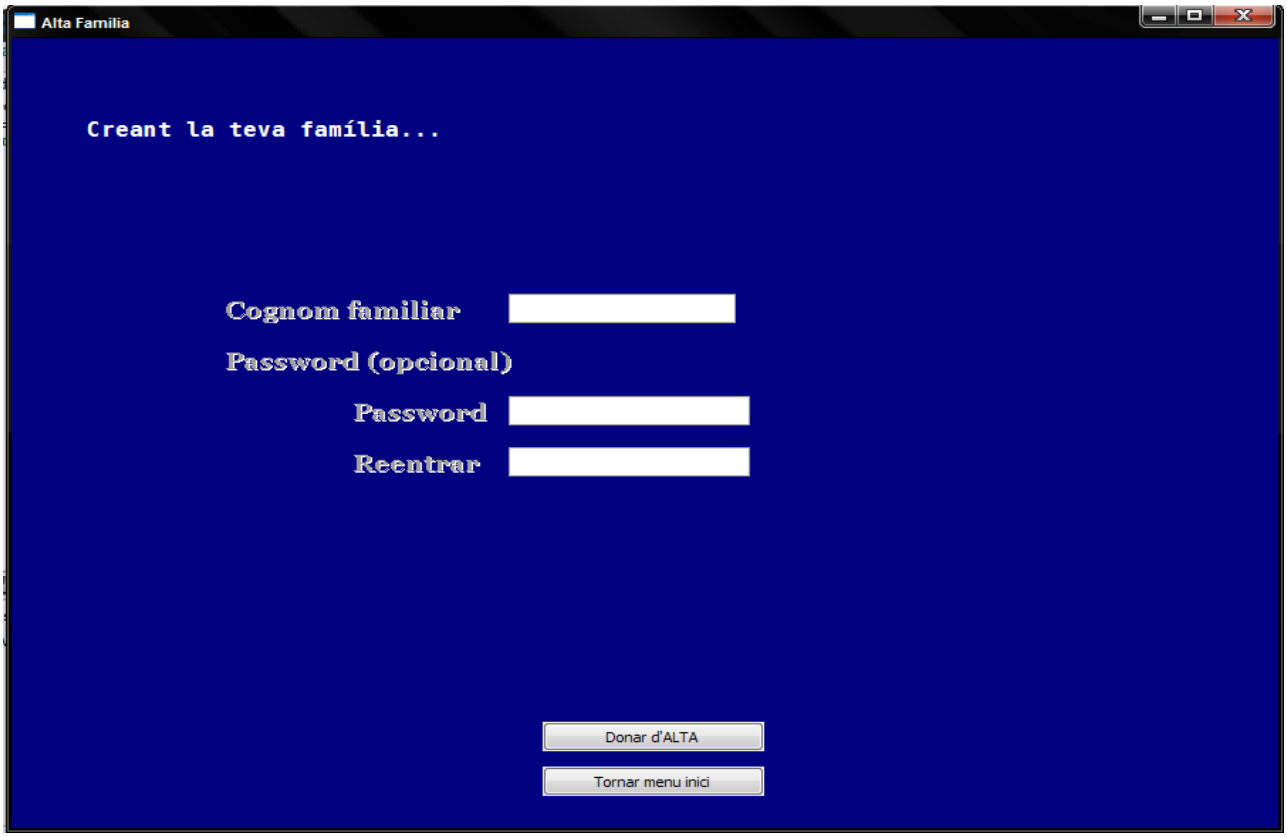
Hi ha dos possibles casos: que l'event només fagi referència a un sol familiar; o bé a dos.

6.5.4 BdFamilia

Aquesta interfície és important ja que divideix els familiars en “grups”. Cal destacar que es permet guardar en un mateix sistema diverses famílies i, per tant, és molt important que ni els familiars, ni la informació dels mateixos es barreji i acabi donant problemes, un d'ells el de la protecció de dades.

6.5.4.1 Mètode *altaFamilia(Familia)*

Evidentment, aquest mètode dóna d'alta una familia a la bdd. La interfície que connecta l'usuari amb la bdd, és la següent:



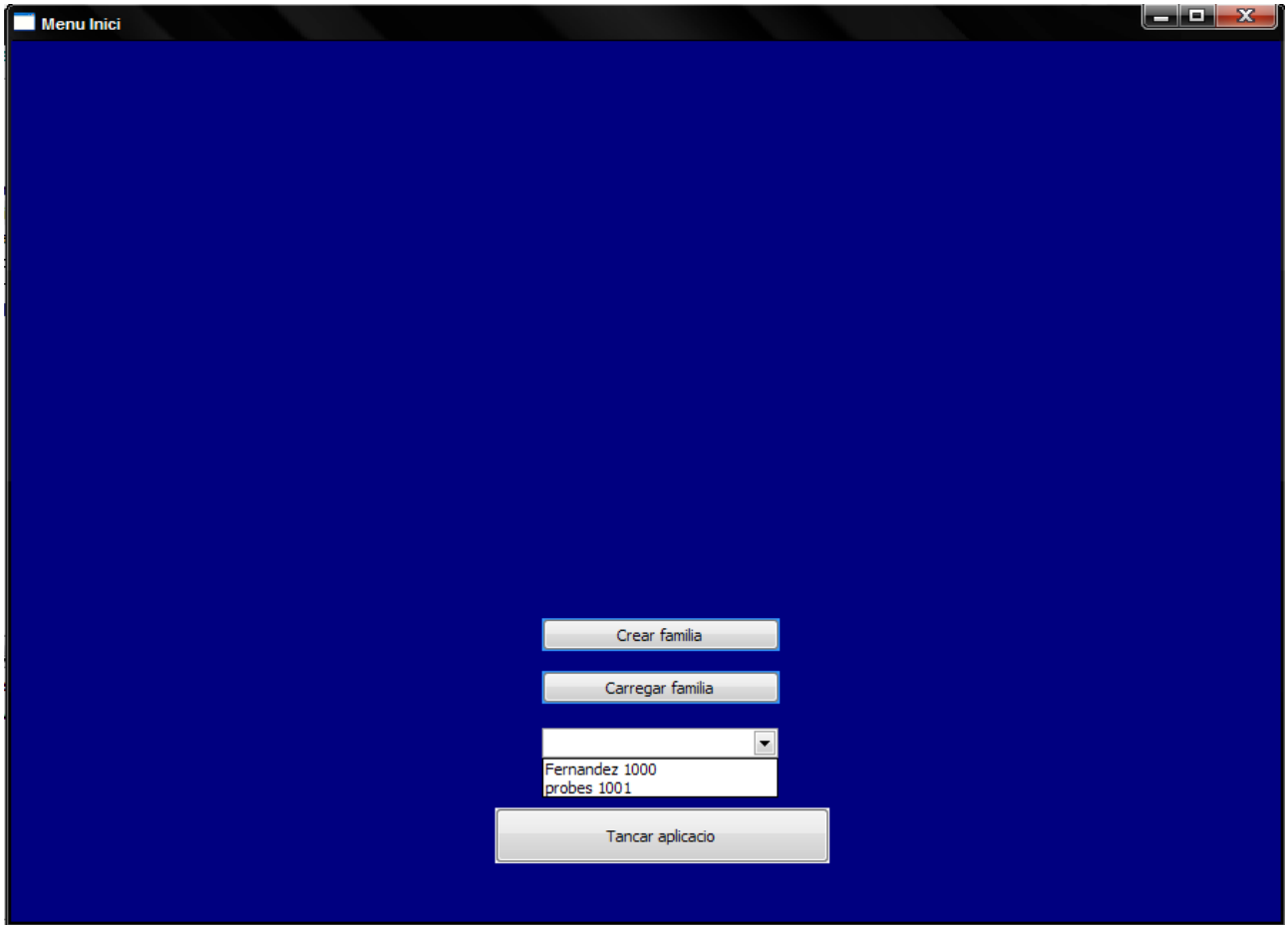
Només hi ha un camp obligatori a emplenar i és el de nom de la família. La clau, com he comentat en diverses ocasions és opcional. Si les dades són correctes i es dóna d'alta, el mètode que estic analitzant, primer li assigna un identificador a la família (la clau primària de les famílies comença a partir de 1000). Finalment un cop assignat, es realitza l'alta amb la sentència:

```
if(pass.length()==0){
        s.executeUpdate("INSERT INTO FAMILIA VALUES("+id+ ",'"+aux.getNom() +
        "',NULL);");
}else{
        s.executeUpdate("INSERT INTO FAMILIA VALUES("+id+",'"+aux.getNom()+"'," +
        aux.getPass()+"');");
}
```

Destacar que s'ha dividit el codi en dues branques, depenent si s'ha introduït password o no.

6.5.4.2 Mètode *getTotesFamilies()*

Aquest mètode s'utilitza únicament a l'inici de l'aplicació



Aquesta ha sigut la forma més senzilla i estètica, en la meva opinió, per mostrar-les pantalla i poder escollir amb aquella que es vulgui treballar.

El mètode que s'analitza, retorna un vector de Families. L'únic que cal remarcar en aquest mètode és l'actualització del mateix:

```
while (rs.next()) {
    Familia c = new Familia(rs.getInt(1), rs.getString(2),rs.getString(3));
    tots.addElement((Familia)c);
}
```

6.5.4.3 Mètode *existeixFamiliaNom(Familia)*

Aquest mètode únicament comproba que la família existeixi. S'utilitza en poques ocasions, i únicament serveix com a referència; m'explico, aquest mètode comproba per nom que n'exisiteixi una, per tant si hi ha dos famílies *Hernández* el mètode retornarà cert, pero no la família desitjada.

6.5.4.4 Mètode *familiaProtegida(Familia)*

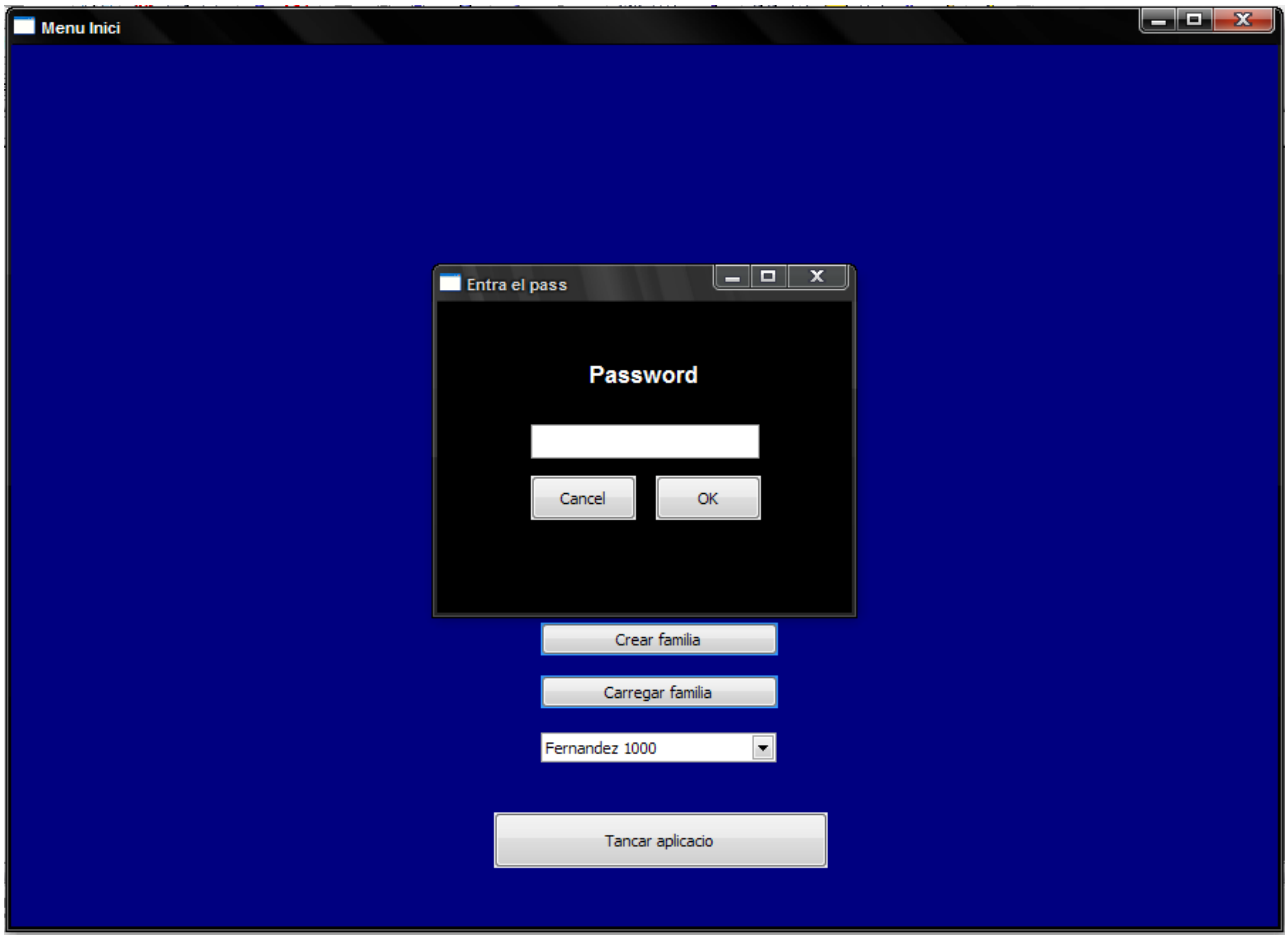
Aquest mètode comproba si la família passada per referència està protegida amb una contrasenya o no. El valor retornat és un enter amb valor: 1 en cas que si que ho estigui i 0, en cas contrari.

6.5.4.5 Mètode *obtenirIdFamilia(String)*

Mètode que s'utilitza en molt poques ocasions. A l'inici de l'aplicacio era molt usat, fins que vaig permetre afegir famílies amb el mateix nom en un mateix sistema. Des d'aquell moment i amb la classe dades la seva utilitat és pràcticament nul.la. Encara que en alguns moments i sempre que només existeixi una sola família amb un nom, és utilitzable amb bon resultat.

6.5.4.6 Mètode *comprobarContrasenya(Familia, String)*

Aquesta funció retorna un cert/fals en funció que la contrasenya de la Familia es correspongui amb l'String. En cas afirmatiu es carregarà la família, mentre que en cas negatiu es demanarà el password, com es pot observar a la foto.



6.5.5 BdFamiliar

Aquesta és la classe amb més mètodes; això es deu a que la classe familiar és la que més atributs té, i per tant la que necessita més accions per actualitzar la seva informació. El llistat dels procediments implementats són...

assignarFamiliar(Familiar, ResultSet)
obtenirFamiliarId(int)
existeixFamiliar(Familiar)
existeixFamiliarNom(String, Familiar)
obtenirNomFamiliar(int)
obtenirId(String)
crearId()
obtenirFamiliarNom(String, String)
obtenirTotsFamiliars(String)
altaFamiliar(Familiar, String)
eliminarFamiliar(Familiar)
eliminarModFamiliar(Familiar)
modificarFamiliar(Familiar, String)
arrelsCompletes(String)
familiarXedad(int, Integer, Integer, String)
obtenirFills(Familiar)
obtenirFamiliarsData(String, String, String)
obtenirGermans(Familiar)

assignarFamiliar(Familiar, ResultSet)

Aquest és l'únic mètode privat de tots els implementats en aquesta classe. El que fa és assignar tots els varlors del resultset al familiar. Hi ha una sèrie de comprobacions com:

```
if(rs.getString(5)!=null){
        nou.setDMort(rs.getString(5));
}else{
        nou.setDMort("");
}
```

6.5.5.1 Mètode *obtenirFamiliarId(int)*

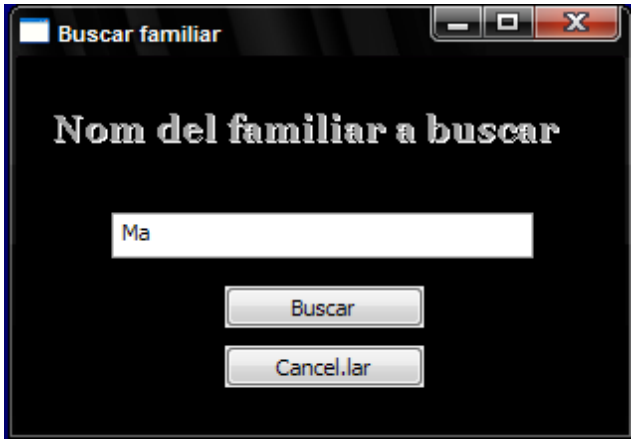
Mètode que obté el familiar a partir del seu identificador. En aquest, com en molts altres, s'utilitza *assignarFamiliar(Familiar,Resultset)*.

6.5.5.2 Mètode *existeixFamiliar(Familiar)*

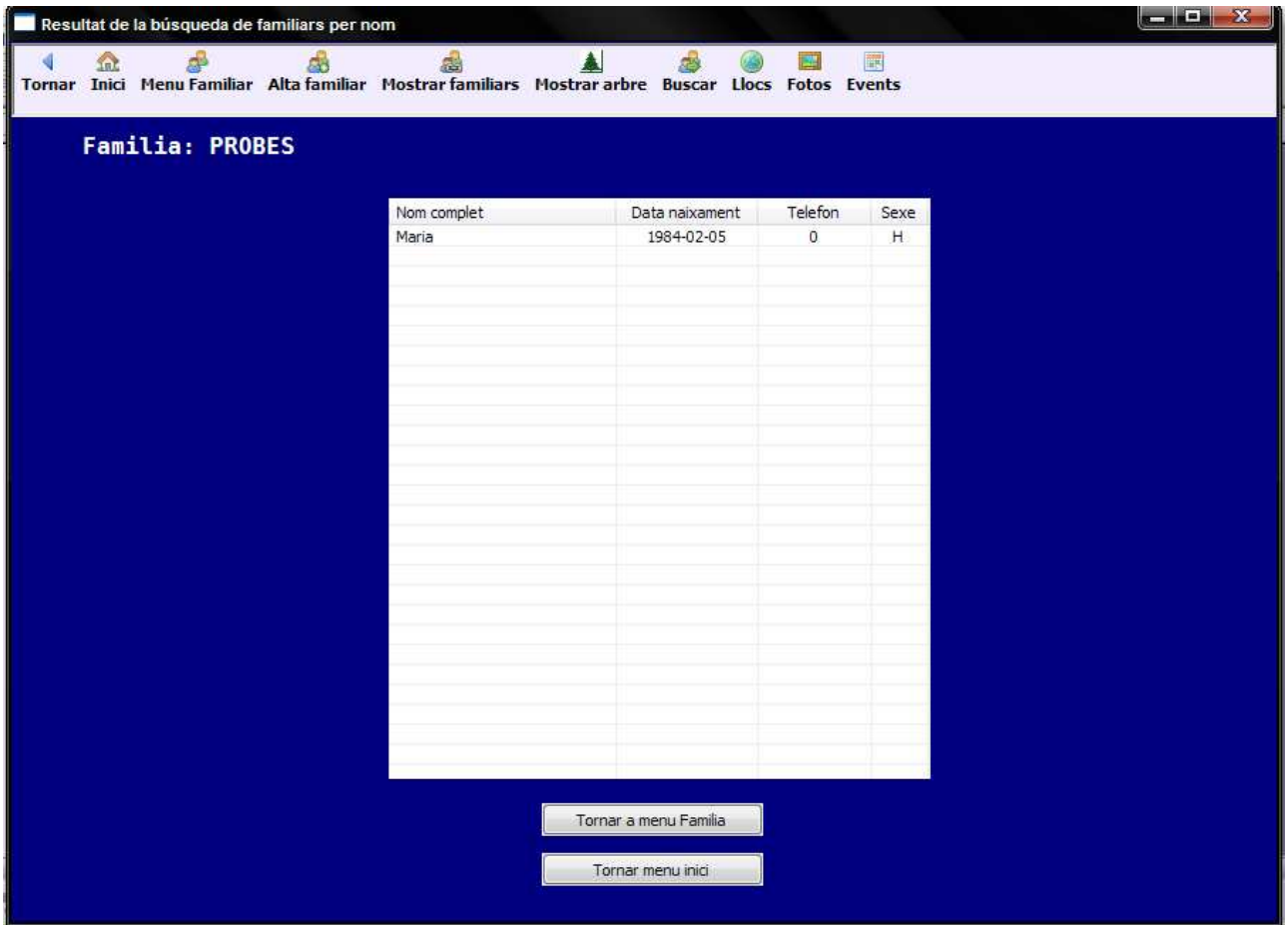
Es comproba si el familiar existeix. L'únic que es “pot” comprobar és l'identificador; ja que com he dit a l'inici pràcticament tots els seus atributs són repetibles: nom,telèfon,mail,... Retorna cert o fals, en funció de la seva existència.

6.5.5.3 Mètode *existeixFamiliarNom(String, Familiar)*

Aquesta acció retorna tots els familiars, d'una mateixa família, amb el nom igual a l'String passat per referència. L'exemple d'aquest mètode és el següent...



Aquí s'introdueix la seqüència a buscar; el resultat és el que es pot veure a continuació:



Com es pot observar, no cal introduir el nom complet per obtenir el resultat. Cap la possibilitat de que els resultats sigui múltiple, per això hi ha una taula que s'emplenarà amb tots els familiars.

La select que recupera, de la base de dades, tots els familiars que comencin per l'String passat és:

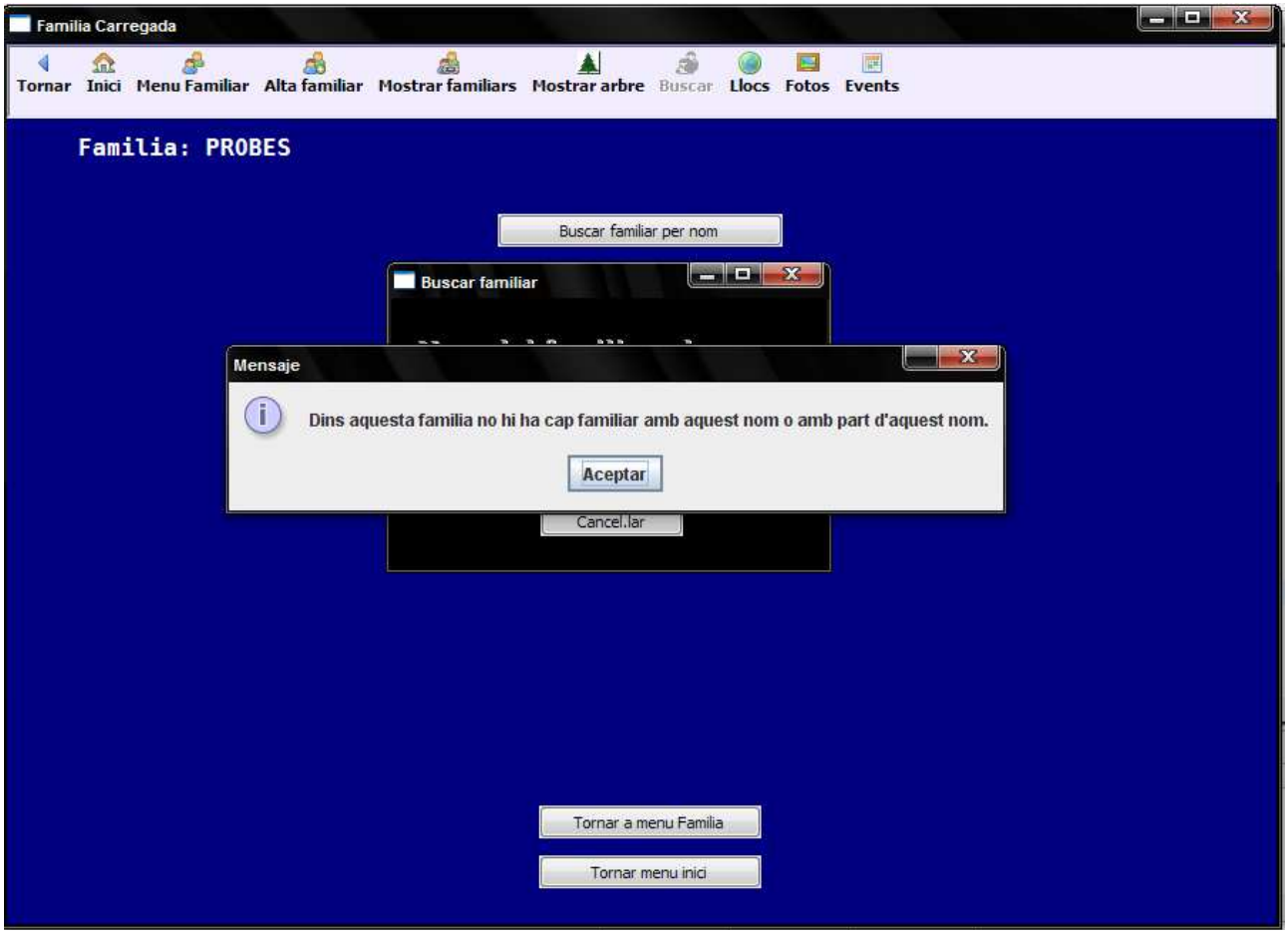
```
s.execute("SELECT COUNT(*) FROM FAMILIAR,FAMILIA " WHERE (SUBSTR(FAMILIAR.NOM, 1, "+nou.getNom().length()+  ")='"+nou.getNom()+"') AND  FAMILIAR.ID_FAMILIA =FAMILIA.ID AND FAMILIA.NOM='"+nomFam+"';");
```

El que es fa es comproba que l'string passat sigui igual que el camp nom de la base de dades desde la posició 1 fins a la posició igual a la llargada de l'String. El codi a més permet introduir el nom complet (nom i cognoms), i d'aquesta forma obtenir un resultat més reduït. Amb aquesta acció es retorna el número de familiars que compleixen la premisa.

6.5.5.4 Mètode *obtenirFamiliarNom(int, String)*

Acció on es passen dues variables; la primera és l'identificador de la família; mentre que la segona és la cadena de caràcters amb la que es desitja buscar el nom dels familiars. Aquesta acció retorna un vector amb tots els familiars amb nom igual a l'String passat.

Sembla il.lògic tenir el mètode anterior, que només retorna el número de membres, mentre que aquest segon retorna tots els familiars. He decidit divdir-ho en dos mètodes perque no és necessari canviar de pantalla si no hi ha cap resultat; és a dir si l'usuari introdueix una seqüència que no concorda amb cap nom dels familiars, llavors no cal mostrar una pantalla buida sinó un missatge com el següent...



Conclusió: el mètode que retorna un enter l'utilitza la pantalla que crea la pantalla resultat, mentre que la interfície resultant crida el mètode que retorna el vector. En cas que la primera crida retorni 0 es mostrarà el resultat de la foto anterior, mentre que en cas contrari la nova interfície, cridant el mètode que obté tots els familiars, els mostrarà.

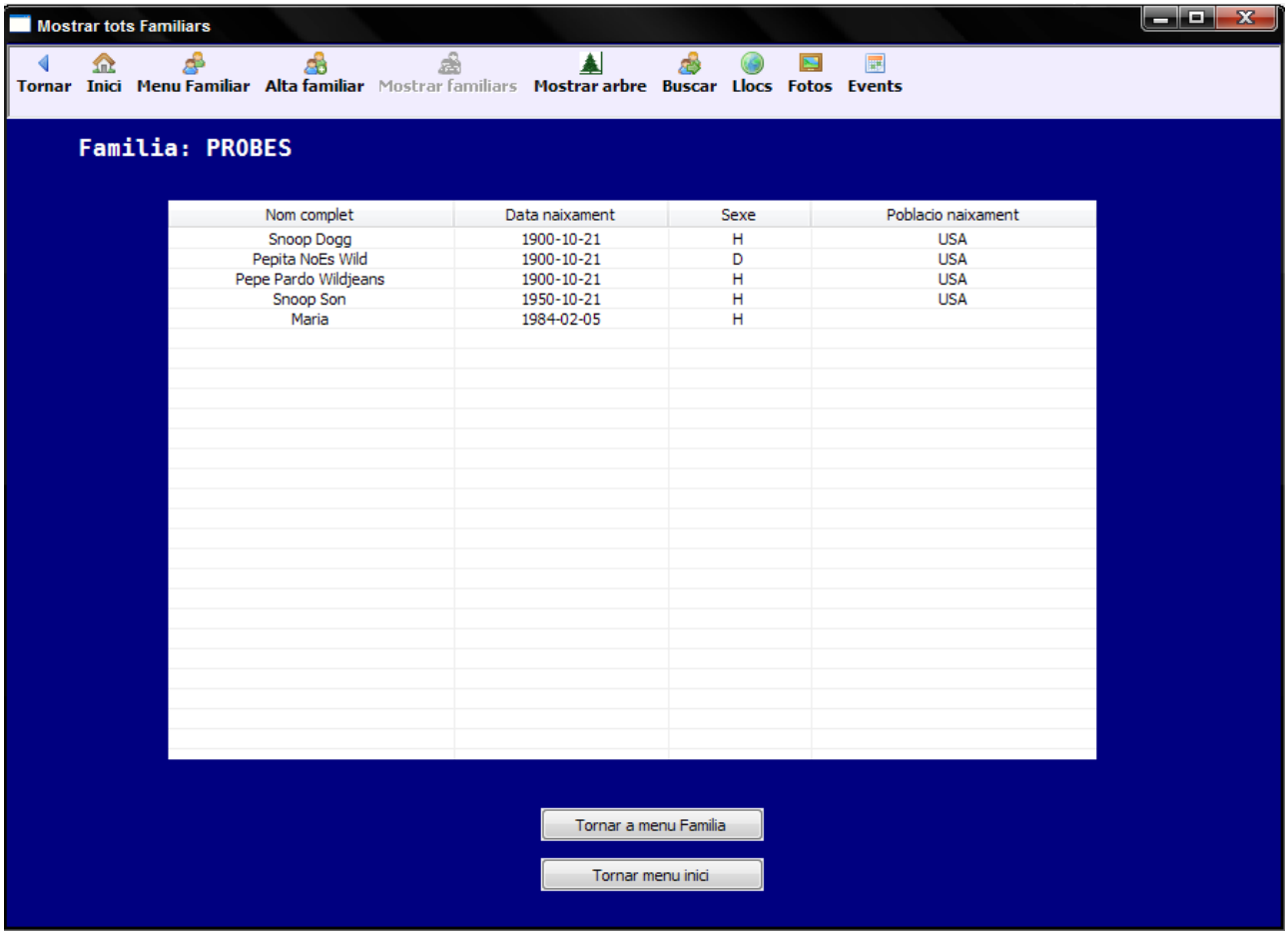
6.5.5.5 Mètode *obtenirNomFamiliar(int)*

Mètode que obté el nom d'un familiar a partir del seu id. Aquest mètode s'utilitza a les interfícies que utilitzen el nom i que no necessiten cap més atribut del membre. En aquest últim cas (són necessaris diversos atributs del familiar) es crida un altre mètode **obtenirFamiliarId(int).**

6.5.5.6 Mètode *obtenirId(String)*

S'utilitza per obtenir l'id d'un familiar a partir del seu nom. Aquest mètode no és gaire usat i la única possibilitat d'utilitzar-lo, és quan només hi ha un familiar amb aquell nom, perque si n'hi hagués diversos, el resultat podria no ser el desitjat.

6.5.5.7 Mètode *obtenirTotsFamiliars(String)*



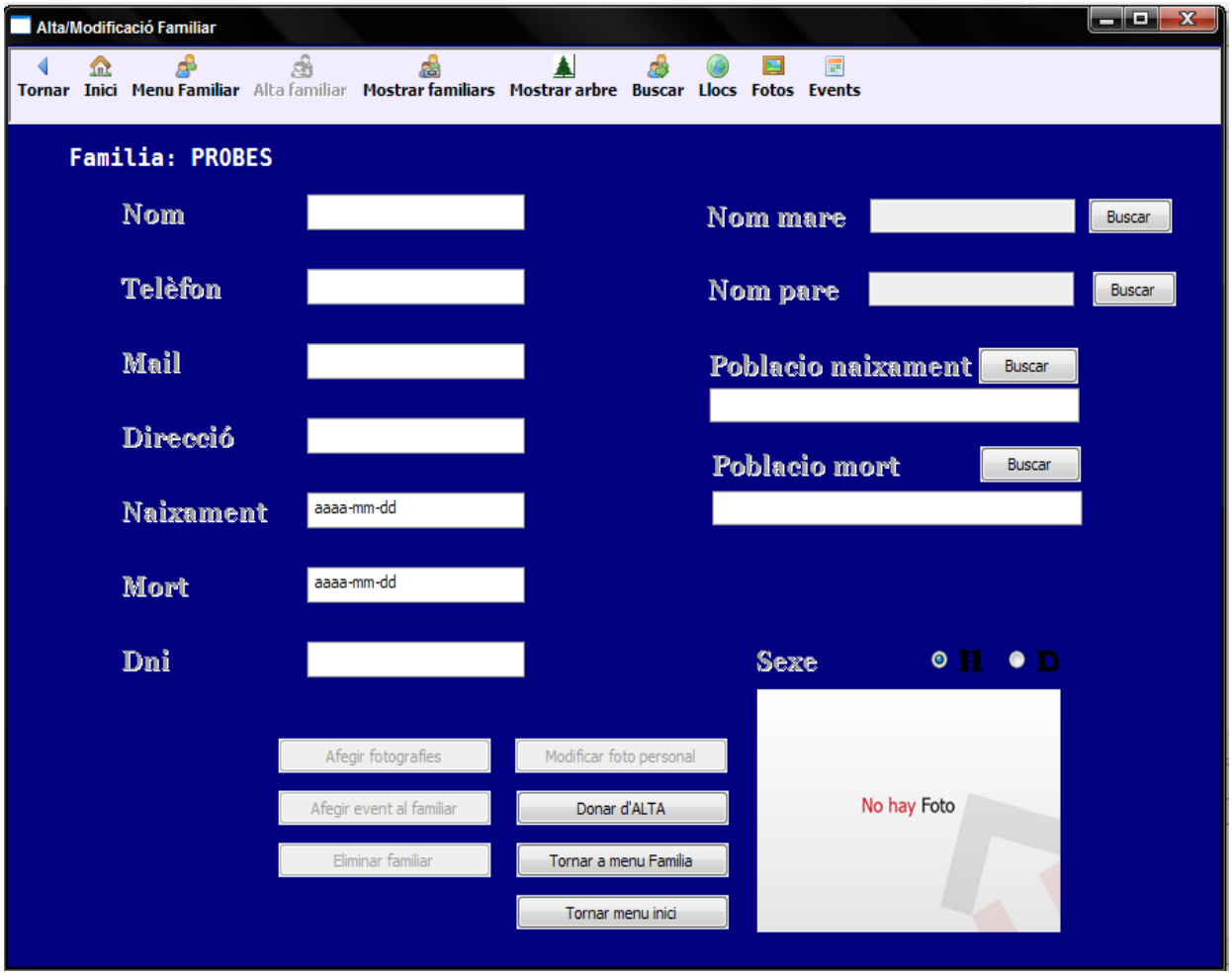
Com s'observa, a més del nom hi ha una sèrie de dades que permetran a l'usuari, seleccionar inequivocament al familiar; només amb el nom, en cas que pare i fill s'anomenin igual, podria

haver-hi problemes d'incongruència de les dades.

L'únic que he tingut en compte alhora de realitzar la select, és la ja esmentada particularitat que poden haver-hi diverses famílies, i per tant alhora de seleccionar tots els familiars s'ha hagut de condicionar amb la família a la que pertanyen. El resultat és un vector de familiars que es mostren a la interfície anterior, i que clicant-hi a sobre és mostren de forma detallada.

6.5.5.8 Mètode *altaFamiliar(Familiar, int)*

Mètode molt important que permet afegir nous familiars a la base de dades. Com he comentat anteriorment, la classe Familiar té molts atributs i aquest procediment ha sigut difícil de tractar ja que s'han hagut de tenir moltes variables en compte. La interfície que el tracta és la



següent...

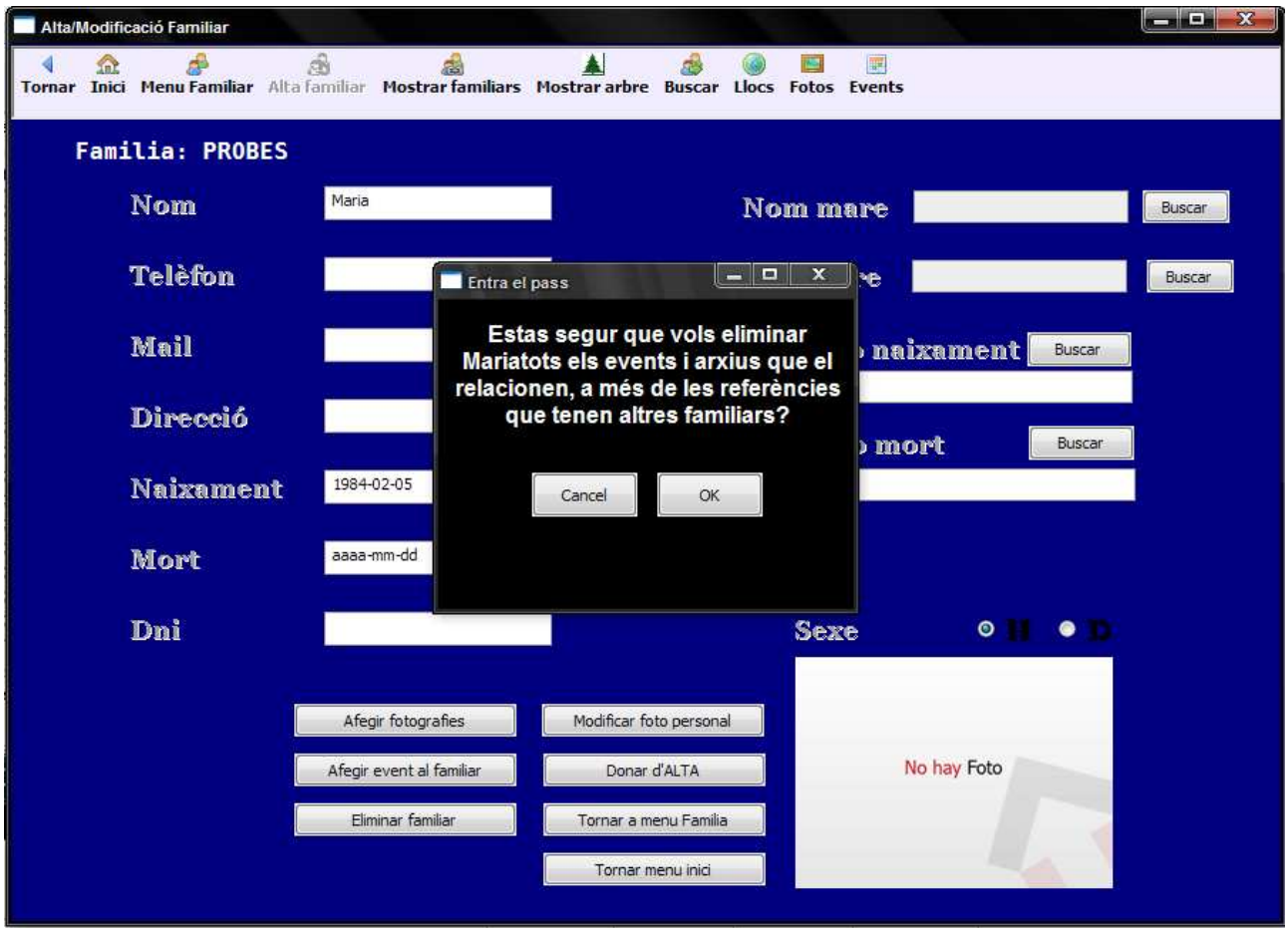
Com es pot veure a la imatge, hi ha molts camps a emplenar, tot i que molt pocs són obligatoris. Pel que fa al codi destacar que primer li assigno un valor a l'identificador del nou familiar. El pas final és realitzar l'alta, en la que hi ha una diferència de branques segons si el familiar és mort o no

```
if(b){
        s.executeUpdate("INSERT INTO FAMILIAR VALUES("+      idf+              ",'"
+nou.getId()+     "',"+idFamilia+",'"+nou.getDNaix()+"','"+nou.getDMort()+"','"
+     nou.getNom()+              "',"+nou.getTelefon()+",'"+nou.getMail()+"','"+
        nou.getAdreça()+ "','" +nou.getLloc()+"' ,'"+nou.getLlocMort()+
        "','"+nou.getIdPare()+"','"+nou.getIdMare()+"','"+nou.getSexe()+"');");
}else{
        s.executeUpdate("INSERT INTO FAMILIAR VALUES(" +idf+"      ,'"+nou.getId()+
        "',"+idFamilia+",'"+nou.getDNaix()+"',NULL,'" +nou.getNom()+"','"
        +nou.getTelefon()     +",'"+nou.getMail()+"','"+     nou.getAdreça()+"','"
        +nou.getLloc()+"',NULL,'"+nou.getIdPare()+   "','"+nou.getIdMare()+   "','"
        +nou.getSexe()+"');");
}
```

Perque en cas que no estigui mort les variables població i data de mort, han de ser introduïdes amb els valors neutres a la bd.

6.5.5.9 Mètode *eliminarFamiliar(Familiar)*

Mètode utilitzat per eliminar un familiar de la base de dades. És un procés més complicat del que pugui semblar ja que, no només borra el familiar de la taula, sinó que a més ha d'eliminar tots els arxius que el relacionen, així com els seus events i també totes les referències que tinguin altres familiars (altres membres el poden tenir com a pare o mare) i també quedaran borrats. Per tant és un pas que l'usuari ha de tenir clar i que intento deixar-ho pal.lès com mostra la següent pantalla:



Pel que fa al codi: l'eliminació del familiar és el següent:

```
sdel.executeUpdate("DELETE FROM FAMILIAR WHERE ID="+nou.getBdd()+";");
```

Després s'esborren tots els arxius i events.

```
GestioArxiu ga=new GestioArxiu();
ga.eliminarTotsArxius(nou);
GestioEvent ge=new GestioEvent();
ge.eliminarTotsEvents(nou);
```

I per acabar s'eliminen les referències que tenen altres familiars:

```
s.executeQuery("select id from familiar where id_familia="+nou.getIdFam()+
"and (id_mare='"+nou.getNom()+"' or id_pare='"+nou.getNom()+"');");
rs = (ResultSet) s.getResultSet();
while(rs.next()){
        Familiar aux=new Familiar();
        aux.assignar(obtenirFamiliarId(rs.getInt(1)));
        if(aux.getIdPare()==rs.getInt(1)){
                smod.executeQuery("UPDATE FAMILIAR SET ID_PARE=0 WHERE
                ID="+rs.getInt(1));
        }else{
        smod.executeQuery("UPDATE FAMILIAR SET ID_MARE=0 WHERE ID="+rs.getInt(1));
}
```

Únicament es seleccionen aquells familiars amb *id_pare* o *id_mare* igual al id del familiar a eliminar; i llavors alhora de reassignar-los l'id comprobo de quin dels dos es tracta.

6.5.6 BdLloc

Classe que gestiona els “llocs” (poblacio, regió, estat). Aquesta classe és molt usada, ja que tant familiars (lloc naixament i mort); com fotos, com events l'utilitzen. He cregut molt important ubicar els objectes en llocs, per donar tota la informació possible. Els mètodes desenvolupats són:

```
obtenirLlocs()
obtenirLloc(String)
obtenirProvincies()
obtenirEstats()
obtenirEstatPerProvincia(String)
obtenirIdProvincia(String)
```

existeixPais(Lloc)

existeixProv(Lloc)

existeixPob(Lloc)

altaLloc(Lloc)

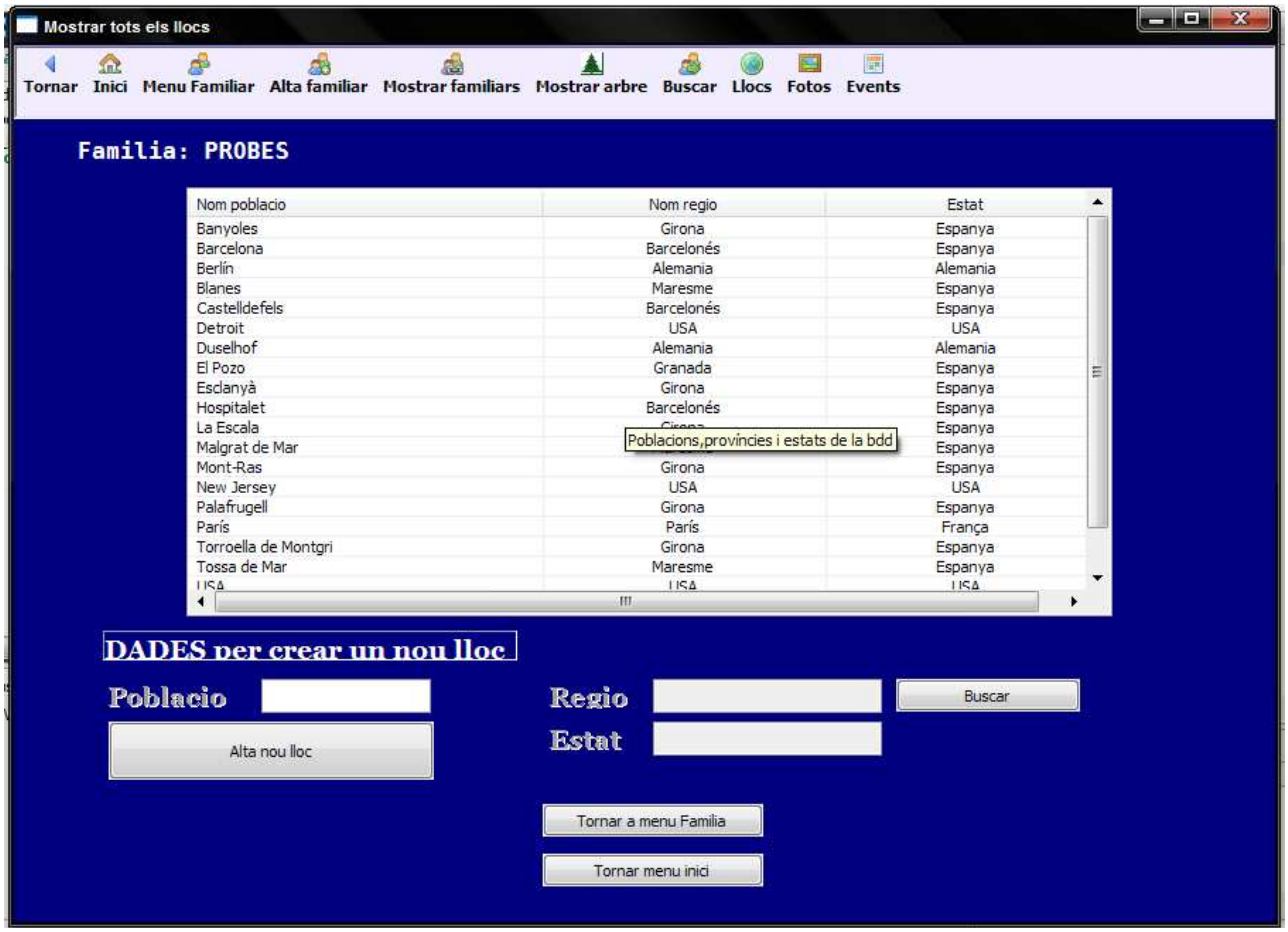
altaRegio(Lloc)

altaEstat(String)

modLloc(Lloc)

6.5.6.1 Mètode *obtenirLlocs()*

Mètode que retorna un vector amb els llocs guardats a la bdd. La interfície que l'usa és la següent:



6.5.6.2 Mètode *obtenirLlocs(String)*

Aquesta acció retorna únicament el lloc, on la població té el nom igual a l'String passat.

6.5.6.3 Mètode *obetnirEstats()*

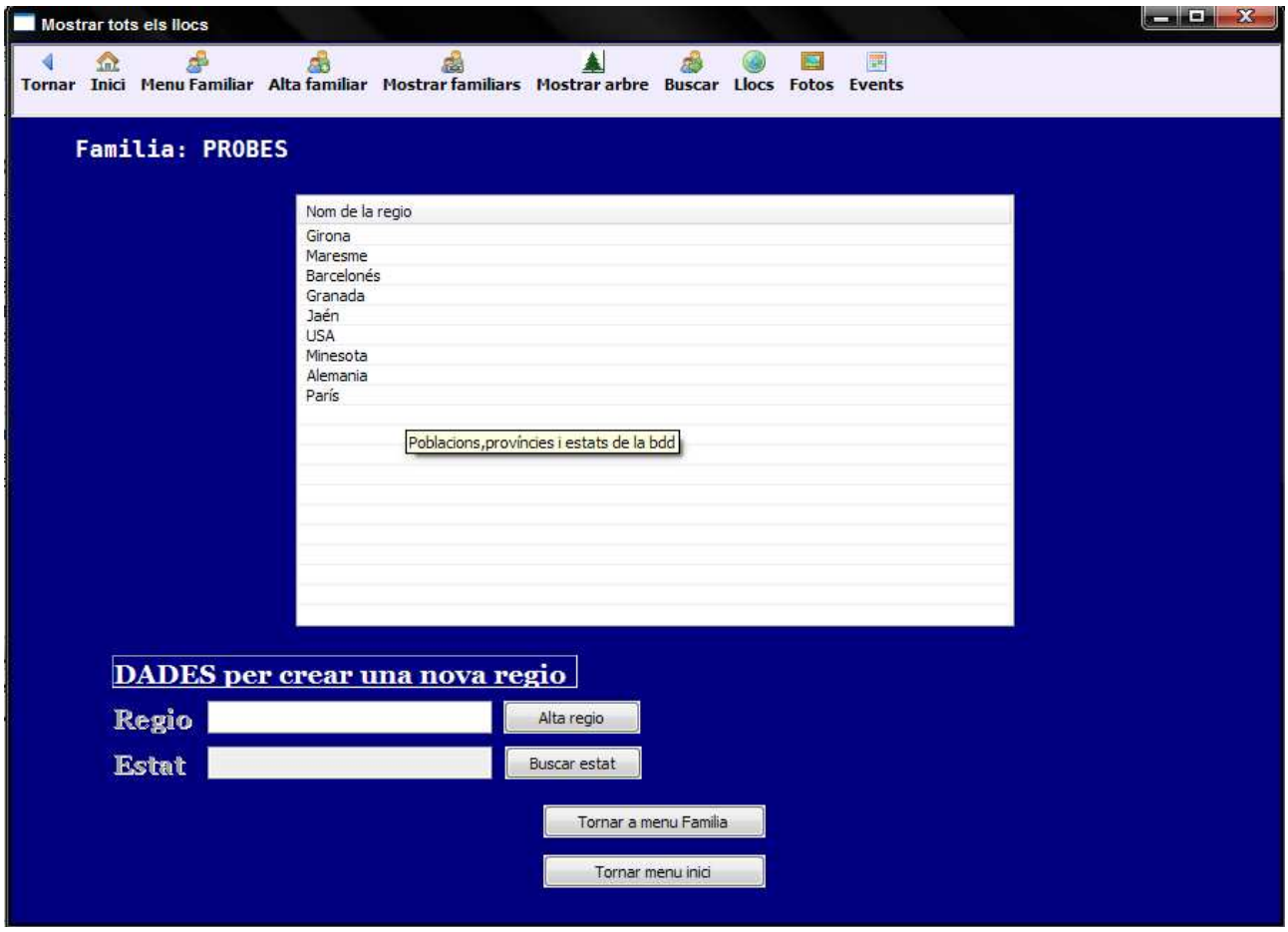
Retorna tots els estats guardats ala bdd.

6.5.6.4 Mètode *obtenirEstatPerProvincia(String)*

Aquesta acció retorna el país d'una regió; s'utilitza quan es vol donar d'alta un nou lloc. El primer que cal fer és obtenir l'identificador de l'estat, a partir de la regió i llavors, amb aquesta dada es pot conèixer el nom de l'estat.

6.5.6.5 Mètode *obtenirProvincies()*

Retorna totes les regions guardades a la bdd. Exemple:



6.5.6.6 Mètode *altaLloc(Lloc)*

Aquest mètode si que té certs aspectes interessants. Per donar d'alta un lloc s'han d'actualitzar tres taules: població, regió i estat. Per tant el que s'ha de fer alhora d'afegir un nou lloc és afegir la població (si és possible, és a dir que no exisiteixi).

```
s1.executeUpdate("INSERT INTO POBLACIO VALUES('"+lugar.getNom()+"',"+idProv+" "
+","+idEstat+"); ");
```

El següent pas si s'ha donat d'alta, si no existeix la regió s'ha de donar d'alta la regió:

```
s2.executeUpdate("INSERT INTO PROVINCIA VALUES(" +idProv+ ",'"+
lugar.getProvincia() +"',"+idEstat+");");
```

I finalment, si s'ha donat d'alta la regió, afegir l'estat (en cas que no exisiteixi).

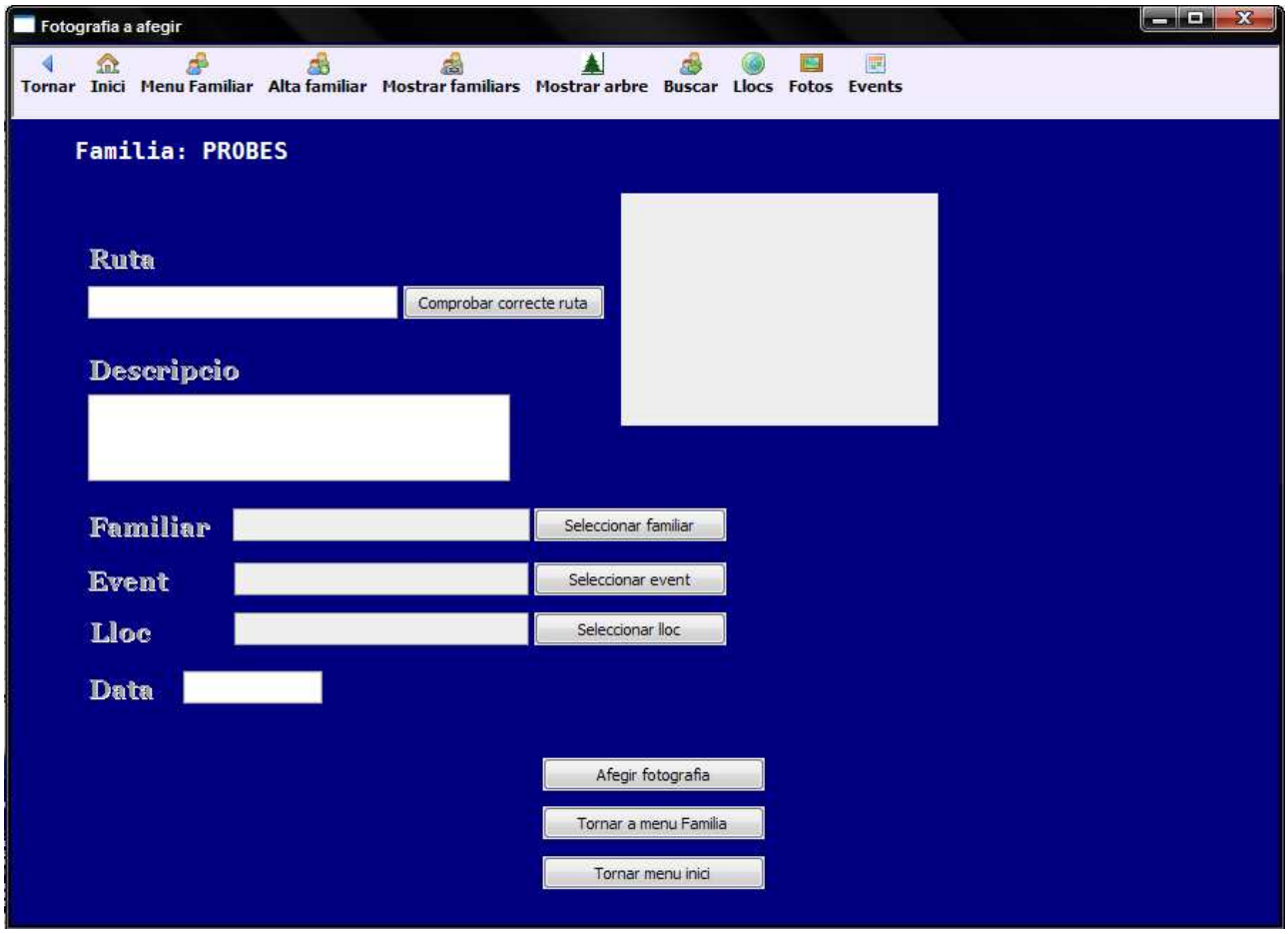
```
s.executeQuery("SELECT ID FROM ESTAT WHERE NOM='"+lloc.getPais()+"';");
```

També hi ha la possibilitat, evidentment, d'afegir regions o estats, de forma separada. En cas que es vulgui afegir una regió, el mètode implementat és pràcticament idèntic a l'últim mètode explicat, sense afegir poblacions. Hi ha altres mètodes en aquesta classe pero son molt intuitius, i quedaran explicats més endevant, durant l'exposició de les interfícies.

6.6 Definició de les classes interfície

6.6.1 Mètode AfegirFotos

Aquesta interfície permet a l'usuari inserir fotos a familiar o events; i és la següent:



Disposa, evidentment, de tots els camps que hi ha a la taula Arxius (excepte el tipus i l'id) que s'assignen de forma automàtica.

El primer camp que apareix és el de la ruta. Si aquest no s'emplena, i s'intenta donar d'alta el resultat és un missatge d'error per pantalla:

Remaracar que al costat de la ruta (s'ha d'introduir la ruta desde la carpeta imatges,

anteriorment explicada) hi ha el boto comprobarCorrecteRuta, que permet visualitzar la foto. És la manera que he trobat per poder pal.liar el no tenir un explorador (ho vaig intentar posar pero, vaig tenir moltes dificultats i ho vaig deixar pel final, en cas de tenir temps, i com es pot veure no n'he tingut). En cas que la ruta sigui correcta es mostrarà la foto redimensionada, mentre que si no es mostrarà una imatge:

El següent camp és la descripció, que també és obligatori. Podria existir la teoria que no ho hauria de ser, però la meva opinió és que alhora de mostrar l'àlbum de fotos és molt més interessant amb informació extra. D'aquesta forma tant l'usuari com qualsevol familiar al veure-la tindrant una informació molt interessant.

Els tres camps següents funcionen de forma molt semblant. No es poden introduir dades directament, sino que s'han de seleccionar d'una llista. En el cas d'events i llocs, és possible que desde la mateixa llista es pugui inserir un nou objecte, en cas que no es trobi el que es desitja. En el cas de familiar, no ha estat possible degut al volum de dades i en cas que es vulgui seleccionar un familiar no introduït, s'haurà de crear-lo primer.

Quan es selecciona un familiar, lloc o event aquest queda directament guardat dins la classe **dades**, d'aquesta forma quan es retorna a la classe, les dades queden a punt per ser carregades. He implementat dos mètodes per poder carregar les dades:

```
private void crearArxiu(Arxiu arxiu,Familiar nou){
        arxiu.setUbicacio(textRuta.getText());
        arxiu.setFamiliar(nou.getBdd());
        arxiu.setLloc(textLloc.getText());
        if(textData.getText().length()==0){
                arxiu.setData("aaaa-mm-dd");
        }else{
                arxiu.setData(textData.getText());
        }
        arxiu.setDescr(textDescr.getText());
        arxiu.setTipus(2);
}
```

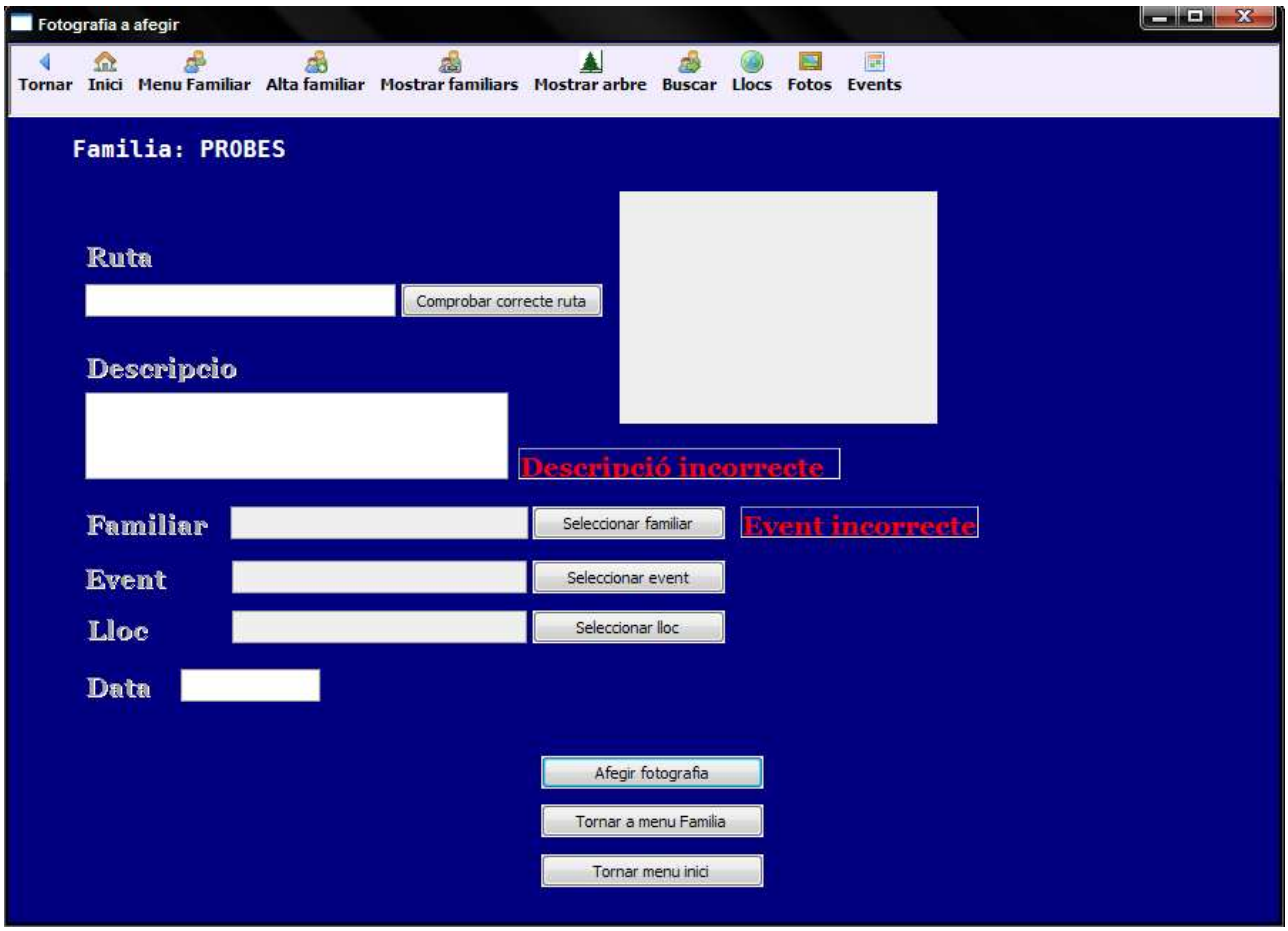
Aquest primer guarda les dades entrades per pantalla i d'aquesta forma al tornar les tindré:

```
private void carregarArxiu(Arxiu arxiu,Familiar nou,Evento event){
        try{
                textRuta.setText(arxiu.getUbicacio());
                textDescr.setText(arxiu.getDescr());
                if(event.getId()>0){
                        textEvent.setText(event.getDesc());
                        textData.setText(event.getData());
                        textLloc.setText(event.getLloc());
                        GestioFamiliar gf=new GestioFamiliar();
                        Familiar familiar=new Familiar();
                        familiar=gf.obtenirFamiliarId(event.getFam1());
                        textFamiliar.setText(familiar.getNom());
```

```
            }else{
                textData.setText(arxiu.getData());
                textLloc.setText(arxiu.getLloc());
                if(!nou.getNom().equals("")){
                    textFamiliar.setText(nou.getNom());
                }
            }
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
}
```

Mentre que aquest segon recupera les dades guardades amb anterioritat, i les que s'han modificat for a de l amateixa classe.

Finalment hi ha el camp data que permet afegir-ne una. És important remarcar que en cas que en cas que no s'empleni un dels camps obligatoris, es mostrarà un misssatge per pantalla a més d'un missatge d'error dins la mateixa pantalla...

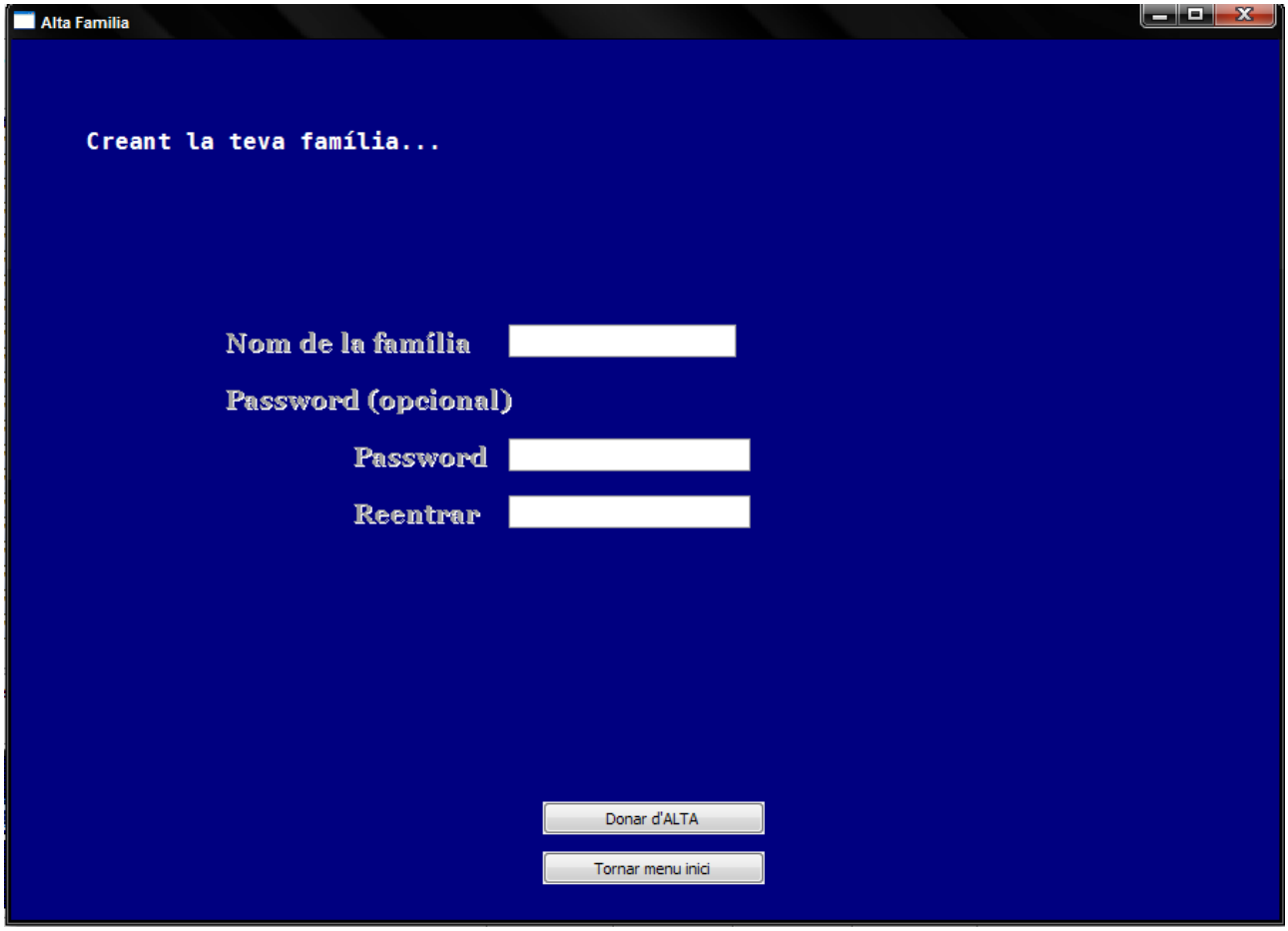


D'aquesta manera l'usuari veurà directament quina informació li queda per emplenar.

En la interfície també s'observa una barra, una mica a l'estil d'un navegador. Aquesta és la mateixa per a totes les interfícies i pertany a una altra classe que s'explicarà posteriorment. Gràcies a aquesta barra o menú de navegació i als dos botons de la part inferior, aconsegueix que l'usuari pugui redireccionar-se ràpidament a la pantalla desitjada (un dels objectius que volia aconseguir alhora de crear el programari).

Per finalitzar aquesta pantalla m'agradaria explicar tres coses: la primera que encara que tots els camps no siguin obligatoris, en el cas que s'afegeixin dades (com ara la data) aquestes són comprobades i en cas de no ser vàlides s'informarà a l'usuari. I segon aspecte important, que si es selecciona un event (el familiar, l'event, i el lloc i la data, sempre que dins l'event tinguin valor) quedaran inicialitzades. D'aquesta forma encara es treurà més feina a l'usuari. I la tercera i última i també important, és que a aquesta interfície s'hi pot arribar per diferents camins, per tant és important saber-ho i tractar cada un dels possibles cassos de forma diferent. Gràcies a la classe dades, tinc tota la informació que necessito tant per accedir a aquesta, com per tornar a l'anterior sense problemes.

6.6.2 Mètode AfegirFamília



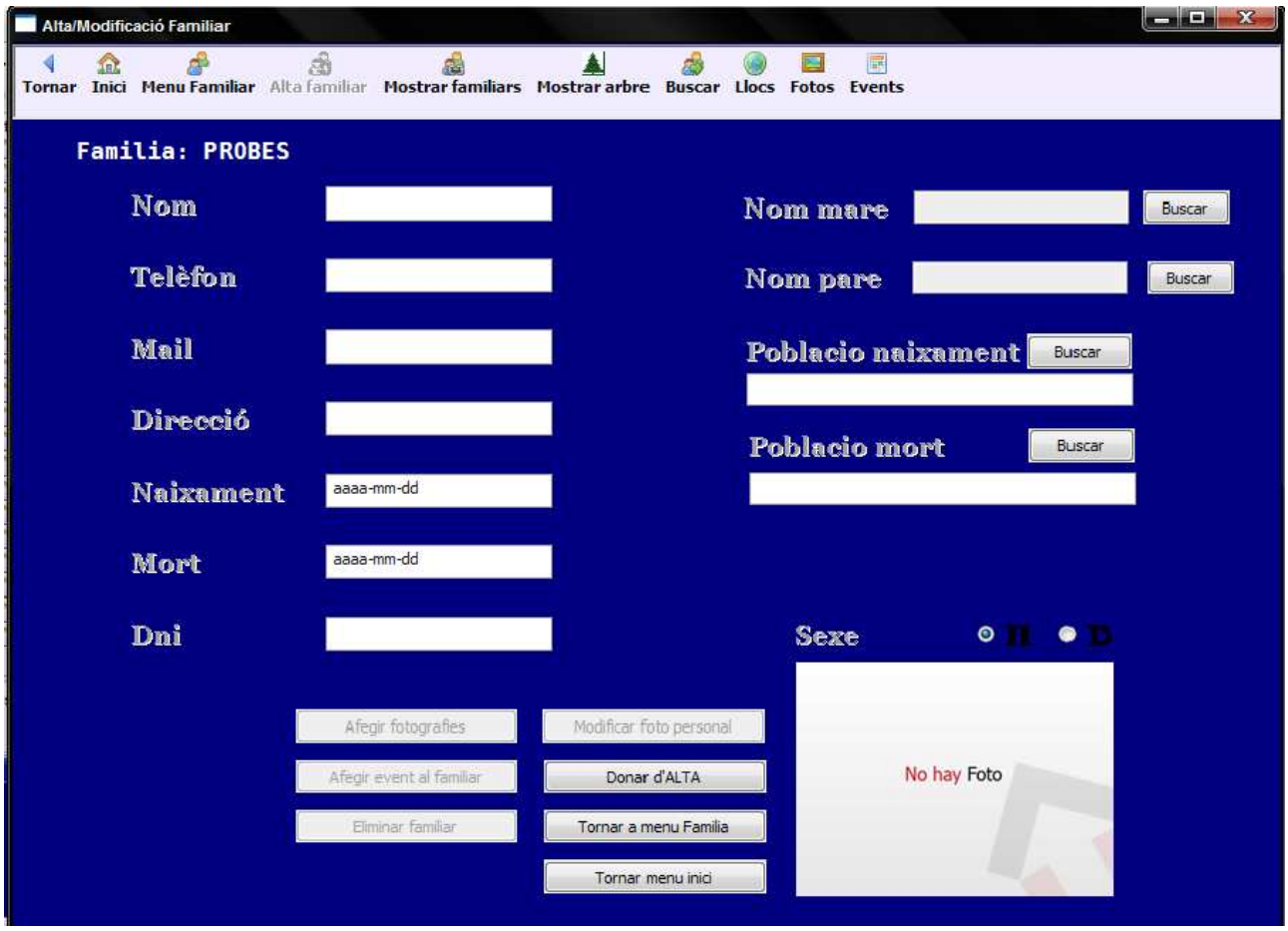
Aquesta interfície és força senzilla i aquest és el seu aspecte. El primer que cal esmentar és que aquesta no té la barra de navegació. Això és degut a que aquesta només es carrega quan l'usuari ja ha seleccionat la família i la està modificant o visualitzant la informació que conté.

Només hi ha tres camps: el primer, que és obligatori, és per insertar el “nom de la família”. Normalment, crec, s'hauria d'introduir el cognom familiar o bé, en cas que existeixi, el “mot” que té la família (molt utilitzat al sud España, on totes les famílies se les coneix per un mot assignat forces generacions passades). I els dos últims són per insertar una clau, sempre que es vulgui protegir la família. No són obligatoris, pero evidentent si es vol protegir la família el valor dels dos camps ha de ser igual, sino es mostra un missatge d'error per pantalla:

Per acabar amb aquesta classe m'agradaria comentar dues coses. La primera que és possible introduir diverses famílies amb el mateix nom, ja que l'aplicació només utilitza l'id (generat de forma automàtica) per realiltzar les operacions. Segon i últim, que l'aspecte que tenen les pantalles pot ser i segurament ho serà, diferent en altres ordinadors. La particularitat d'haver programat l'aplicació amb SWT permet que l'aspecte de les pantalles, sigui el que els usuaris tenen predefinits per defecte. Aquest aspecte important, almenys estèticament, funciona també en qualsevol SO, adoptant sigui quin sigui la forma de les finestres, que es tingui escollit.

6.6.3 Mètode AltaFamiliar/ModificacioFamiliar

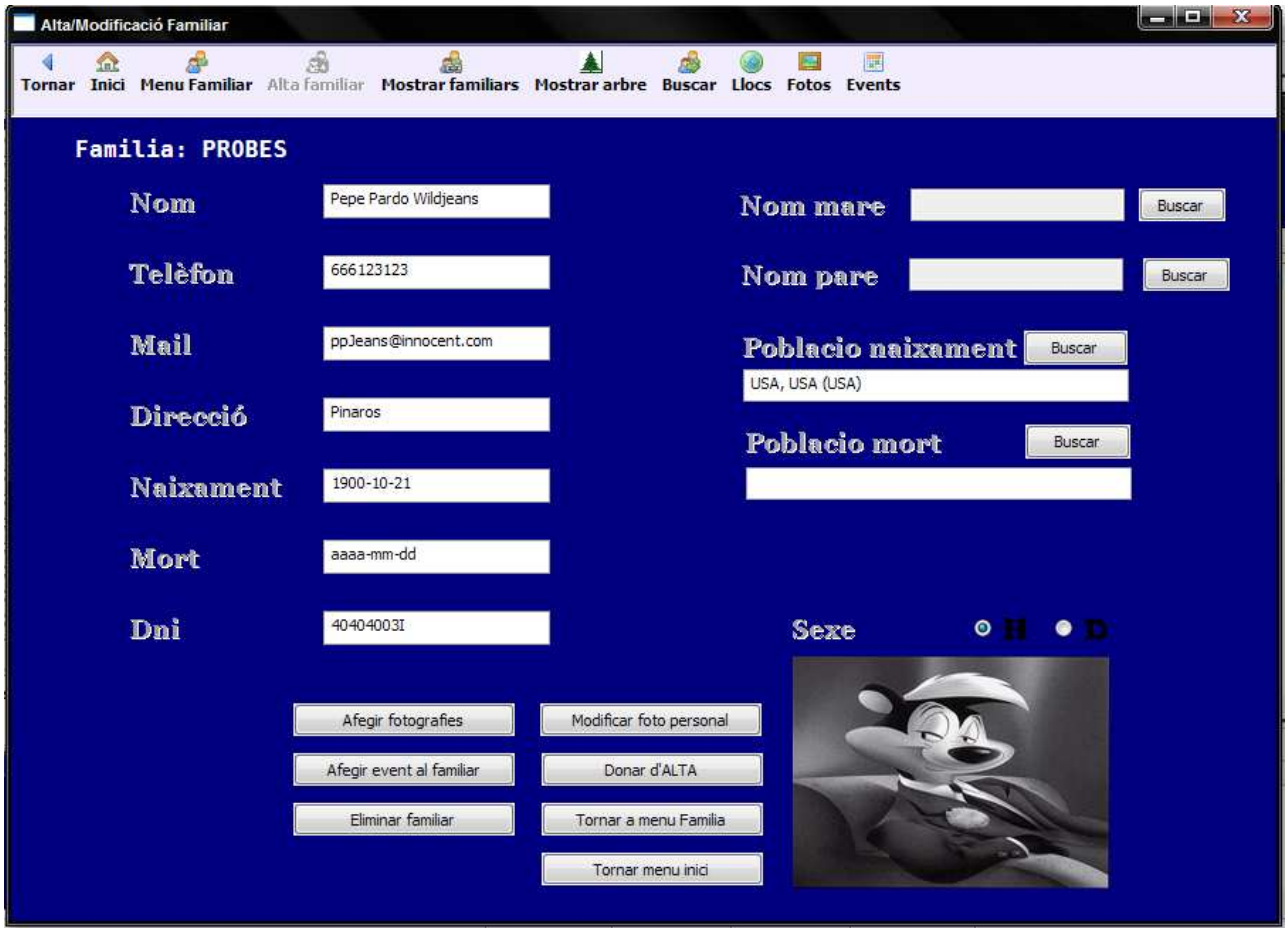
Aquesta interfície es relaciona directament amb la classe familiars, i permet afegir un nou membre a la família. Com s'ha comentat anteriorment és una classe amb un gran volum de dades i, per això, probablement aquesta és la classe més complicada de totes les que hi ha, o almenys la que té més codi. El seu aspecte...



Es pot veure com hi ha molts camps, tot i que molt pocs són obligatoris. Com en interfícies anteriors, en cas que no s'hagi emplenat un camp obligatori, la interfície li fa saber al usuari, així com també en cas que hi hagi errors. Hi ha moltes comprobacions realitzades: tant el telèfon, com el mail, com la data (de naixament i mort); han estat camps tractats, perquè no hi hagi la possibilitat d'afegir dades incorrectes.

El nom de la mare i el del pare, són escollits d'una llista, així com la població de naixament i mort. També hi ha la possibilitat d'escollir sexe. Destacar que hi ha una sèrie de botons deshabilitats, així com també una fotografía abaix a la dreta, que no poden ser utilitzats. Comentar que aquesta mateixa interfície s'utilitza per modificar un familiar.

Per tant en el cas que estiguem modificant el familiar botons com afegirFotografia, afegirEvent o modificarFotoPersonal, seran utilitzables. Si es modifica la foto personal i s'escull una correcte, en totes les interfícies que aparegui el familiar, es mostrarà la seva foto, co mes pot veure a continuació:



Quan es modifica un familiar tota la seva informació és modificable. L'únic que cal és eliminar la informació que hi havia i editar-la. Fins i tot la fotografia personal es pot canviar per una altra, així com el nom de la mare o el pare, o el lloc de naixament.... absolutament TOT és modificable. Si es vol que la informació editada queda guardada a la base de dades és imprescindible que s'apreti el botó donar d'alta, ja que sino no quedarà guardada.

Per acabar amb aquesta classe... per crear la interfície la única variable que es necessita es dades. Hi ha un detall molt important a comentar abans de seguir: aquesta classe té dos familiars un és **alta** i l'altre és **nou**. Alta s'utilitza evidentment alhora de donar d'alta i el nou s'utilitza alhora d'anar canviant de pantalles. L'únic inconvenient que hi ha, i que seria una bona millora a realitzar, és següent: si l'usuari està en una pantalla on apareix un familiar, com per expemple mostrarFamiliar i llavors s'inserta un familiar nou; llavors, si el familiar torna enrere, la pantalla que apareix és la que toca, **mostrarFamiliar** pero les dades no seran les del familiar que hi havia, sino que seran les del nou familiar donat d'alta.

La raó del que he comentat és que només hi ha dos familiars dins la classe dades, quan n'hi hauria d'haver un, cada cop que es mostra, es crea o es modifica la informació de diferents membres de la família. És una millora que no he implementat, que no repercuteix a l'hora d'executar el programa, però que és interessant comentar. Una possible solució seria implementar dins la classe dades un vector de Familiars, que guardés a més quin hi havia a cada pantalla, d'aquesta forma al tornar enrere la informació que es mostraria, seria exactament la vista amb anterioritat.

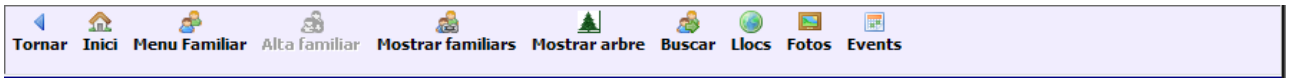
6.6.4 Mètode AltaRealitzada

Aquesta és una interfície molt senzilla, que avisa a l'usuari que la bdd ha estat modificada. S'utilitza quan es dóna d'altaf algún obtjecte (familiar, arxiu,...); quan es modifica o bé quan s'esborra. El seu aspecte és el següent

Com es pot la única funció que té és confirmar a l'usuari que la operació s'ha realitzat amb èxit i prenent ok, el programari el dirigirà al lloc pertinent.

6.6.5 Mètode Barra

Aquesta és una interfície especial ja que no existeix sola, sino que apareix en totes les altres interfícies. La seva funció principal és la de possibilitar a l'usuari una sèrie d'accessos directes, per facilitar la navegació...



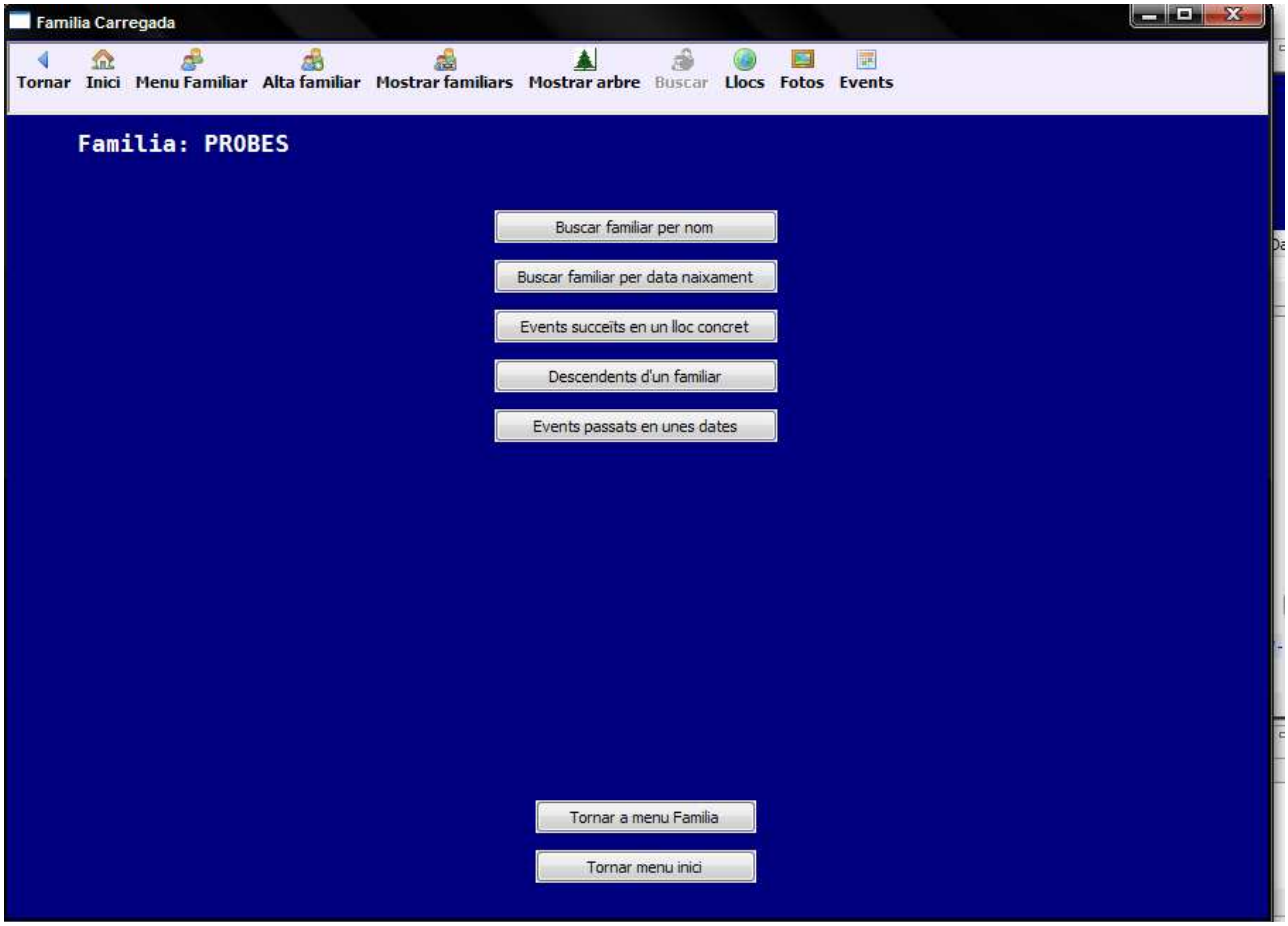
La barra implementada té una sèrie de botons que permeten arribar a totes les opcions del programa, almenys a les més usades, amb un sol click. D'aquesta classse cal destacar, a més del comentat, que hi ha força codi per poder implementar el tornar; ja que hi ha aquest codi:

```
if(anterior.equals("FamiliaCarregada")){
        dades.eliminarPagina();
        try {
                FamiliaCarregada nova = new FamiliaCarregada(dades);
        } catch (Exception e1) {
                e1.printStackTrace();
        }
}else if(anterior.equals...
```

Per a cada una de les interfícies; l'usuari sempre pot tornar enrere, i s'ha tingut que tenir en compte.

6.6.5 Mètode Buscar

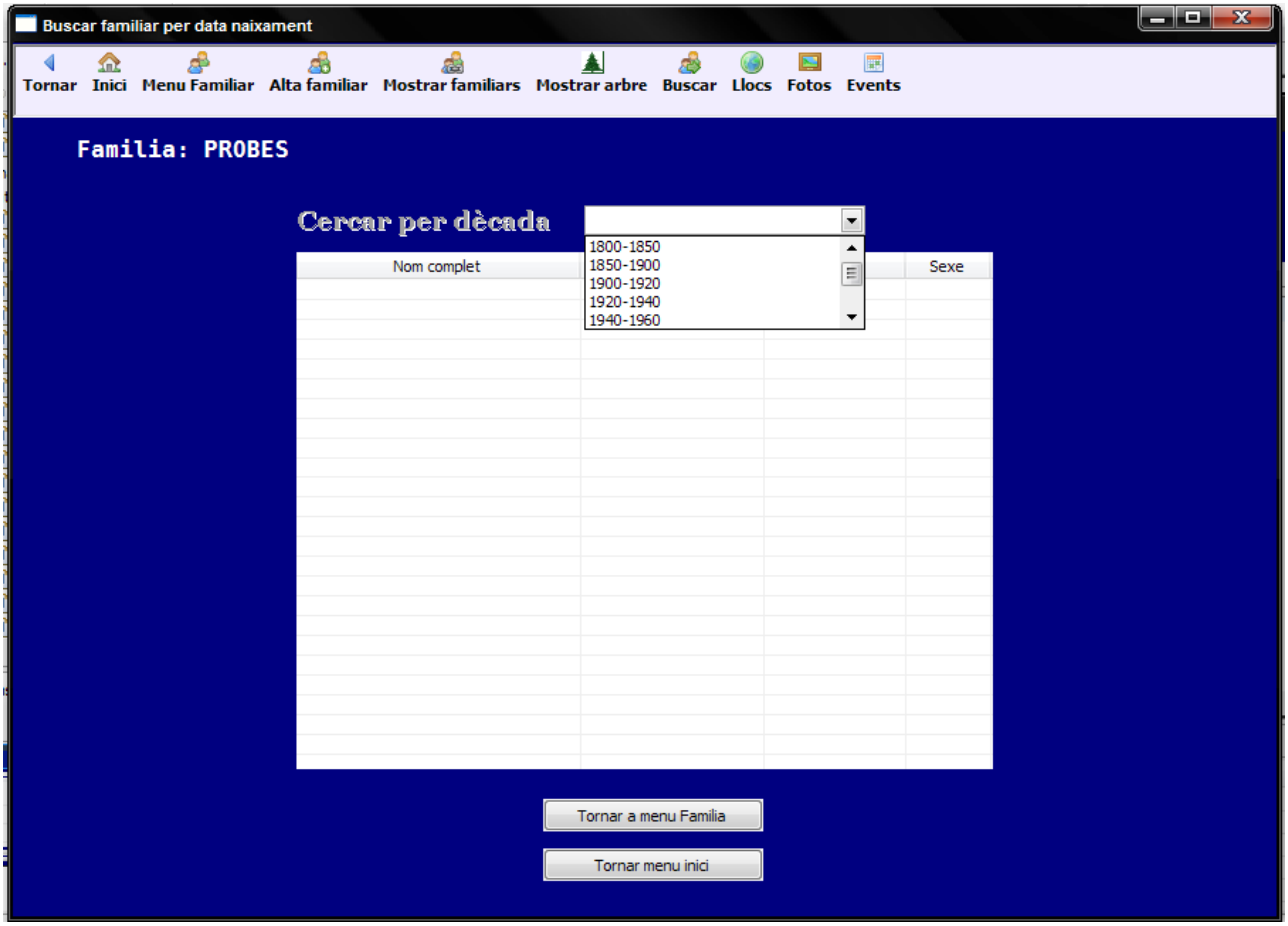
Aquesta interfície permet visualitzar les dades de diverses formes:



Com es pot observar, a part de la barra de navegació, hi ha una sèrie de botons que permeten realitzar diferents cerques. Per tant aquesta pantalla és únicament un menú que enllaça amb altres pàgines.

6.6.7 Mètode BuscarDataNaixament

Aquesta és una de les opcions de l'anterior intefície....



... com es pot observar a dalt hi ha la opció cercar per dècada. Hi ha una sèrie de dècades introduïdes i l'usuari pot escollir quina és la que li interessa veure. Hi havia la possibilitat de fer que a la pantalla l'usuari introduis manualment les dates escollides, però al final em vaig decantar per aquesta opció. El resultat que es dona, si exisiteix, son tots els familiars nascuts entre les dues dates, escollides per l'usuari.

Com només comento el més important del codi, crec que és important recalcar el que permet realitzar accions quan l'usuari, per exemple, clica sobre algun dels objectes. Per expemple el combo dècades: un cop inicialitzada la pantalla, es creen les variable i s'inicialitzen els valors (en aquest cas s'emplena el vector amb les dècades desitjades). Un cop fet s'assigna un **listener** perquè d'aquesta forma el codi s'activarà quan es detecti que l'usuari ha clicat (també es poden afegir listener quan l'usuari s'hi mogui per sobre,...).

```
comboDecad.addSelectionListener(new   org.eclipse.swt.events.SelectionListener()
{
      public void widgetSelected(org.eclipse.swt.events.SelectionEvent e) {
            try {
                  //inicializacio de les variables ...
                  familiars=gf.obtenirFamiliarsData(data1, data2,nomFam);
                  if(!(familiars .elementAt(0)).equals(er)){
                        for(int i=0;i<familiars.size();i++){
                              //emplenar la taula
                  }else{
                  //missatge error per pantalla
            }
      }
}
```

El mètode, primer borra tot el que conté la taula. Posteriorment obté tots els familiars nascuts entre les dues dates passades; i finalment s'emplena la taula amb: nom complet del familiar, la data de naixament (així també es comproba que el resultat sigui correcte), el telèfon i el sexe.

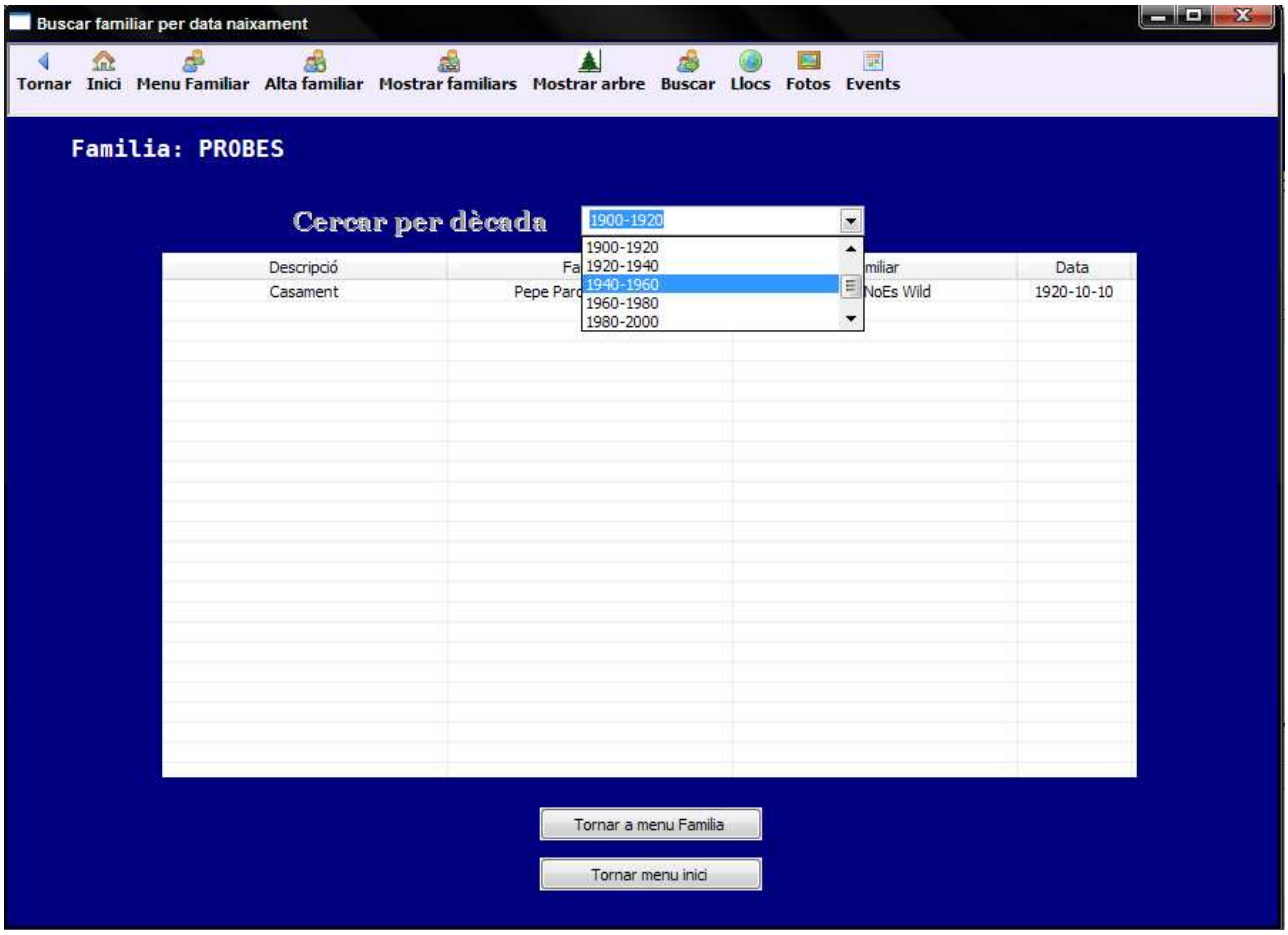
Un cop l'usuari té els resultats, si desitja veure la informació detallada sobre qualsevol, l'únic que ha ha de fer és clicar sobre un d'ells i directament el programari el redireccionarà a la pàgina principal del familiar.

6.6.8 Mètode BuscarEventData

Interfície molt semblant a l'anterior. El que fa és mostrar tots els events succeïts entre dues dates. La implementació és casi exacta a l'anterior, això si, s'ha tingut que crear nous mètodes a la classe bddEvent, per poder obtenir els resultats desitjats.

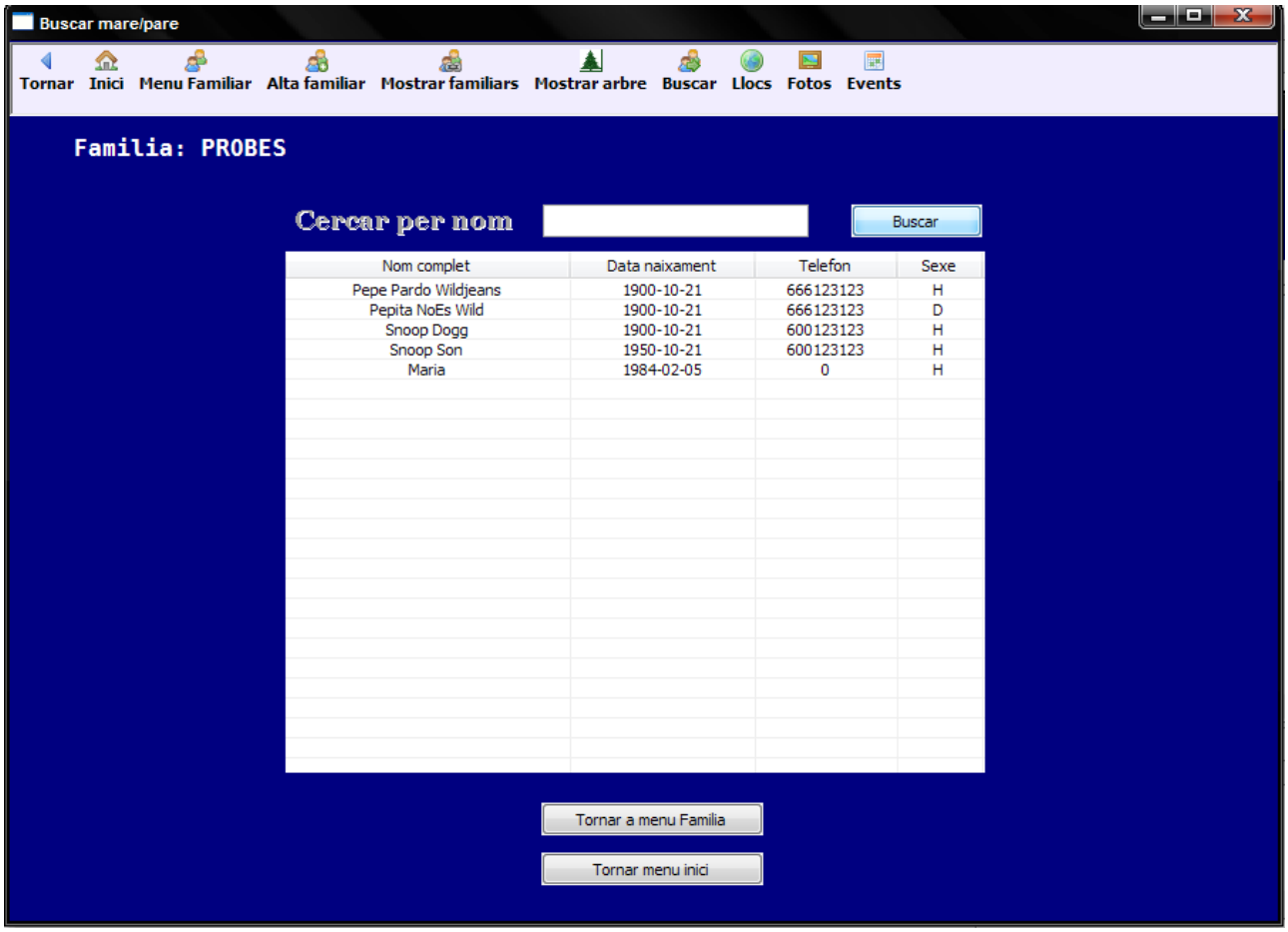
Com abans, l'usuari pot seleccionar alguna de les dècades i el resultat seran tots els events succeïts en aquells deu anys. Llavors, com abans, l'usuari disposa la possibilitat de veure de forma detallada aquells events, com també fer qualsevol altre de les operacions disponibles.

En cas que la dècada desitjada no tingui cap event, a l'usuari se li informa amb un missatge d'error. La interfície és la següent:



6.6.9 Mètode BuscarMarePare

Aquest mètode permet seleccionar la mare o el pare d'un familiar. S'utilitza quan es vol donar d'alta un familiar i es desitja seleccionar algún dels projenitors:



Com ja havia comentat a l'inici, he desitjat, encara que no quedi expressat de forma explícita, englobar l'actualitat de les parelles. Per tant si es desitja seleccionar el pare, es permet seleccionar una dona i a la inversa. Com a resultat és possible que dos dones siguin (pare,mare) i per tant de forma explícita queda implementada qualsevol tipus d'associació de familiars així com també la adopció.

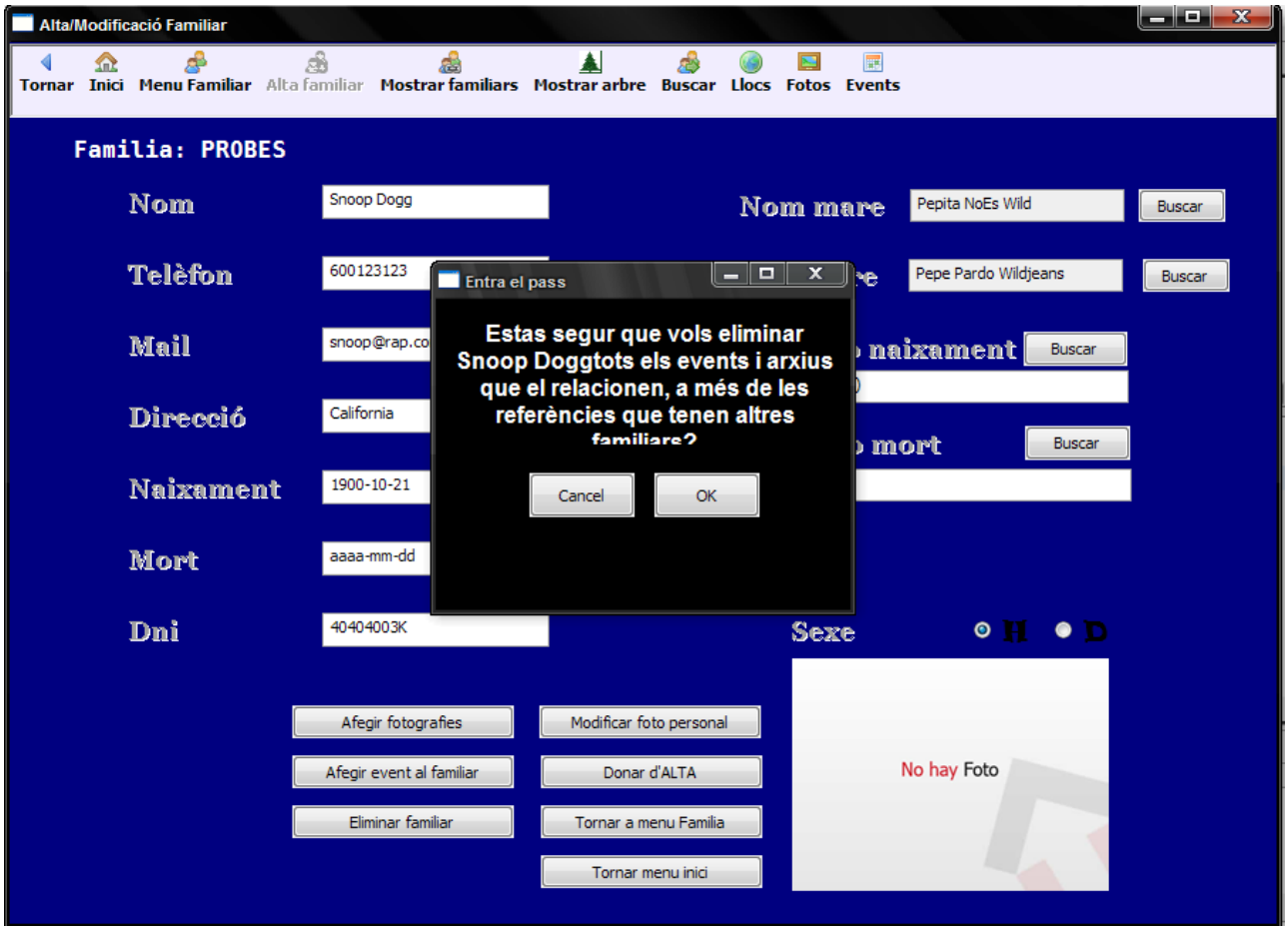
Com mostra la imatge adalt hi ha el nom de la família. A més he afegit la possibilitat de buscar per nom (s'utilitza el mètoe de la bddFamiliar buscarFamiliarNom). D'aquesta manera en cas que hi hagi molts resultats a la taula, la cerca és reduirà molt.

Finalment explicar que, quan es crida la interficie es passa una enter, que hem permet saber si el que es desitja seleccionar és la mare o el pare. De la forma en que he implementat el codi, un

cop escollit el familiar, aquest queda guardat com a mare o pare dins la variable familiar dins de **dades.** D'aquesta forma al retornar, la pantalla (altaFamiliar o qualsevol altre) podrà carregar totes les dades d'aquesta variable per pantalla i no s'hauran de reentrar.

6.6.10 Mètode ConfirmacióBorrar

Amb aquesta pantalla se li demana a l'usuari la confirmació que desitja eliminar algun objecte de la bdd:



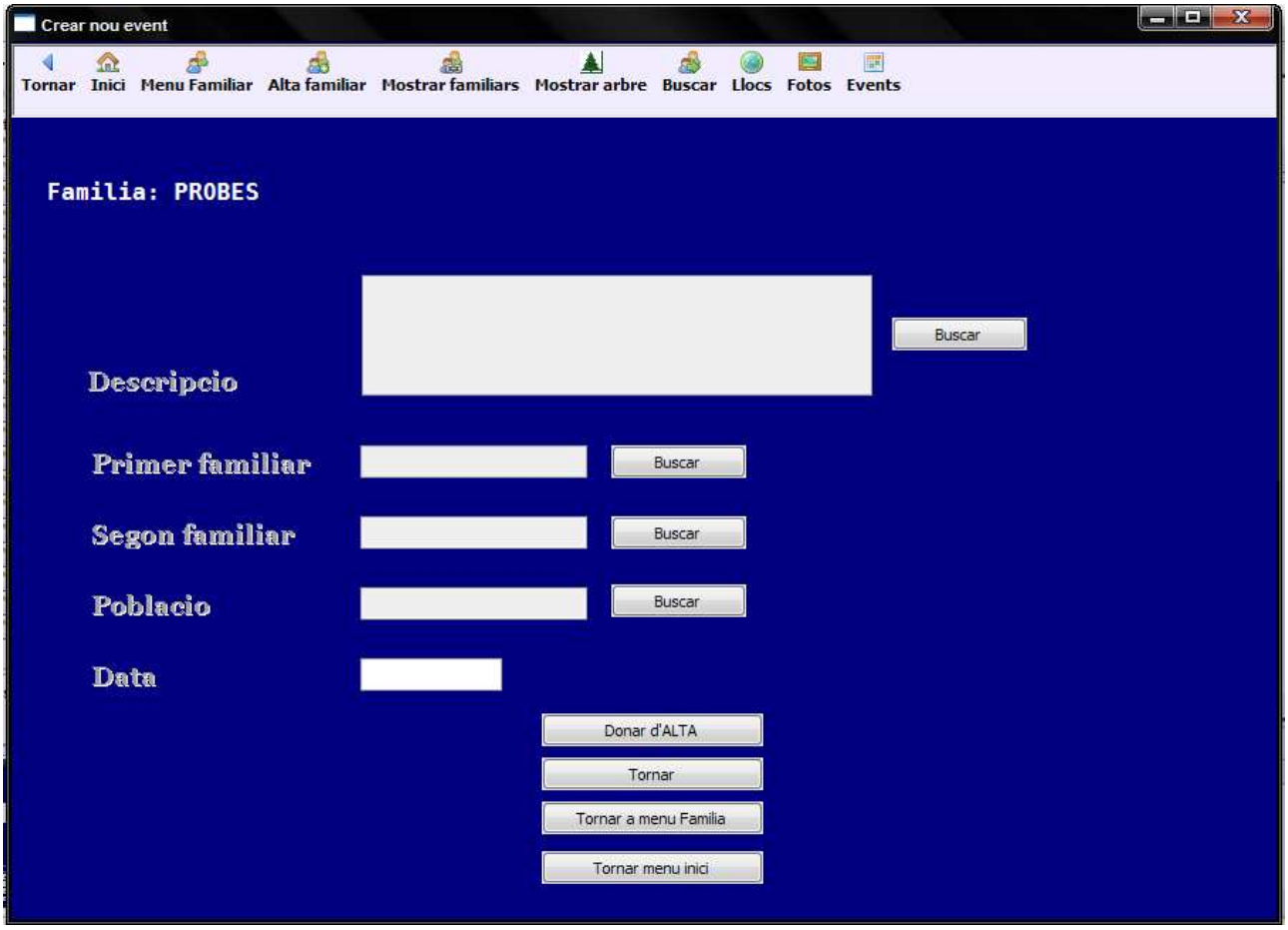
En aquest cas es demana si es vol eliminar un familiar, tot i que també s'utilitza aquesta mateixa, per eliminar arxiu o events. Dins el codi s'ha tractat cada una de les possibilitats, tant per canviar el text que es mostra, com també per cridar una funció o una altra de la bdd. En cas negatiu, es tanca la finestra i l'usuari pot continuar, mentre que en cas negatiu s'esborra el que s'ha seleccionat i si tot funciona correctament, es mostrarà la interfície abans comentada **AltaRealitzada.**

Per acabar m'agradaria comentar que a aquesta interfície se li passa la pantalla anterior, ja que si s'acaba procedint amb la baixa, la pantalla que l'usuari estava veient s'ha de tancar i redirigir

cap a la pantalla inicial.

6.6.11 Mètode CrearNouEvent

Aquí l'usuari pot crear nous events:



Com sempre, alguns camps no són obligatoris (en aquest cas únicament no ho són, segon familiar i data). L'únic valor que es pot afegir manualment és la data, ja que tots els altres s'han de seleccionar d'una llista (tot i que en cas de no haver-hi el desitjat, es podrà afegir desde aquella interfície, excepte els familiars).

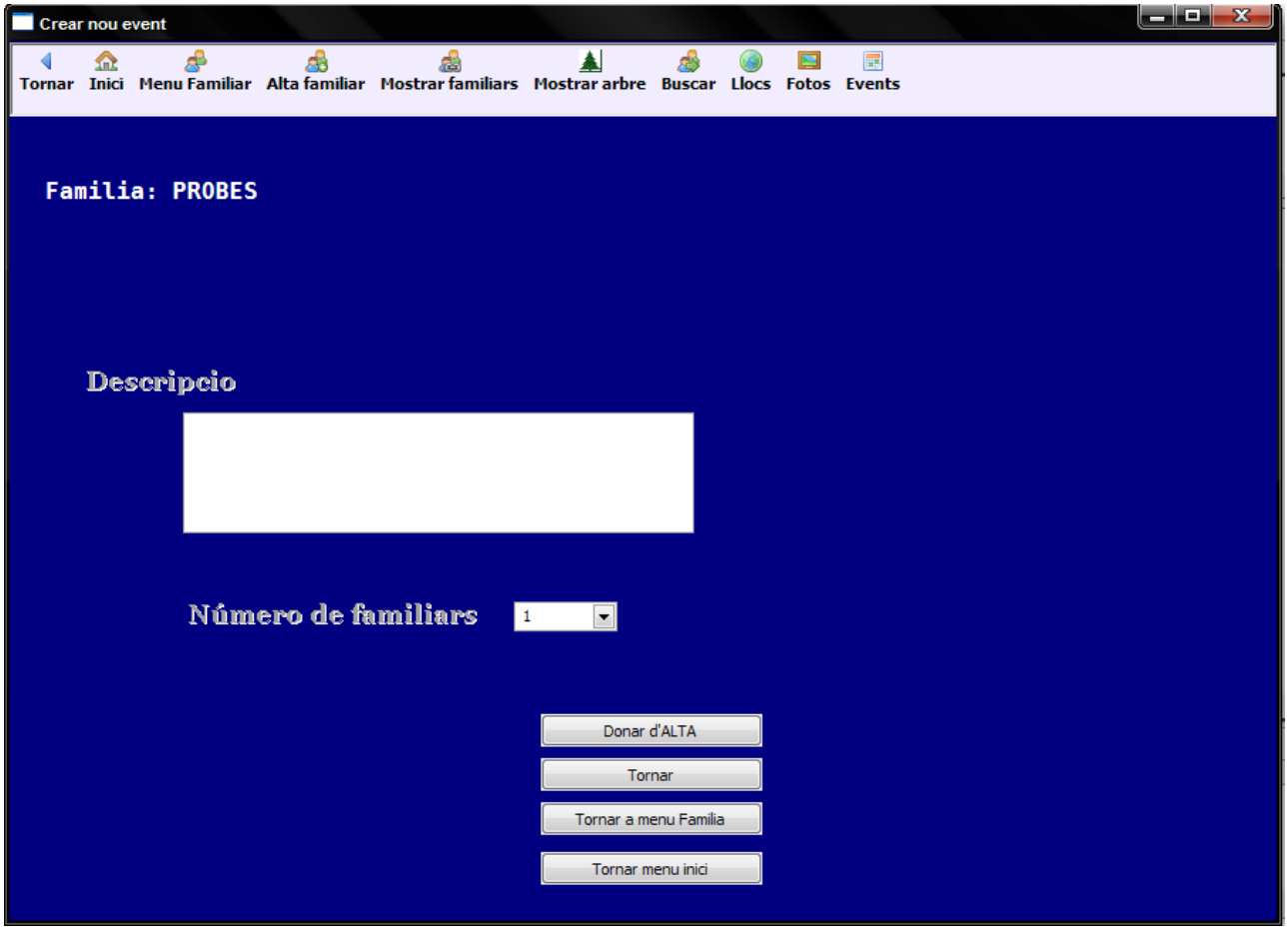
Hi ha events que només engloben un familiar, en aquest cas directament quan es seleccioni el segon familiar quedarà deshabilitat. Com en casos anteriors quan no hi hagi valor en els camps obligatoris l'usuari rebrà un missatge d'error, i en cas que la data sigui incorrecte, serà comprobada i en cas negatiu, l'usuari també serà informat.

Quan es canvia d'interfície per seleccionar algun familiar, la descripció o el lloc, les dades introduides queden guardades dins la classe **dades** i d'aquesta forma es pot canviar d'interfície sense perdre la informació, ja emplenada.

Ja per acabar, comentar que aquesta mateixa pantalla s'utilitza per modificar un event. Per fer-ho alhora, el mètode que es crida és el mateix **altaEvent(event)**. Aquest mètode el que fa és comprobar si existeix l'event (per identificador) i en cas positiu el modifica i en cas negatiu el crea.

6.6.12 Mètode CrearNouTipusEvent

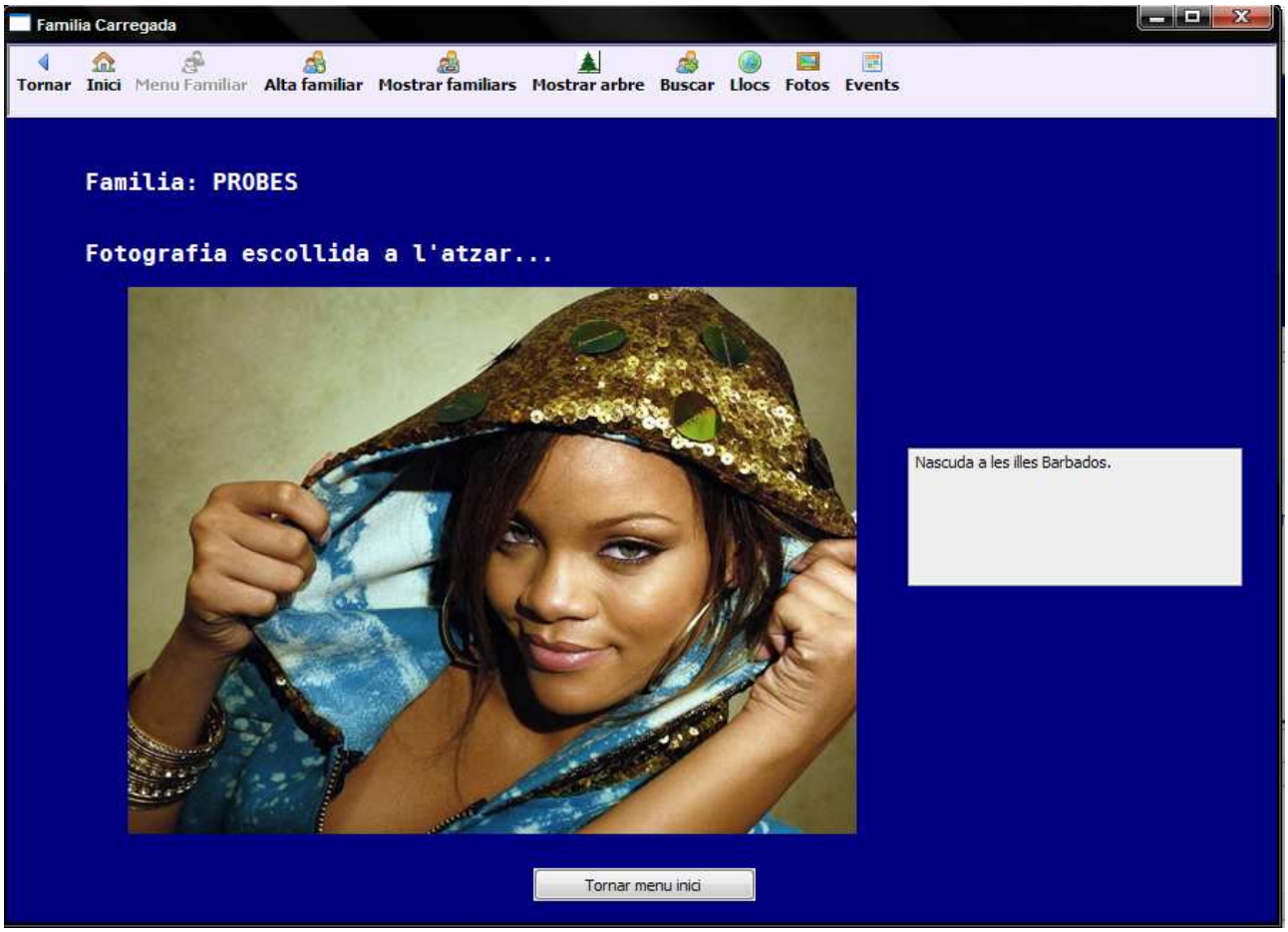
Primer recordar que un tipus event és: un casament, un divorci,... mentre que un event és el casament entre X e Y, podent afegir una informació extra. La interfície té l'aspecte següent...



En aquesta interfície només hi ha dos camps. El primer és la descripció i el segon és el nombre de familiars que fan referència al nou tipus d'event. Al donar d'alta directament s'assigna un valor a l'identificador.

6.6.13 Mètode FamiliaCarregada

Aquesta és la classe que es mostra a l'inici, després de carregar la família desitjada. S'utilitza únicament com a punt de pas. Gràcies a la barra inicial permet realitzar qualsevol de les opcions del programari. El seu aspecte és el següent...

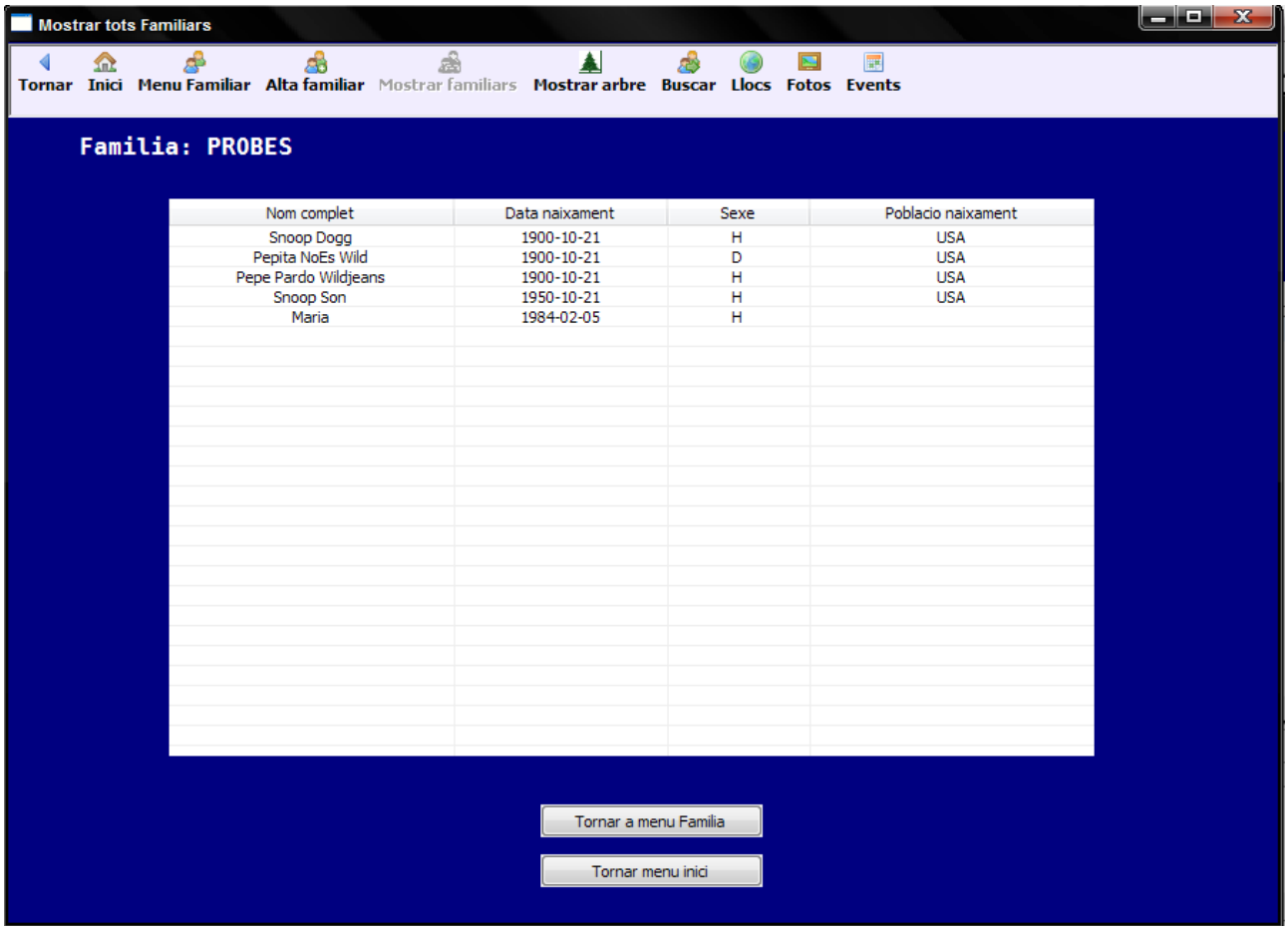


Vaig decidir no repetir els mateixos accessos directes que hi havia a la barra i per això vaig pensar en implementar alguna cosa diferent. El que es fa és escollir una de les fotos, de la bdd i mostrar-la juntament amb la seva descripció. He cregut que era millor mostrar aquesta informació que deixar la pàgina sense cap tipus de servei.

En cas que no s'hagi inserit cap fotografia a la bdd, per exemple la primera vegada que s'inicia l'aplicació, es mostrarà una imatge, que informa que no hi ha cap foto a la bdd. Crec que aquesta idea, és molt interessant, especialment si hi ha moltes fotografies guardades, ja que és probable que no s'hagin vist totes; i d'aquesta forma, com es carrega una foto de forma aleatoria sempre es poden veure fotos no visulitzades.

6.6.14 MètodeFamiliarMostrarTotsFamiliars

Es mostren tots els familiars existents a la bdd.

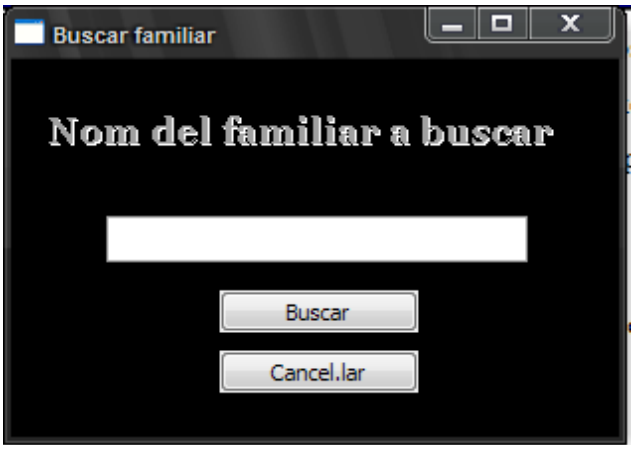


Es mostra el nom, la data de naixament (perque si hi ha dos familiars amb el mateix nom, amb la data és més senzill seleccionar el familiar desitjat), el sexe i la població de naixament. A més totes aquestes variables seran les que probablement seran emplenades alhora de donar d'alta un membre.

Com sempre que apareix una llista, en cas que es cliqui sobre algún dels camps es mostrarà el familiar detalladament (**MostrarFamiliar)** que quedarà explicat més endevant.

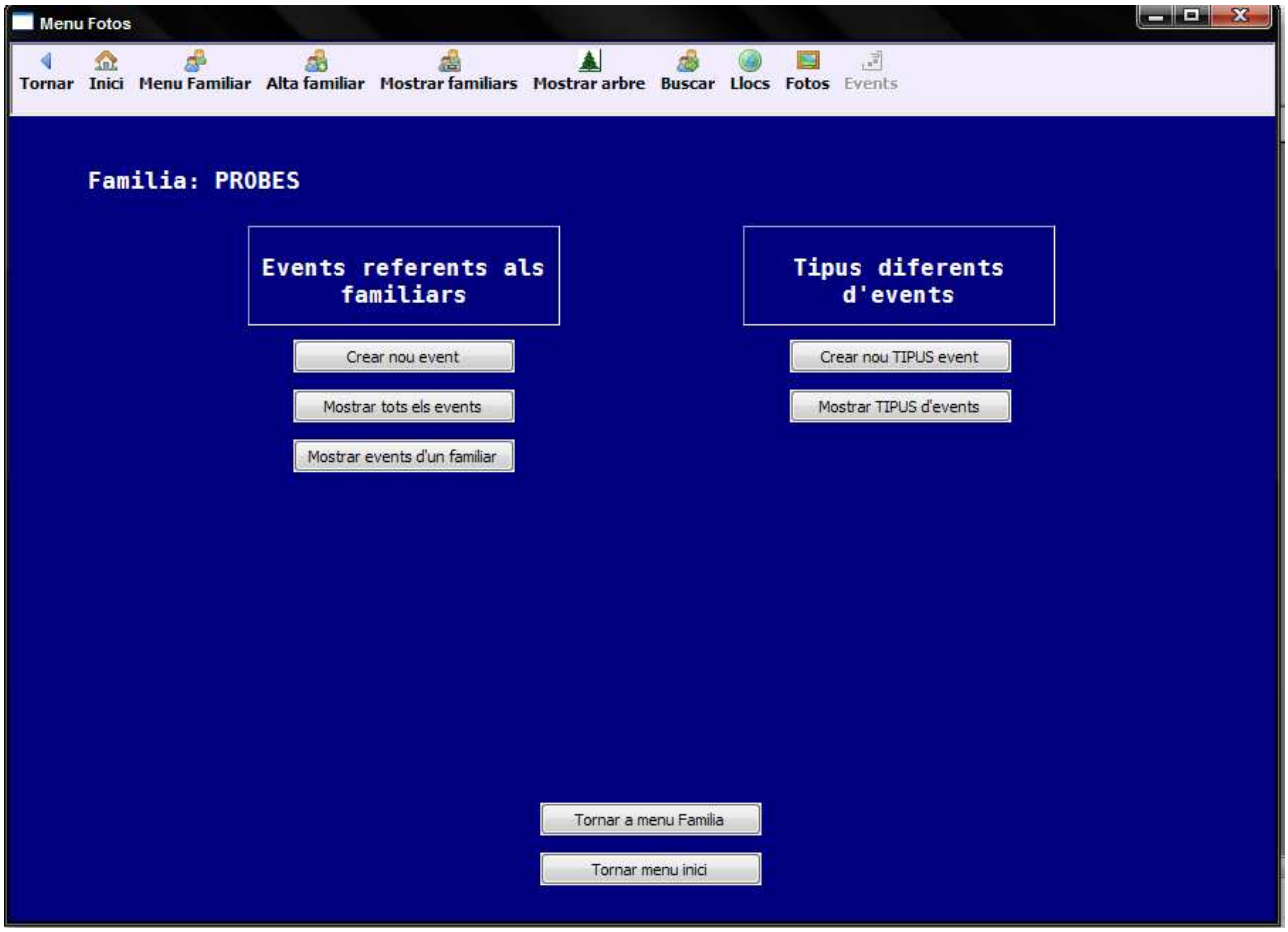
6.6.15 Mètode BuscarFamiliarNom

La interfície:



Es tracta d'una pantalla molt senzilla. Utilitza el mètode **buscarFamiliarNom**, explicat anteriorment. Un cop introduït el nom, en cas que existeixi algun familiar, es mostrarà en una nova pantalla, mentre que si no es mostra un missatge d'error.

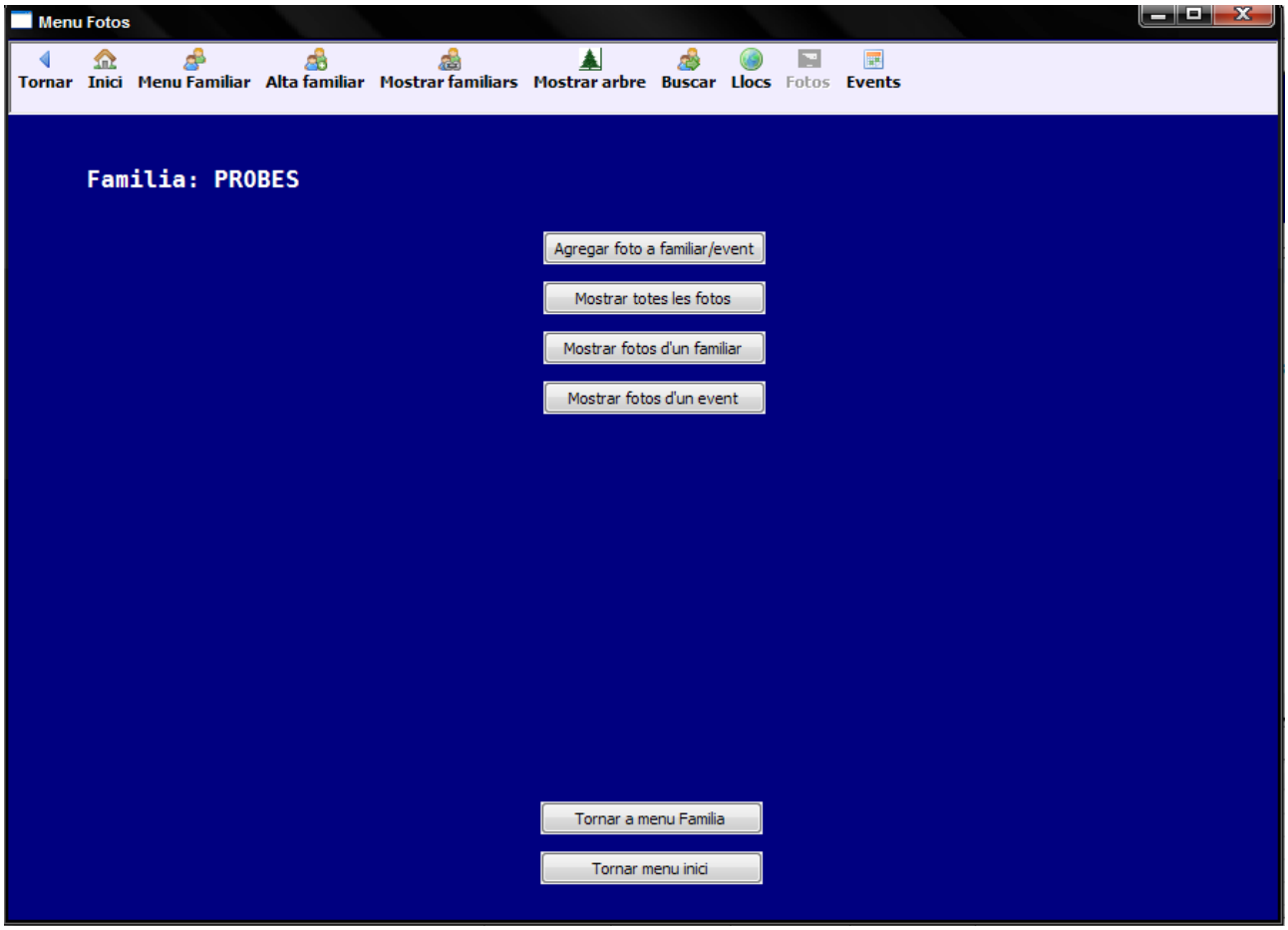
6.6.16 Mètode MenuEvents



En la primera versió d'aquesta pantalla, tots els botons estaven junts i la veritat no quedava clar. (Anteriorment ja he explicat la diferència entre event i tipus d'event, i no hi tornaré). La divisió que he implementat, crec que ajudarà molt a l'usuari. A la part esquerra hi ha les operacions relacionades amb els events, mentre que a la dreta les de tipus d'event.

6.6.17 Mètode MenuFotos

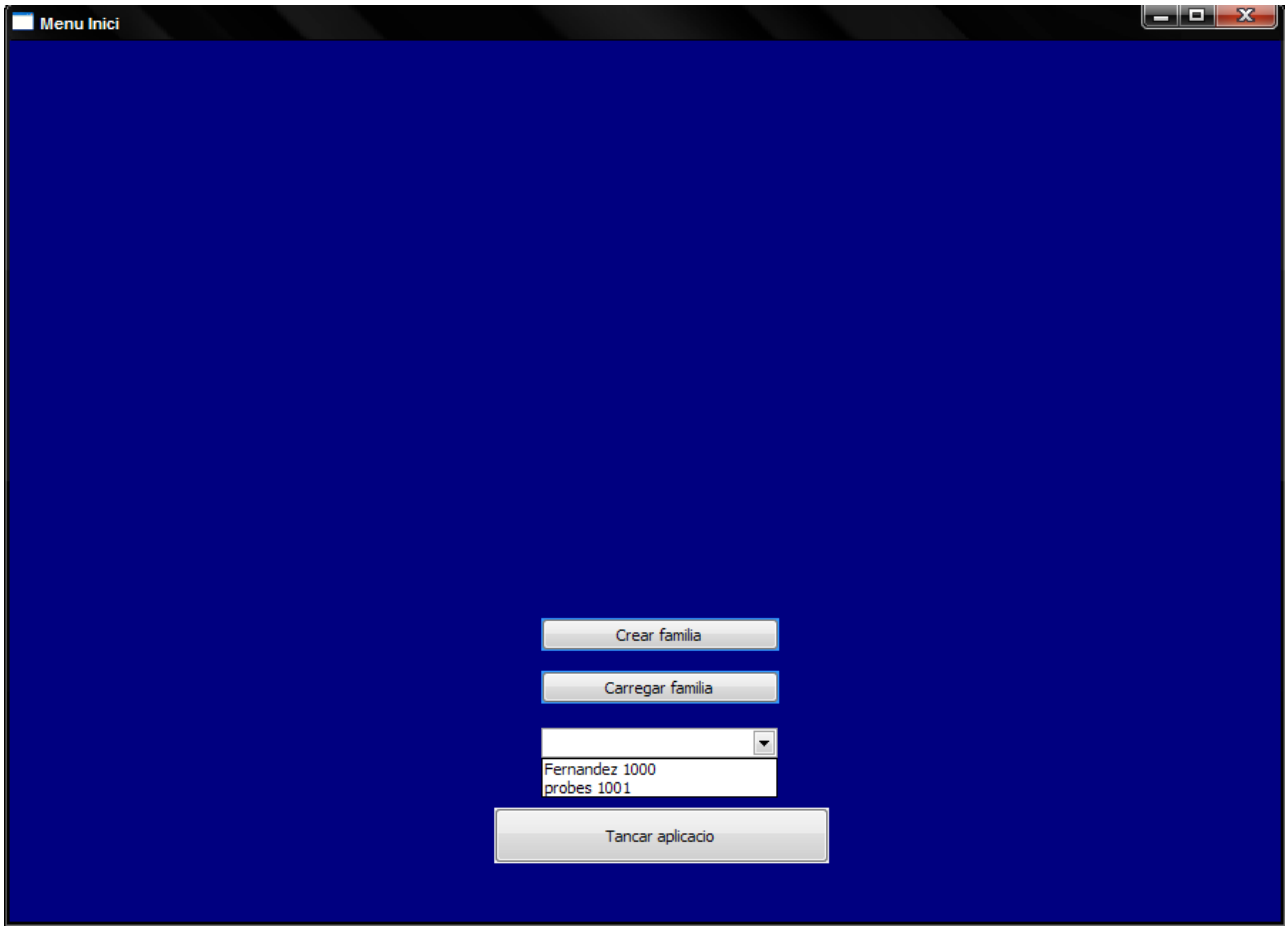
Intefície que permet realitzar totes les operacions relacionades amb fotos:



Permet inserir fotos (a familiars i arxius), mostrar totes les existents, mostrar les relacionades amb un familiar o bé les d'un event en concret. Al ser una pantalla menú igual que l'anterior, no hi ha gaires coses interessants a comentar, ja que únicament serveixen de pont entre l'anterior i la operació que realment vol fer l'usuari.

6.6.18 Mètode MenuInici

És la primera pantalla amb la que es troba l'usuari:

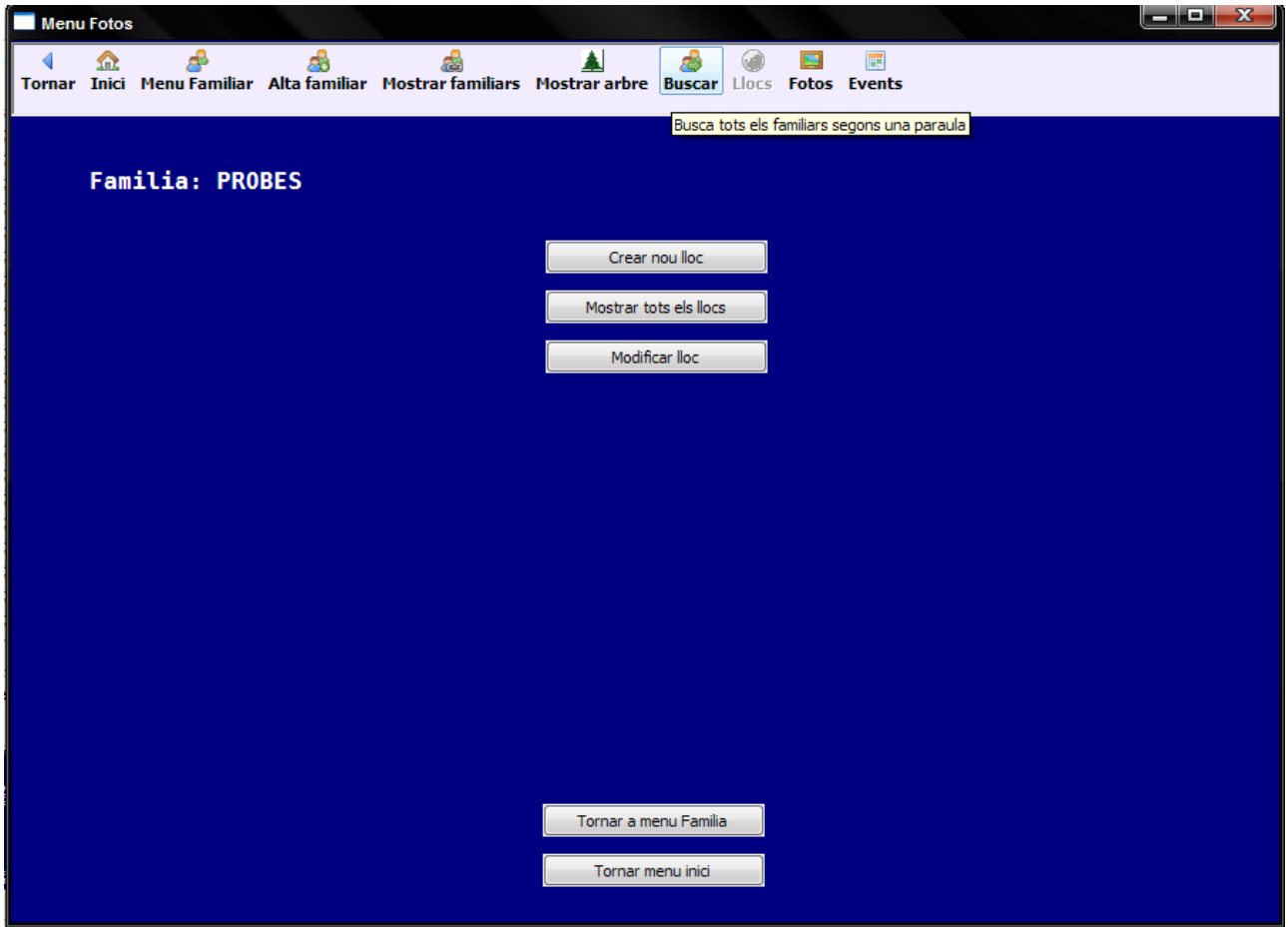


Hi ha dues opcions crear una nova família o carregar-ne una de ja existent. El combo que apareix desplegat inicialment no és visible per l'usuari, només quan aquest selecciona carregar-ne una, es mostra. En ell es carreguen totes les famílies, juntament amb el seu id. La rao de mostrar-se són dues (una molt més important que l'altra): la primera que si no l'escric, i hi ha diverses familíes amb el mateix nom, no es pot garantitzar que es mostri la família desitjada; i segon que d'aquesta manera també l'usuari pot distingir millor quina és la que vol.

En cas que es seleccioni una família sense pass, es rediregirà directament a famíliaCarregda, mentre que si està protegida, es demanarà la seva clau.

6.6.19 Mètode MenuLlocs

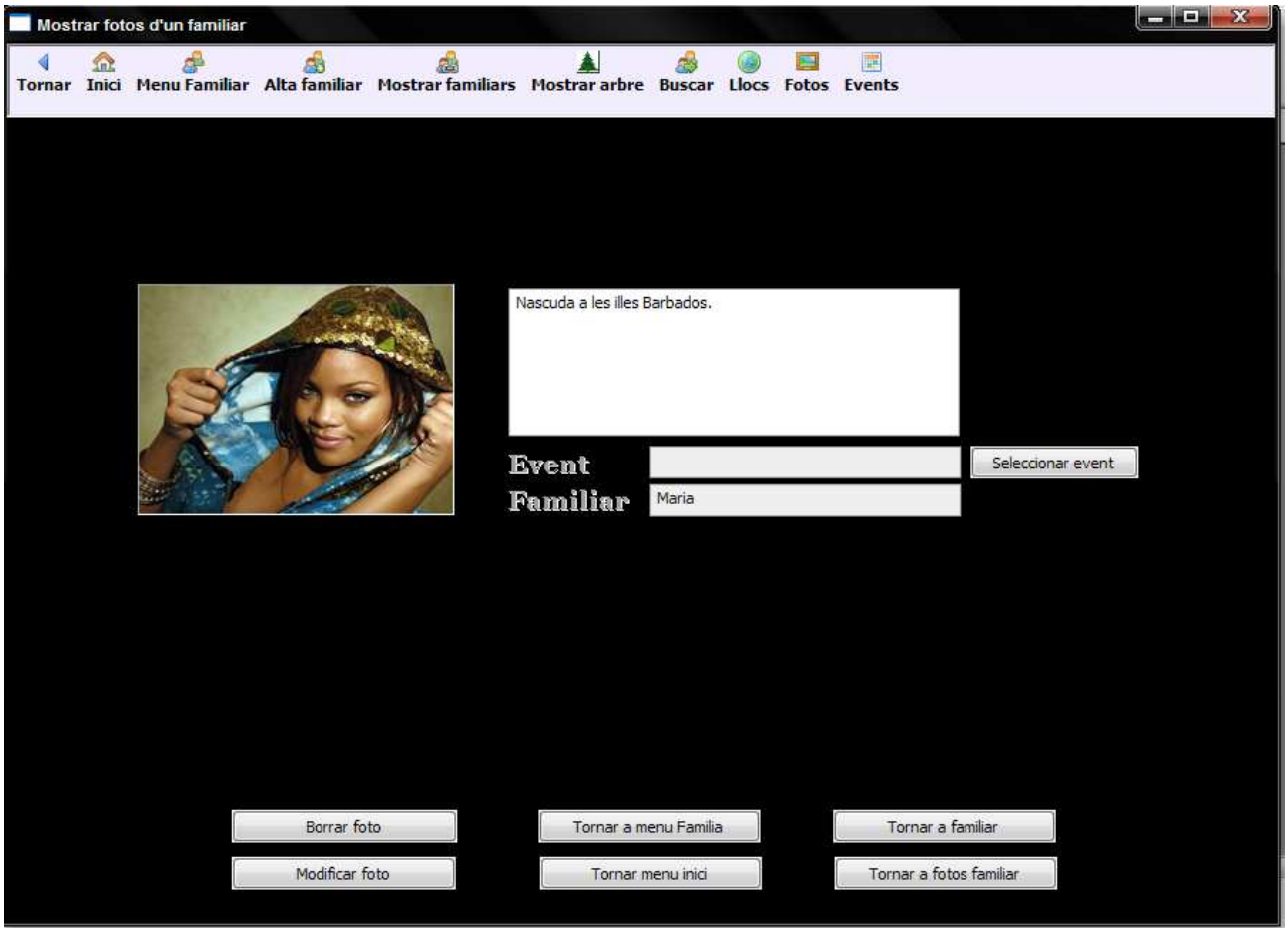
El seu aspecte:



Permet crear un nou lloc, mostrar-los tot o modificar-ne algun.

6.6.20 Mètode ModificarFotoPersonal

Aquesta interfície permet a més de modificar una foto, borrar-la.

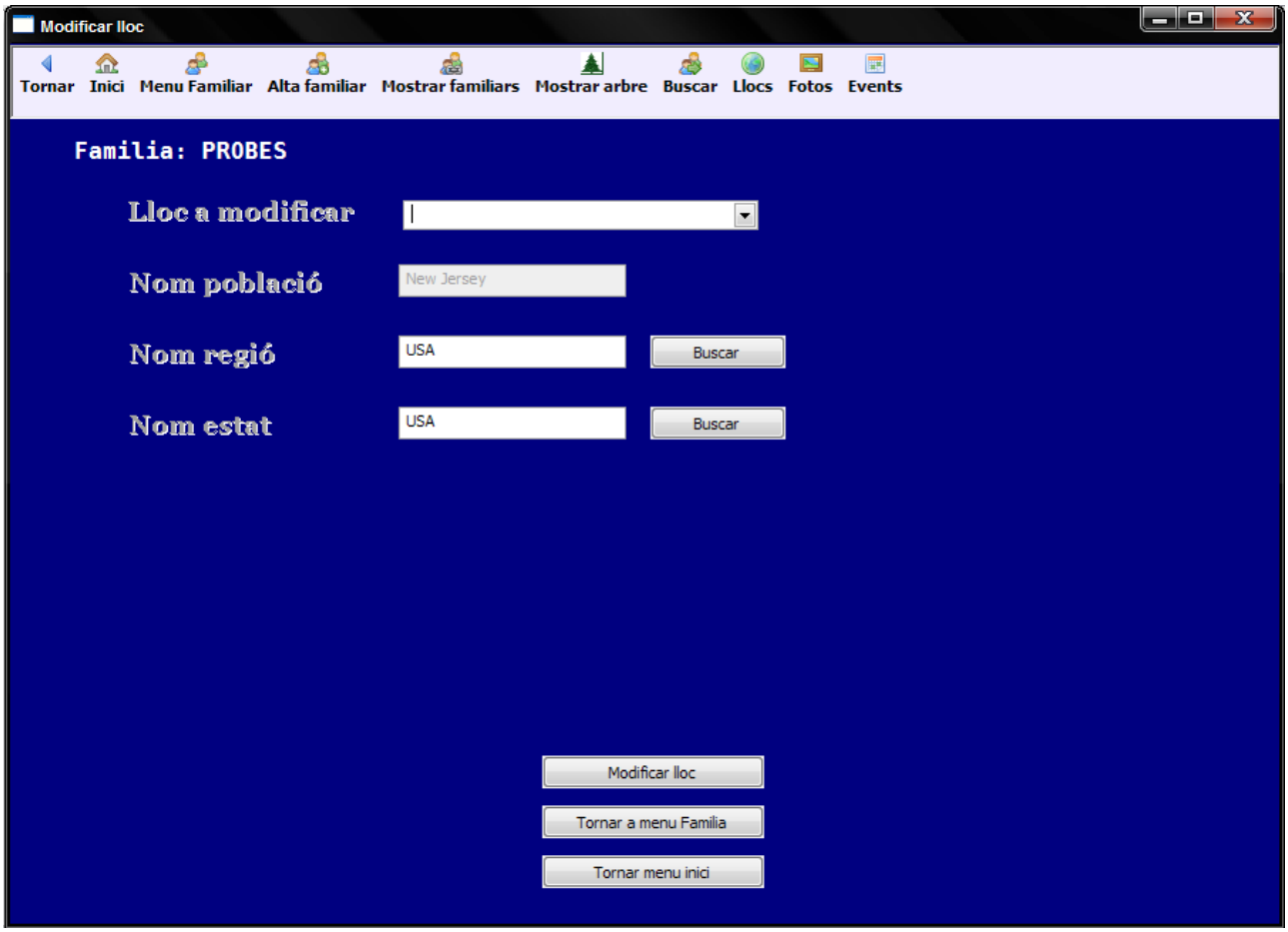


A la part esquerra es mostra la foto personal. A més també es mostra la descripció, l'event que la relaciona i el familiar a qui pertany. Per modificar la descripció es pot fer de forma directa, mentre que per modificar l'event s'ha de seleccionar d'una llista i llavors modificar-lo.

Per acabar comentar que he dividit les possibles accions en tres grups. El primer grup, situat abaix a la dreta fa referència a la modificació de la bdd. Un dels botons permet borrar la foto (primer es demana conformitat), mentre que el segon modifica tots els valors que s'ha retocat a la pantalla. El segon grup, el del mig, son botons que permeten tornar a pantalles principals, mentre que el tercer permet tornar a interfícies relacionades amb familiars (ja sigui per mostrar tots els familiars, o bé per mostrar les fotos del familiar).

6.6.21 Mètode ModificarLloc

La interfície és la següent:

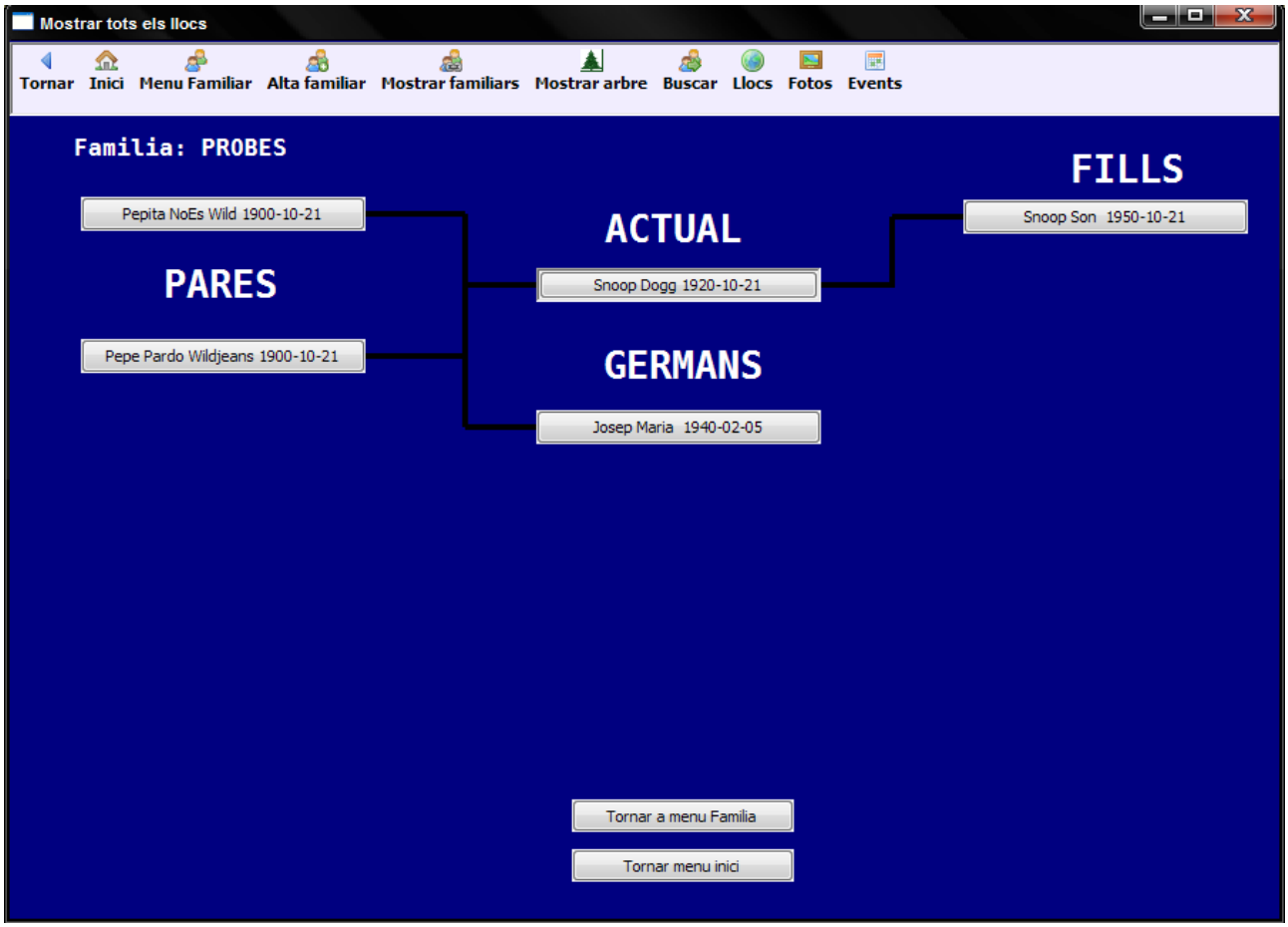


A la part superior hi ha un combo que permet seleccionar el lloc que es vol modificar. Quan es selecciona un, apareixen totes les seves dades per pantalla. Tant la regió com l'estat es poden modificar (seleccionant d'una llista), mentre que el nom de la població no és modificable.

En cas que l'usuari desitgi modificar el lloc, l'aplicació comprobarà que les dades son correctes i en cas afirmatiu, per mitjà del mètode **modLloc** de la classse **Bdd**, el lloc quedarà modificat.

6.6.22 Mètode MostrarArbre

Aquesta interfície mostra per pantalla tres generacions de la família. Es mostra un "referent", els seus pares, els seus germans i els seus fils. Per arribar a aquesta pantalla l'usuari, primer ha de seleccionar d'una llista el familiar que vol que sigui el "referent". Del familiar escollit es mostren els pares, els germans i els fills. El seu aspecte és el següent:



El referent apareix en la part central, on hi ha **Actual.** La informació que es mostra és: el nom i la data de naixament. Gràcies a aquesta data l'usuari de l'aplicació podrà distingir entre familiars amb el mateix nom, i jo podré trobar el familiar, perquè únicament amb el nom no podria seleccionar-lo inequivocament. En cas que es cliqui sobre d'ell, es rediregirà a l'usuari a la pantalla del membre de la família seleccionat, mostrant-lo detalladament.

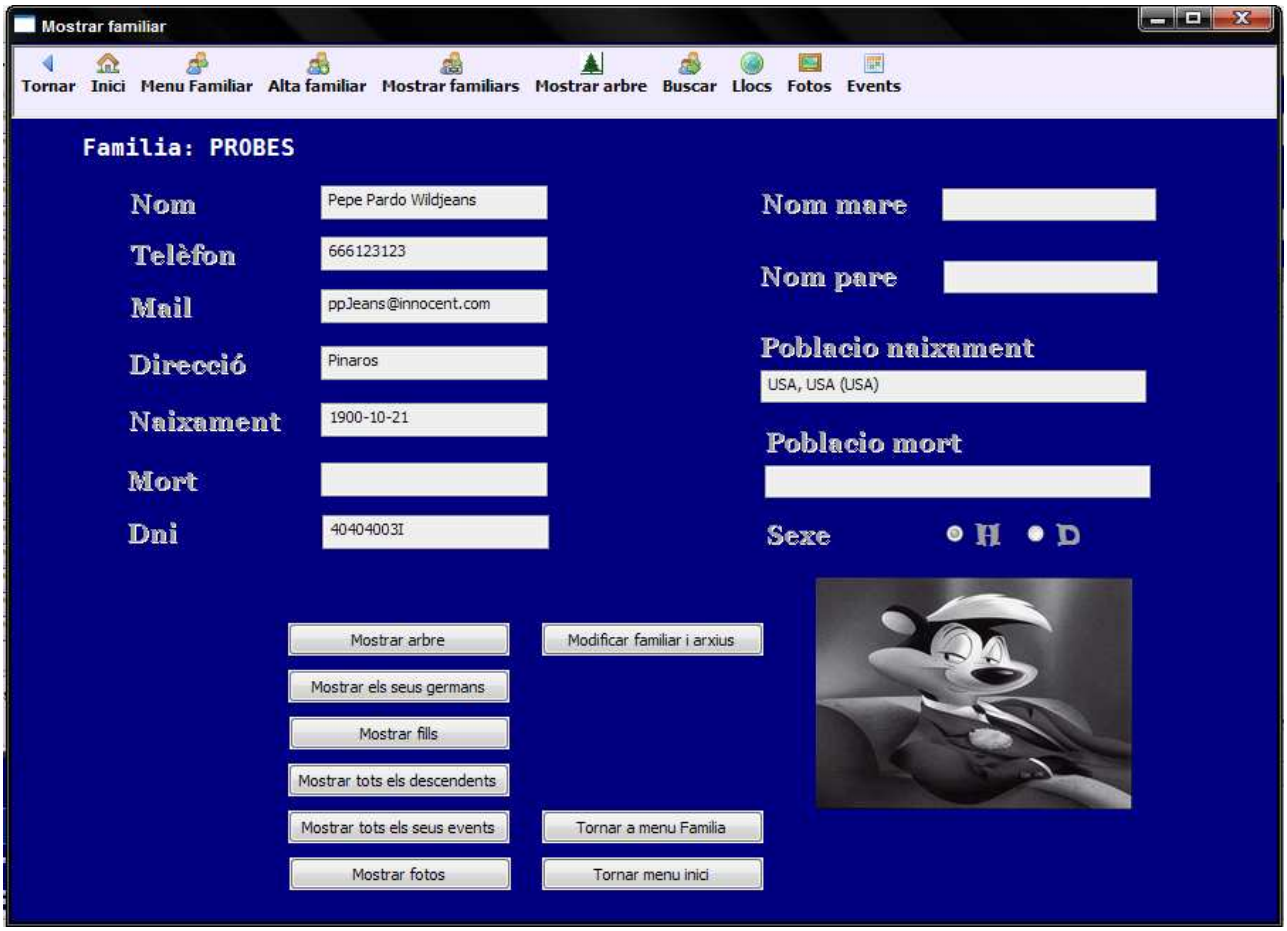
A l'esquerra apareixen els pares. En cas que no hagin estat introduïts es veuran els botons, pero no contindran cap informació a més d'estas deshabilitats. Aquest cop i, en tots els casos excepte el del referent, quan es clica sobre algun familiar el que es fa és cridar aquesta mateixa interfície pero canviant l'arrel. D'aquesta manera es permet la navegació de l'arbre de generació en generació.

Sota el referent hi ha tots els seus germans. Segons els germans que hi hagi s'aniran creant botons i també les línies decoratives. Cal remarcar que com a màxim he permès dibuixar 8 germans (contant el referent); quan se'n tenen més en comptes d'apareixer per pantalla, es mostrarà un botó que permetrà veure'ls en una llista.

Finalment a la dreta apareixen tots els fills del referent. Com abans el nombre màxim de fills pot ser 8, en cas que n'hi hagi més, únicament es mostrarà un botó per veure'ls en una llista.

6.6.23 Mètode MostrarContacte

Aquesta interfície mostra tota la informació possible d'un familiar:



S'assembla molt a la pantalla modificació d'un familiar i precissament per poder-hi accedir, només s'hi pot arribar des d'aquesta pàgina. Permet veure tota la informació del familiar, i en cas que es vulgui modificar s'haurà de seleccionar la opció de modificar-lo. La diferència entre **mostrarContacte** i **modFamiliar,** a més de ser l'evident és que les opcions que hi ha en la interfície modificar són per canviar dades (modificar foto, afegir fotos, afegir event i modificar el propi familiar); per contra les opcions que es mostren en aquesta pantalla són només per conèixer al familiar.

Altres de les opcions possibles són: veure el seu arbre familiar, siguen en ell el referent; també permet mostrar els seus germans o fills (es mostra una nova pantalla amb una llista de familiars seleccionables, per ser mostrats), mostrar els seus events o mostrar les seves fotos. Hi ha la opció de veure els seus descendents i és una interfície interessant que es comentarà més endevant i permet mostrar tots els membres de la família descendents del familiar actual. Cal aclarir que si en cas que no hi hagi germans, fills.. es mostra un missatge per pantalla i no es redirigeix a l'usuari.

Finalment comentar que també es mostra la imatge, que el familiar ha seleccionat com a familiar; en cas de no fer-ho es mostrarà una imatge com la següent:

Imatge que ha estat reduïda per mitjà del mètode següent:

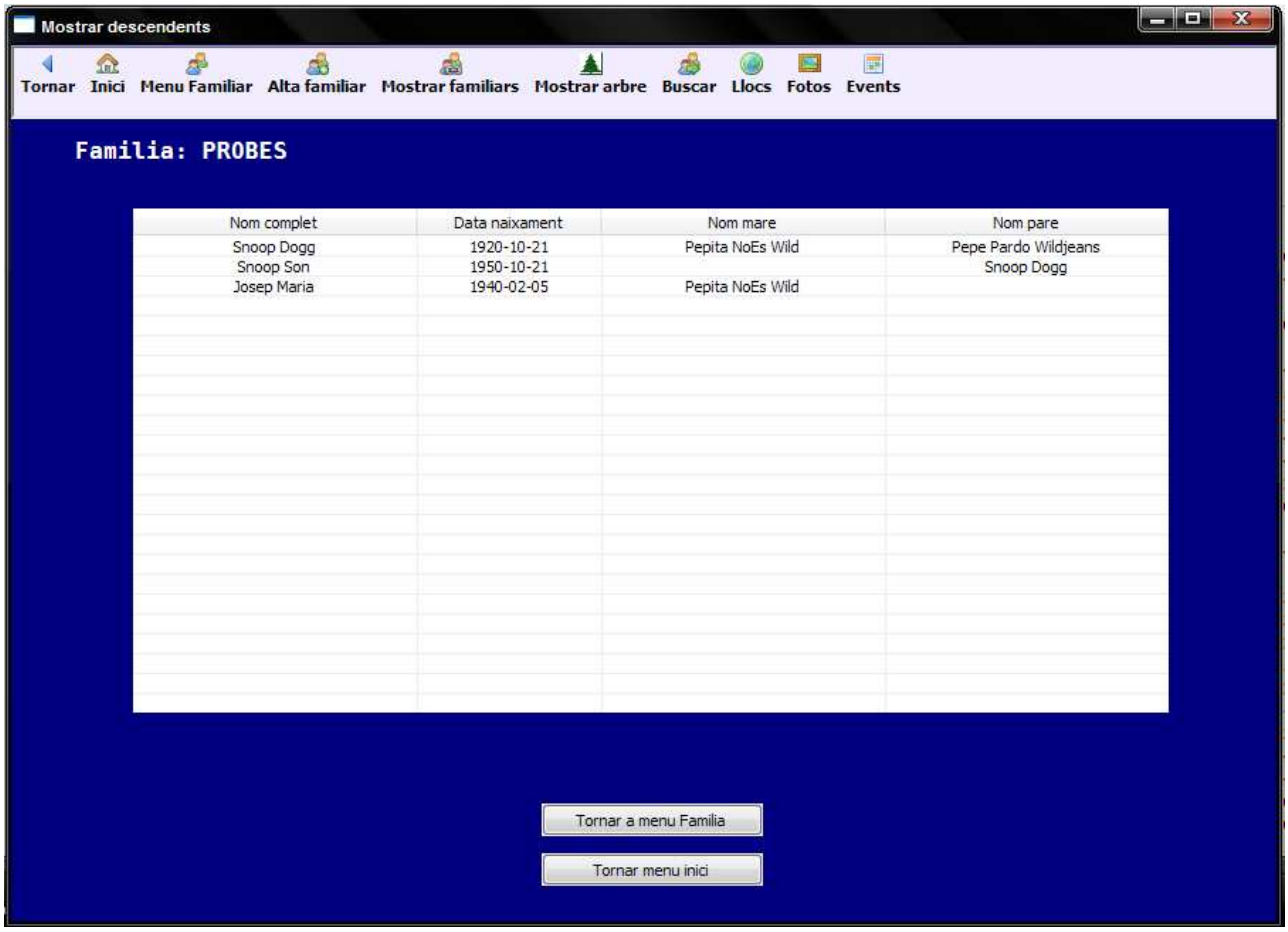
```
if(ubicacio!=""){
        Image original=new Image(null,Display.getCurrent().getClass().
                getResourceAsStream(ubicacio));
        Image scaled = new Image(Display.getDefault(),original.getBounds().width,
                original.getBounds().height);
        GC gc = new GC(scaled);
        gc.setAntialias(SWT.ON);
        gc.setInterpolation(SWT.HIGH);
        gc.drawImage(original, 0, 0,
        original.getBounds().width, original.getBounds().height,
                0, 0, foto.getBounds().width, foto.getBounds().height);
        gc.dispose();
```

```
            original.dispose();

            foto.setImage(scaled);

}else{
//En cas que no hi hagi ubicació s'ha de fer el mateix que abans, pero carregant la foto anterior.
```

Aquest mètode és el que s'usa en totes les interfícies que es mostra una foto. Per finalitzar comentar que el tamany del label que permet ubicar la foto en pantalla, està escalat d'una foto de 1024*768, per tant les fotos amb aquesta mida (la més utilitzada en fotos), quedaran reduides mantenint el seu aspecte.

6.4.3.24 MostrarDescendents

Intefície que mostra, com una llista, tots els familiars descendents d'un altre, seleccionat per l'usuari:



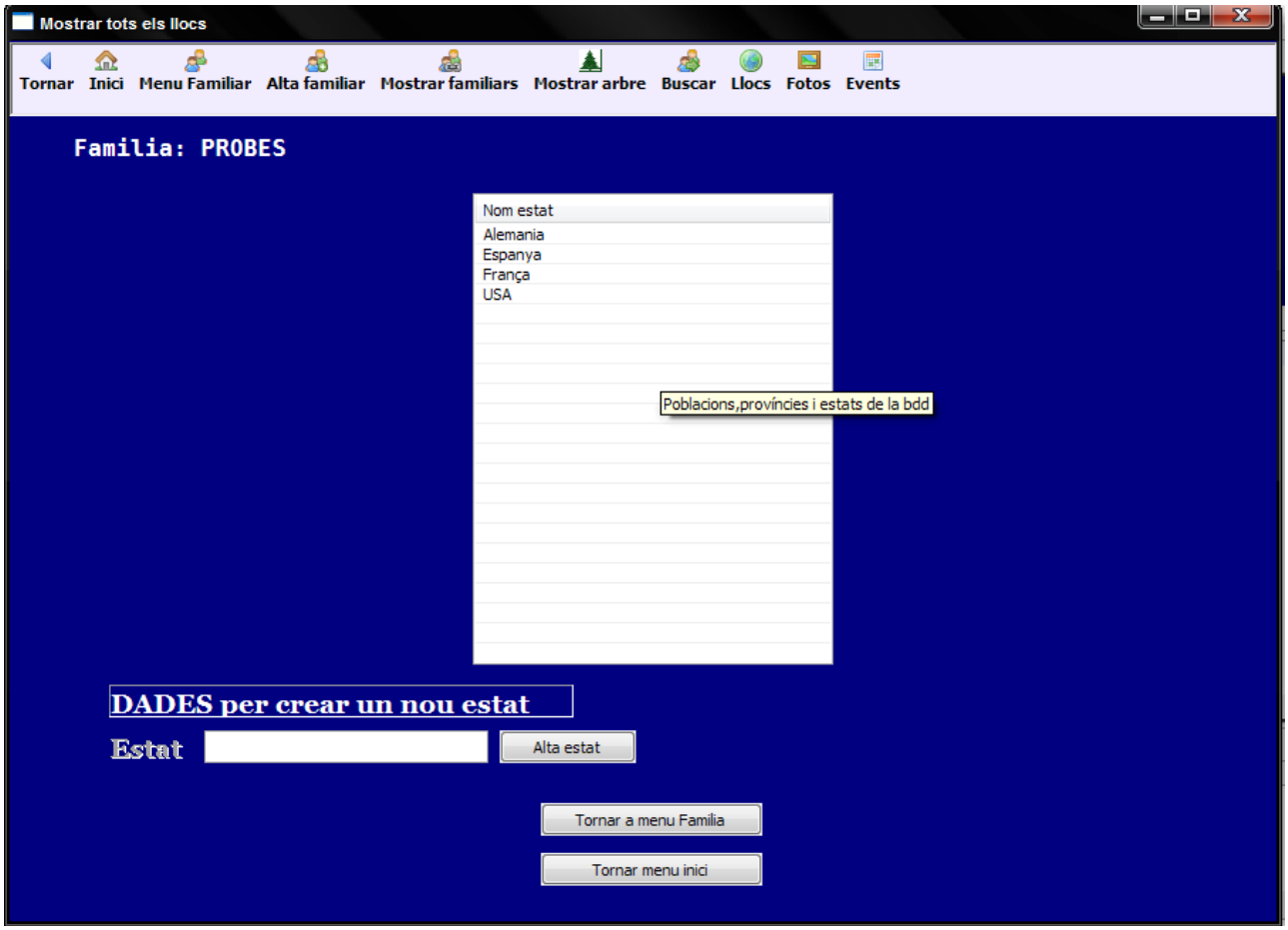
A la llista es mostra el: nom complet, la data de naixament i finalment el nom del pare i la mare. Com sempre, cada un dels familiars que hi ha a la llista, en ser clicats, es mostra de forma detallada.

L'únic que queda per comentar és el codi recursiu que s'implementa. Només hi havia froma vàlida d'obtenir tots els descendents a partir de l'acutal i era dissenyant una funció recursiva que anés recorrents els fills del membre de la família que es cridava en el mètode. Al final vaig crear-lo de la següent forma:

```
private void buscarDescendents(Familiar nou, Vector descendents) throws Exception{
        Vector fills=new Vector();
        GestioFamiliar gf=new GestioFamiliar();
        fills=gf.obtenirFills(nou);
        if(fills.elementAt(0).equals("Error")){
        //aquest familiar no té fills.
        }else{
                for(int i=0;i<fills.size();i++){
                        descendents.add(fills.elementAt(i));
                        buscarDescendents((Familiar)fills.elementAt(i),descendents);
                }
        }
}
```

6.6.25 Mètode MostrarEstat

El seu aspecte és el següent:

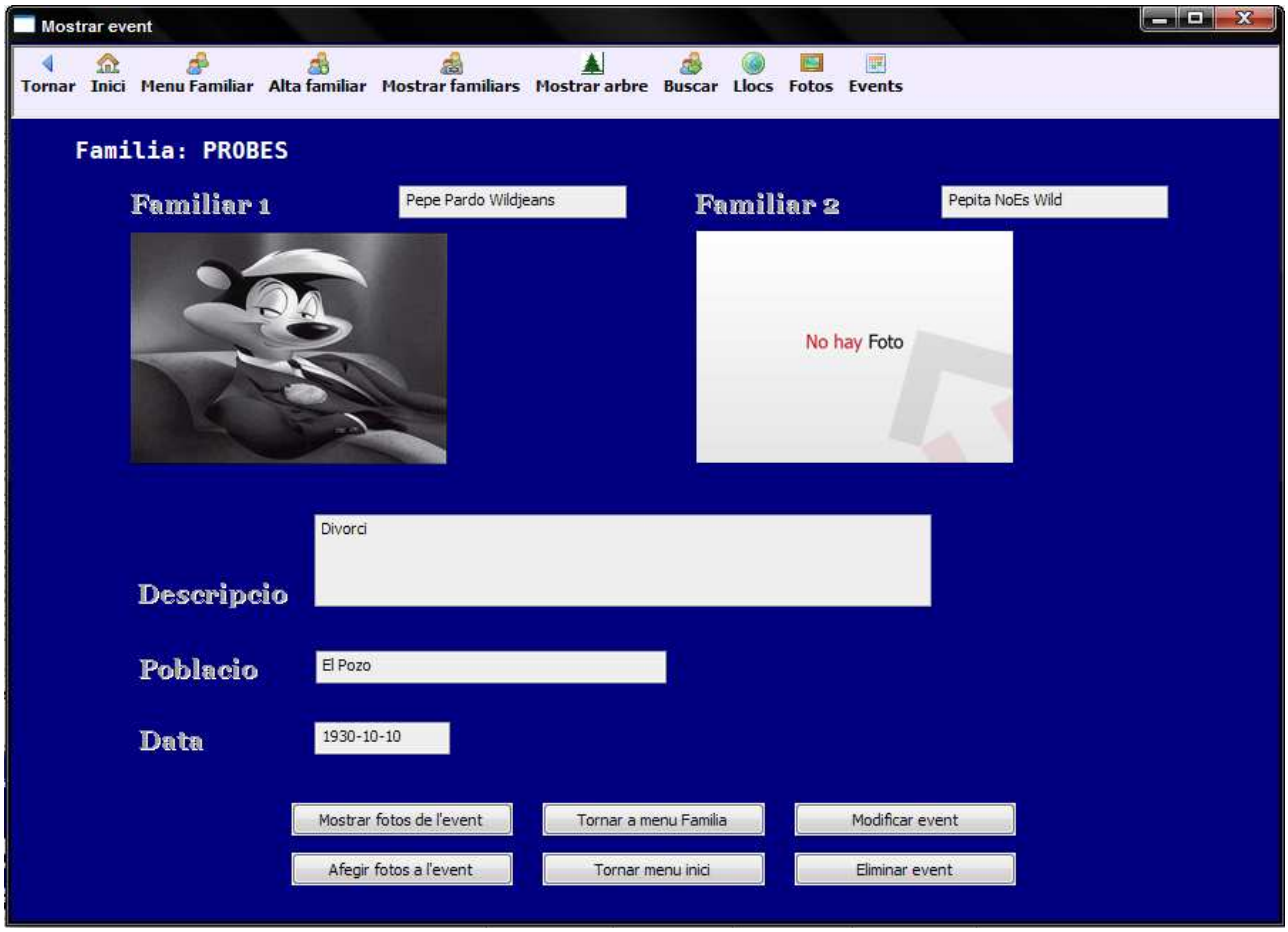


El que es mostra a la part superior és una llista d'estats i a la part inferior la possibilitat de crear-ne un de nou. En cas que l'estat que es vulgui introduir existeixi es mostrarà un missatge d'error, i en cas contrari es donarà d'alta i podrà ser escollit. Aquest mètode s'utilitza quan es vol donar d'alta una nova regió o un lloc.

Per arribar-hi primer s'ha de passar forçosament per mostrarRegió. Per tant quan es selcciona un dels estats de la llista és per donar d'alta alguna nova regió. (Quedarà més clar quan expliqui la interfície mostrarRegió).

6.6.26 Mètode MostrarEvent

Amb aquesta pantalla l'usuari pot visualitzar un event de forma detallada:

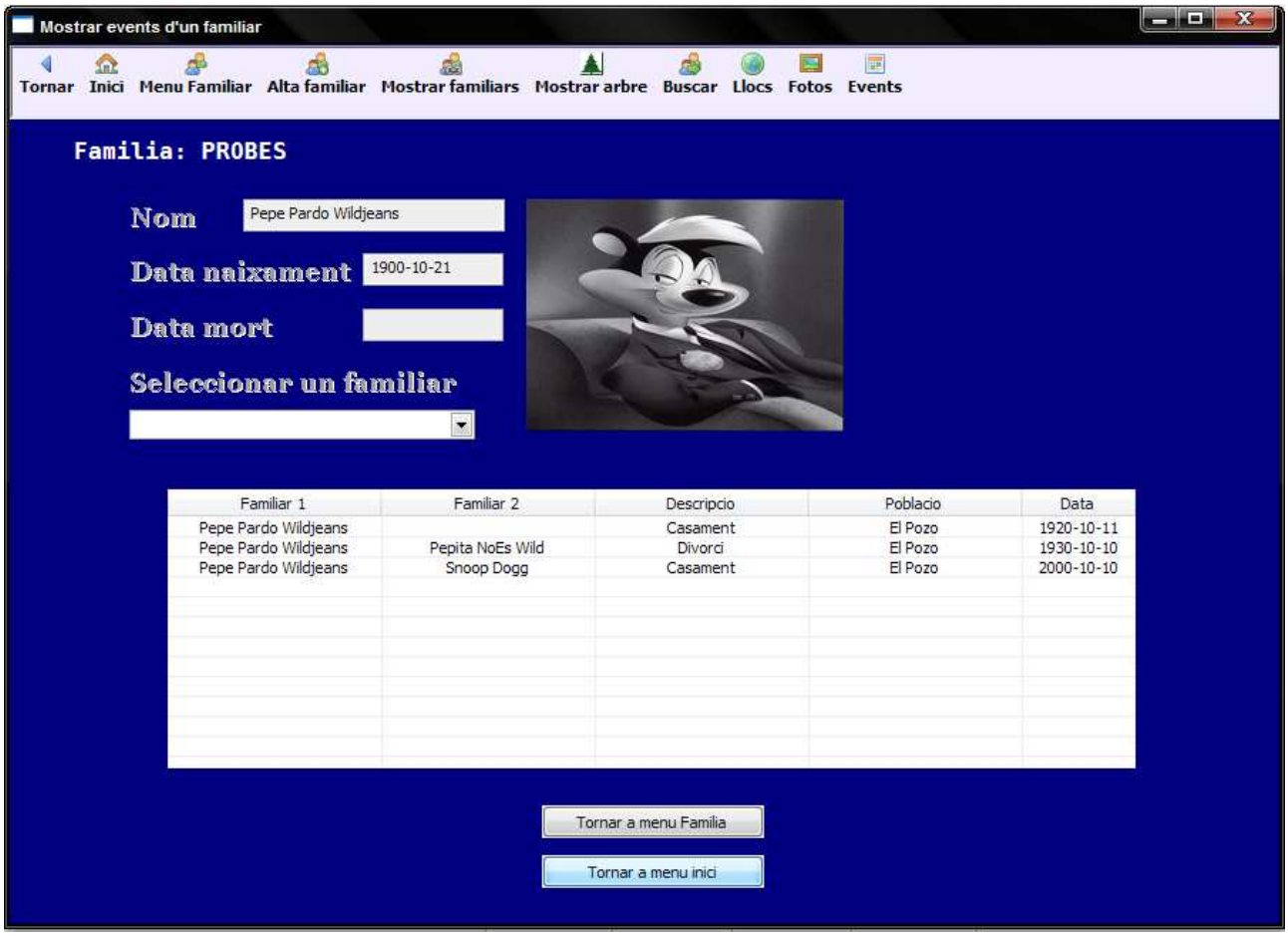


A la part superior apareixen el nom dels familiars de l'event (en cas que sigui un event d'un sol familiar, només n'apareixerà un); juntament amb les seves fotografies personals. Després es mostra la descripció, la població on va ocòrrer l'event i la data. Hi ha diverses opcions dividides en tres grups. A la part esquerra hi ha l'apartat de fotos: mostrar les fotos de l'event, així com també poder-ne afegir de noves.

A la part central hi ha les opcions de tornar a pantalles anteriors. I per finalitzar existeix la possibilitat de modificar l'event o bé eliminar-lo. Com sempre alhora d'eliminar es demana confirmació, a més en aquest cas l'eliminació comporta l'eliminació de tots els arxius que s'hi relacionen.

6.6.27 Mètode MostrarEventsFamiliar

La següent interfície permet a l'usuari veure un llistat amb tots els events relacionats a un familiar. El seu aspecte és el següent:

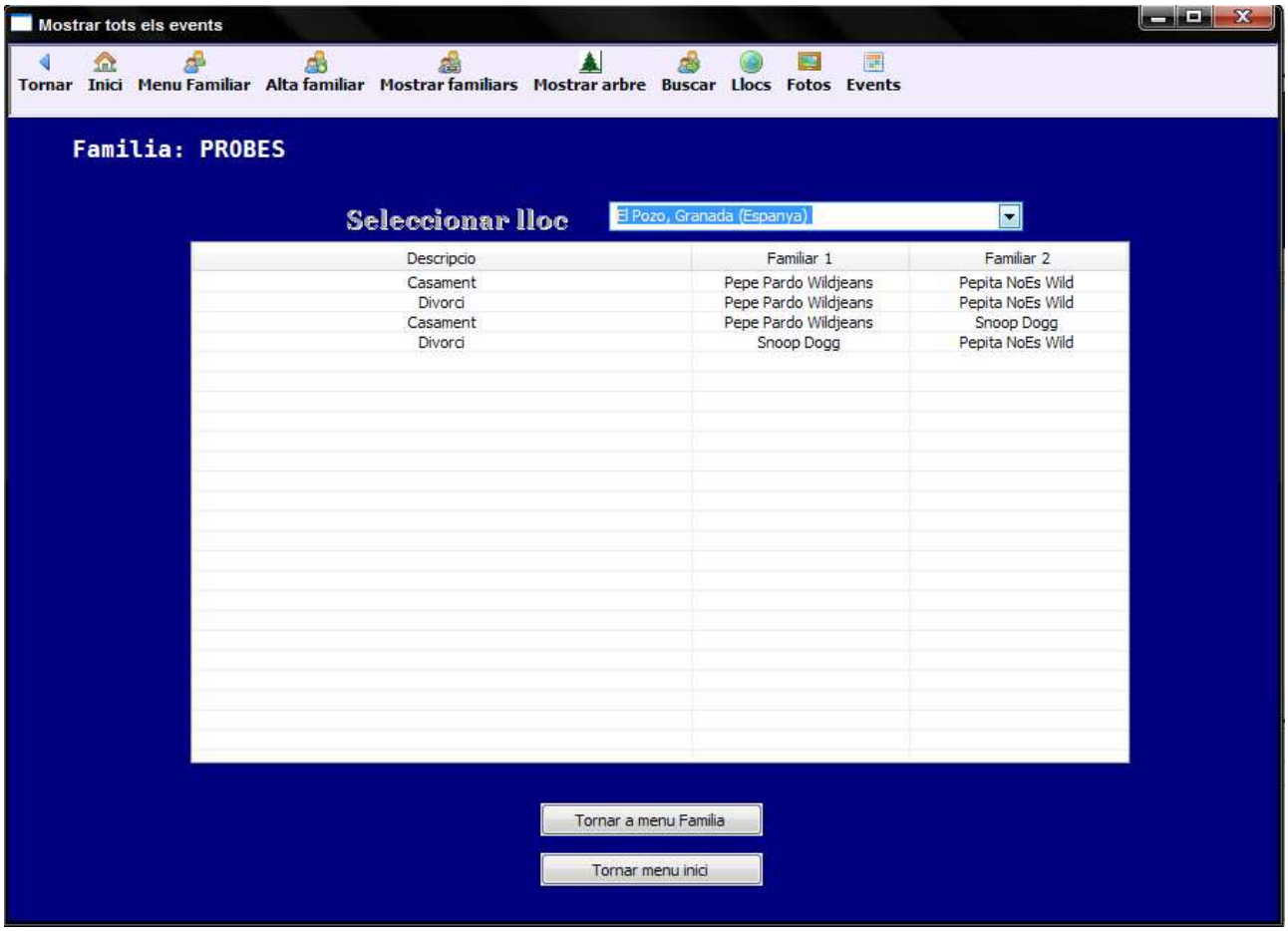


A més de algunes de les dades personals del familiar així com la seva foto, hi ha una llista de tots els events, en els quals hi forma part. Es mostren ambdos membres que el fomen, una descripció, la població i la data. A més si es selecciona algun es mostrarà de forma detallada, en una altra pantalla.

Finalment comentar, que he posat un combo amb tots els membres de la família. D'aquesta forma si es volen veure els events d'altres familiars només s'ha d'escollir el que es desitji. A més com el text és modificable, és molt més senzill trobar el familiar que es vol.

6.6.28 Mètode MostrarEventsLloc

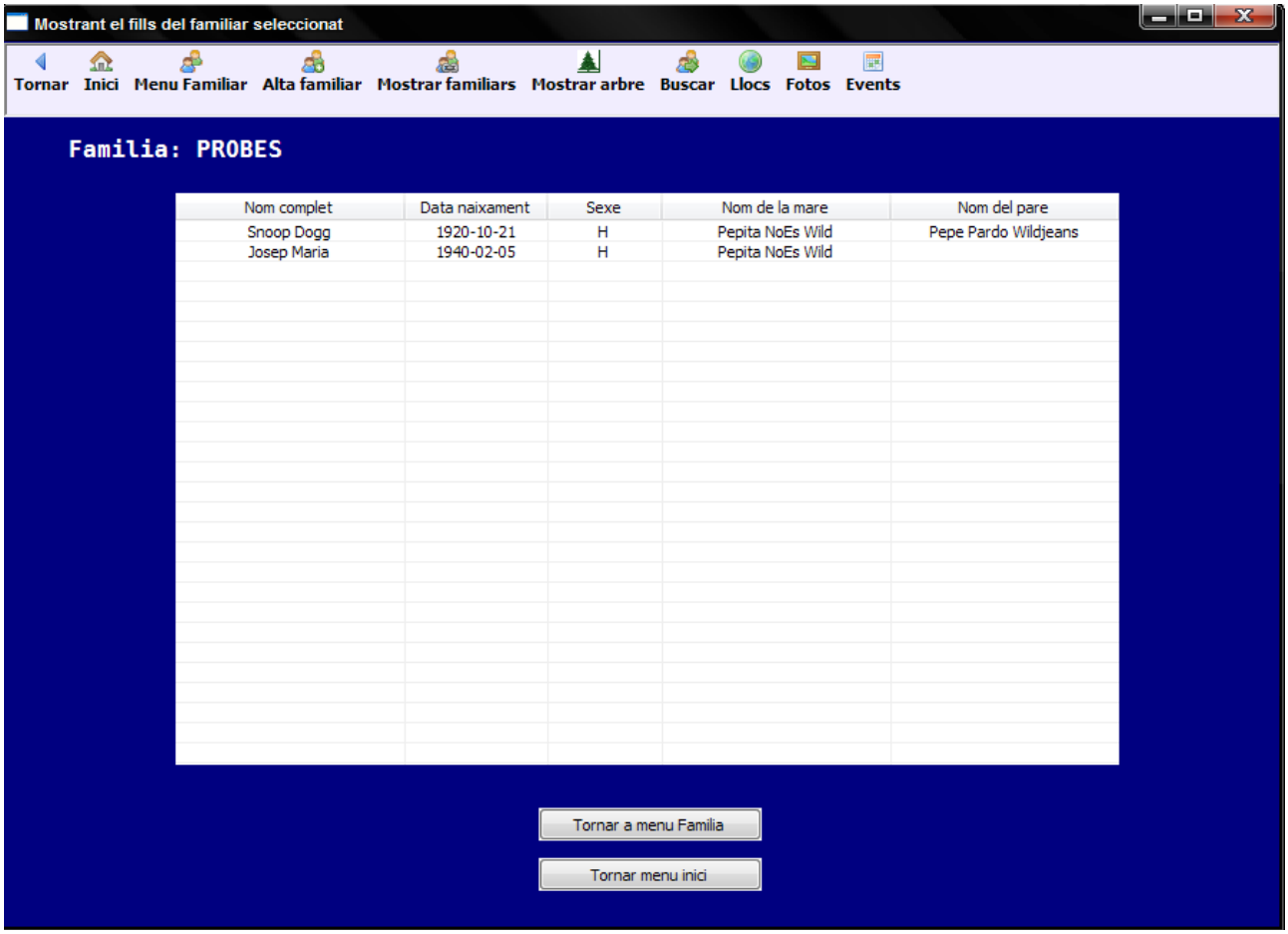
Es mostren tots els events que han passat en un indret en concret:



En la part superior hi ha un combo que permet seleccionar el poble que l'usuari vulgui. És editable, per tant, permet ceracr-lo de forma més ràpida. També s'hi pot observar una llista on hi ha tots els events que hi han succeït. La informació que es mostra és la descripció i el nom dels familiars (o del familiar en cas que sigui de un).

6.4.3.29 MostrarFills

Aquesta és una interfície semblant a la de mostrarDescendents, tot i que aquesta és una més senzilla. MostrarDescendents utilitzava en múltiples ocasions **buscarFills** (de la classe bdd), mentre que aquesta només ho utilitza una vegada, ja que només es volen obtenir els fills d'algun membre de la família:



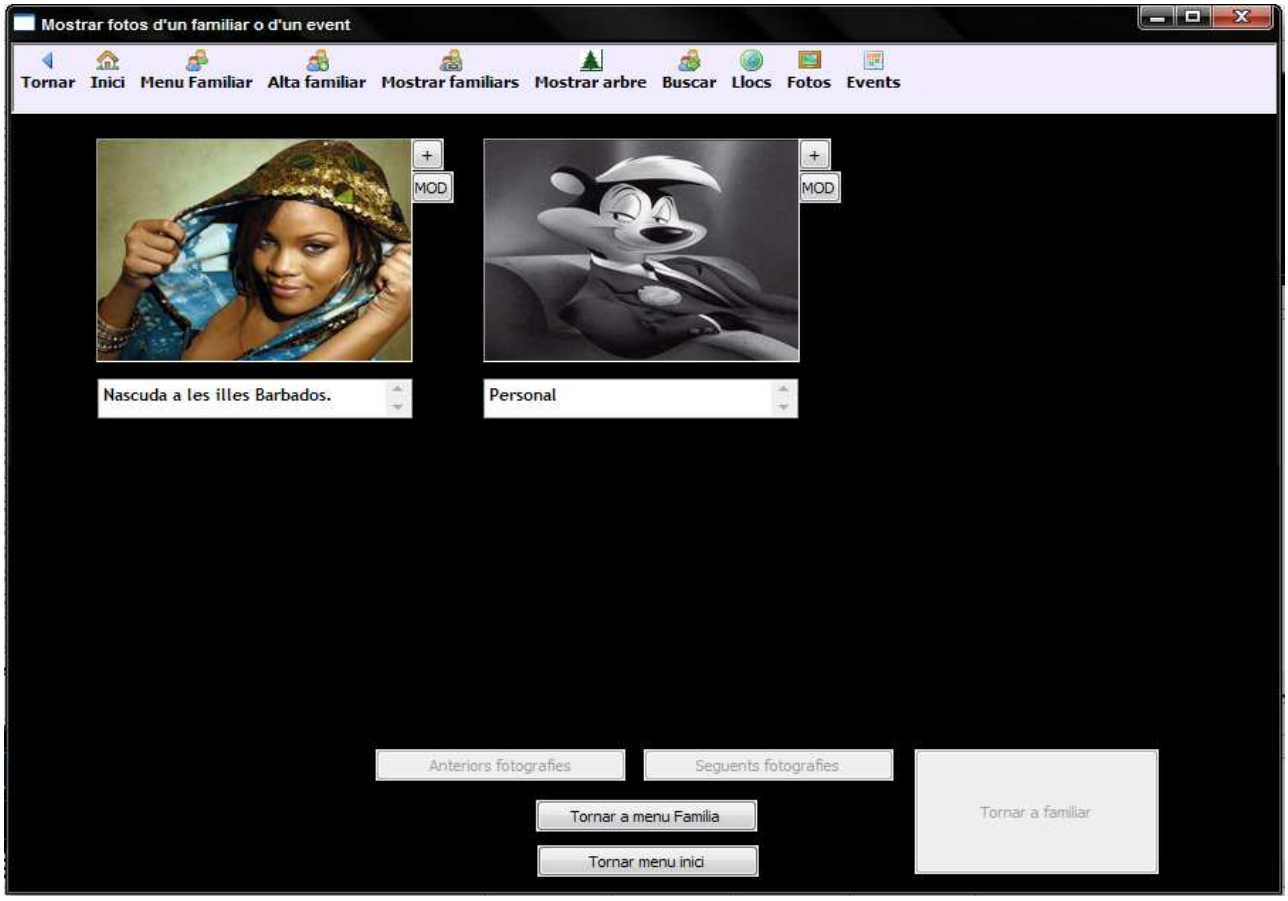
S'ha implementat una llista amb tots els fills d'algun dels familiars. Com s'ha comentat anteriorment, si es desitgen trobar els fills d'algun familiar, no comprobo si és un home o una dona. El que faig és una cerca per id, pero comparant tant a l'id del pare, com de la mare. D'aquesta forma obtinc, tot i que no està expressat de forma explícita, que parelles homosexuals tinguin fills. El resultat pot ser que en ocasions un familiar aparegui com a mare i en un altre cas com a pare.

6.6.30 Mètode MostrarFotosFamiliar

Aquest és un mètode que pot ser cridat especialment quan es mostren fotos per pantalla ja que hi ha la opció de veure-les a pantalla completa. Aquesta opció permet veure en detall una fotografía. En cas que la fotografia sigui més gran de 1024*768 (resolució estandard) es redueix a aquesta mida, mentre que si és més petita es mostra al seu tamany natural.

6.6.31 Mètode MostrarFotosFamiliar

Aquest mètode mostra totes les fotos del familiar seleccionat. Com que aquesta mateixa interfície, en realitat, també s'utilitza per mostrar les fotos d'un event, o d'una família, el que vaig decidir al implementar-ho va ser que se li passes per referència totes les fotografies (dins d'un vector), a la interfície. D'aquesta forma aconseguiria que ella mateixa pogués fos reutilitzable i que a més, en cas que no hi hagués cap fotografía, desde la classe que es cridés a aquesta interfície es sabria que s'havia de mostrar un missatge d'error tal que no hi ha fotografies que mostrar.

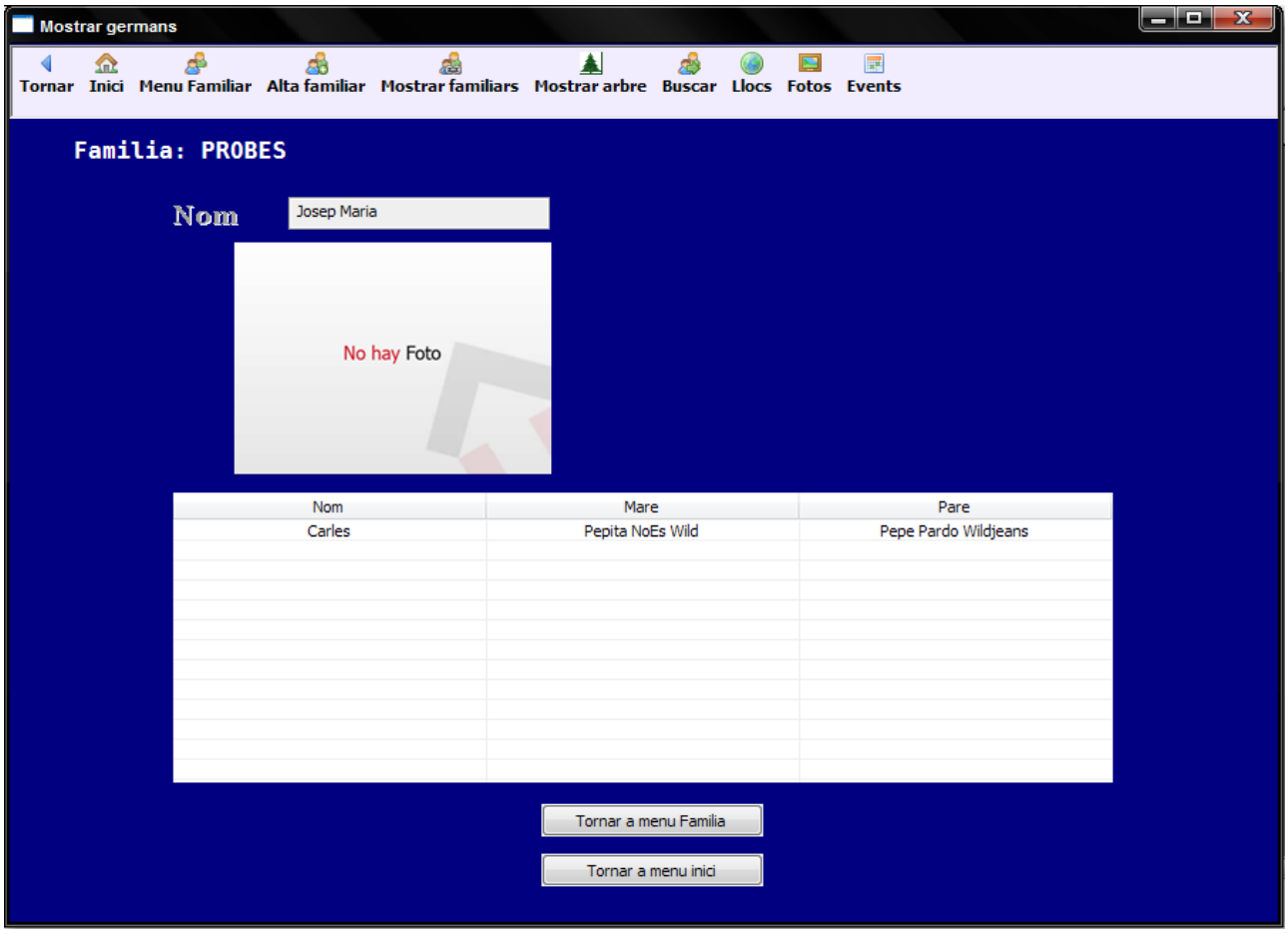


Com es pot veure a la imatge, es mostren les fotos amb la seva descripció a més de dos botons al costat de les mateixes. El que hi ha un + permet mostrar la foto a pantalla completa, mentre que el **MOD** permet modificar la fotografia.

En total poden haver-hi sis fotos per pantalla. En cas que el total de fotografies de la família sigui més de sis, s'activaran els botons següents i anteriors fotos, i es podrà anar veient totes les fotografíes que hi ha a la bdd.

6.6.32 Mètode MostrarGermans

Intefície que mostra a l'usuari tots els germans d'algun familiar:



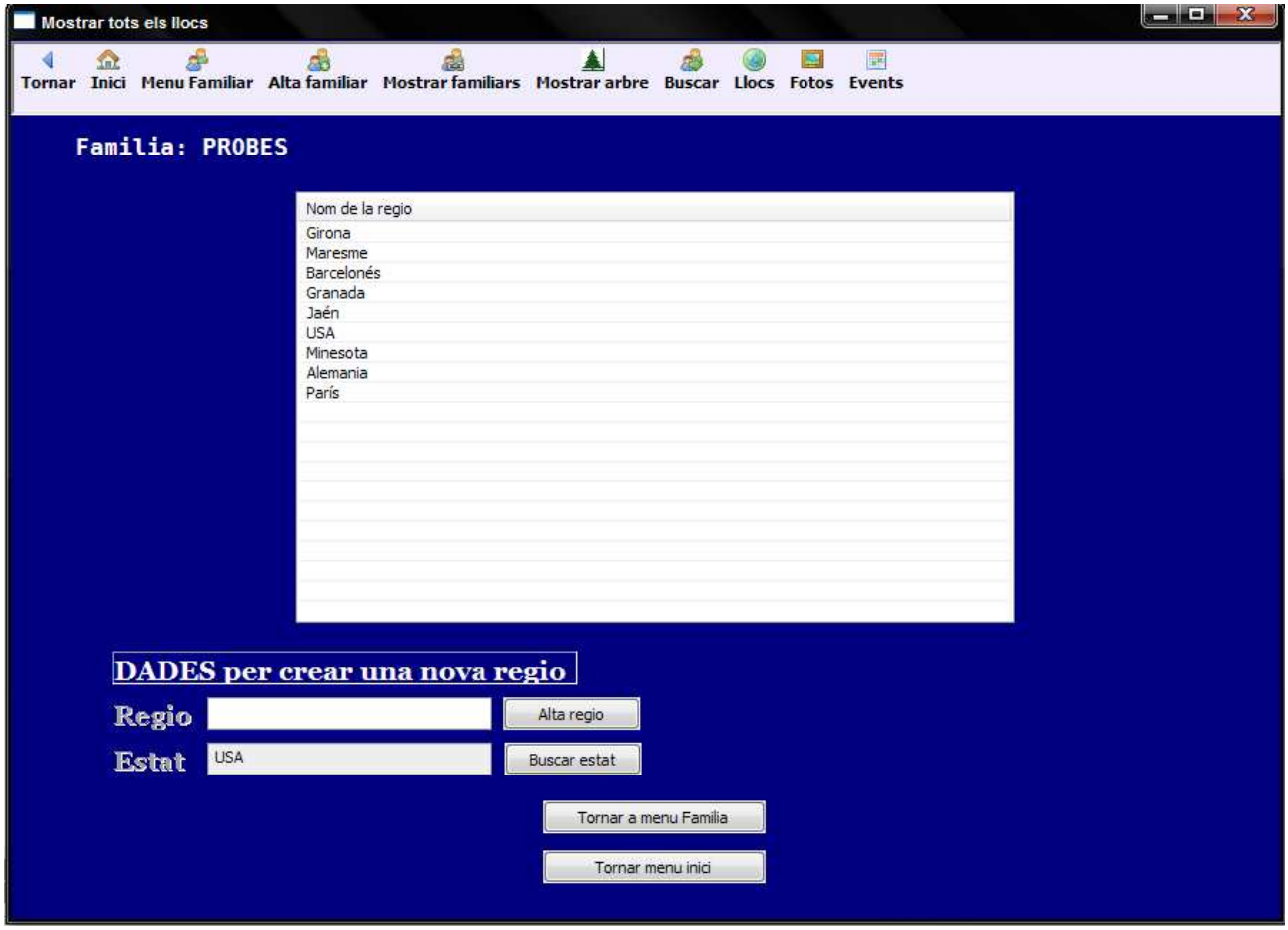
Es mostra el nom del familiar seleccionat i la seva foto, en la part superior. A més he codificat la llista de germans, entrats a la bdd. Dels germans apareix el seu nom, el de la seva mare i el del seu pare.

6.6.33 Mètode MostrarLlocs

Aquesta interfície no s'utilitzarà gaire, crec. Únicament mostra per pantalla la informació d'un lloc. En realitat tota la informació dels llocs apareix a **seleccionarLloc** que és la pantalla que s'utilitza, per a poder arribar a aquesta. Pero la rao de crear aquesta pantalla va ser que es poguessin afegir fotos als llocs. D'aquesta forma, també, els llocs i els arxius haguessin quedats relacionats, a més de la possibilitat d'afegir una nova vista a les ja existents. Aquest últim punt no ha estat implementat, i seria una bona millora a realitzar.

6.6.34 Mètode MostrarProvíncies

Es mostren totes les regions (a l'inici vaig posar províncies tot i que al final es va decidir, canviar-ho per alguna cosa més genèrica com regió):

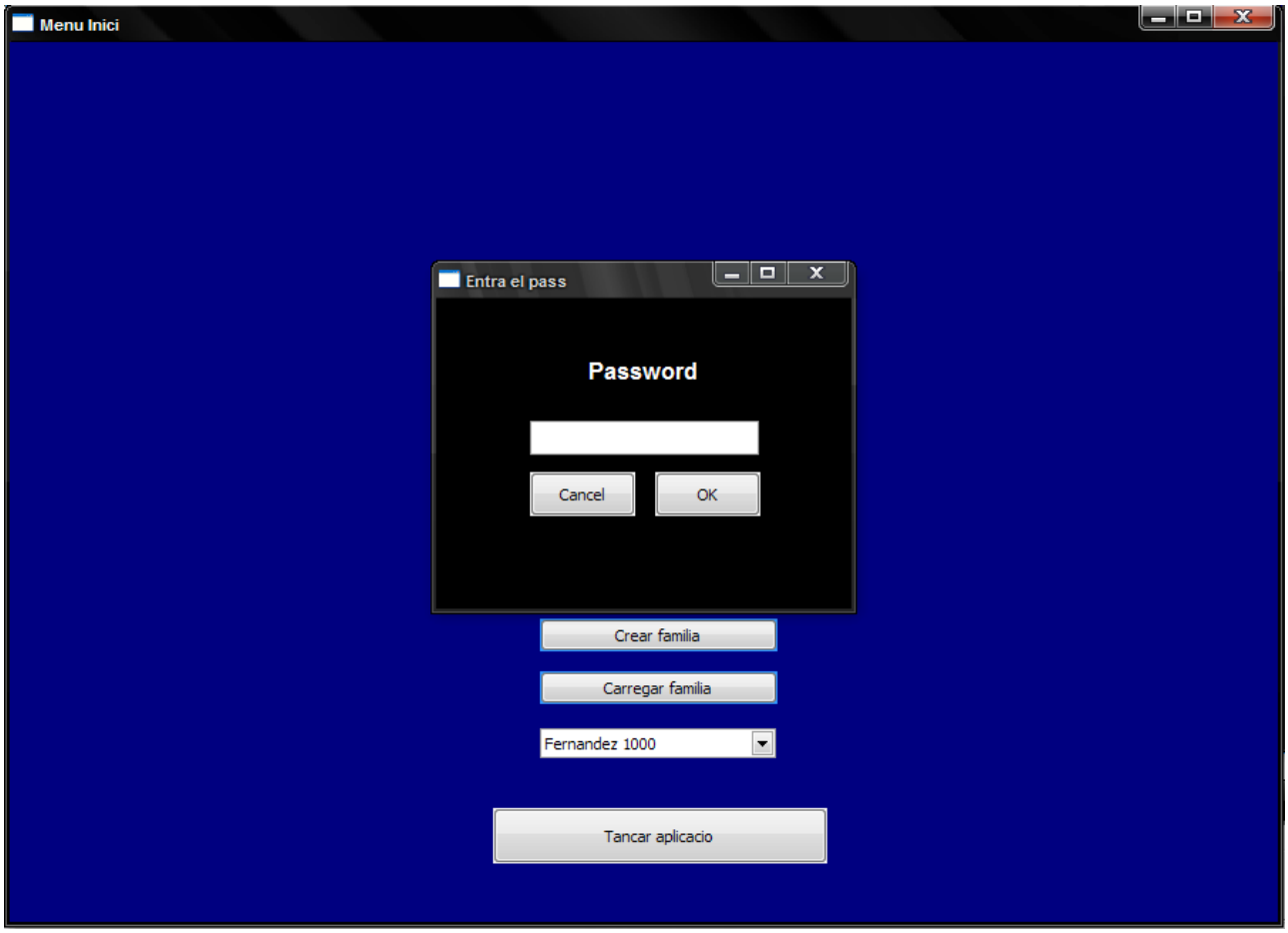


La funcionlitat d'aquesta pantalla és precissament escollir una regió perque sigui utilitzada per **SeleccionarLloc** (que després explicaré amb detall). Per tant si l'usuari desitja utilitzar alguna de les regions existents només l'ha de clicar. En cas que aquesta no es trobi introduida, té la opció de crear-ne una, dins la mateixa interfície.

A la part inferior hi ha dues opcions buscar estat i alta regió. Si es vol donar d'alta una regió primer cal seleccionar l'estat. Un cop seleccionat només cal escriure el nom de la nova regió i en cas que no estigui ja introduït a la bdd, es donarà d'alta inmediatament; i un cop donat d'alta apareixerà a la llista de regions possibles.

6.6.35 Mètode Password

Quan al menú inici es selecciona una família protegida s'ha d'introduir el password correctament. La pantalla que apareix és la següent:



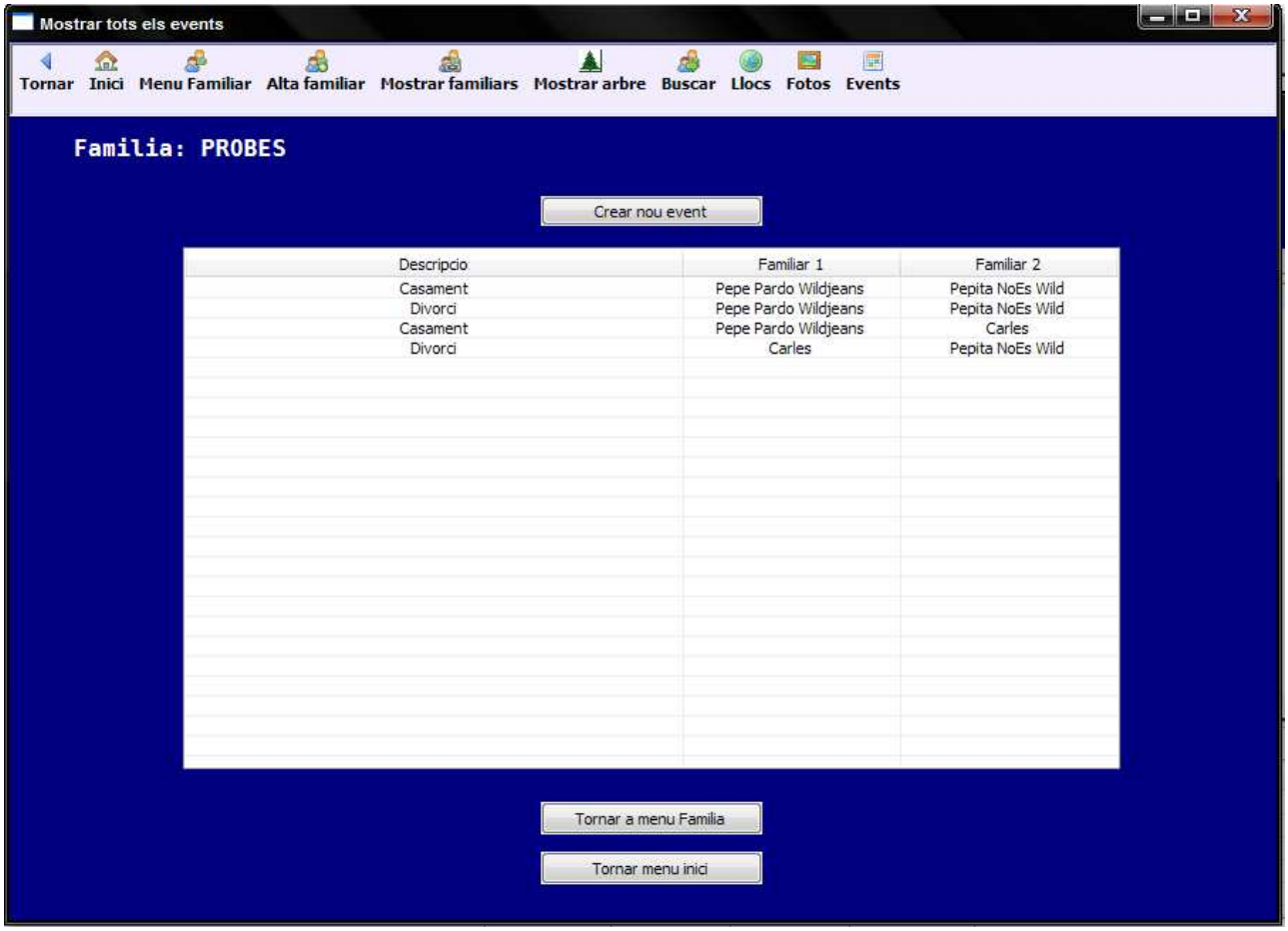
I en cas que la clau intruduida no sigui la correcta apareix el següent:

6.6.36 Mètode ResultatBuscarNom

Aquí es mostra a l'usuari tots aquells familiars que corresponguin amb la seqüència de caràcters escrita per l'usuari. Únicament es mostra el nom, la data de la naixament, telèfon i el sexe. I en cas de seleccionar-lo es mostra de forma detallada.

6.6.37 Mètode SeleccionarEvent

Aquestes últimes interfícies que queda per explicar han sigut les més complicades de desenvolupar, ja que s'hi pot accedir de moltes altres diferents. Això fa que s'hagi de comprobar sempre de quina interfície es ve i segons això, decidir a quina s'ha d'accedir quan l'usuari seleccioni algun dels objectes de la llista. El seu aspecte és el de la imatge:



En aquesta pantalla es mostren tots els events de la bdd, que corresponguin a la família actual (en aquest cas proves). Es mostra la descripció de l'event així com els familiars que el formen.

La part complicada del codi és la següent:

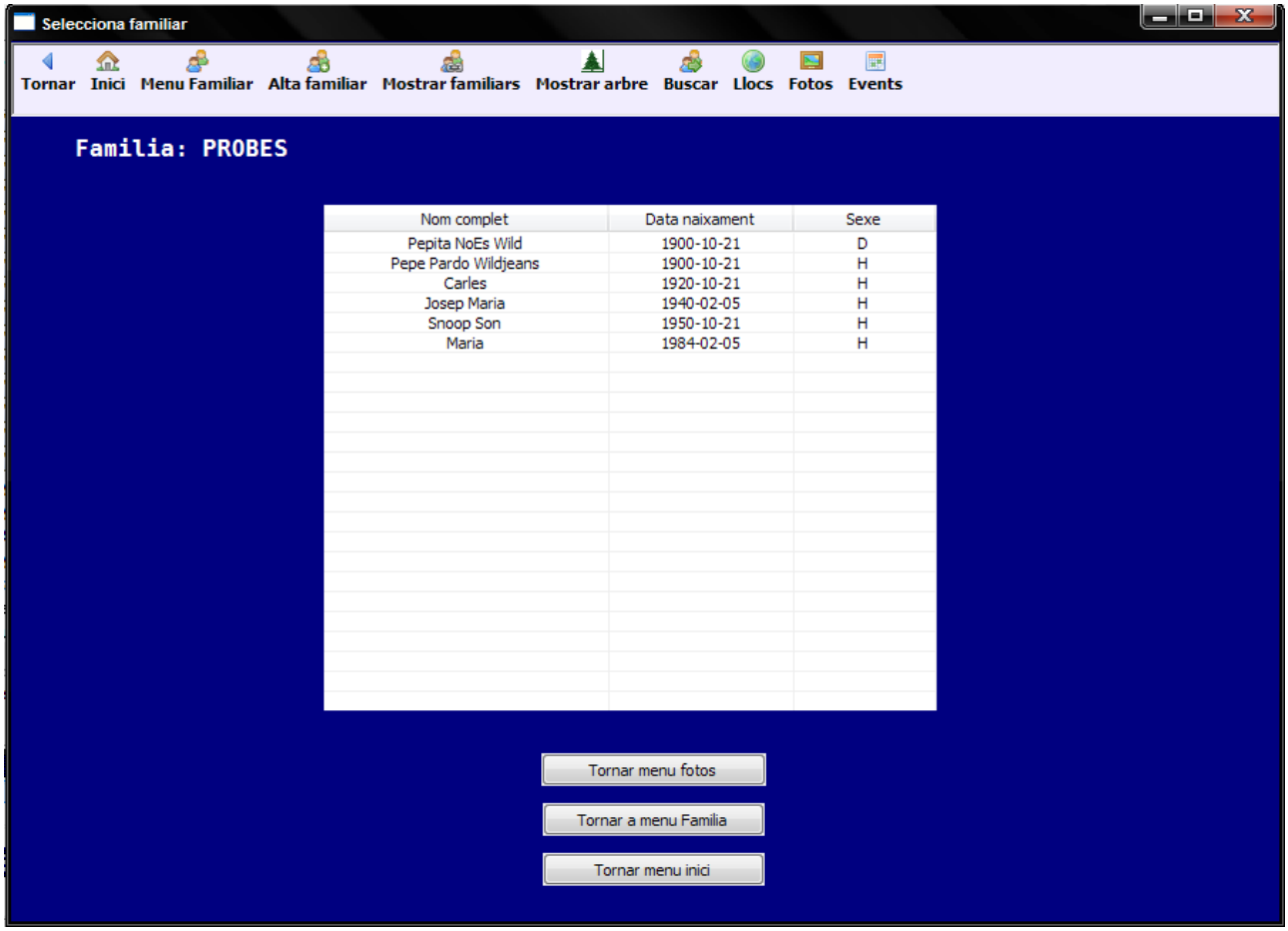
```
if(opcio.equals("MenuFotos")){
        //inicialització de les variables .....
        MostrarFotosFamiliar nova=new MostrarFotosFamiliar(dades,fotografies,1);
}else if((opcio.equals("MenuEvents"))||(opcio.equals("MostrarEvent"))){
        sShell.close();
        MostrarEvent nova=new MostrarEvent(dades);
}else if(opcio.equals("ModificarFoto")){
        //falta codi...
        ModificarFoto nova=new ModificarFoto(dades,fotografies,dades.getArxiu());
}else if(opcio.equals("AfegirFotos")){
        sShell.close();
        AfegirFotos nova=new AfegirFotos(dades,1);
}else{
        sShell.close();
        AfegirFotos nova=new AfegirFotos(dades,2);
}
```

Bona part del codi l'he borrat, perque quedi només la idea de les diferents possibilitats que hi ha. Per finalitzar destacar que aquesta pantalla s'utilitza, per exemple, quan es vol afegir una fotografia a un event. Quan s'ha de seleccionar l'event, el programari redirigeix a l'usuari cap a aquesta pantalla i un cop seleccionat l'event, aquest es guarda dins la variable dades. Al guardar-ho en aquesta classe quan es torna a afegir foto i es carreguen les dades, ja apareix l'event que acabava de seleccionar.

En la primera versió que havia fet hi havia un problema. Quan l'usuari, que ja venia de pantalles anteriors, per seleccionar l'event i aquest no estava introduït, havia de: primer anar al menú events i crear un nou event i llavors tornar a començar el procés d'afegir una foto a un event. Per tant al final vaig decidir afegir la opció de que desde aquesta mateixa pantalla es pogués crear. D'aquesta forma l'usuari guanya en interactivitat amb l'aplicació, pero la feina que s'ha realitzat per comprobar de totes les pantalles que es ve i d'arrossegar totes les variables d'una interfície a a una altra ha sigut molt major.

6.6.38 Mètode SeleccionarFamiliar

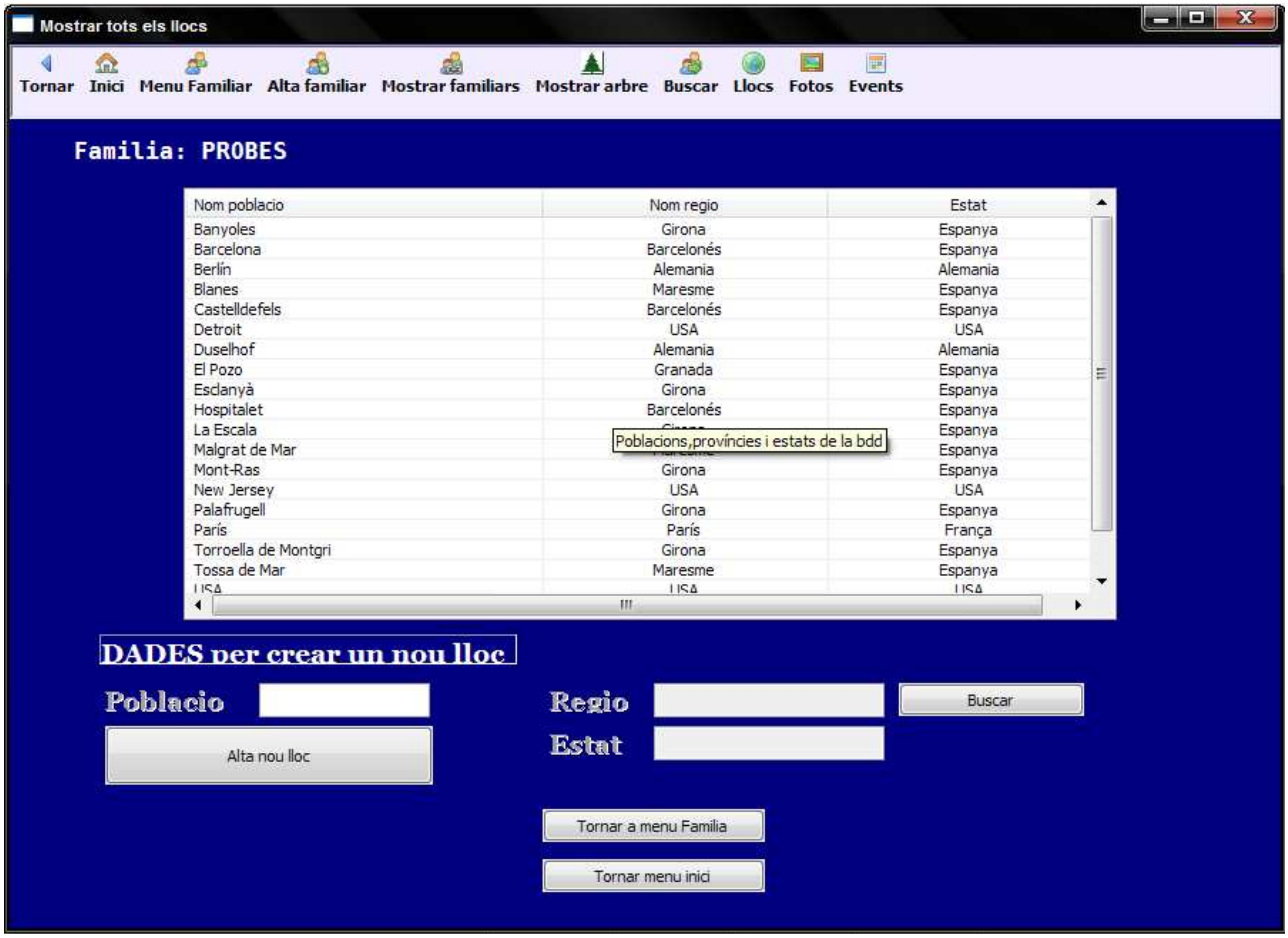
Aquest mètode s'assembla moltíssim a l'anterior, però en aquest cas es tracta de familiars i no d'events:



La taula es carrega amb tots els familiars, de la família, que hi hagi guardats a la bdd. De cada un es mostra el nom, la data de naixament i el sexe; mentre que si l'usuari el selecciona per l'acció que desitgi. Si, per exemple, la pantalla anterior era per afegir una foto al familiar, al clicar sobre algun dels membres de la família, significarà que serà el propietari de l'arxiu, i quedarà guardat com a tal.

6.6.39 Mètode SeleccionarLloc

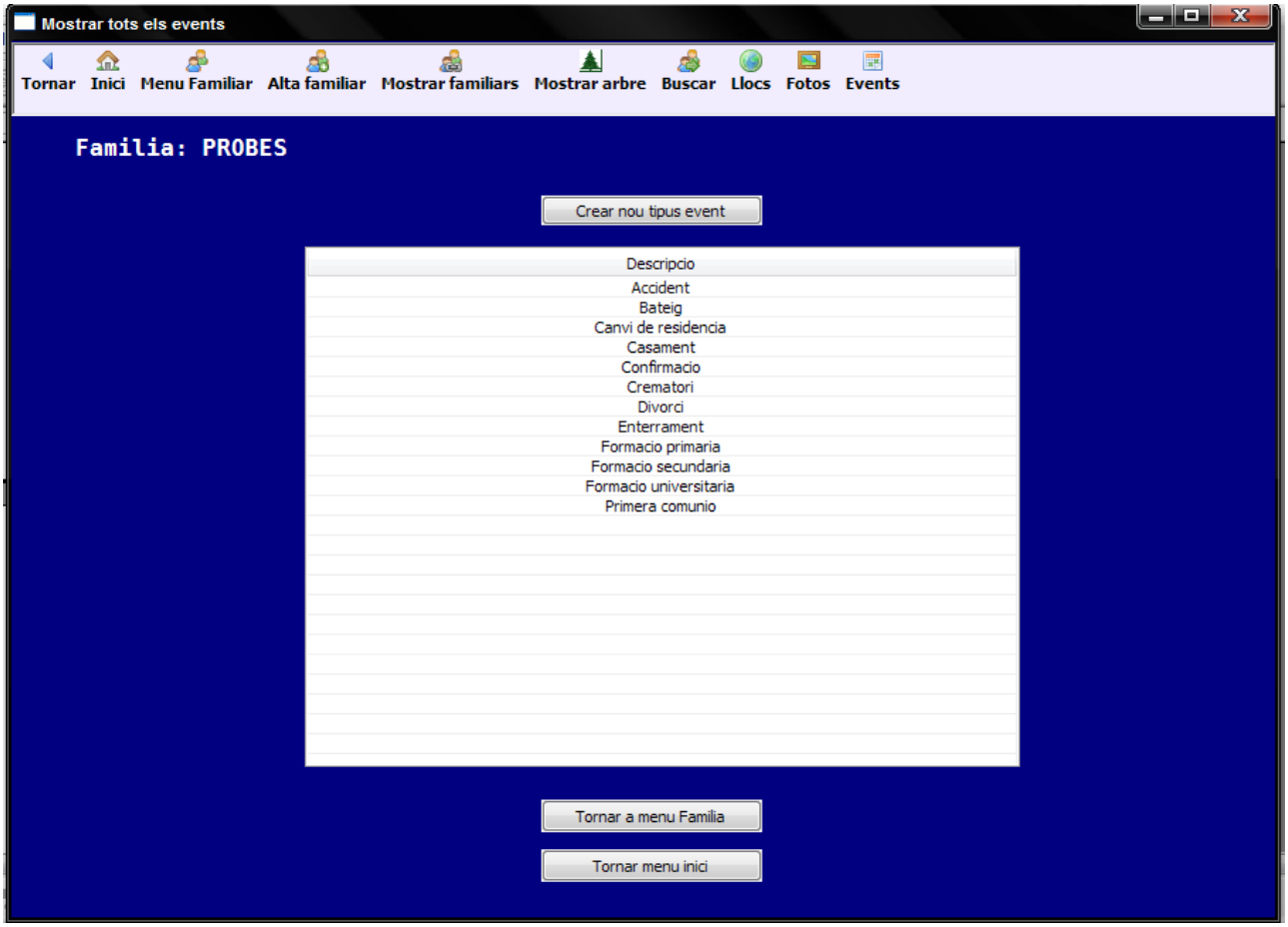
També s'assembla a les due anteriors. Permet seleccionar, en aquest cas un lloc, guardat a la bdd:



En cas que el lloc no estigui introduït a la bdd, és possible crear-ne un. Un cop estigui donat d'alta serà utilitzable de forma inmediata. Cal dir, com abans, que afegir aquesta opció a l'usuari alhora de seleccionar un lloc, l'ajuda força, pero la implementació per poder-ho fer, també ha sigut important.

6.6.40 Mètode SeleccionarTipusEvent

Aquesta interfície permet a l'usuari seleccionar un tipus d'event determinat. Si l'usuari no troba el desitjat és possible crear-ne de nous. Aquest és el seu aspecte...



6.7 Apartat web

6.7.1 Programari extra utilitzat

És important remarcar que eclipse és una eina molt potent i, que per tant alhora de decidir-me per utilitazar algun programari no va haver-hi cap dilema. A més com tot el projecte ha estat codificat amb aquest software, l'elecció encara era més evident. Tot i això va ser imprescindible descarregar el plugin WTP (web tools project) a més de descarregar el Tomcat, eina complementària tot i que necessària per poder carregar les pàgines en algun servidor.

6.7.2 Justificació i explicació del codi

La idea principal de la pàgina web és que diverses persones de la mateixa família puguin conèixer certa informació de la mateixa família via web, sense necessitat d'utilitzar el programa, tot i que hi ha, evidentment, una compatibilitat absoluta. D'aquesta forma només que un dels familiars crei un compte d'usuari a la pàgina i comenci a emplenar-la, els altres familiars la podran retocar i afegir informació personal, per acabar, entre tots, recopil.lant molta informació de la família.

La primera versió de la pàgina web no era gaire bona i es va remodelar totalment. Finalment s'han utilitzat servlets, listener i totes les eines relacionades amb JSP, per acabar obtenint una pàgina web més segura i estable, i que utilitza molts recursos dels que ofereix el programari eclipse. També al utilitzar aquesta metodologia, s'ha aconseguit reutilitzar totes les classes que s'havien generat amb el programari.

L'apartat web és millorable en certs aspectes, com es comentarà en un dels punts posteriors. Un dels punts importants a millorar és que la meva idea inicial era que la pàgina web pogués realitzar totes les accions que podia fer el programari, tot i que finalment només en pot efectuar una part. Tot i això els requisits bàsics s'han comlert.

La pàgina inicial de la web és la de login. En cas que l'usuari no estigui registrat es dona la possibilitat de donar-se d'alta introduint el nom que vol utilitzar, com també la seva contrasenya. En cas que l'usuari estigui donat d'alta i, les dades siguin correctes es mostra la pantalla principal de l'usuari. En aquest punt hi ha dues opcions que l'usuari tingui assignada una família o que no. En cas que no tingui assignada cap família, l'usuari no podrà realitzar cap operació.

En cas que l'usuari conegui el nom i la contrasenya de la família a la bdd, la pot seleccionar, tot i que també hi ha la possibilitat de crear-ne una de nova. A la web, igual que a l'aplicació, l'usuari té la possibilitat de crear i consultar diferents famílies. Quan l'usuari selecciona una família se li queda predeterminada per totes les altres sessions que inicia, pero en qualsevol moment la pot canviar, sempre que, evidentment, coneixi el nom i la contrasenya de la família.

Un cop carregada la família apareixen les possibilitats que la web ofereix, que no són totes les que ofereix el programari, explicat anteriorment. Existeix la possibilitat de mostrar tots els familiars de la família. També es poden crear nous familiars, modificar les dades dels mateix (inclòs la modificació dels progenitors, per mitjà de la selecció d'algun dels familiars ja introduïts), es poden eliminar familiars i també es mostra l'arbre del familiar, tot i que de forma detallada.

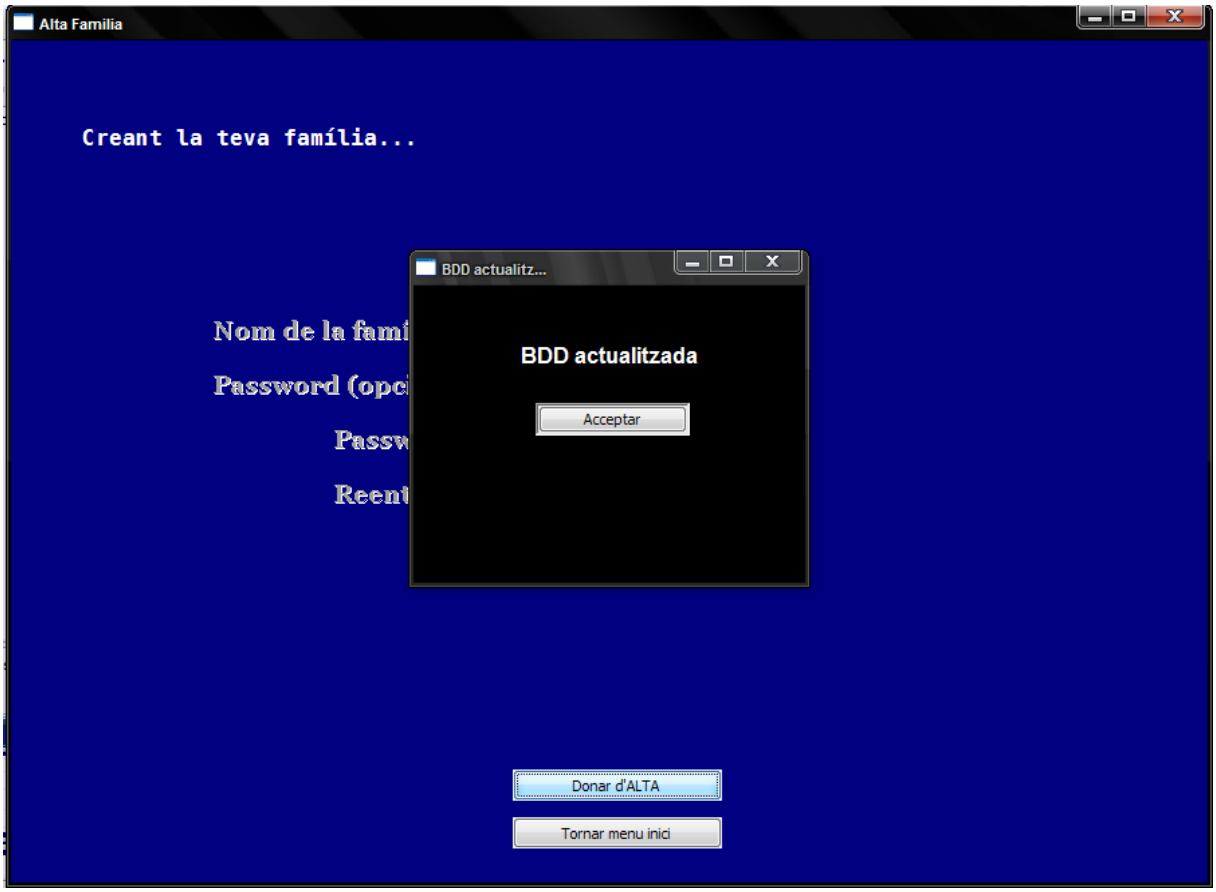
7. Altres aspectes

7.1 Probes d'execució

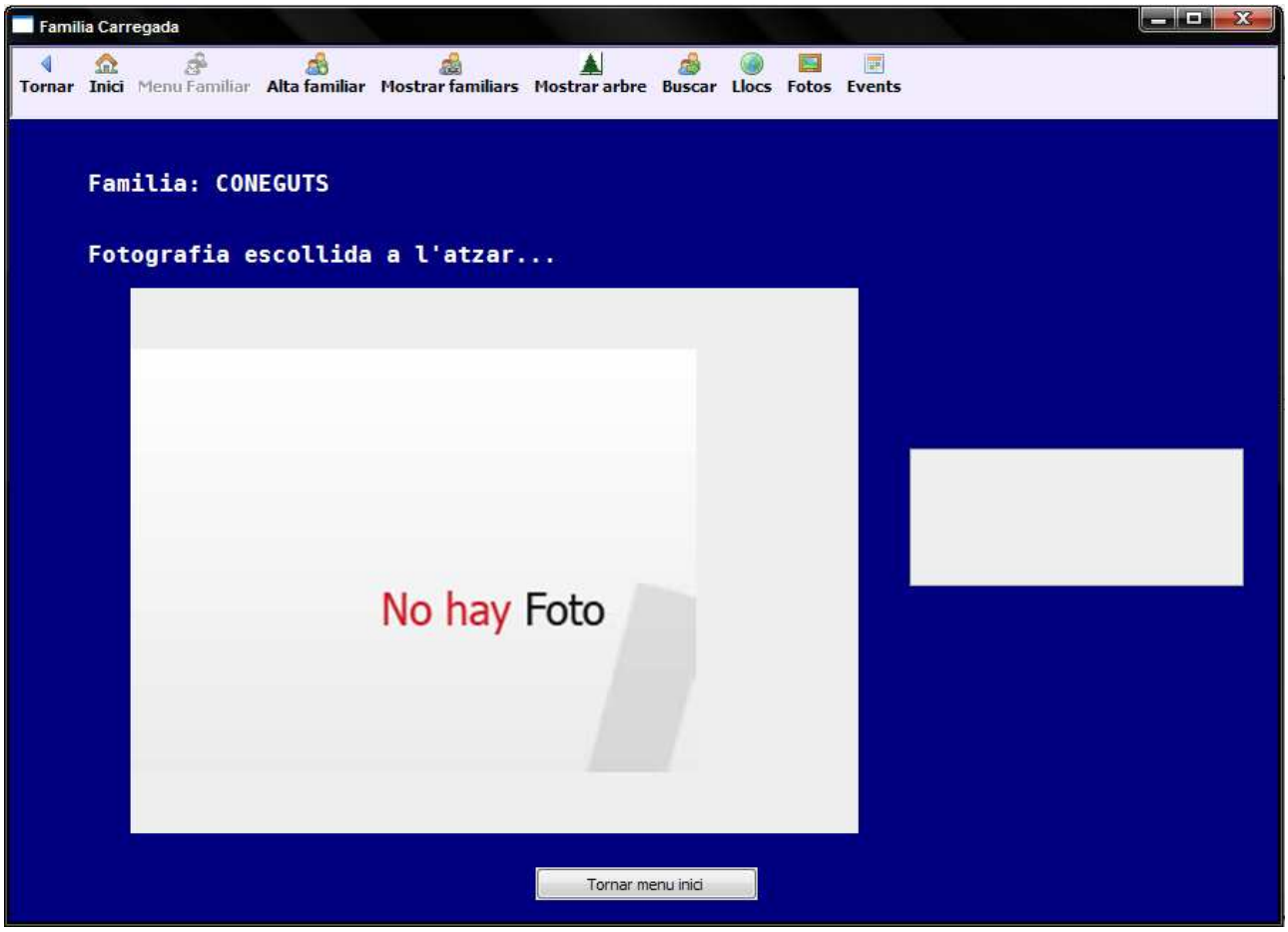
A continuació realitzaré un seguit de proves per mostrar com el programari funciona correctament. En principi realitzaré les opcions més importants i no repetiré proves semblants, només ho comentaré. Part de les proves d'execució realitzaden han estat efectuades per alguns amics i altres familiars. Gràcies a això alguns apartats que no quedaven del tot clars, han estat modificats per obtenir una interacció amb l'usuari el més gran possible, que era un dels principals requisits que volia obtenir amb aquesta aplicació. Finalment destacar que el nivell dels usuaris era força diferent, d'aquesta forma encara el resultat de les proves, ha estat el més ampli possible.

7.1.1 Creació de dades

La primera proba a realitzar, és la de crear una familia. En crearé una anomenada **coneguts** amb contrasenya, igual al nom de la família.



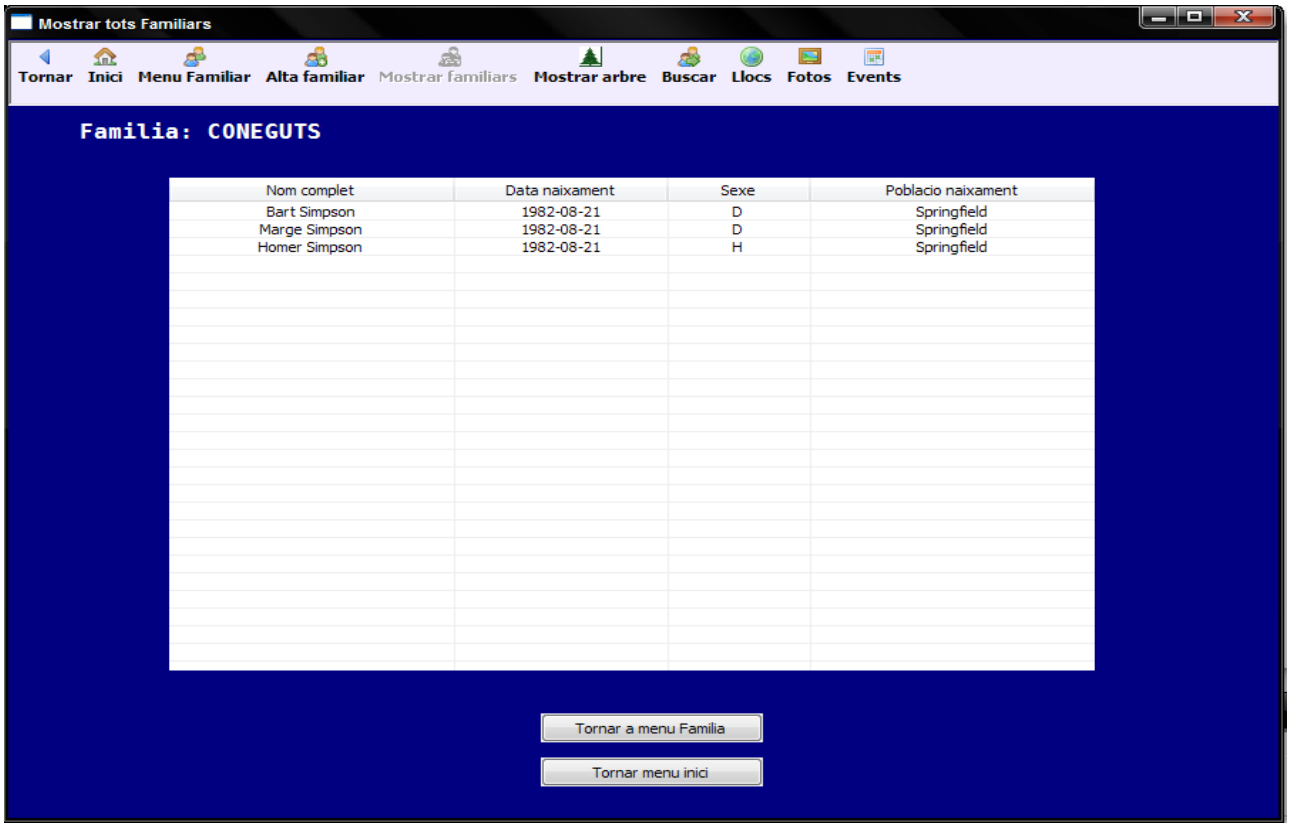
Després de donar d'alta la família, apareix un missatge tal que la bdd ha estat actualitzada. En el següent pas directament es redirigeix a l'usuari cap a la pantalla de **familiaCaregada**. Com és la primera vegada que s'entra no s'ha carregat cap fotografia i el resultat és el següent:



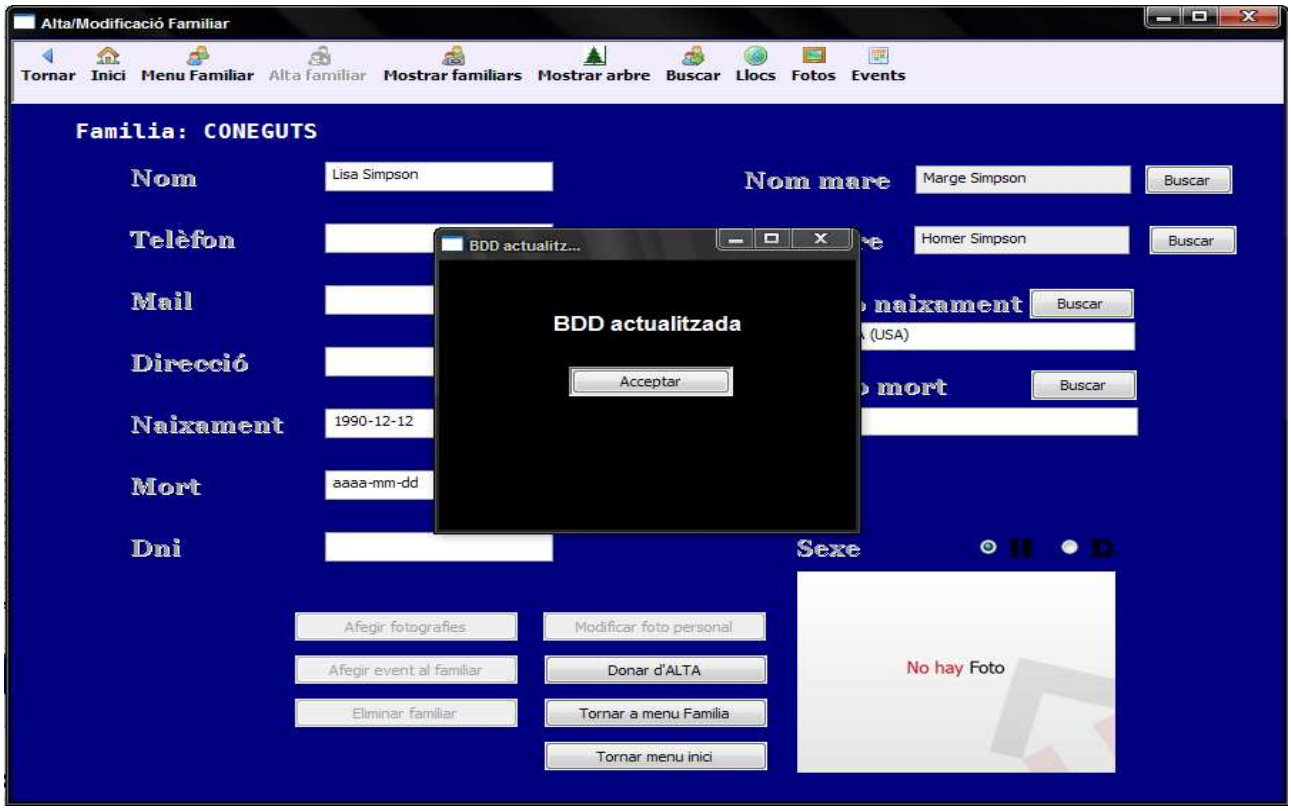
Els familiars que he afegit a la bdd són:

```
INSERT INTO FAMILIAR VALUES (10,'40404000A',1002,'1982-08-21' ,NULL,'Homer Simpson' , 0 ,'', '', 'Springfield' ,NULL, 0, 0, 'H');

INSERT INTO FAMILIAR VALUES (11,'',1002,'1982-08-21' ,NULL,'Marge Simpson' , 0 ,'', '', 'Springfield' ,NULL, 0, 0, 'D');

INSERT INTO FAMILIAR VALUES (12,'',1002,'1982-08-21' ,NULL,'Bart Simpson' , 0 ,'', '', 'Springfield' ,NULL, 10, 11, 'D');
```

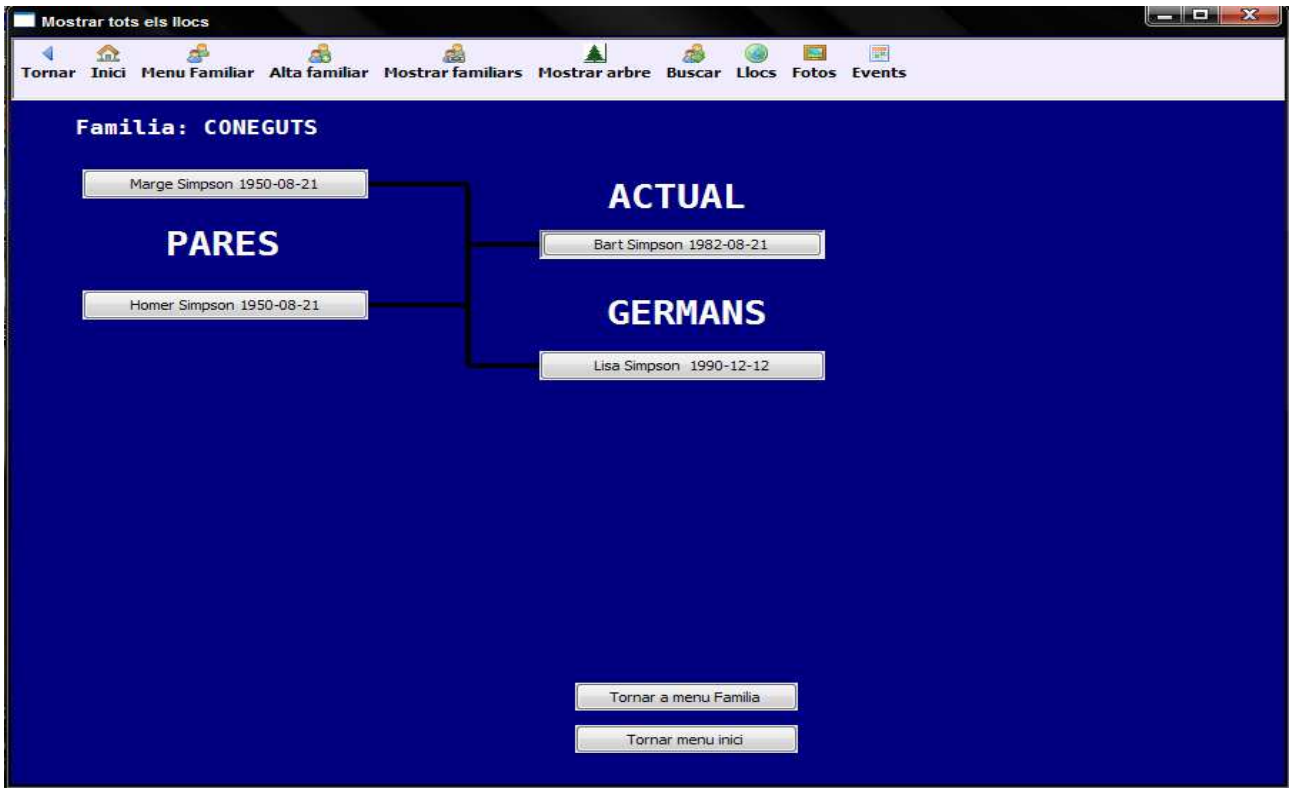
Aquestes dades les he introduït directament desde mySql i comprobaré que s'hagin introduït correctament:



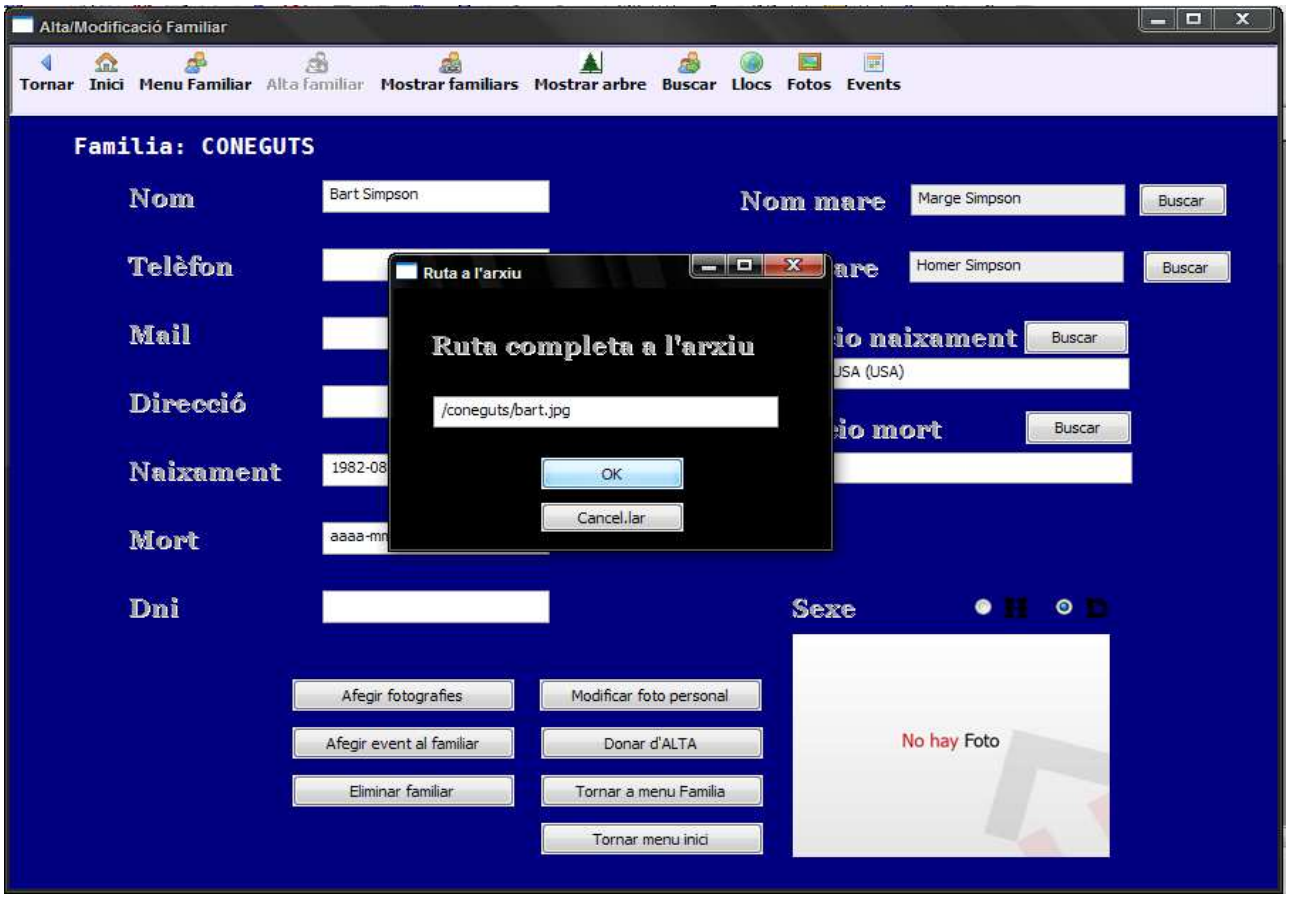
Com es pot comprobar les dades han estat correctament introduïdes. Ara introduiré un familiar desde l'**altaFamiliar**...



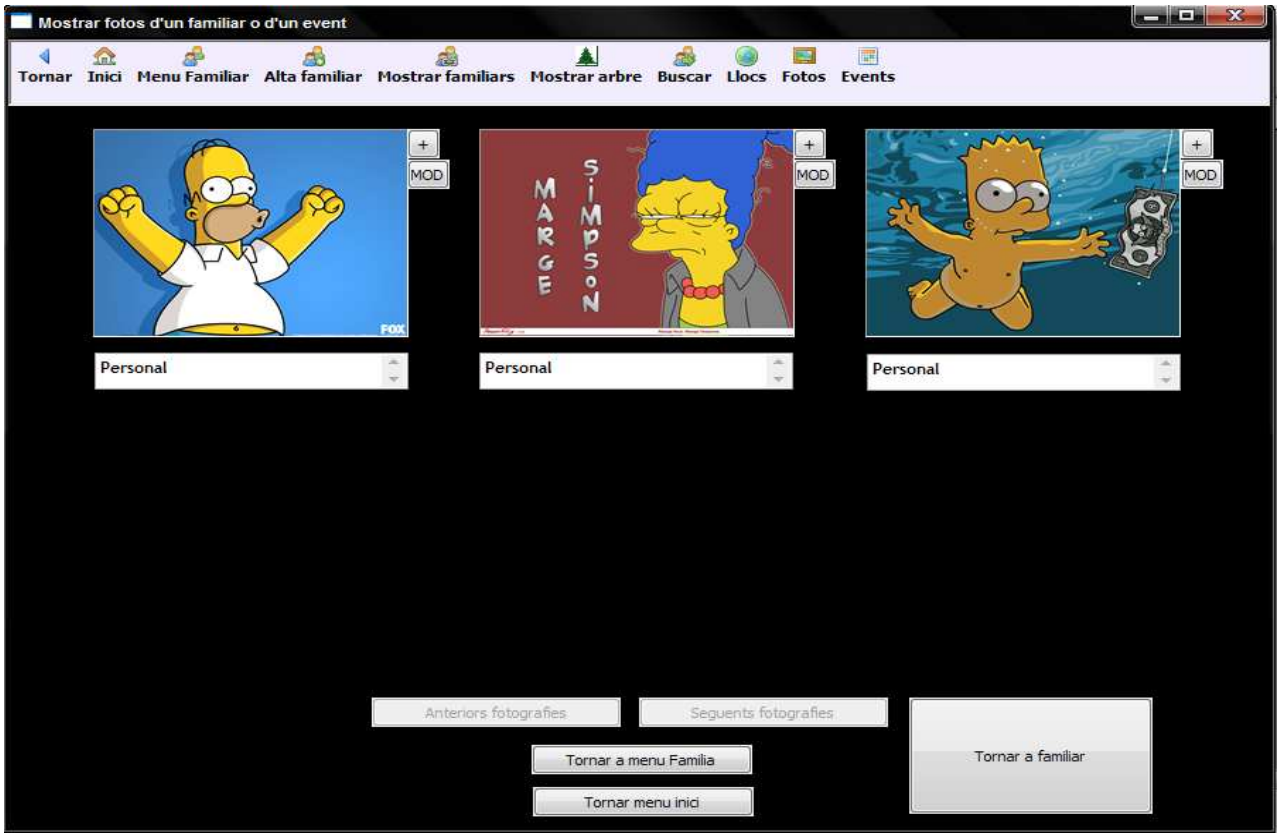
Ara mostraré com queda l'arbre familiar, després d'afegir aquest nou familiar:



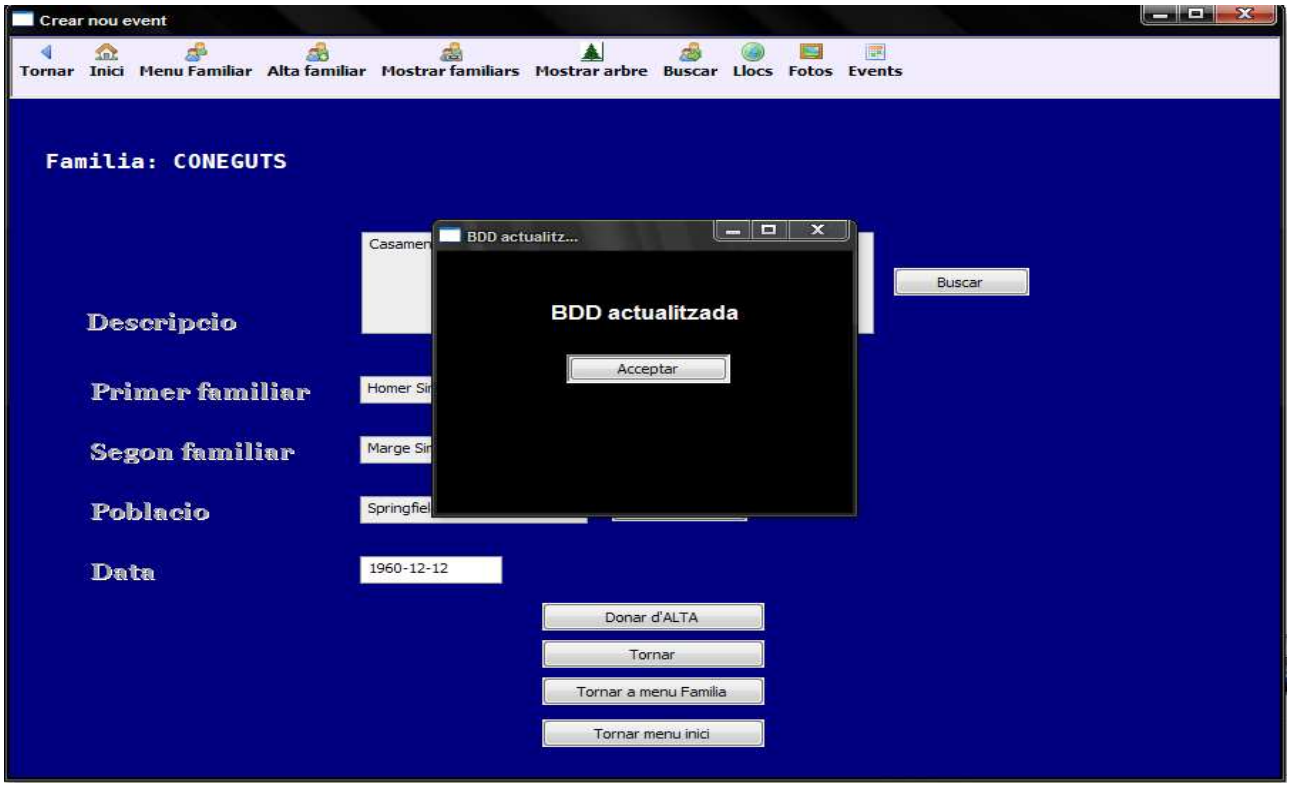
Com es pot observar s'ha introduït el nou familiar i el funcionament de l'arbre també és correcte. Ara afegiré alguna imatge als familiars:



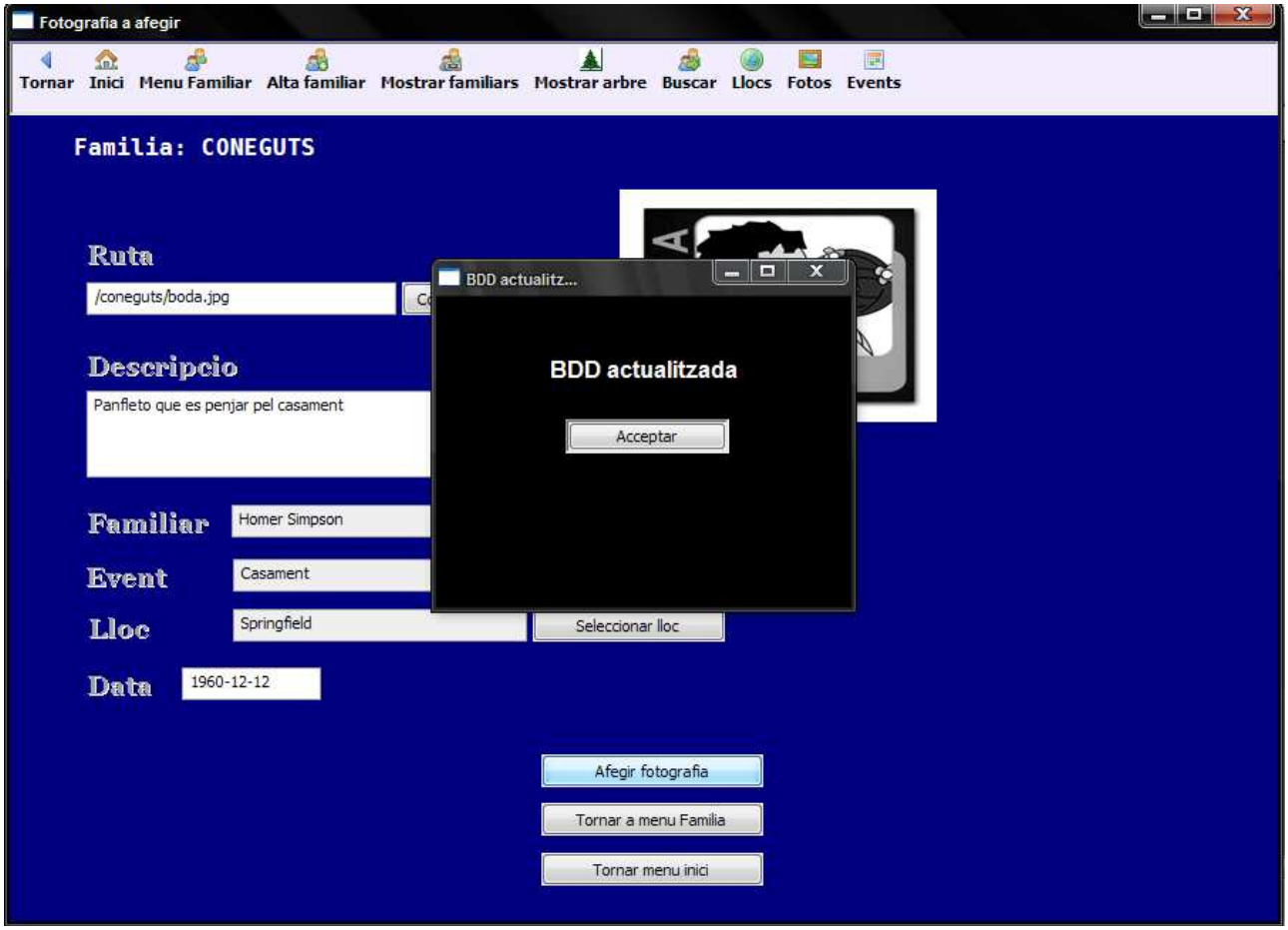
La fotografia ha estat introduida ja que al tornar la foto ha estat carregada directament. El que faré es mostrar totes les fotos afegides:



Ara afegiré un event casament:



i ara afegiré una foto a aquest event:



El programari té moltes opcions (no he mostrat cap de les possibles cerques que es poden realitzar), pero crec que entre el que he mostrat anteriorment, el que he mostrat ara i el que es mostrarà el dia de la presentació hi ha proves suficients.

7.2 Problemes trobats

El primer problema que vaig tenir va ser la impossibilitat de crear una connexió entre la bdd que volia utiliztar OBase (del paquet de programes d'OpenOffice). Per crear la bdd amb aquest programari no vaig tenir cap dificultat, pero alhora de poder-ho connectar amb l'eclipse, em va ser impossible. Vaig intentar buscar per internet ajuda i fins i tots vaig enviar alguns emails, pero ni vaig trobar gaire bibiliografia sobre el tema, ni vaig conseguir que em responguessin a la pregunta. Al final es va obtar per la opció MySql, amb el que no vaig tenir cap problema ni per crear la connexió ODBC ni tampoc per sincronitzar-lo amb l'Eclipse.

Un altre dels problemes amb que hem vaig trobar, tot i que aquest va tenir solució molt més ràpida, era que l'eclipse sense extres, no té la possibilitat de treballar las classes interfície de forma directa sobre el gràfic, com per exemple **Jbuilder**. Pero buscant en la pàgina www.eclipse.org, vaig trobar-la i amb el paquet descarregable Visual Editor, el problema va quedar resolt; ja que permet treballar directament sobre les classes VisualClass, de forma que agilitza molt la creació de les interfícies.

Un altre dels problemes que m'he trobat és la impossibilitat d'utilitzar un explorador de carpetes i obtenir el resultat. És a dir, per exemple, alhora d'inserir la ruta d'una fotografia, l'ideal seria posar un botó per explorar les carpetes del sistema operatiu i acabar seleccionant l'arxiu desitjat. Ho vaig implementar, pero un cop seleccionat no aconseguia obtenir el resultat correcte. Després d'intentar-ho molt, vaig decidir obtar per la solució d'introduir la ruta manual (desde la carpeta imatges, situada com a accés directe a l'escritori i d'aquesta forma ajudar a afegir noves fotots).

Un dels problemes més importants amb que em vaig trobar va ser el de la navegació de les pantalles. Per redireccionar a l'usuari a les múltiples opcions, no hi havia cap problema. Aquest residia quan es volia tornar enrere. Si es volia tornar a una pantalla, únicament es tornava a carregar la pantalla pero no les dades que s'havien mostrat i, això en molts casos mostrava dades que no eren les desitjades. Per tant la primer solució per la que vaig apostar, va ser passar tot un seguit de variables per referència (en realitat una variable per cada un dels possibles objectes Familiar, Arxiu, Event, Lloc; a més d'un String **donVinc** que contingués el nom de la interfície anterior).

Aquesta primera solució funcionava força bé pero encara tenia algun problema. Tornar a la pantalla anterior funcionava correctament, fins i tot carregava les dades que l'usuari havia seleccionat. Pero si es volia tornar dues o tres enrere, ara feia un bucle ja que tornava la última pantalla visitada. Per tant es feia un bucle entre la última i la penúltima, ja que a la varialbe donVinc quedava guardada únicament la última pantalla visitada.

Vaig intentar perfeccionar-ho, i vaig canviar String **donVinc** per Vector, que funcionés com una pila. La solució en quant a tornar a les pantalles visitades funcionava perfectament. Per finalitzar vaig decidir que era millor agrupar-ho tot en una classe **Dades.** Aquesta solució funciona, casi perfectament tot i que té un problema que comentaré en l'apartat millores.

També vaig tenir certs problemes per equivocacions alhora d'escollir el camp clau de les taules de la bdd. Per exemple seleccionar el dni com a camp clau (en aquest programari no és una bona opció perque segurament, molts familiars de generacions passades no tenen dni), tampoc escollira la ruta d'un arxiu com el seu camp clau (ja que en cas de que dos familiars escolleixin el mateix arxiu, en cas que un decidís borrar-lo, borraria també el de l'altre familiar); i altres casos semblants.

La solució en aquests casos va ser fàcil, tot i que moltes vegades va ser costosa en el temps, perque hi havia ja tota la classe que es relacionava amb la bdd. El problema, al final, va quedar resolt creant un nou camp, a pràcticament totes les taules que era ID i que s'inicialitzava de forma automàtica. A més aquesta solucó acabava permetent, que gairebé tots els camps fossin modificables.

7.3 Possibles millores

Començaré per explicar la millora abans comentada del vector **DonVinc.** La solució que vaig acabar implementant si que funciona, en quant a que si es va tornant es pot arribar a l'inici recorrent totes les pantalles que havia visitat l'usuari. El problema és que dins de la classe dades, es guarden: dos familiars (un per l'ús de qualsevol de les interfícies i un altre per donar d'alta), un event, un arxiu i un lloc. D'aquestes variables es modifica el seu valor cada cop que es canvia de classe.

Per tant quan es canvia de pantalla i es tornar, si no s'ha realitzat cap operació el que l'usuari observa en la nova pantalla, és exactament el que havia vist anteriorment. Pero en cas que modifica alguna de les variables, de la classe **dades**, el que veurà serà la pantalla correcta pero les dades no ho seran. Aquest problema no fa que l'aplicació funcioni incorrectament ni, tampoc serà una circumstància que passarà sempre (ja que moltes vegades el programari s'utilitzarà per visualitzar dades i com no es modificaran les dades, el resultat a tornar SI serà el correcta, sempre).

Crec que si que és possible la solució del problema anterior tot i que la implementació pot ser molt costosa en temps. Una possible solució seria que les variables de la classe **dades** no fossin objectes, sino vectors. S'hauria de tractar igual a aquests objectes, que al vector DonVinc, ja que es tracta exactament de la mateixa situació.

Una altra possible millora seria la de mostrar de forma explícita molts dels detalls que si que han estat implementats. Per exemple no queda expressat per pantalla que una parella sigui formada per dues persones del mateix sexe, encara que es possible; tampoc el tema de les adopcions, com tampoc la possibilitat del canvi de sexe d'una persona. Aquests aspectes, com altres, queden implementats, degut a la forma en que he programat el codi. Son fets possibles tots els anteriors, i en les millores a realitzar serien aspectes que haurien de quedar pal.lesos a les interfícies gràfiques, i no només en el codi.

Una altra de les millores seria la possibilitat d'afegir fotografies a llocs. La meva idea inicial era poder-ne afegir almenys una, tot i que seria una bona idea poder associar, com es fa amb els familiars, inifinitat de documents gràfics.

Evidentment una part molt millorable és l'apartat web. Seria interessant que aquesta part tingués la possibilitat d'efectuar les mateixes operacions que el programari, ja que serien dues eines compatibles i complementaries, tot i que amb estils diferents. Tot i que amb les funcions que realitza s'assoleixen els requisits inicials, seria interessant que qualsevol usuari pogués actualitzar la bdd de la web. El procés per verificar que la base de dades modificada és reemplaçable no seria gens trivial, ja que o bé no es pot actualitzar una base de dades per una altra sense verificar les dades que en aquesta hi ha, o bé s'hauria de guardar una còpia de seguretat dins el servidor de la web, per si hi hagués possibles errors.

7.4 Treball realitzat durant l'el.laboració del projecte

Ara que el projecte està pràcticament acabat al 100%, crec important recopil.lar tots els passos recorreguts per aconseguir-ho.

El primer pas I, potser més important va ser decidir la temàtica. Aquesta part hem va costar força ja que no acabava de trobar una temàtica interessant, que hem permetés implicar-me el necessari, tenint en compte la magnitud del treball. Per aconseguir decidir-me vaig necessitar aproximadament unes 7 setmanes.

Posteriorment vaig intentar cercar, en bibliografia diversa I per internet, tot el material possible sobre la gestió de famílies. La cerca per internet va ser força més senzilla que la bibliogràfica. Amb tot entre la recopil.lació de la informació, la transcripció a paper de la mateixa I la comparació entre altres programaris I el que jo volia desenvolupar, el temps total per aquest part va ser d'una setmana.

La següent etapa va ser decidir l'entorn de programació, així com també altres programaris necessaris per al desenvolupament del projecte. Aquesta etapa no va ser molt llarga en temps, pero cal destacar que s'han de comparar les diferents opcions possibles (tant de llenguatge, com de software), així com redactar-ho per poder-ho explicar a la documentació final, com es pot llegir en apartats anteriors. La duració aproximada va ser de 3 setmanes.

Un cop escollida la temàtica i els programes necessaris que s'utilitzarien, en la següent etapa havia d'analitzar amb deteniments quins eren els requisits funcionals del sistema (que era exactament el que volia aconseguir amb la realització del projecte). A més dins aquesta etapa, es pot incloure el desenvolupament dels diagrames anteriorment explicats, que ajuden a entendre millor els requisits finals de l'aplicació. La durada aproximada d'aquesta etapa va ser d'unes tres setmana.

El següent pas va ser la codificació del projecte. Aquesta, evidentment ha estat la part que més feina ha compartat, com també dificultats. En total he estat unes 21 setmanes. Mentre que el període de codificació de la web va ser de 4 setmanes aproximadament.

Finalment l'últim pas va ser realitzar la documentació necessària per poder presentar-ho. Aquesta etapa va ser molt més llarga del que m'esperava i va ser d'unes 15 setmanes. A mida que anava fent la documentació del projecte vaig poder apreciar certes imperfeccions de l'aplicació final, que van ser modificades. Per tant es podria dir que aquesta etapa va solapar-se amb la de codificació, ja que va permetre millorar certs aspectes incorrectes.

11. Conclusions

Tot i que hi ha moltes possibles millores a realizar crec que els objectius inicials han estat assolits. La interfície permet agregar informació de familiars, de llocs i associar-hi infinitat d'arxius i events (a més de la possibilitat de crear-ne de nous). També s'ha creat una web que permet realitzar part de les accions del programari i que permet a diferents usuaris consultar una mateixa família online. A més s'ha permés la possibilitat de guardar diferents famílies en el mateix sistema.

Crec, a més, que les interfícies son clares i han estat foça treballades, perque a més de tenir un cert aspecte, siguin lo més intuitives per l'usuari. La barra de navegació ràpida, permet a l'usuari aconseguir realitzar qualsevol de les possibles operacions, de forma ràpida sense tenir que passar moltes pantalles. I aquest era un dels objectius personals, després d'haver vist el funcionament d'altres aplicacions, ja que era un aspecte en el que fallaven. Volia aconseguir una navegació ràpida, còmoda, senzilla, intuitiva i per tant de ràpida adaptació per a qualsevol usuari; i crec, que s'ha aconseguit.

Comentar també que les diverses opcions presents i la multiplicitat de vistes disponibles ajuden molt a que l'usuari no només pugui anar afegint informació, sino que a més permet que la pugui buscar de formes molt diverses. A més també hi ha moltes vistes tant pels familiars, com per les fotos com pels events. Tot això fa que cercar el que l'usuari vol és molt senzill, pràctic i a vegades original (depenent de la cerca que s'esculli).

Finalment dir que crear aquest projecte de final de carrera, m'ha fet recollir en un mateix programari bona part dels coneixements adquirits en les diferents assignatures de la carrera.

11. Bibliografia

La veritat és que gaires llibres no utilitzat per realitzar l'aplicació. Gairebé tota la informació la he recollit d'internet. A continuació mostraré els llibres utilitzats, i només algunes de les adreces que més m'han ajudat a crear el programari...

Llibres:

Daum, Berthold. *Eclipse 3 para desarrolladores en Java.* 1a. ed. Lützelbach: Anaya Multimedia, 2005.

Harris, Robert i Warner, Rob. *The Definitive Guide To Swt And Jface.* 2a. ed.. Apress 2007.

Pàgines web:

The Eclipse Foundation (2008). *Eclipse - an open development platform.* Recuperat el 8 d'octubre del 2007, desde http://www.eclipse.org/

Ajpdsoft (2008). *Instalar y trabajar con la clase Visual Editor de Eclipse(Java).* Recuperat el 10 d'octubre del 2007, desde http://www.ajpdsoft.com/modules.php?

IBM (2005). Extending The Visual Editor:Enabling support for a custom widget. Recuperat el 10 d'octubre del 2007, desde http://www.eclipse.org/articles/Article-VE-Custom-Widget/customwidget.html

IBM (2008). *Build GUIs with the Eclipse Visual Editor project*. Recuperat el 12 d'octubre del 2007, desde http://www.ibm.com/developerworks/library/os-ecvisual/

Collabnet (2008). OpenOffice the open and productivity suite. Recuperat el 13 d'octubre del 2007, desde http://www.openoffice.org/

StarUml (2007). StarUml the open source UML/DMA Platform. Recuperat el 5 de novembre del 2007,desde http://www.staruml.com/

Sun Microsystems (2007). Java para Windows – Firefox/Mozilla. Recuperat el 12 de novembre del 2007, desde http://www.java.com/es/download/windows_xpi.jsp?locale=es&host=www.java.com

Eclipse (2007). SWT snippets. Recuperat el 14 de novembre del 2007, desde http://www.eclipse.org/swt/snippets/#table

My Heritage Ltd. (2007). Family tree builder – programa gratis de genealogía. Recuperat el 20 de novembre del 2007, desde http://www.myheritage.es/family-tree-builder

Deudos.com (2007). Explora su árbol familiar. Recuperat el 20 de novembre del 2007, desde http://www.deudos.com/

Millenia (2007). Legacy Family tree genealogy software. Recuperat el 20 de novembre del 2007 desde http://www.legacyfamilytree.com/tipsNGSJulAug99.asp

Intellectual reserve, Inc. (2007). Family search, where generations meet. Recuperat el 20 de novembre del 2007 desde http://www.familysearch.org/eng/paf/

Google (2008). Google. Recuperat moltes vegades desde 8 d'octubre del 2007, desde http://www.google.es/

Wikipedia (2008). Wikipedia, la enciclopedia libre. Recuperat el 7 d'abril del 2008, desde http://es.wikipedia.org/wiki/Programaci%C3%B3n_Extrema

9 Annex

9.1Manual d'usuari

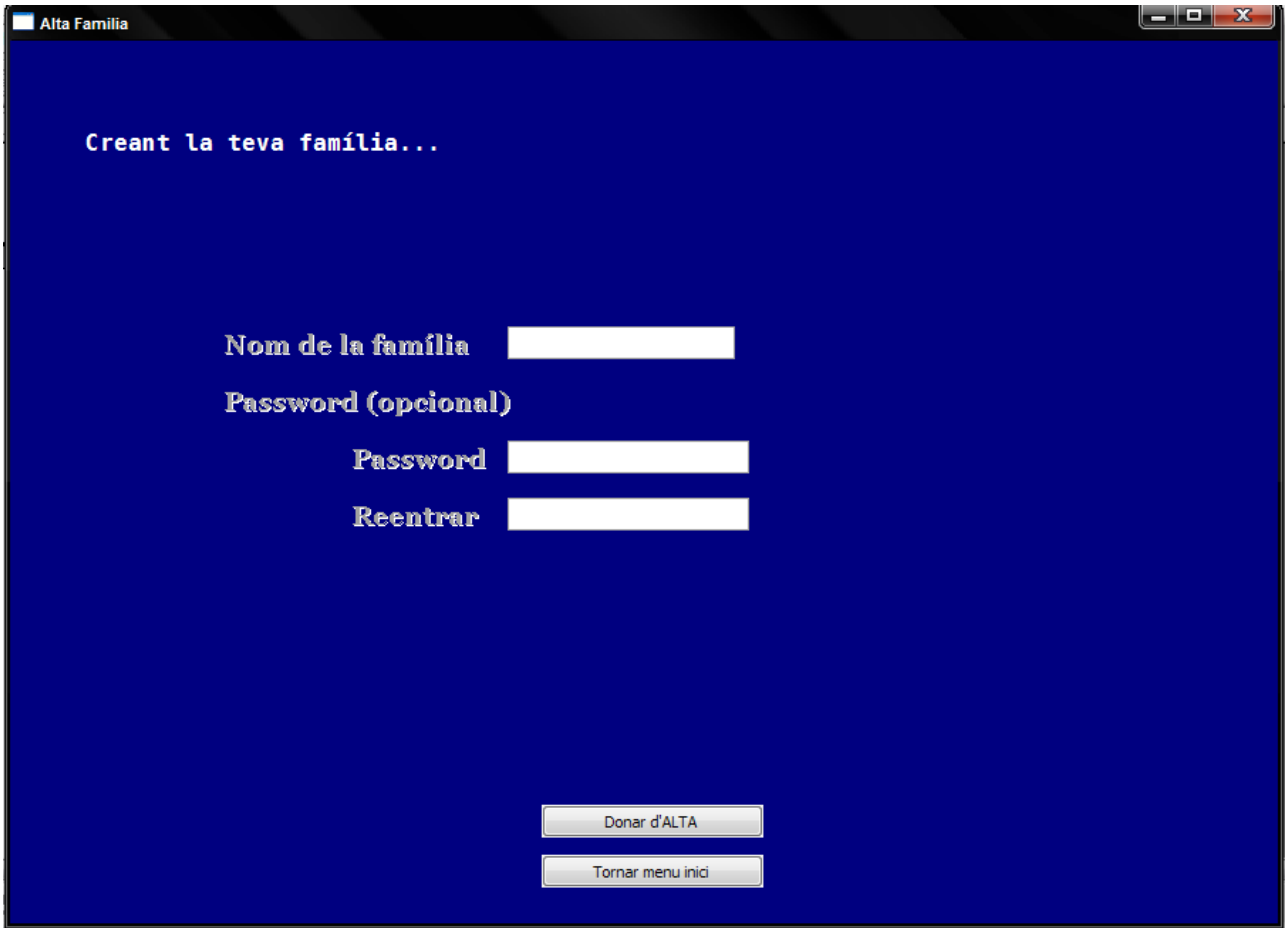
9.1.1 Requisits del sistema

El programa funciona sobre qualsevol sistema operatiu, pero té alguns requisits obligatoris. Els components de la computadora no són importants, ja que funcionarà amb qualsevol ordinador més o menys actual. El que si és necessari...

- 100Mb d'espai lliure a memòria. (aquest és el mínim ja que si es volen afegir arxius al programari, és necessari que l'espai sigui molt més gran).
- Java 6. En cas que no estigui instal.lat, en el cd hi ha l'aplicació. A més posteriorment els programes incluits en el cd, també s'explicarà com instal.lar-los.
- MySql (inclòs al cd)
- Qualsevol processador de textos (inclòs OpenOffice al cd)
- En cas de voler publicar per internet, caldrà conexió a internet a més d'algun navegador.

9.1.2 Manual d'usuari

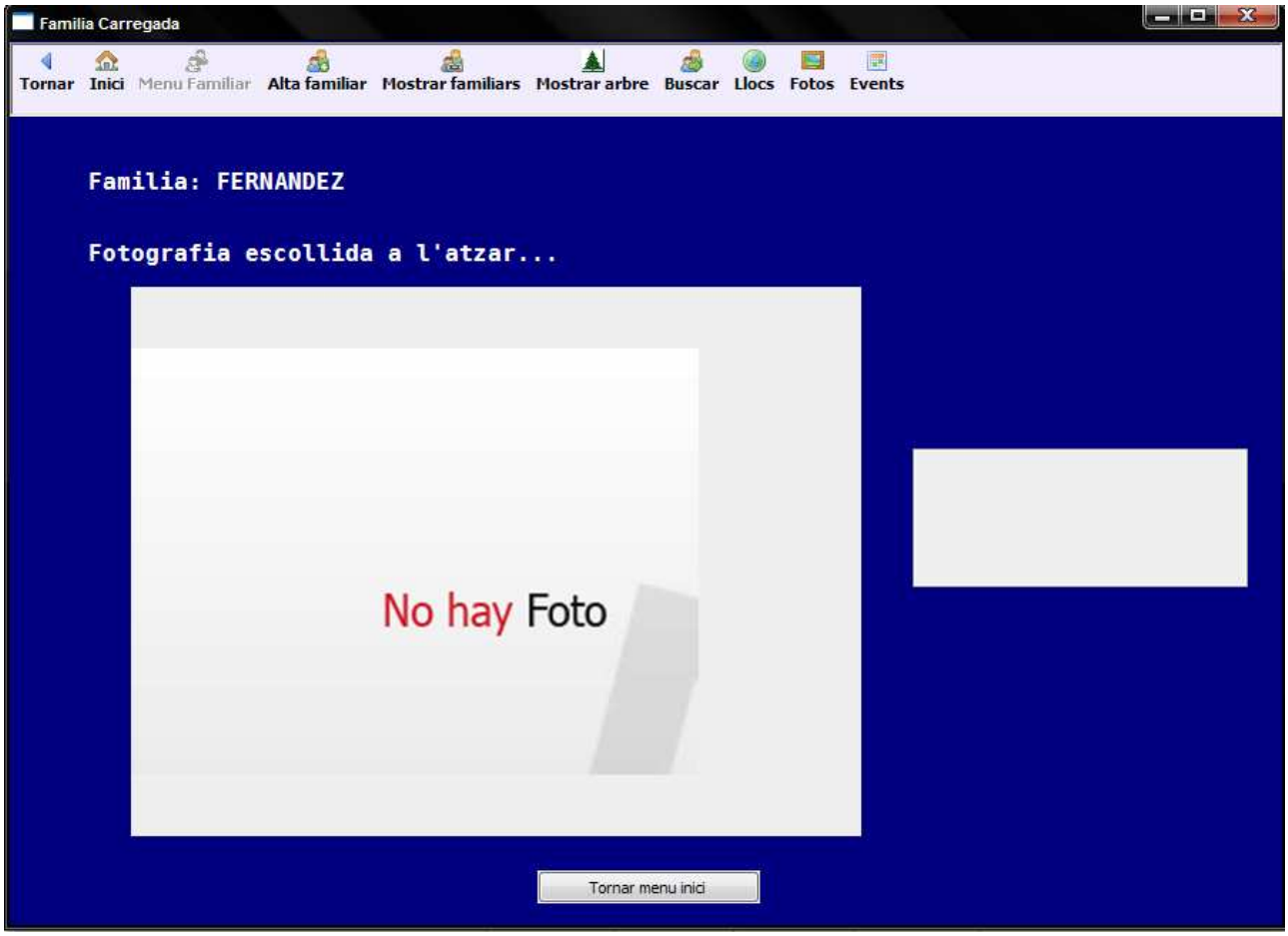
La primera pantalla que apareix a l'aplicació permet carregar o crear una família. En cas que sigui la primera vegada que s'accedeix al programari cal crear-ne una de nova. Per crear-la cal seleccionar la opció **crear família**. La pantalla que apareixerà si es tria aquesta opció és la següent:



En el primer camp s'ha d'introduir el nom que li vulgui donar a la família. El més habitual és el primer cognom. El nom que se li assigni anirà apareixent a la part superior en totes les pantalles, de tots els familiars. Els dos últims camps a emplenar són per afegir una clau, en cas que es desitgi. En cas que les dades introduïdes no siguin correctes (el password no és el mateix en els dos camps) es mostrarà un missatge d'error per pantalla; per contra es rediregirà a l'usuari a la pantalla inicial de la família.

En posteriors execucions del programa, no s'haurà de crear una nova família, sino que s'haurà de seleccionar la família de la llista que apareix, al clicar sobre **carregar família.**

Després de carregar la família, si encara no s'ha afegit cap fotografía apareixerà la següent pantalla:

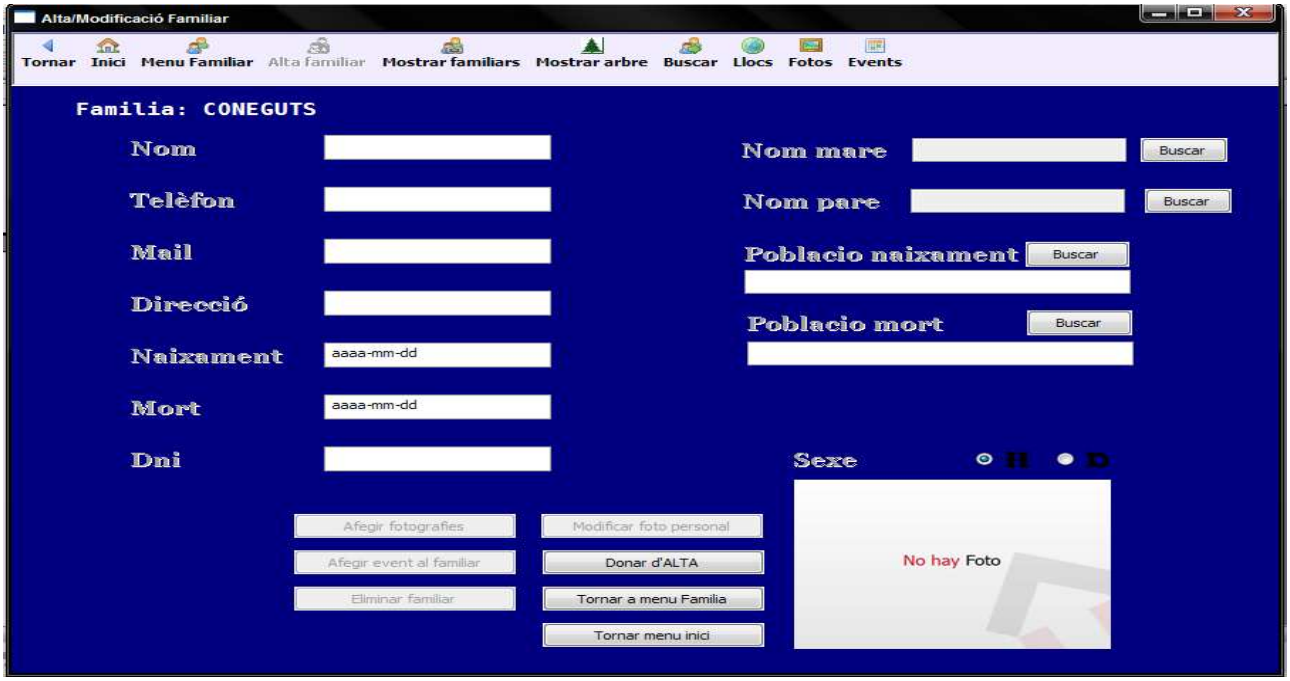


Com es pot observar a la foto a la part de dalt apareix una barra. Aquesta serveix per desplaçar-se per totes les opcions del programari. Les opcions més importants tenen un accés directe, mentre que les altres tenen una pantalla pont per escollir entre les múltiples opcions.

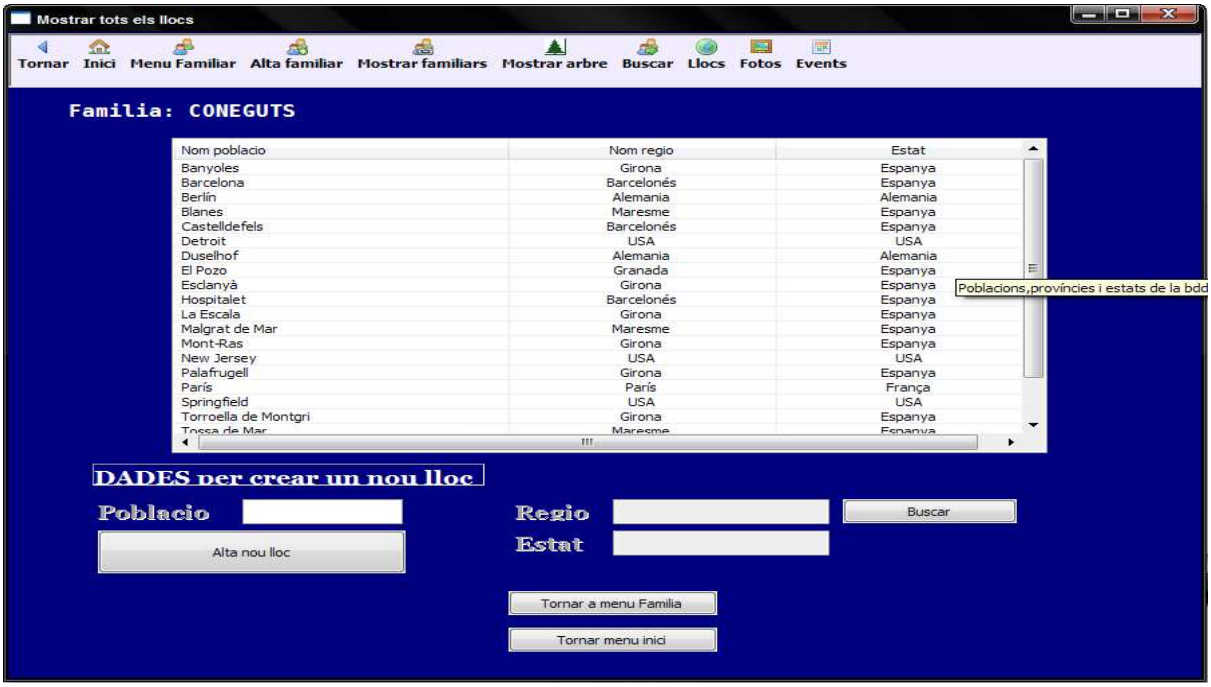
Després apareix el nom de la família i una fotografía de la bdd escollida a l'atzar. Si encara no s'ha introduït cap fotografia, apareixerà aquesta imatge, tal que no hi ha cap foto a la bdd. Mentre que si la família ja n'ha guardat algunes apareixerà una escollida a l'atzar, i la seva descripció a la dreta.

A la part inferior apareix el botó tornar al menú inicial, s'utilitza en cas d'haver carregat una família no desitjada. Aquesta operació també es pot fer clicant sobre la opció inici de la barra de navegació.

Ara explicaré les possibles opcions de la barra de navegació. La primera que apareix és tornar i permet tornar a visitar la pantalla anterior, amb les mateixes dades que hi havia carregades la vegada anterior. La segona és per tornar al menú inicial, la que permet carregar o crear una nova família. La tercera icona permet donar d'alta un familiar i el seu aspecte és el següent...

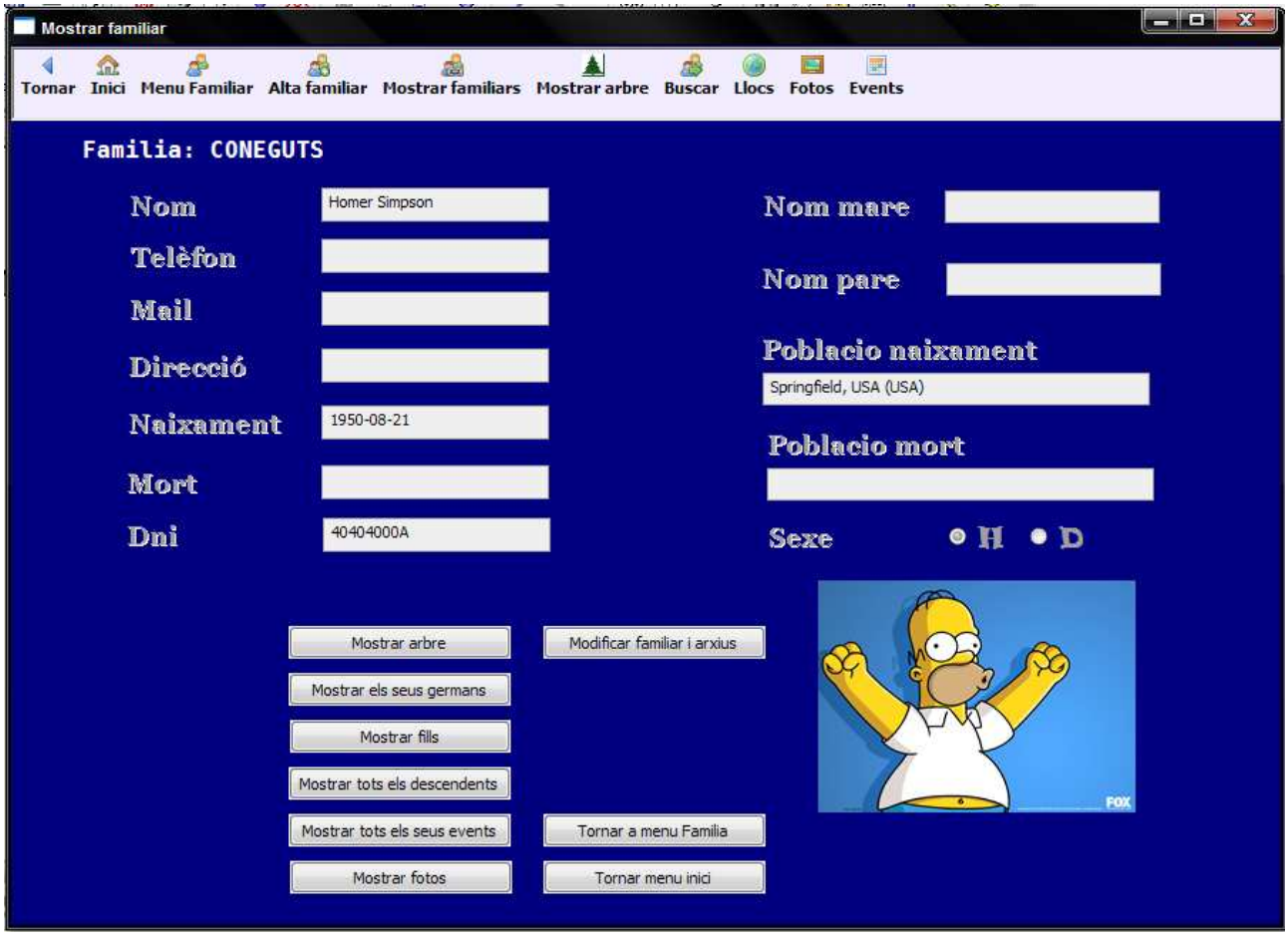


No tots els camps són obligatoris, encara que a més informació millor, tant per les cerques que es podran realitzar, com pels altres familiars que vulguin veure les dades personals. Destacar que per introduir la mare, el pare, o la població; no es pot fer de forma directa, com tots els altres camps, sino que s'ha de seleccionar d'una llista, com es pot veure a continuació:



Un cop introduïdes les dades i es clica sobre donar d'alta es donarà d'alta el familiar a la bdd (sempre que els valors introduïts siguin correctes, i camp camp obligatori estigui en blanc).

La quarta opció del menú permet veure tots els membres de la família. Al seleccionar-ne algun mostra de forma detallada, com en el següent exemple:

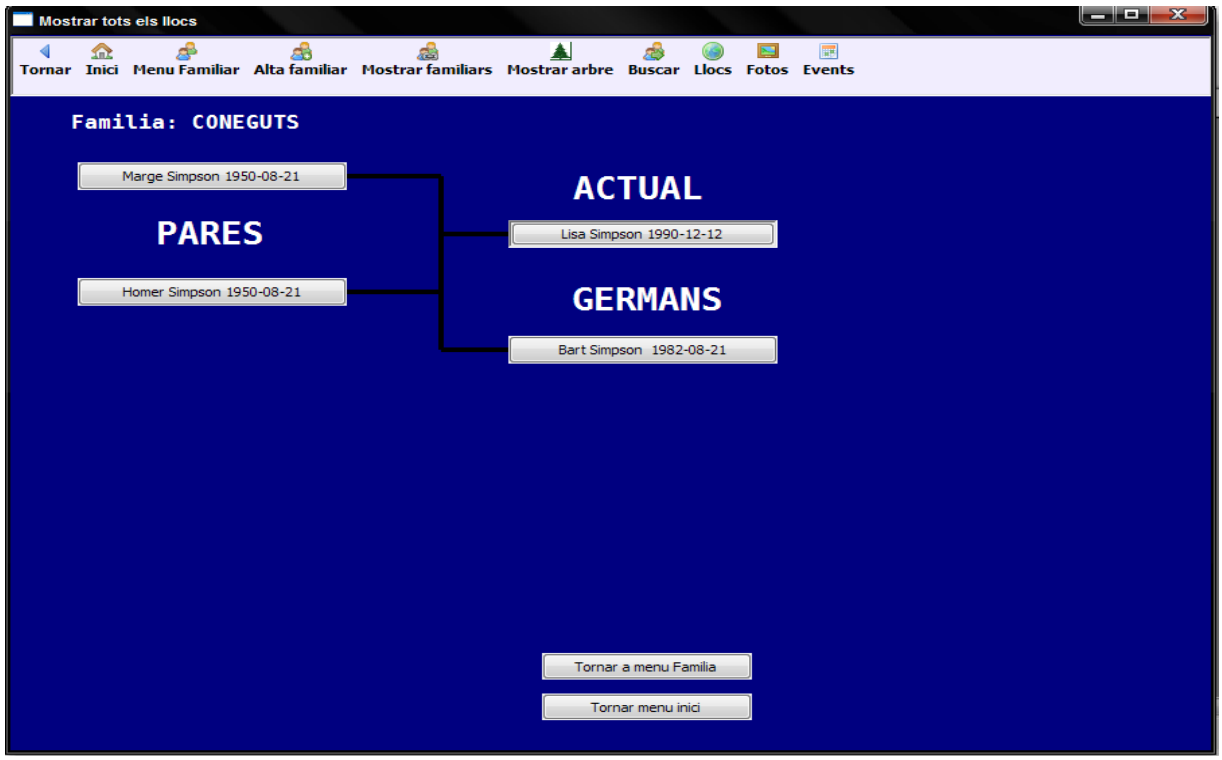


Com es pot observar en l'exemple, es mostren les dades personals introduides a més de la seva foto personal. També hi ha un seguit d'operacions extres que permeten mostrar tot tipu d'informació extra tal com (els fills, tots els seus descendents, germans, el seu arbre, els events que el relacionen així com totes les seves fotos). En els casos on apareixen familiars, el resutlat en cas de seleccionar alguna de les opcions, és una taula (semblant a la vista amb anterioritat) que mostra certa informació del familiar i que en cas de triar-ne un, ens dona la informació completa (com en la imatge de dalt).

En aquesta mateixa interfície també hi ha la opció de modificar el familiar. En aquest cas hi ha la possibilitat de modificar les dades, a més d'afegir-li fotos, events o fins i tot modificar la fotografia personal (que és la que es mostra, quan es fa referència al familiar). I també hi ha la possibilitat d'eliminar-lo, pero compte si això es fa es borraran el familiar, tots els arxius i events relacionats amb ell i totes les referències que tiguin els familiars. És a dir, si s'esborra un familiar que té un fill, aquesta refrerència quedarà borrada.

Cal recalcar els passos que s'han de seguir per poder afegir un arxiu a la bdd. El primer que s'ha de fer és copiar la fotografia que es vol afegir dins la carpeta, que hi ha ubicada a l'escritori anomenada **imatges**. En cas que es vulgui crear una carpeta per la família s'haurà de crear de forma manual. Un cop fet aquest pas, alhora d'afegir les imatges s'haurà d'introduir la ruta desde la carpeta imatges. En el cas que es mostra a les fotos, si s'ha creat una carpeta anomenada **Coneguts**, quan es fa referència a una foto.... ***/coneguts/NomFoto.**** .... aquest seria un exemple. L'extensió de l'arxiu pot ser qualsevol.

Una altra de les opcions existents a la barra de navegació és mostrar l'arbre genealògic. Primer, pero, s'ha de seleccionar el familiar d'una llista i llavors es mostra el seu arbre. Com a exemple:

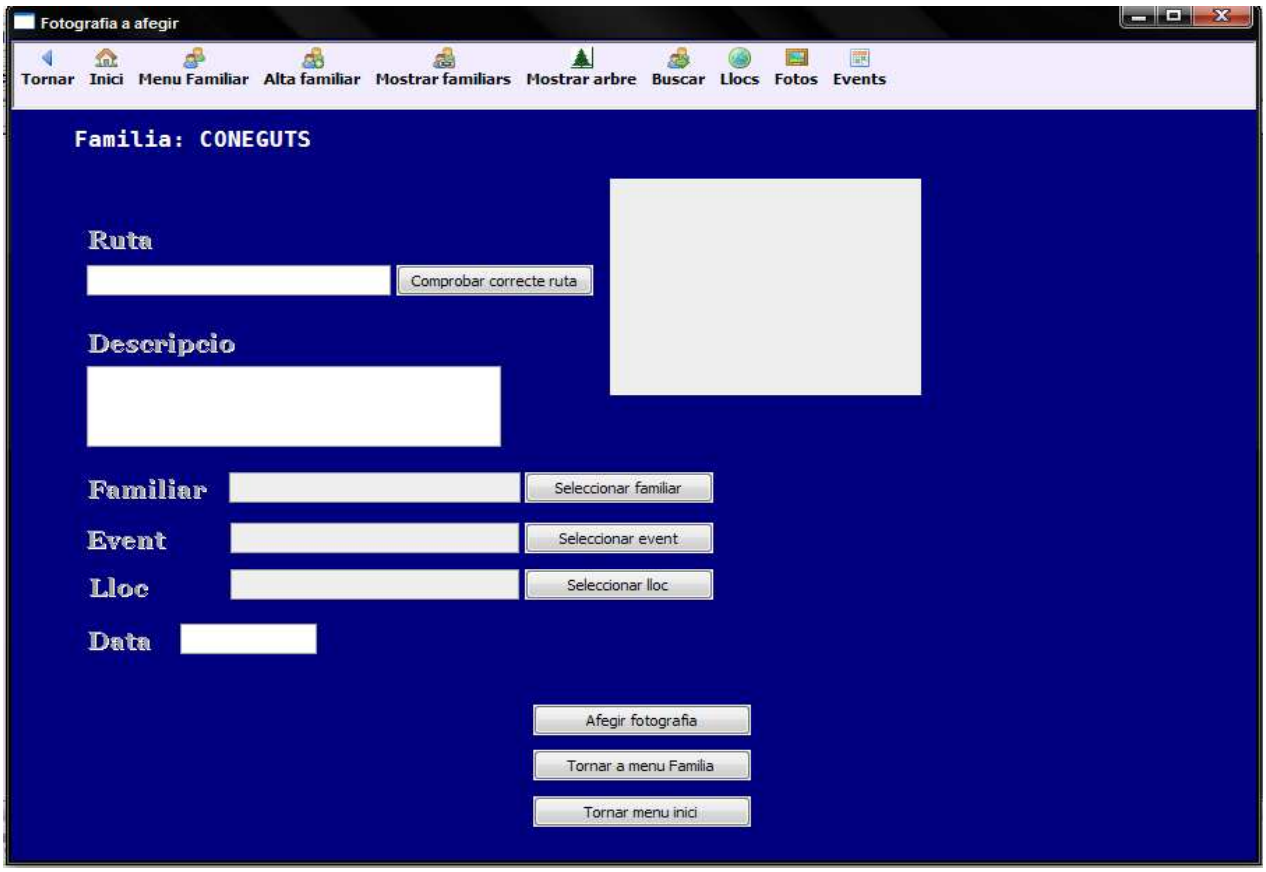


A la part alta centrat, hi ha el familiar actual. En cas de seleccionar-lo, es mostraria de forma detallada. A la seva esquerra hi ha els seus pares, abaix els germans i a la dreta els seus fills. En cas que siguin 8 o més germans o tingui 8 o més fills, no es mostraran com a la imatge, sino que apareixerà una icona per veure'ls en forma de llista. En cas que es seleccioni a qualsevol dels familiars, excepte l'actual, es tornarà a carregar l'arbre pero amb l'actual igual al familiar escollit.

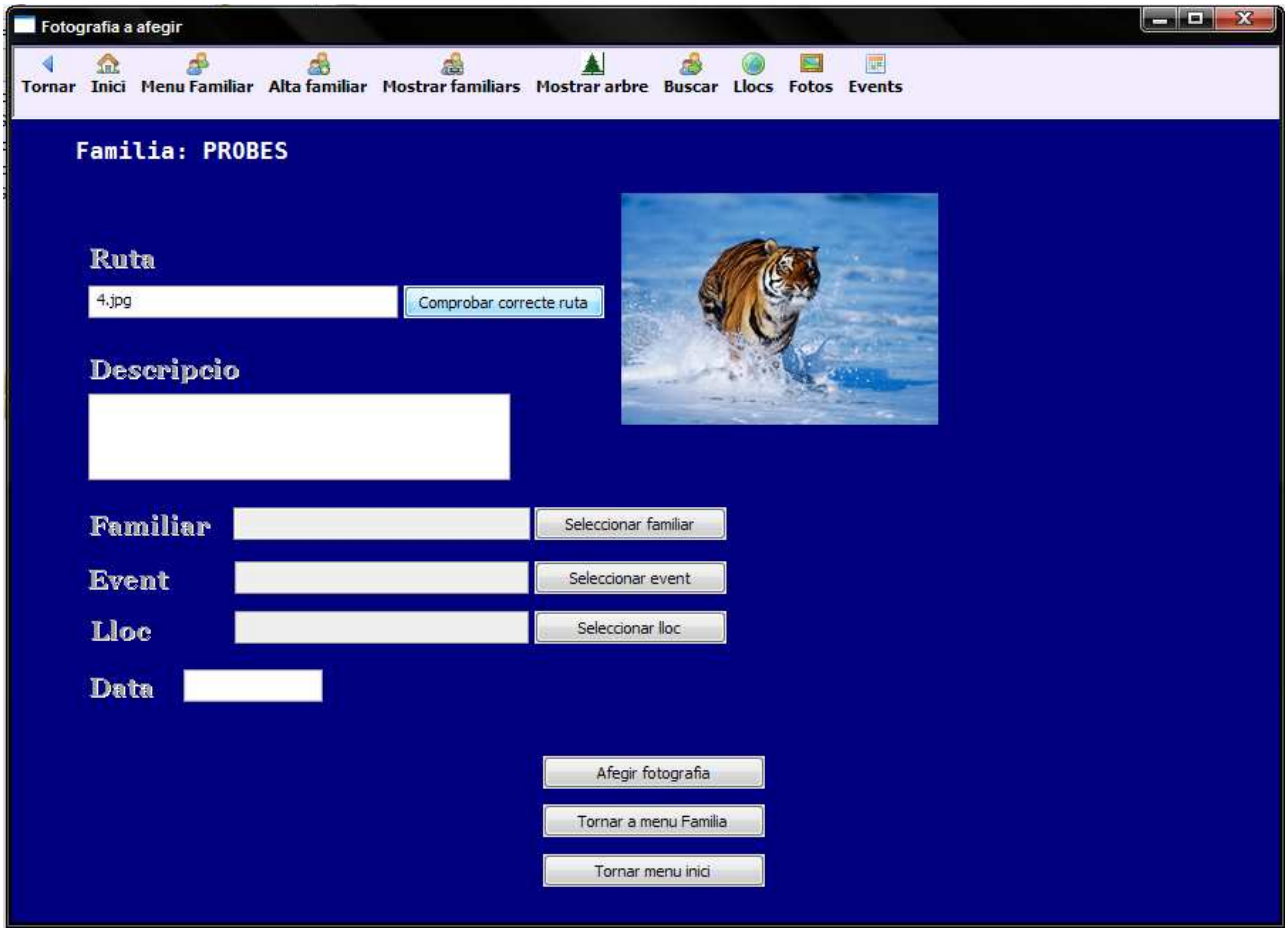
Una altra de les opcions és **buscar**. Aquí hi ha predefinides un conjunt de vistes perque d'aquesta forma es pugui buscar algun familiar en concret, per veure'l detalladament, o bé per veure un conjunt de familiars o arxius dins un paràmetres establerts.

El següent menú **lloc** permet la creació, modificació i visualització dels llocs.

Un altre opció que hi ha és **fotos**. Aquí hi ha la possibilitat tant d'afegir fotograies o bé mostrar-les totes o segons uns criteris preestablerts. En cas que es seleccioni afegir una foto hi ha certes particularitats a explicar. Primer caldrà copiar la foto (com s'ha explicat anteriorment) a la carpeta imatges. Un cop fet s'haurà d'accedir a la següent pantalla



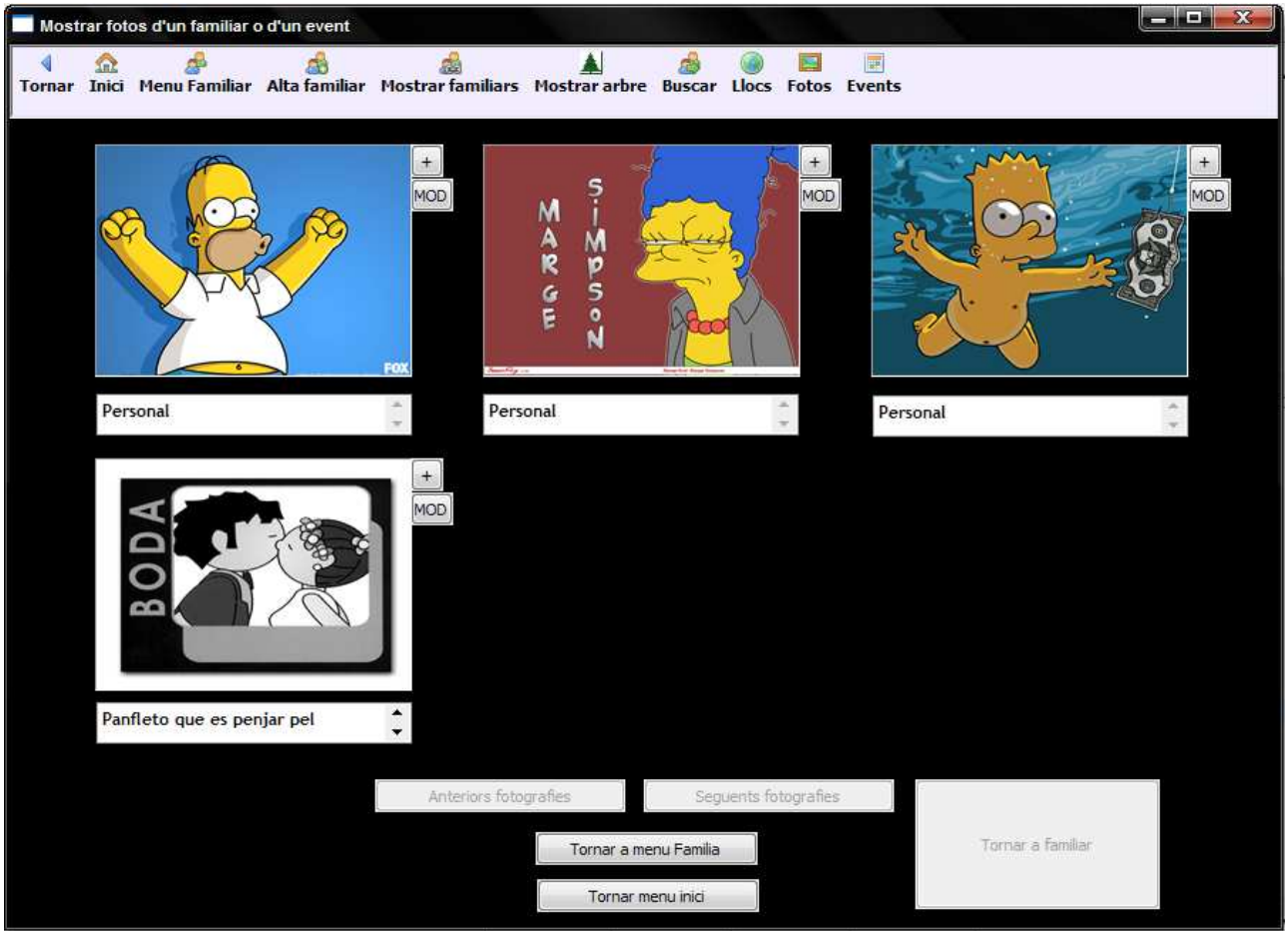
El primer que s'ha d'emplenar és la ruta. Si es vol comprobar si la ruta és correcta s'haura de clicar el botó de la dreta, com en l'exemple:



Com es pot veure, la imatge era correcta, ja que s'ha carregat. En cas que no s'hagi carregat la foto desitjada o ens aparegui una foto amb un missatge que no hi ha foto, significa que la ruta introduida no és la correcta.

Llavors s'ha d'emplenar la descripció i el familiar, els altres camps son optatius. En cas que es seleccioni l'event de la llista, probablement (si havien estat inicialitzats) totes les altres dades (familiar, lloc i data) quedaran emplenades de forma automàtica, ja que s'utilitzaran les dades ja guardades dins un event.

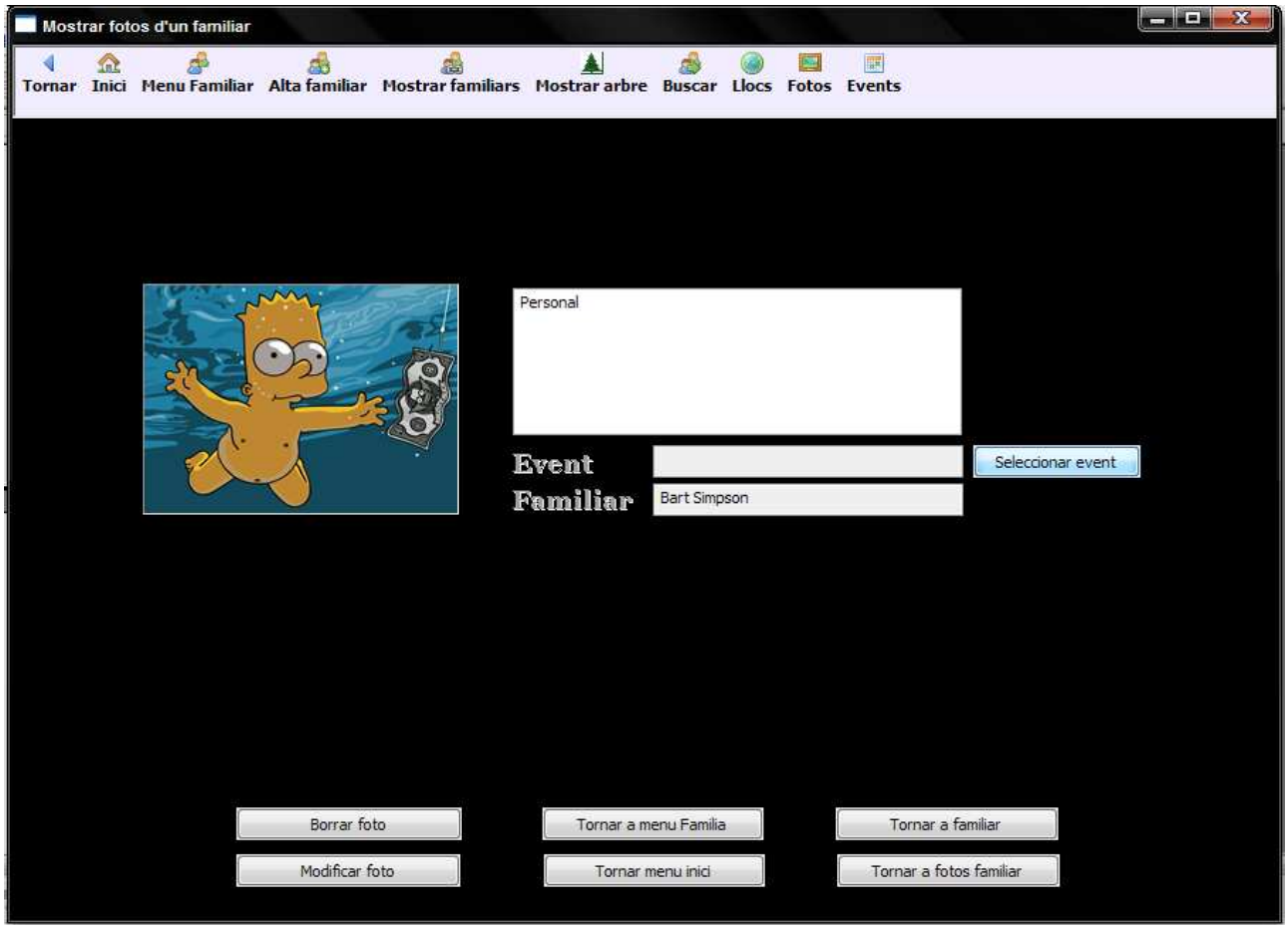
Si es selecciona qualsevol de les opcions de mostrar fotografies, la pantalla que s'obtindrà és la següent amb les fotografies pertinents, segons la cerca seleccionada:



En l'exemple es mostren totes les fotos d'una família, tres d'elles personals i una d'elles relacionada amb un event. Com es pot veure a més de la foto hi ha la descripció de la mateixa, la possibilitat de veure-la a pantalla completa (botó + que hi ha a la dreta a la part superior de la foto) i també es pot modificar.

Com a màxim poden aparèixer 6 fotos per pantalla, pero en cas que n'hi hagi més, el botó següents fotografies, permet anar a les pàgines següents. Si es decideix modificar una fotografia, es podrà canviar la seva descripció, canviar l'event al que fa referència (sempre que es tracti d'un event relacionat amb el familiar al que ja pertany la foto) o fins i tot eliminar-la. Cal destacar que una mateixa fotografia pot ser utilitzada per diferents familiars, tot i que poden tenir diferents descripcions, per la qual cosa no implica que si un la borra, l'altre la perd.

La pantalla modificar fotografía té el següent aspecte:



Per acabar amb els menús hi ha el menú dels events. Aquest permet afegir informació extra sobre els membres de la família. Informació com per exemple canvi de residència, casament divorci.... Si hi ha algun tipus d'event (casament,...) que no aparegui a la bdd, sempre se'n poden crear de nous a gust de l'usuari, que a més podran ser utilitzats, tant per altres familiars de la mateixa família, com també d'altres famílies (igual passa amb els llocs nous que es creen o modifiquen).

I per últim comentar que es possible mostrar tots els events, o segons familiar o veure les fotos d'alguns... En conclusió a més de la opció d'afegir events, llocs, familiars i arxius, de modificar-los, de borrar-los, de relacionar els familiars de moltes formes, inclús de formes inventades per l'usuari; hi ha la possibilitat de visualitzar les dades de moltes formes diferents (per any, per lloc,....) i això permet una gran interactivitat amb l'aplicació.