Anàlisi orientada a objectes

Benet Campderrich Falgueras
Recerca Informàtica, SL

P00/05007/00303

UOC
www.uoc.edu

Índex

Introducció 5

Objectius 6

1. El paper de l'anàlisi 7
1.1. La relació entre la recollida de requisits i l'anàlisi 7
1.2. La relació entre l'anàlisi i el disseny 7
1.3. La utilitat de l'anàlisi 8

2. Paquets d'anàlisi i paquets de serveis 9
2.1. Els paquets d'anàlisi 9
2.2. Els paquets de serveis 10

3. Revisió dels casos d'ús 12

4. Especificació de les classes d'anàlisi 13
4.1. Identificació de les classes d'entitats 14
4.2. Especificació dels atributs de les classes d'entitats 15
4.2.1. Casos especials en els atributs 15
4.3. Identificació de les relacions entre classes 16
4.3.1. Jerarquies d'herència 16
4.3.2. Herència múltiple 18
4.3.3. Interfícies 18
4.3.4. Associacions 19
4.3.5. Agregacions 20
4.4. Identificació de les classes de frontera, les classes de control i de les operacions. Diagrama estàtic d'anàlisi 21

5. Especificació formal dels casos d'ús 22

6. Anàlisi de la interfície d'usuari 23

7. Exemple 24
7.1. Revisió dels casos d'ús 24
7.2. Paquets d'anàlisi i de serveis 24
7.3. Identificació de les classes d'entitats 25
7.4. Especificació dels atributs de les classes d'entitats 26
7.5. Relacions 26
7.5.1. Relacions d'herència 26
7.5.2. Associacions 28
7.5.3. Agregacions 28

7.6. Identificació de les classes de frontera, les classes de control i de les operacions 29
7.7. Especificació formal dels casos d'ús 31

Resum 37

Activitats 39

Exercicis d'autoavaluació 39

Solucionari 40

Glossari 40

Bibliografia 40

Introducció

Aquest mòdul tracta de l'anàlisi del programari amb tècniques orientades a objecte, aplicant les notacions i conceptes d'UML i seguint el cicle de vida del Rational Unified Process. Dins d'aquest cicle de vida, l'anàlisi i el disseny constitueixen un sol component del procés, però nosaltres les considerarem dues etapes diferents, per raons que s'aniran explicant al llarg del mòdul.

En els apartats següents es veurà detalladament el paper de l'**anàlisi** en relació amb l'etapa que el precedeix, la recollida i documentació de requisits, i la que la segueix, el disseny (sempre amb el benentès que considerem que el desenvolupament de programari no és un procés lineal, sinó que el cicle de vida és iteratiu i incremental, i que, per tant, les seves etapes es repeteixen per a diferents parts del programari fins a completar-lo); aquí diem, però, que l'anàlisi és la primera etapa del desenvolupament del programari pròpiament dit, i que consisteix a traduir els requisits a una forma més adequada per a ser la base de partida d'aquest desenvolupament.

Dins l'anàlisi distingirem els passos següents:

- La revisió dels casos d'ús.
- La identificació de les classes d'anàlisi.
- La identificació de les relacions entre classes.
- L'especificació formal dels casos d'ús.
- L'anàlisi de la interfície d'usuari.

Objectius

L'objectiu d'aquest mòdul és que l'alumne aprengui a fer l'anàlisi de programari orientat a objectes fent servir els conceptes i notacions d'UML. Aquest objectiu es compon dels següents:

1. Entendre el paper de l'anàlisi i les diferències amb l'etapa anterior –la recollida i documentació de requisits– i la següent –el disseny.

2. Comprendre les diferències entre els tres tipus de classes que es consideren en l'anàlisi.

3. Aprendre a elaborar el diagrama estàtic d'anàlisi.

4. Saber formalitzar els casos d'ús mitjançant diagrames d'interacció i altres.

5. Aprendre a fer l'anàlisi de la interfície d'usuari.

1. El paper de l'anàlisi

1.1. La relació entre la recollida de requisits i l'anàlisi

L'etapa de recollida i documentació de requisits estableix els requisits del futur sistema de programari amb prou detall perquè es pugui començar a desenvolupar. Se suposa que hi ha un acord entre els usuaris (o llurs suposats representants) i els desenvolupadors del programari sobre aquests requisits, cosa que vol dir que els requisits estan expressats d'una manera poc formalitzada, ja que altrament només serien intel·ligibles per a professionals del desenvolupament de programari. Però uns requisits en aquesta forma no són una bona base per a construir un programari de manera sistemàtica.

Per tant, una primera comesa de l'anàlisi és la de traduir els requisits a un llenguatge més formal, que en el mètode que seguim són els models i diagrames d'UML; així s'obté el que s'anomena **model de l'anàlisi**.

Quins han estat els productes de l'etapa de recollida i documentació de requisits? Bàsicament, la descripció de les funcions del programari en forma de casos d'ús, d'una banda, i de tasques dels usuaris, de l'altra; el model del domini o el model del negoci i el glossari són només documents previs o, a tot estirar complementaris, d'aquells dos. Ara bé un desenvolupament orientat a objectes ha d'utilitzar un formalisme de classes i objectes que en els casos d'ús i les tasques no apareix encara.

Per tant, una segona comesa de l'etapa d'anàlisi serà la identificació d'unes classes fonamentals que seran la base de la implementació del programari. Una tercera comesa serà l'expressió dels casos d'ús en termes d'aquestes classes.

1.2. La relació entre l'anàlisi i el disseny

Tant a l'anàlisi com al disseny es descriu el futur programari de manera orientada a objectes; la diferència fonamental està en el fet que a l'anàlisi es descriu el funcionament del futur programari, sense entrar en la problemàtica de la implementació.

De fet, l'anàlisi es pot fer sense tenir en compte en quin llenguatge de programació s'implementarà, i fins i tot és molt convenient de fer-ho així (llevat,

potser, dels noms de classes, atributs i operacions). Tanmateix, hi ha un alt grau de continuïtat entre l'anàlisi i el disseny, ja que la majoria de definicions de classes de l'anàlisi es fan servir després en el disseny; en els mètodes de desenvolupament no orientats a objectes, en canvi, es fan servir models diferents a l'anàlisi i al disseny.

1.3. La utilitat de l'anàlisi

Jacobson, Booch i Rumbaugh esmenten quatre raons que poden justificar l'elaboració d'una documentació d'anàlisi separada de la de disseny:

Lectura complementària

Trobareu una ampliació de la justificació de l'elaboració d'una documentació d'analisi separada de la de disseny a l'obra següent:

I. Jacobson; G. Booch; J. Rumbaugh (2000). *El proceso unificado de desarrollo de software.* Addison-Wesley.

1) Es pot analitzar tot el programari amb un cost relativament baix i després implementar-lo per parts d'una manera coherent, ja que l'anàlisi n'haurà donat una perspectiva global.

2) La documentació d'anàlisi dóna una visió general del programari, però sense arribar als detalls de la implementació que va molt bé per a adquirir ràpidament un coneixement del programari.

3) De sistemes de programari en què la seguretat és crítica, se'n poden fer diverses implementacions partint d'una única anàlisi; el mateix es pot fer si es desitja encarregar-ne diverses implementacions per a quedar-se amb la millor.

4) Per a fer de nou un sistema antic amb tecnologia actual, la descripció d'aquest feta per mitjà d'anàlisi pot ser una base de partida molt més convenient que la implementació existent, en la qual hi ha molts detalls que no cal tenir en compte.

L'ús que es fa de l'anàlisi a les diferents fases del cicle de vida del Rational Unified Process pot ser un d'aquests tres:

- El model de l'anàlisi es va actualitzant i es fa servir en tot el cicle de vida.

- El model de l'anàlisi es considera un resultat transitori i quan s'entra de ple en el disseny i la programació s'abandona, els problemes d'anàlisi que apareguin es resolen a nivell de disseny i programació .

- No hi ha model d'anàlisi, o bé s'integra amb la documentació dels requisits –i llavors cal que aquesta sigui molt més formal, cosa que fa difícil que la puguin entendre la majoria d'usuaris– o bé es passa directament dels requisits al disseny; això darrer vol dir posar-se a resoldre un problema que no ha estat especificat formalment i, per tant, de manera prou clara i coherent, cosa que només podria ser aconsellable en casos extremadament senzills.

2. Paquets d'anàlisi i paquets de serveis

Per poc gran que sigui un projecte de programari caldrà dividir la documentació en paquets d'UML, cadascun dels quals contindrà classes, casos d'ús o altres elements d'UML. En una divisió ben feta, els **paquets** compliran dues condicions:

1) Seran **coherents**, és a dir, els elements del mateix paquet estaran fortament relacionats.

2) Seran **poc dependents** els uns dels altres, és a dir, hi haurà poques connexions entre elements de paquets diferents.

Es distingeixen dos nivells de paquets: **paquets d'anàlisi** i **paquets de serveis**; uns i altres es basen en la funcionalitat (és a dir, en cap cas en els requisits no funcionals). Cada paquet de servei està inclòs dins un sol paquet d'anàlisi.

Un cop descompost el programari en paquets, o un paquet d'anàlisi en paquets de servei, convé identificar les dependències entre paquets; si es considera que n'hi ha massa, es canvia la descomposició.

2.1. Els paquets d'anàlisi

Els paquets d'anàlisi constitueixen una divisió del sistema de programari que té sentit des del punt de vista dels experts en el domini; cada paquet d'anàlisi correspon a un o diversos subsistemes sencers, i pot servir per a repartir la feina de l'anàlisi entre diverses persones o equips.

La descomposició del programari en paquets s'estableix quan hom s'ha fet una idea que es considera prou fiable de la quantitat de feina i del nombre i complexitat dels diagrames, situació a la qual es pot haver arribat tant al començament de l'etapa d'anàlisi com un temps després. Es poden assignar paquets separats als grans processos del negoci o bé als actors primaris, i en tot cas, els casos d'ús entre els quals hi ha relacions d'extensió, inclusió o generalització s'han d'assignar al mateix paquet.

Vegeu el concepte de *classe d'entitat* en l'apartat 4 d'aquest mòdul didàctic.

Pot succeir que una classe (generalment una classe d'entitat) s'utilitzi en més d'un paquet; si hi ha diverses classes compartides pels mateixos paquets es pot fer un

paquet a part amb aquestes, mentre que si només hi ha una classe, se la pot deixar fora dels paquets; també es pot desenvolupar la classe dins un paquet –aquell en què es creïn els objectes, per exemple– mentre que els altres la farien servir. En tot cas, entre els paquets que utilitzen la mateixa classe o classes i la classe o paquet esmentat s'estableixen les relacions de dependència corresponents.

2.2. Els paquets de serveis

Els paquets de serveis són subdivisions dels paquets d'anàlisi des d'un punt de vista que podríem anomenar *comercial*, és a dir, són opcions separades des del punt de vista de les organitzacions clients*.

*** Podríem dir que els paquets de serveis són opcions de catàleg comercial.**

Ja sabem que els casos d'ús són autònoms, en el sentit que cada un l'engega de manera independent l'actor primari respectiu; tanmateix, hi ha casos d'ús relacionats en el sentit que no és possible engegar-ne un si abans no se n'ha dut a terme un altre (per exemple, no es pot emetre una factura per a un client si abans no s'ha creat la comanda corresponent); en aquests casos és lògic que tots dos casos formin part del mateix paquet de serveis. En canvi, un cas d'ús relatiu a l'obtenció d'estadístiques sobre el temps transcorregut entre la creació d'una comanda i l'emissió de la factura corresponent podria formar part d'un paquet de serveis opcional d'estadístiques, que unes empreses clients del programari comprarien i d'altres, no (i això encara que l'actor dels tres casos d'ús fos el mateix).

Els paquets de serveis presenten les característiques següents:

- Comprenen casos d'ús relacionats.
- Generalment els seus casos d'ús tenen a veure només amb un actor o, en tot cas, amb pocs.
- Són indivisibles, en el sentit que un client o té un paquet de serveis complet, o no el té.
- Poden ser incompatibles o, ben al contrari, un pot necessitar-ne un altre o diversos.
- En un cas d'ús poden intervenir diversos paquets de serveis.
- Hi ha molt poques relacions de dependència entre elements de paquets de serveis diferents i, en conseqüència, es poden analitzar, dissenyar i més endavant mantenir de manera totalment independent o gairebé.

- Es poden reutilitzar, si més no entre diferents configuracions del sistema de programari.

La descomposició dels paquets d'anàlisi en paquets de serveis es fa assignant un paquet de serveis a cada conjunt de casos d'ús opcional.

3. Revisió dels casos d'ús

La base de partida per a l'anàlisi és la documentació sobre els casos d'ús elaborada en l'etapa anterior. Per tant, convé que aquesta sigui precisa i detallada. Si la documentació s'havia fet amb l'objectiu principal de ser la base d'un acord entre usuaris i desenvolupadors, pot ser que no compleixi aquestes condicions i caldrà revisar-la.

Hi podria haver mancances com ara que no es mostrin les relacions entre casos d'ús o que hi hagi algunes ambigüitats en la terminologia utilitzada en les descripcions textuals. Si no s'han estudiat prou les relacions entre casos d'ús, podria ser que hi hagi dos casos que fan funcions semblants, potser de manera contradictòria. És possible que la descripció de la funció d'un cas d'ús feta en llenguatge ordinari sigui ambigua quant a les combinacions de condicions; llavors pot ser convenient d'escriure-la de nou en forma de pseudocodi.

4. Especificació de les classes d'anàlisi

Es consideren els tres **tipus de classes d'anàlisi** que esmentem a continuació:

1) Les **classes de frontera*** representen en el nivell d'anàlisi la interfície d'usuari per pantalla. N'hi ha d'haver almenys una per cada paper de cada actor; per tant, cada una representa la interfície d'usuari entre un cas d'ús i un actor. Les classes de frontera representen objectes gràfics complexos com finestres, diàlegs per pantalla i menús; en aquesta etapa no es pretén descriure els detalls del format d'aquests objectes –i, per tant, normalment no tindran atributs– però poden tenir relacions d'agregació.

*** En anglès, *boundary classes*.**

Exemple d'agregació entre classes de frontera

Una relació d'agregació podria ser la que hi ha entre un format de pantalla i un botó d'aquesta, quan s'engega algun procés.

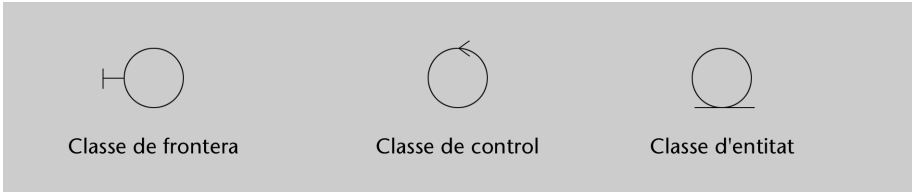
2) Les **classes d'entitats** corresponen als objectes del domini, és a dir, aquells que modelen entitats o esdeveniments del món real dels quals el programari ha de fer servir informació (que són els atributs d'aquestes classes). Molts d'aquests objectes hauran de ser **persistents**, és a dir, han de durar més que el procés que els crea, cosa que obliga a desar-los en algun fitxer o base de dades.

3) Les **classes de control** corresponen a objectes interns del programari* i no persistents. Les operacions d'aquest tipus de classes contenen la part principal dels algorismes de l'aplicació (només queden fora la verificació de la informació entrada per la pantalla i la lectura i gravació dels objectes persistents). Generalment no tindran atributs. En un cas d'ús n'hi pot haver qualsevol nombre –rarament no n'hi haurà cap– però si n'hi ha més d'una, n'hi ha d'haver una que controli el conjunt del cas d'ús.

*** En el sentit que ni són visibles per als usuaris ni modelen res del món real.**

El fet de distingir aquests tres tipus de classes dóna més estabilitat a les classes de cadascun d'ells; així, si es modifica la manera com es presenta la informació a l'usuari sense canviar-ne el processament de les dades, només cal modificar classes de frontera; si es canvien els algorismes de processament de les dades sense canviar-ne l'especificació de les dades ni llur presentació als usuaris, només caldrà modificar classes de control. Per a cada cas d'ús, es fa un diagrama de col·laboració on figuren les classes dels tres tipus i les operacions que es demanen l'una a l'altra.

La notació per a aquests tipus de classes és la següent:



Aquesta notació, igual que aquesta tipificació de les classes, no forma part d'UML.

4.1. Identificació de les classes d'entitats

Probablement és el pas més difícil i també el que té més repercussions sobre els passos i etapes posteriors.

La identificació de les classes d'entitats consisteix a identificar unes primeres classes per mitjà de les quals es puguin especificar els casos d'ús en forma d'interaccions.

Moltes d'aquestes classes passaran al disseny i a la implementació, etapes en les quals es definiran moltes més classes, el paper de les quals serà d'implementació, no pas conceptual.

Busquem classes en el sentit habitual de conjunts d'objectes homogenis que tenen unes mateixes dades (atributs) i comportament (operacions); a més –i a diferència de quan s'elabora el model del negoci– ens interessen les classes de dins del programari i no pas de l'entorn.

Classes d'entorns i d'entitats

Les classes de l'entorn poden ser actors dels casos d'ús; en canvi, les classes que siguin el model de coses de l'entorn tractades pel programari sí que són classes d'entitats.

Regla poc recomanable per a determinar les classes fonamentals

Una de les regles més utilitzades per a determinar quines han de ser les classes fonamentals és buscar els substantius que hi hagi dins la documentació textual dels requisits i tot seguit revisar aquesta llista des de diversos punts de vista; aquesta no és una tècnica gaire aconsellable perquè dins la documentació textual hi ha sempre molts substantius que no corresponen a classes, sovint hi ha classes importants que no es designen explícitament per un substantiu i fins i tot pot passar que els desenvolupadors –que no han de tenir necessàriament una formació lingüística sòlida– tinguin dificultats per a identificar els substantius, sobretot quan són idèntics a formes verbals o adjectius.

Heus ací algunes fonts de classes d'entitats:

- Una bona font de classes, si existeixen, són les **biblioteques de classes** ja definides i programades on es puguin trobar classes que tinguin relació amb els objectius del programari.

Avantatge de les biblioteques de classes

Un dels avantatges de les biblioteques de classes és que no solament aconseguirem identificar classes sinó també reutilitzar-les.

- **Classes suggerides pels experts en el domini**.

•La **documentació (revisada) dels casos d'ús**, especialment les descripcions textuals i el glossari: termes que apareixen sovint, entitats definides explícitament i termes que es donen per coneguts. (Tots ells poden ser, per exemple, persones i papers de persones, objectes físics, esdeveniments, unitats organitzatives –empreses, grups d'alumnes, equips, departaments–, llocs), eliminant els sinònims i adoptant la forma singular. S'exclouen immediatament aquells termes que són massa vagues o que corresponen a entitats exteriors al sistema, i també en general tot el que siguin accions o processos, que és més probable que siguin operacions (a menys que tinguin atributs).

Res no impedeix de fer ús de totes tres fonts alhora.

Tret del cas de classes ja implementades, convé eliminar aquelles que tinguin alguna d'aquestes característiques:

a) Classes sense atributs: això indica que el sistema no en té informació per tractar.

b) Classes sense operacions, és a dir, que no tenen un paper ni actiu ni passiu en cap operació.

c) Classes que tenen només un atribut; pot ser que realment no sigui una classe sinó un atribut d'una altra.

d) Classes amb un sol objecte: si es dóna aquest cas, cal mirar si n'hi ha d'altres amb atributs i operacions similars, i intentar unificar-les; també cal mirar si no val més considerar els atributs i operacions d'aquest únic objecte com a atributs i operacions de classe.

4.2. Especificació dels atributs de les classes d'entitats

Els atributs de les classes d'entitats són generalment dades esmentades explícitament en els casos d'ús; això vol dir que els usuaris les usen directament.

Com que el model de l'anàlisi convé que sigui independent del llenguatge de programació, els tipus dels atributs seran tipus abstractes* i no pas tipus suportats per un llenguatge de programació concret; tanmateix convé que s'estableixi quins són aquests tipus. Per la mateixa raó, els noms dels atributs no han d'estar sotmesos necessàriament a les restriccions dels noms de cap llenguatge de programació concret, però per tal d'evitar haver de canviar els noms en arribar al disseny, és convenient que els noms d'atribut tinguin un format compatible amb el de la majoria de llenguatges de programació.

*** Per exemple, enter sense indicar-ne la precisió.**

Format dels noms d'atributs compatible amb la programació

Un format compatible amb la programació podria ser, per exemple, atributs constituïts només per lletres, xifres i *underscores*, sense que aquests puguin figurar al principi ni al final. Això val per a tots els noms en general: noms d'operacions, de classes, de paràmetres, etc.

4.2.1. Casos especials en els atributs

Ara veurem un seguit de casos especials en els atributs:

1) Atributs el valor dels quals és compartit per diversos objectes: seran **atributs de classe**.

2) Atributs no aplicables a tots els objectes de la classe: es pot crear una superclasse que tingui només aquells atributs que siguin aplicables a tots els objec-

tes i després una subclasse o més amb atributs que siguin aplicables a tots els objectes respectius.

3) Atributs amb valors repetitius: de vegades pot ser convenient definir-los com a classe a part amb una associació *n* a 1 amb la primera, sobretot si són atributs compostos o parells d'atributs amb la mateixa cardinalitat.

4) Atributs derivats (edat, etc.): només s'inclouran si figuren en sortides descrites en els casos d'ús, mentre que aquells que siguin simplement un recurs per a reduir la durada dels processos a canvi d'utilitzar més espai a memòria o a la base de dades no s'introduiran fins al disseny, que és quan es té en compte aquest punt de vista.

4.3. Identificació de les relacions entre classes

Arribats en aquest punt, tenim una llista en principi completa, per bé que provisional, de les classes del programari.

Classes d'entitats i taules relacionals

Les classes d'entitats amb els seus atributs són semblants a les taules d'una base de dades relacional (ja s'ha esmentat abans que moltes de les classes d'entitat són persistents); doncs bé, és com si les taules estiguessin sense normalitzar. Així com la normalització fa canviar el contingut de les taules en termes d'atributs i de files, i també fa definir taules noves, també hi haurà atributs que passaran d'una classe a una altra i es definiran classes noves. Però, tanmateix, convé tenir clars els límits d'aquesta analogia: ni hi ha una correspondència clara entre els passos de la normalització de taules relacionals i els de la revisió de la llista de classes, ni aquesta està tan formalitzada com aquella, ara com ara.

S'examinaran les relacions d'herència, les associacions i les relacions d'agregació; tot i que en parlarem en aquest ordre, és un procés iteratiu en el qual pot passar, per exemple, que la identificació d'una associació faci canviar una jerarquia d'herència.

4.3.1. Jerarquies d'herència

Cal establir les jerarquies d'herència, cosa que es fa per dues vies, la generalització i l'especialització.

Especialització

Vegeu la definició d'*herència per especialització en* el subapartat 4.2.1 del mòdul "UML (I): el model estàtic" d'aquesta assignatura.

Pel que fa a l'especialització, cal tenir en compte el següent:

1) Només cal afegir una subclasse si els seus objectes tindran atributs, operacions o associacions que no tindran la resta d'objectes de la superclasse. Cal evitar sistemàticament subclasses corresponents als diferents estats que poden tenir els objectes de la superclasse; només s'ha de definir una subclasse per a un estat si es dóna alguna de les circumstàncies esmentades.

2) Si es pot, cal evitar la doble especialització, és a dir, que d'una superclasse es distingeixin dos conjunts independents de subclasses. De vegades això pot ser conseqüència del fet que la superclasse sigui en realitat la combinació de dues classes*.

*** Com el *join* de dues taules en una base de dades relacional.**

Exemple de combinació de dues classes

Considerem l'exemple següent, adaptat d'un de Richter: si els comptes d'un banc es divideixen, des d'un punt de vista, en comptes corrents i comptes a termini, i, des d'un altre, en comptes de persones i comptes d'empreses, seria millor distingir dues classes diferents –comptes i clients– unides per una associació i dues subclasses a cadascuna: comptes corrents i comptes a termini, a comptes, i persones i empreses, a clients.

Generalització

Vegeu la definició d'*herència per generalització* en el subapartat 4.2.2 del mòdul "UML (I): el model estàtic" d'aquesta assignatura.

Quant a la generalització, cal tenir en compte aquests punts:

1) Si diverses classes de significat semblant tenen atributs o operacions en comú, cal considerar la possibilitat de definir una superclasse comuna a totes aquestes classes que n'agrupi tots els atributs i operacions en qüestió. Perquè els atributs comuns puguin posar-se dins la superclasse, cal que estiguin definits igual a totes les classes, però les operacions només cal que tinguin la mateixa signatura (o almenys el mateix nom) i una funció anàloga a totes les classes, ja que les operacions de la superclasse poden ser abstractes.

La superclasse serà abstracta, ja que tots els seus objectes pertanyeran a alguna de les classes inicials (a menys que en definir la superclasse ens adonem que hi ha objectes d'aquesta que no són de cap d'aquelles classes i que, per tant, faltaven classes per identificar). Ara bé, que la superclasse sigui abstracta no vol dir que ho hagin de ser totes les seves operacions –aquelles operacions que es facin de la mateixa manera en totes les subclasses seran operacions concretes de la superclasse.

2) Convé situar tots els atributs i operacions comuns al més amunt possible dins la jerarquia d'herència, encara que les operacions hagin de ser abstractes.

3) Havent definit una superclasse, pot passar que una subclasse tingui tots els seus atributs i operacions heretats, és a dir, que no en tingui cap de propi; llavors es pot suprimir aquesta subclasse, i la superclasse serà evidentment concreta.

4) No té sentit que una superclasse abstracta tingui només una subclasse (i tampoc se'n prevegin més en el futur): caldrà suprimir la superclasse. En canvi, pot ser normal que passi això mateix amb una superclasse concreta, per exemple, de resultes de la situació del punt anterior.

5) Una superclasse abstracta pot heretar tant operacions concretes com operacions abstractes, i les seves operacions pròpies* poden ser tant abstractes com concretes.

*** Les operacions pròpies d'una classe són aquelles que no ha heretat.**

6) Què es pot fer si hi ha tres subclasses, i un atribut o operació és comú només a dues d'aquestes? Una solució és afegir un nivell intermedi a la jerarquia d'herència, de manera que les dues classes en qüestió siguin subclasses d'una que tingui l'atribut o operació esmentat; ara bé, si també hi hagués un atribut o operació que fos comú a les subclasses primera i tercera, caldria una nova subclasse intermèdia, i llavors la primera subclasse seria subclasse de les dues subclasses intermèdies, ja que té totes dues operacions, i per tant estaríem en un cas d'herència múltiple.

En el cas d'una operació, hi ha també l'opció d'incloure-la dins la superclasse de les tres subclasses i redefinir-la com una operació nul·la dins la tercera subclasse (això només seria viable si la superclasse és abstracta, ja que si fos instanciable i es creés un objecte d'aquesta que també fos de la tercera subclasse, li seria aplicable l'operació tal com s'hauria definit a la superclasse). Parlant de l'agregació veurem una altra solució a aquest problema.

Vegeu les agregacions en el subapartat 4.3.5 d'aquest mòdul didàctic.

4.3.2. Herència múltiple

L'herència múltiple consisteix en el fet que una classe té diverses superclasses i, per tant, hereta atributs, operacions, associacions i agregacions de totes aquestes.

Hi ha conflicte si la subclasse hereta dos atributs, operacions o associacions amb el mateix nom i diferents propietats (**herència repetida**), ja que en principi no se sap quin preval, i cal establir un criteri, poc o molt arbitrari, per a decidir-ho. Alguns llenguatges de programació no suporten l'herència múltiple, i llavors en l'etapa de disseny cal adaptar el diagrama estàtic, però no per això s'ha de renunciar a fer servir l'herència múltiple en l'anàlisi si és escaient.

4.3.3. Interfícies

Recordareu que les interfícies d'UML equivalen a classes abstractes sense atributs i amb totes les operacions abstractes.

Vegeu les interfícies d'UML en el subapartat 4.4 del mòdul "UML (I): el model estàtic" d'aquesta assignatura.

En el lloc de les subclasses, definides estàticament, hi ha les classes que implementen la interfície, que es poden substituir sense modificar-la. Això fa que les interfícies siguin una alternativa a les classes abstractes que pot ser avantatjosa des del punt de vista de la flexibilitat.

4.3.4. Associacions

Hi ha una associació entre classes quan els objectes d'una necessiten la col·laboració d'objectes de l'altra per a dur a terme les seves operacions

Recordeu

En els diagrames de col·laboració els missatges entre objectes "circulen" per les associacions.

Per aquesta raó, és obvi que hi ha associacions entre les classes de frontera i almenys algunes classes de control, d'una banda, i entre classes de control i classes d'entitats, de l'altra; les classes de control que no estiguin associades a classes de frontera estaran associades a d'altres classes de control.

En relació amb les associacions convé tenir en compte el següent:

1) Tant les associacions com els papers que hi fan les classes poden tenir noms; moltes vegades no cal donar alhora noms a l'associació i a tots els seus papers, ja que un o altre seria banal, però si l'associació no té nom, n'ha de tenir almenys un dels seus papers.

2) Entre dues classes (o més) hi pot haver més d'una associació, amb significat diferent (i llavors pot ser que d'aquestes associacions enllacin els mateixos objectes concrets o no). Cal que siguin diferents o bé els noms d'aquestes associacions o bé els noms dels seus papers.

Associacions diferents entre dues classes

Considerem l'exemple següent, adaptat de Richter: en una classe de vols comercials, hi ha la companyia, el número de vol, l'aeroport d'origen i el de destinació, el dia i l'hora de sortida, i tots els vols amb el mateix número tenen la mateixa companyia i els mateixos aeroports d'origen i de destinació; una opció millor, que evitaria aquestes repeticions, seria substituir la classe per dues: una que tingués el número de vol, la companyia i els aeroports d'origen i de destinació i una altra que tingués la data i hora de sortida, amb una associació d'una a diverses entre les dues (això és anàleg al pas de la segona a la tercera forma normalitzada en bases de dades relacionals).

3) Si existeixen associacions entre la classe A i cadascuna de les subclasses de la classe B, és millor substituir-les per una única associació entre A i B; no solament el diagrama és més senzill, sinó que si s'afegeix una nova subclasse, no cal afegir una nova associació.

4) Hi pot haver una associació (binària o no) entre una classe i ella mateixa.

Exemple d'associació entre una classe i ella mateixa

Tots els empleats d'una empresa (menys un) tenen un cap que és també empleat; ara bé, un empleat no pot ser cap d'ell mateix, i en general un objecte no pot estar associat a ell mateix, ja que aquesta situació podria produir un bucle sense fi.

5) És convenient una classe associativa quan es dóna una d'aquestes circumstàncies: l'associació té atributs o bé hi ha una classe la vida dels objectes de la qual està lligada a la dels enllaços de l'associació i només té un objecte per a cada enllaç.

6) Convé revisar cada associació amb cardinalitat múltiple en els dos papers* per a determinar si no hauria de ser una classe associativa.

*** És a dir, les associacions *m* : *n*.**

4.3.5. Agregacions

Les agregacions es poden considerar un cas particular de les associacions, en què el significat de l'associació és que un paper és part de l'altre en algun sentit, i certament convé aplicar-lo, en general, sempre que ens trobem en casos d'aquesta mena.

Vegeu les associacions en el subapartat 6.1 del mòdul "UML (I): el model estàtic" d'aquesta assignatura.

L'agregació, però, té algunes propietats que la poden fer útil també per a altres situacions en les quals pot substituir d'altres relacions, principalment d'herència, amb vista al fet que el programari desenvolupat sigui més flexible:

- La classe composta delega en les classes components aquelles operacions i atributs que fan referència a elles; això no és imprescindible, però és molt recomanable en general.

- Els components es poden crear i esborrar en temps d'execució, mentre que les subclasses s'han de definir en temps de compilació.

- Els components –sobretot si l'agregació és composició– no cal que siguin visibles des de l'exterior, cosa que vol dir que es pot canviar l'estructura dels objectes d'una classe composta sense afectar altres classes.

- Entre dues classes hi pot haver diverses relacions d'agregació (que llavors s'hauran de distingir pel nom), de la mateixa manera que hi pot haver diverses associacions.

Algunes aplicacions no òbvies del concepte d'*agregació* són aquestes:

1) Per a utilitzar l'agregació com a alternativa a les subclasses, cal fer el següent: els atributs i operacions que serien específics de la subclasse es posen en una nova classe que es defineix com a component de l'anterior amb una cardinalitat mínima de zero, de manera que aquells objectes de la classe inicial que han de tenir els atributs i operacions en qüestió tindran aquest component i els altres no el tindran. Per a canviar de subclasse un objecte caldria esborrar-lo i crear-lo altre cop, ara a la subclasse nova, amb la qual cosa les eventuals referències a ell deixarien de ser vàlides; amb l'agregació, simplement caldria esborrar de l'objecte el component corresponent a la subclasse antiga i afegir-li el de la nova.

2) Com que el valor màxim de la cardinalitat d'un component pot ser més gran que 1, el cas dels atributs amb valors repetitius descrit en un altre suba-

Vegeu el cas dels atributs amb valors repetitius en el subapartat 4.2.1 d'aquest mòdul didàctic.

partat pot ser resolt amb agregació (en principi, composició) en comptes d'associació.

3) El fet que en una agregació o composició els components poden ser compartits suggereix una alternativa a l'herència múltiple, que a més soluciona l'herència repetida, ja que no hi haurà dos atributs o operacions amb el mateix nom sinó només un.

4) El cas dels atributs o operacions compartits per dues subclasses de tres esmentat en un altre subapartat es pot resoldre posant tots els atributs i operacions no comuns a dues subclasses en una classe nova que seria component compartit de totes dues.

Vegeu el cas dels atributs o operacions compartits en l'epigraf "Generalització" del subapartat 4.3.1 d'aquest mòdul didàctic.

5) L'herència múltiple també es pot resoldre al revés de com s'ha fet abans: fent que la subclasse sigui la classe composta i les superclasses les seves classes components! Aquesta solució permet afegir superclasses sense haver de modificar la subclasse.

4.4. Identificació de les classes de frontera, les classes de control i de les operacions. Diagrama estàtic d'anàlisi

De la descripció textual dels casos d'ús surten les operacions de les classes de frontera i també, de manera més indirecta, les de les classes de control; les de les classes d'entitats moltes vegades hi estan implícites, però gairebé sempre són òbvies.

Operacions

1) Les operacions de les classes de frontera són: *introduir* i *presentar dades*.

2) Les operacions de les classes d'entitats són: *afegir, esborrar, llegir, regravar*, o equivalents.

Una manera sistemàtica de cercar i documentar les classes de frontera, de control i les operacions dels tres tipus de classes és elaborar diagrames de col·laboració simplificats dels casos d'ús representant les classes dels tres tipus. Després convé fer un diagrama estàtic en el qual figurin totes les classes dels tres tipus, amb els atributs i operacions identificades i amb totes les relacions entre elles, que és el **diagrama estàtic d'anàlisi**.

5. Especificació formal dels casos d'ús

Quan arribem en aquest punt, tenim descrit de manera formal –mitjançant el diagrama estàtic d'anàlisi– el que podríem anomenar l'*estructura estàtica del programari que es desenvolupa*; resta representar de manera formalitzada el comportament del sistema, i la manera més natural de fer-ho és descrivint formalment els casos d'ús.

Al pas d'especificació de les classes d'anàlisi s'han elaborat uns diagrames de col·laboració simplificats dels casos d'ús, la finalitat dels quals no era descriure de manera formal i detallada el procés de cada un, sinó identificar les classes de frontera i de control i les operacions d'aquestes i de les classes d'entitats. La forma simplificada que s'hi ha fet servir no té en compte la seqüència dels missatges ni el fet que puguin ser repetits un cert nombre de vegades o subjectes a condicions; aquesta especificació pot ser suficient per a alguns casos d'ús senzills, però d'altres necessitaran una especificació complementària; per a aquests, caldrà fer diagrames de seqüència o de col·laboració complets, o, de vegades, diagrames d'activitats.

També es poden fer diagrames d'estats per a il·lustrar, més que casos d'ús en particular, el comportament d'algunes classes d'entitats o de control; en el cas de les primeres, el diagrama d'estats segurament repetirà des d'un altre punt de vista el que ja s'hagi especificat en els casos d'ús afectats, mentre que en el cas de classes de control, pot formar part de la descripció de llur comportament.

6. Anàlisi de la interfície d'usuari

L'anàlisi de la interfície d'usuari parteix de la informació següent:

- la documentació de les tasques futures,
- les especificacions d'usabilitat,
- la llista de les classes frontera,

i té com a objectiu un esquema del contingut de cada finestra associada a una classe de frontera, llevat de les anomenades *finestres secundàries,* que només serveixen per a coses com ara presentar missatges a l'usuari o demanar-li confirmacions o la introducció del valor d'un paràmetre.

És molt important comentar amb els usuaris el contingut de les finestres i la interacció entre l'actor i el sistema.

7. Exemple

Ja hem vist un exemple de recollida i documentació de requisits; ara continuarem el procés fent l'anàlisi per al mateix cas.

Vegeu l'exemple explicat en l'apartat 5 del mòdul "Recollida i documentació de requisits" d'aquesta assignatura.

7.1. Revisió dels casos d'ús

Vegeu la documentació textual al subapartat 5.6.3 del mòdul "Recollida i documentació de requisits" d'aquesta assignatura.

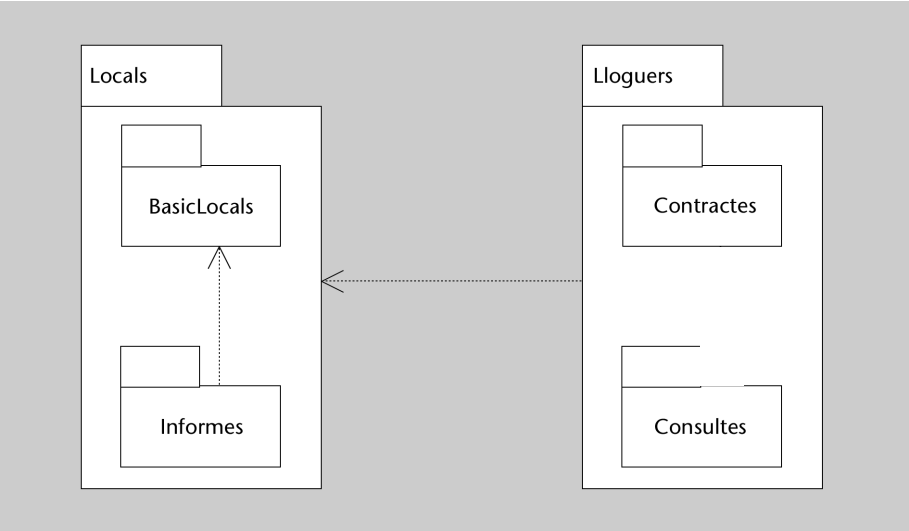
Els casos d'ús obtinguts en l'etapa anterior són senzills i sembla que tal com són ja es poden utilitzar com a base per als passos posteriors. Ara bé, en la documentació textual, en la part d'"Altres requisists", s'indica que s'ha de poder consultar, modificar i esborrar tota la informació; per a complir amb aquest requisit caldria afegir alguns casos d'ús, trivials en general; no s'ha fet, perquè no aportarien cap concepte nou important i allargarien molt l'exemple.

En tot cas, convé tenir present que en sistemes de programari reals la informació s'ha de poder modificar (per corregir-la, si més no), que tota la informació s'ha de poder consultar –sovint de diverses maneres– i que rarament hi ha informació eterna: s'ha d'anar esborrant aquella que ja no faci falta (de vegades pot ser que en comptes d'esborrar totalment la informació que ja no es fa servir es copiï en un fitxer històric, normalment en suports no permanentment *online* com ara disquets i cintes magnètiques).

7.2. Paquets d'anàlisi i de serveis

Evidentment, a la realitat, en un cas tan senzill com aquest només hi hauria un paquet; però si atenem al grau de dependència entre els casos d'ús podríem distingir dos paquets d'anàlisi, *Locals*: que comprèn els casos d'ús relatius a locals i propietaris (els casos d'ús 1, 2 i 3), i els referents a contractes i llogaters (la resta); dins del primer podríem distingir dos paquets de serveis: *BasicLocals* i Informes si considerem opcional la introducció de l'informe de l'inspector, i *Lloguers*, dins del qual es podria considerar que els casos d'ús 6 i 7 són opcionals i constitueixen un paquet de servei *Consultes* separat de *Contractes*, que comprèn els casos d'ús 4 i 5.

L'esquema de les relacions entre els paquets seria el següent:



Recordeu

Les fletxes com les del diagrama indiquen relacions de dependència.

7.3. Identificació de les classes d'entitats

Començarem per identificar les classes d'entitats a partir dels casos d'ús. Per a cada cas d'ús s'indiquen les classes que s'hi troben; quan una classe ja s'havia identificat en un cas d'ús anterior, el seu nom va seguit d'un asterisc, mentre que les classes que són dubtoses van seguides d'un interrogant.

- Cas d'ús número 1: "Afegir local". Classes: *Local*, *Propietari*, *Zona*?
- Cas d'ús número 2: "Afegir propietari". Classes: *Propietari**, *Particular*? *Empresa*?
- Cas d'ús numero 3: "Introduir informe". Classes: *Local**, *Installació*? *Inspector*?
- Cas d'ús número 4: "Introduir contracte". Classes: *Local**, *Llogater*, *Contracte*, *Venedor*?
- Cas d'ús número 5: "Afegir llogater". Classes: *Llogater**.
- Cas d'ús número 6: "Afegir eventual llogater ". Classes: *LlogaterEventual*.
- Cas d'ús número 7: "Buscar locals". Classes: *Local**.

Comentaris

Les classes *Zona* i *Installació* es descarten perquè no tenen cap atribut i, per tant, es poden considerar atributs d'altres classes. *Inspector* i *Venedor*, que no s'esmenten en la descripció dels casos d'ús, però sí en les qüestions a aclarir i les seves respostes, estan en el mateix cas. *Particular* i *Empresa* es descarten com a classes (serien subclasses de *Propietari* i de *Llogater*) perquè no tenen ni atributs ni operacions pròpies.

Per tant, la primera llista de classes d'entitats és aquesta: *Local*, *Propietari*, *Llogater* i *EventualLlogater*.

7.4. Especificació dels atributs de les classes d'entitats

En la llista que veiem a continuació, presentem els atributs de les classes d'entitats corresponents a l'exemple que considerem:

- *Classe Local: zona*(string), *tipus*(string), *numero*(integer), *adreca*(text), *superficie*(real), *caracteristiques*(text), *volum*(real), *caracteristiques_polivalent*(text), *NIF_propietari*(string), *accessibilitat*(text), *installacions*(text), *agent*(string), *inspector*(string).

Classe

L'atribut *nom* de *Propietari*, *Llogater* i *LlogaterEventual* conté els cognoms i nom o bé la raó social segons que el propietari sigui una persona física o una empresa.

- Classe *Propietari*: *NIF*(string), *nom*(text), *adreca*(text), *telefon*(string).
- Classe *Llogater*: *NIF*(string), *nom*(text), *adreca*(text), *telefon*(string).

Recordeu

Convé que els noms dels atributs respectin el format suportat pels compiladors; per tant, és millor no posar-hi accents ni caràcters especials.

- Classe *EventualLlogater*: *NIF*(string), *nom*(text), *adreca*(text), *telefon*(string), *tipus_local*(string), *zona*(string), *superficie_minima*(real), *diversos*(text).
- Classe *Contracte*: *NIF_llogater*(string), *NIF_propietari*(string), *venedor*(string).

7.5. Relacions

7.5.1. Relacions d'herència

Com que les botigues-magatzem tenen un atribut propi, el *volum*, es defineix *BotigaMagatzem* com a subclasse de *Local*. Atès que els polivalents tenen també un atribut propi, les característiques_polivalent, es defineix la subclasse *Polivalent* de *BotigaMagatzem*; com que els polivalents són també oficines, també n'han de ser subclasse i, per tant, cal definir *Oficina* com a subclasse de *Local*.

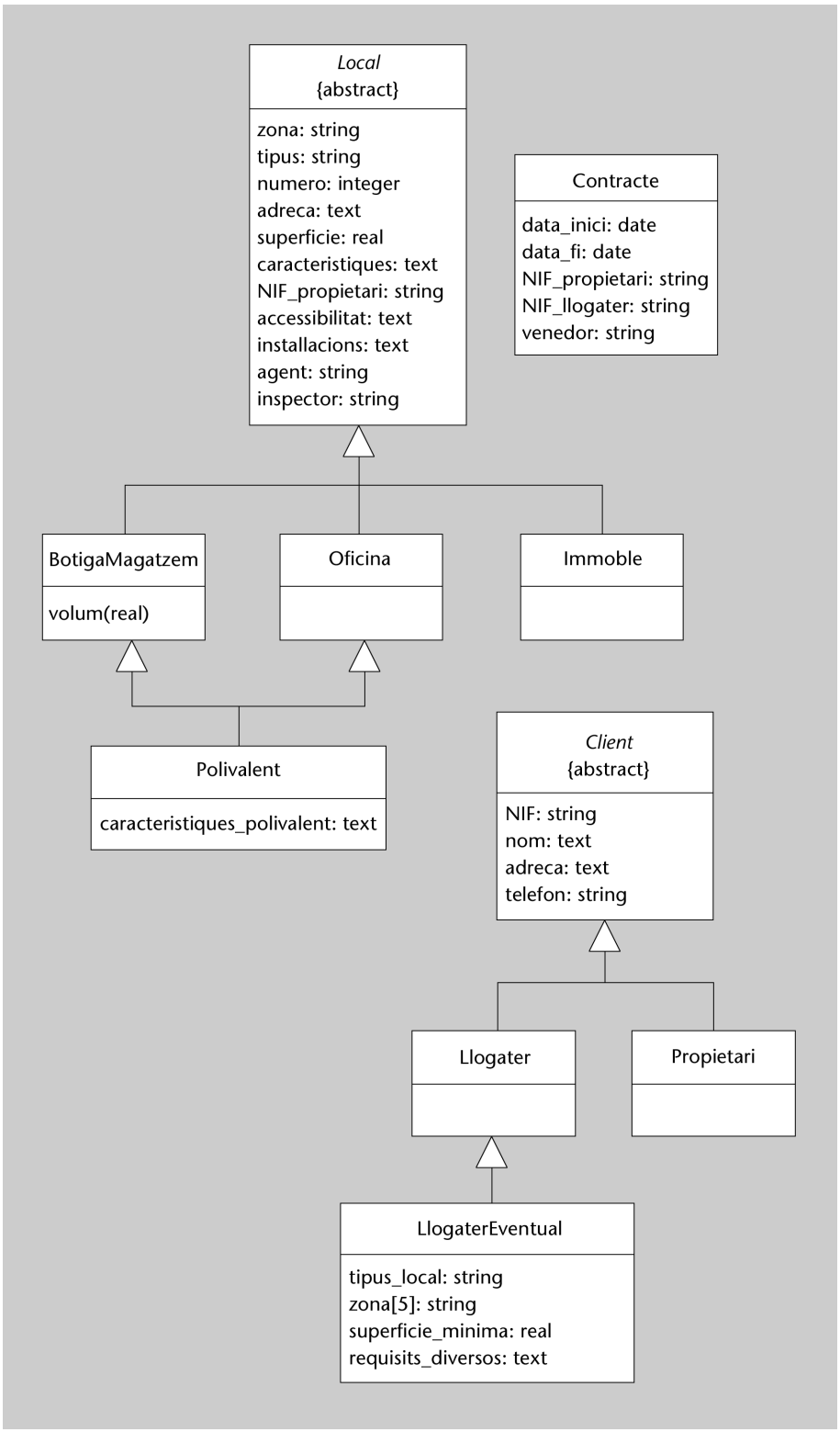
La subclasse *Immoble* s'afegeix per claredat del diagrama, ja que tots els locals que no són botigues-magatzem ni oficines són immobles; per aquesta raó, *Local* és una classe abstracta.

Els atributs comuns a *Propietari* i *Llogater* es posen dins una superclasse abstracta, *Client*; *EventualLlogater* és subclasse de *Llogater* a conseqüència dels atributs.

Observeu

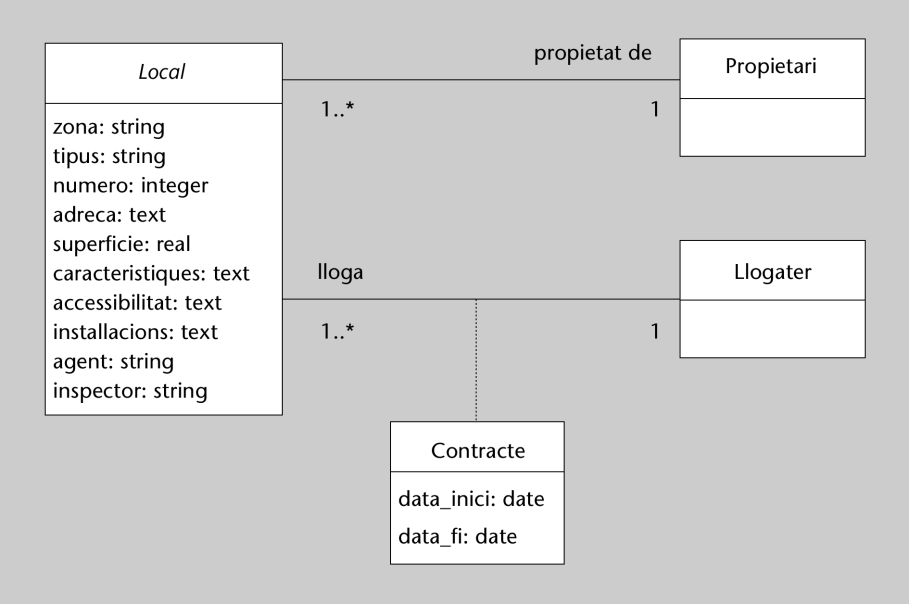
Per claredat, el fet que una classe és abstracta s'indica de dues maneres: amb el nom en cursiva i amb la propietat *abstract*; recordeu, però, que amb una sola n'hi ha prou.

Com que resulta que *Propietari* i *Llogater* no tenen més atributs que els heretats, podríem pensar a suprimir-les, però no es fa perquè tenen comportament diferent: *Propietari* té apartaments i *Llogater* els lloga.


Local
{abstract}
zona: string
tipus: string
numero: integer
adreca: text
superficie: real
caracteristiques: text
NIF_propietari: string
accessibilitat: text
installacions: text
agent: string
inspector: string
Contracte
data_inici: date
data_fi: date
NIF_propietari: string
NIF_llogater: string
venedor: string
BotigaMagatzem
volum(real)
Oficina
Immoble
Polivalent
caracteristiques_polivalent: text
Client
{abstract}
NIF: string
nom: text
adreca: text
telefon: string
Llogater
Propietari
LlogaterEventual
tipus_local: string
zona[5]: string
superficie_minima: real
requisits_diversos: text

7.5.2. Associacions

En aquest diagrama podeu veure les associacions de l'exemple:



L'associació entre *Propietari* i *Local* es dedueixen del cas d'ús 1: "Afegir local", i la classe associativa *Contracte* es dedueix del cas d'ús 4: "Introduir contracte".

Fixeu-vos que l'associació *Contracte* s'estableix amb la superclasse *Local* i no pas amb les seves subclasses, ja que és comuna a totes; en canvi, les relacions s'estableixen amb les subclasses *Propietari* i *Llogater* per la raó contrària.

Observeu també que s'han pogut suprimir els atributs *NIF_propietari* i *NIF_llogater* de les classes *Contracte* i *Local**, perquè són redundants amb les associacions.

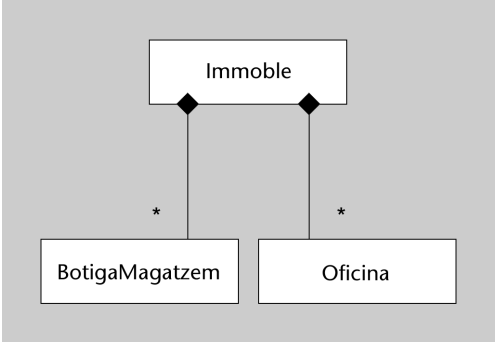
*** A la classe *Local* només s'ha suprimit el primer atribut.**

Implementació de les associacions

Podria ser que a l'hora de programar les associacions s'implementessin precisament mitjançant atributs com els que acaben de veure (més una llista dels codis dels locals de cada propietari), i llavors caldria tornar a afegir-los; però també es podrien implementar les associacions d'altres maneres.

7.5.3. Agregacions

Segons els requisits, un immoble es compon de botigues magatzem i/o oficines; per tant, entre aquestes tres subclasses hi ha les agregacions corresponents. Aquestes agregacions són composicions, atès que cada botiga magatzem o oficina pertany a un sol immoble.



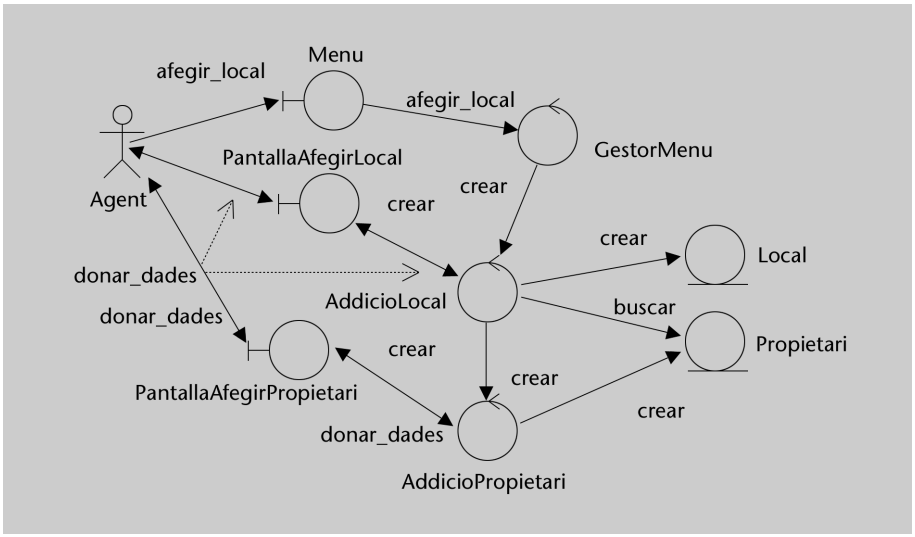
7.6. Identificació de les classes de frontera, les classes de control i de les operacions

Per a cada cas d'ús, farem un diagrama de col·laboració simplificat.

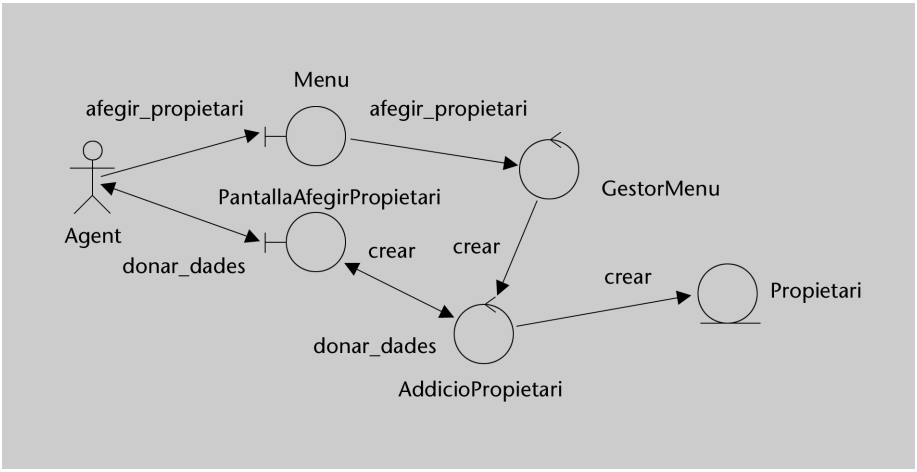
A tots els casos d'ús s'ha posat que l'actor demana una opció a la classe de frontera *Menu*, que correspon al menú de l'aplicació, i aquesta la passa a la classe de control *GestorMenu*, que crida la classe de control principal del cas d'ús. Els noms dels missatges seran operacions de les classes destinatàries; aquells que van de les classes de frontera a l'actor no li demanen cap operació com és lògic.

Cas d'ús 1: "Afegir local"

Aquest cas d'ús comprèn el cas d'ús 2: "Afegir propietari", que, com sabem, l'estén quan no s'ha trobat el propietari.

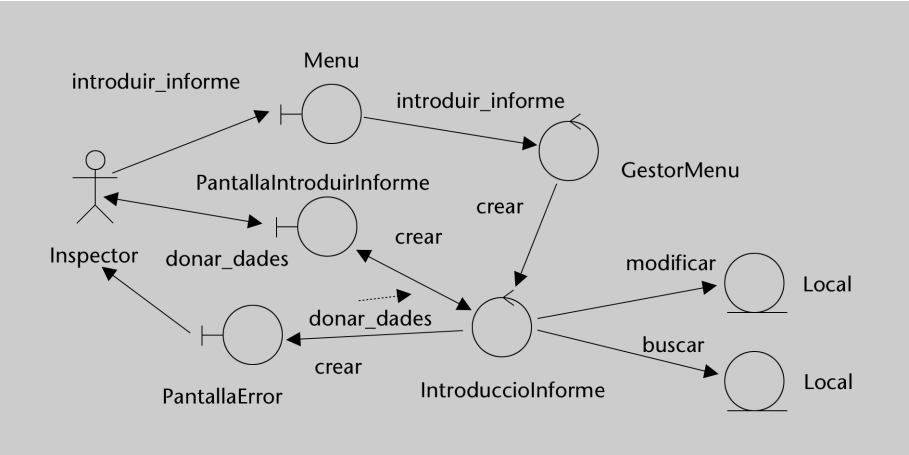


Cas d'ús 2: "Afegir propietari"



Cas d'ús 3: "Introduir informe"

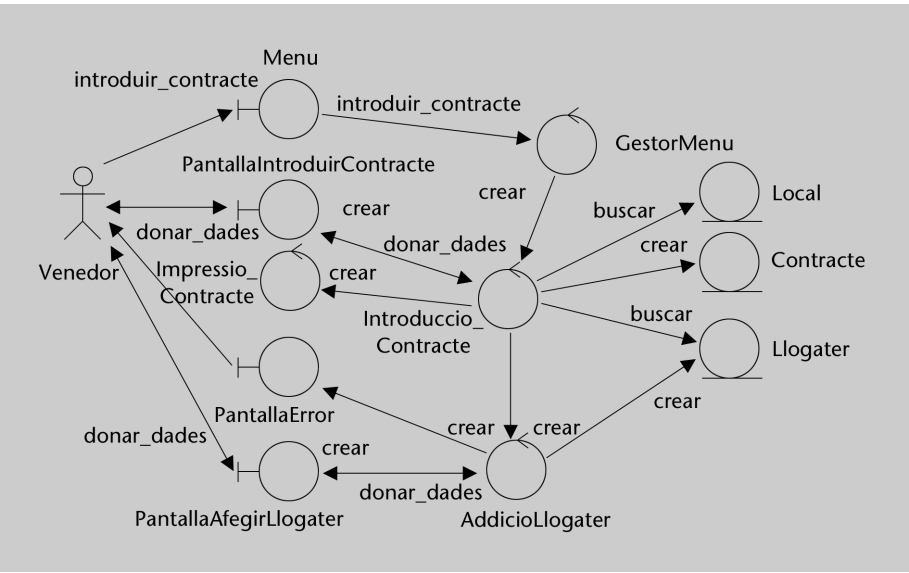
Com es pot veure al diagrama de col·laboració d'aquest cas d'ús:



Es busca el local i, si no es troba, s'envia un missatge d'error a l'actor; si s'ha trobat, un cop introduïdes les dades de l'informe es modifica l'objecte *Local* donant valors als camps corresponents i gravant-lo de nou a la base de dades.

Cas d'ús 4: "Introduir contracte"

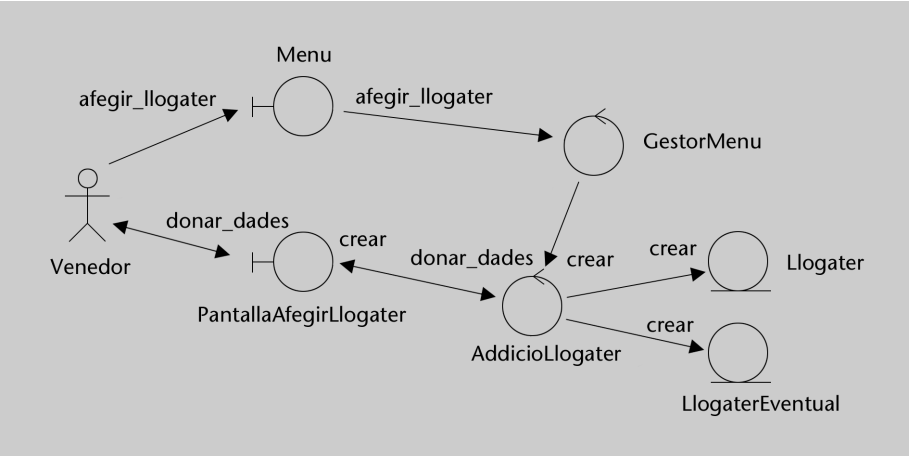
Com podeu observar en el diagrama, si no es troba el llogater es crea (cas d'ús 5: "Afegir llogater"), però si no es troba el local s'envia un missatge d'error a l'actor. La classe *ImpressioContracte* imprimeix les dades del contracte.



Casos d'ús 5 i 6: "Afegir llogater" i "Afegir eventual llogater "

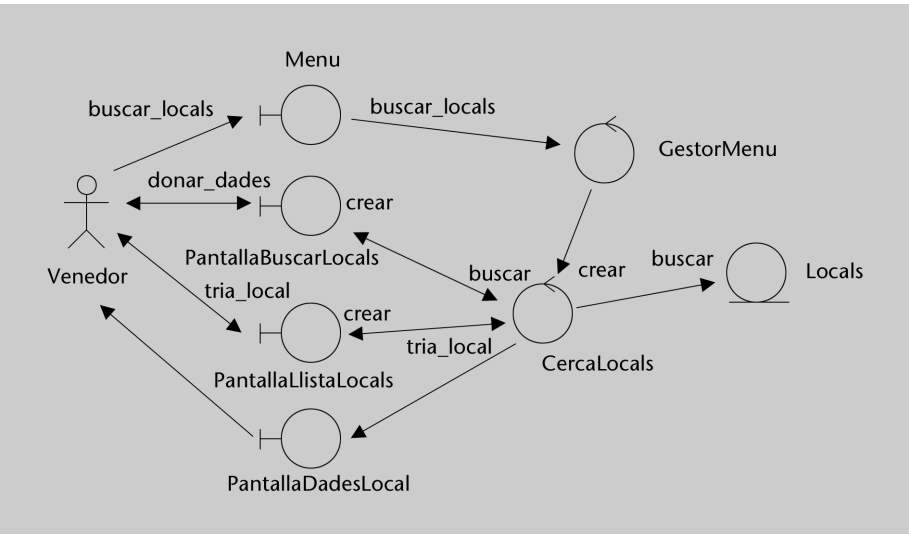
Sabem que el cas d'ús 6 és una especialització del cas d'ús 5, ja que difereix d'aquest en què s'hi afegeixen els valors dels atributs que indiquen quina

mena de locals vol el eventual llogater , que estan a la subclasse *LlogaterEventual*; per tant, els podem convertir en un únic cas d'ús que crea o bé un objecte de *Llogater* o bé un *LlogaterEventual*, respectivament.



Cas d'ús 7: "Buscar locals"

Com podeu veure en el diagrama de col·laboració:



L'actor dóna els arguments de la consulta; li surt una llista dels codis dels locals que compleixen les condicions, dins la qual l'actor pot triar un local de la llista i se n'hi presenten les dades (totes o bé totes llevat l'adreça del local i el NIF del propietari).

Activitat

7.1. Feu ara el diagrama estàtic d'anàlisi complet, preferentment amb una eina CASE.

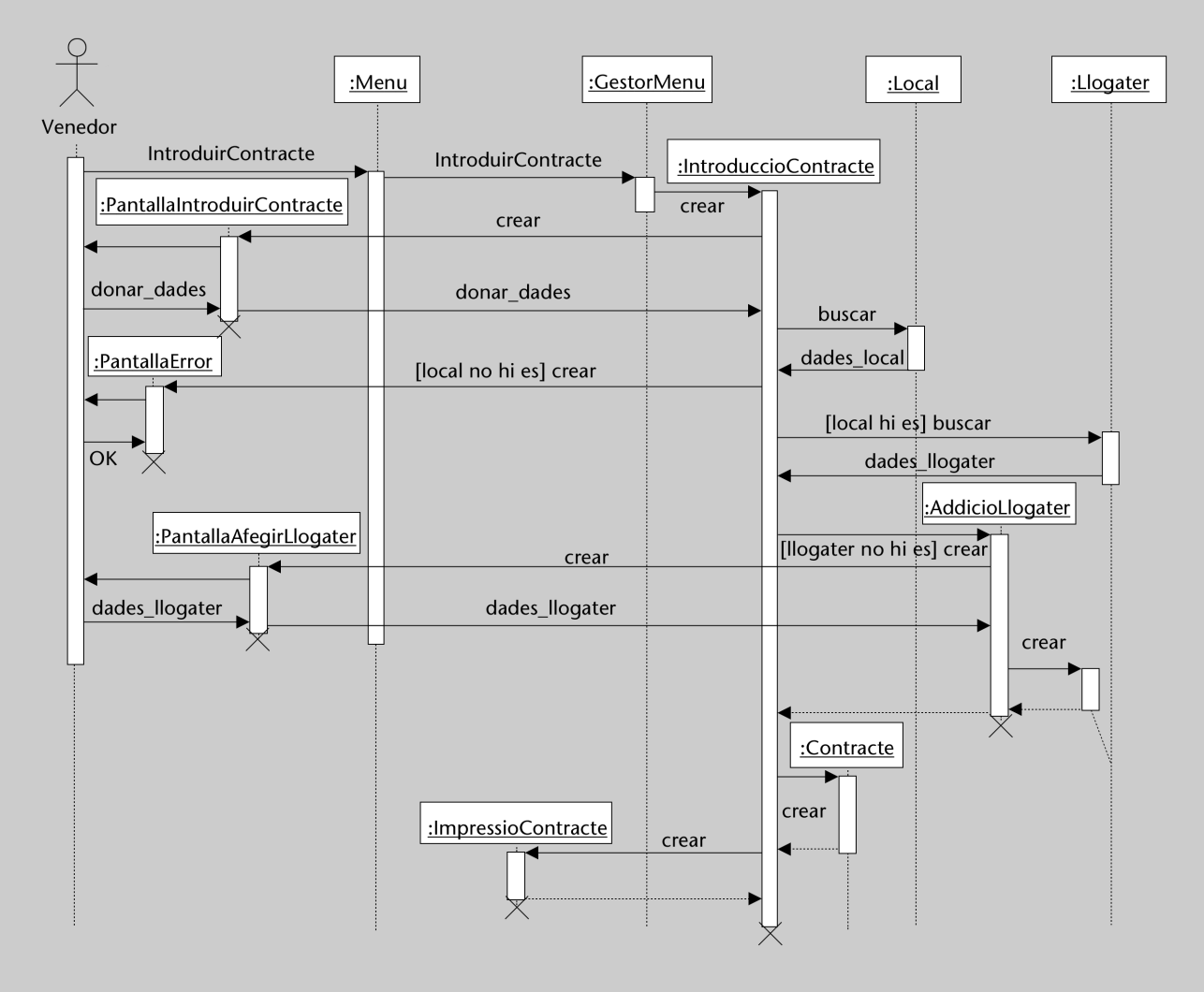
7.7. Especificació formal dels casos d'ús

Segurament que haureu observat que en alguns dels diagrames de col·laboració simplificats que acabem de veure hi ha missatges que només s'envien quan

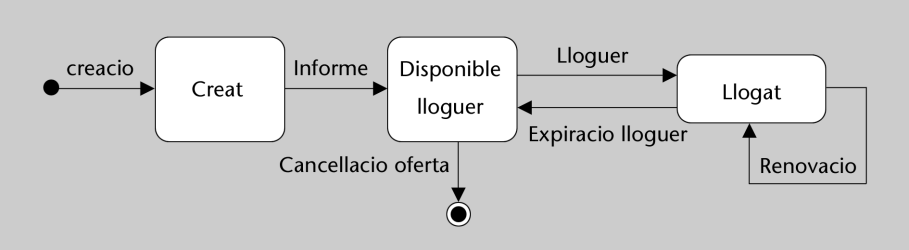
es compleix una condició, i que també, de vegades, entren i surten diversos missatges d'una mateixa classe sense que quedi clar en quin ordre ho fan; per tant, caldrà fer una especificació més detallada d'aquests casos d'ús.

Com que ja n'hem fet un diagrama de col·laboració –ni que sigui simplificat– ara farem servir el diagrama de seqüències, perquè dóna una visió dels casos d'ús que és complementària de la que ja tenim; només es farà per a un cas d'ús, el més complicat, el cas d'ús 4: "Introduir contracte". També veurem el diagrama d'estats de l'única classe en què no és trivial, la classe *Local*.

a) Diagrama de seqüències del cas d'ús 4: "Introduir contracte":



b) Diagrama d'estats de la classe *Local*:



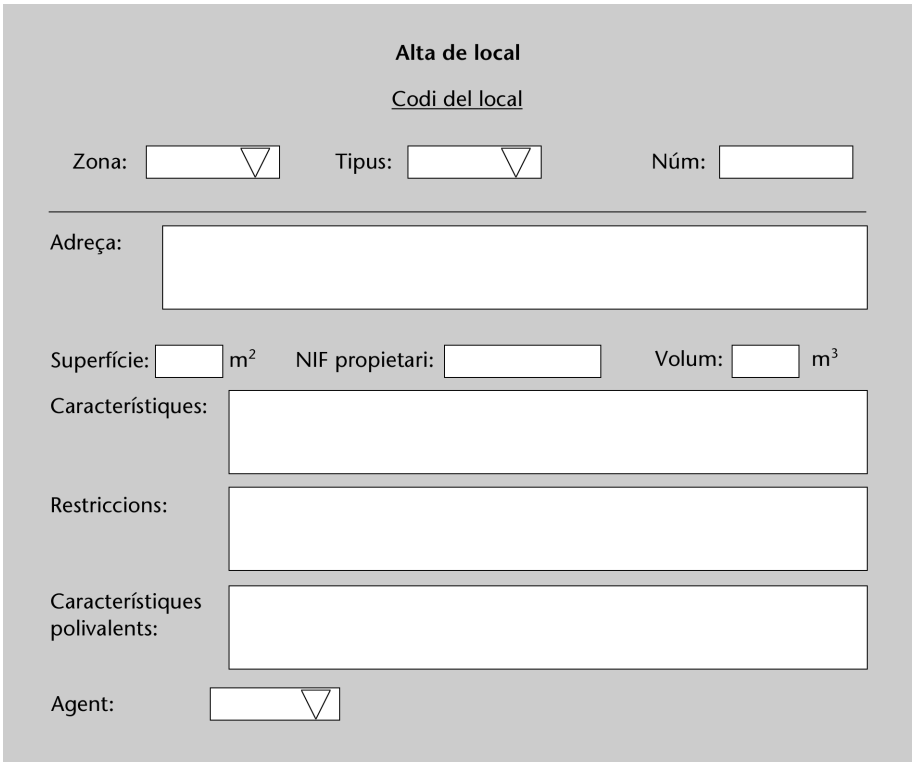
Activitat

7.2. Feu els diagrames de seqüències del resta dels casos d'ús.

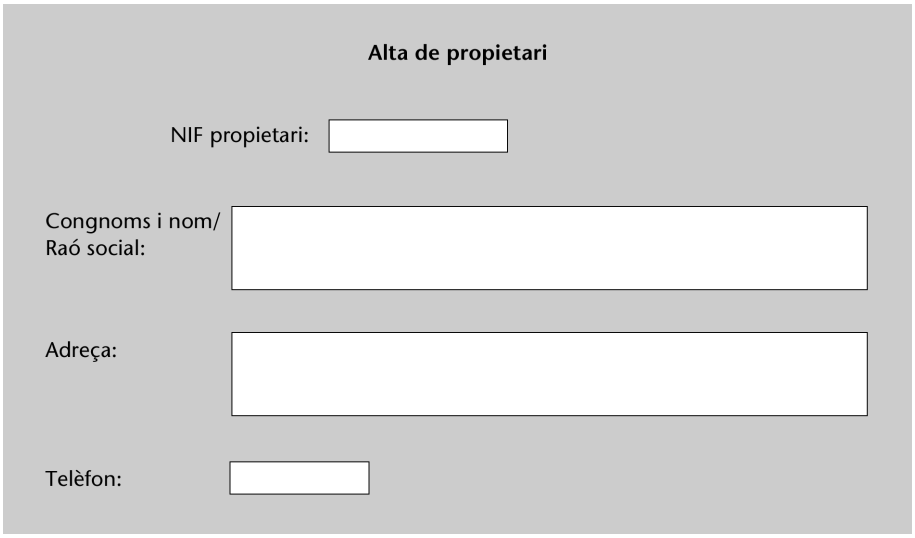
7.8. Anàlisi de la interfície d'usuari

Vegem l'esquema de finestra corresponent a cada classe de frontera.

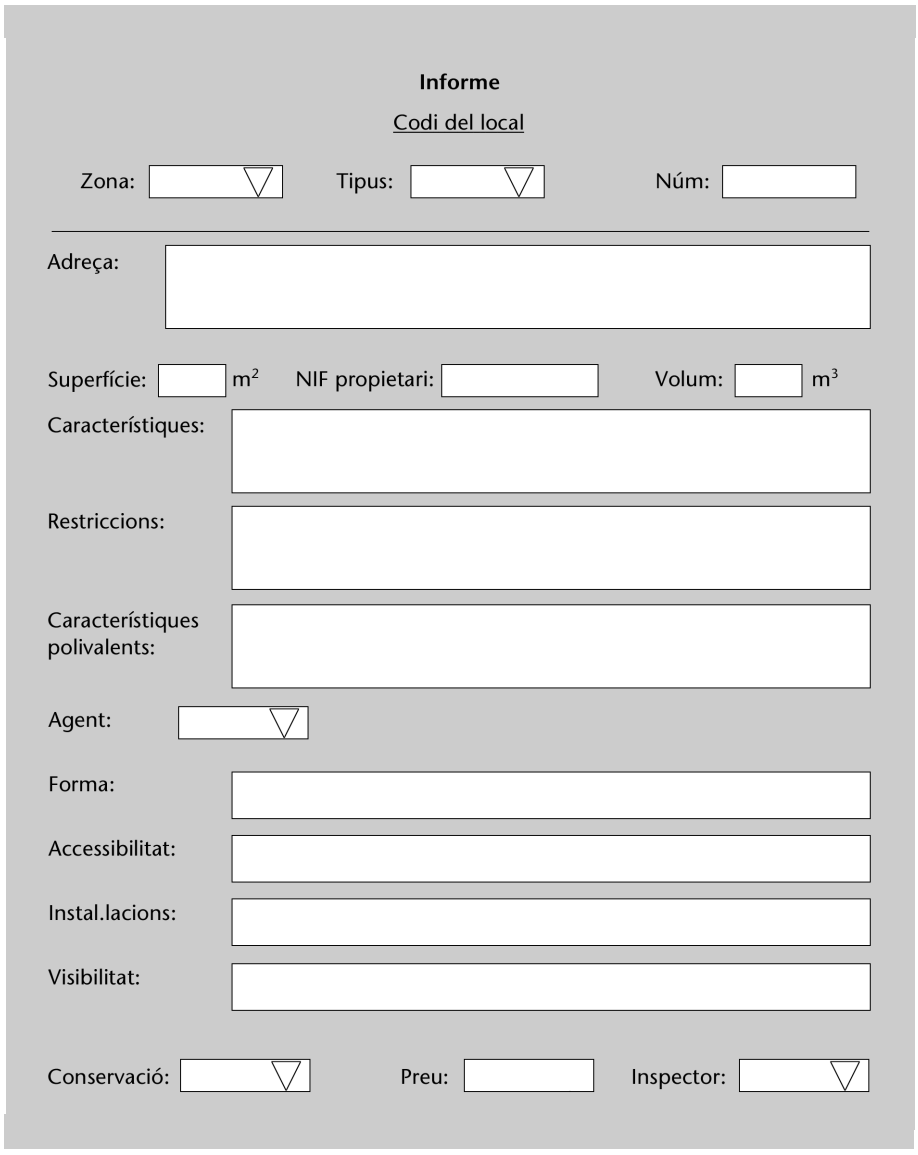
1) Classe *PantallaAfegirLocal*



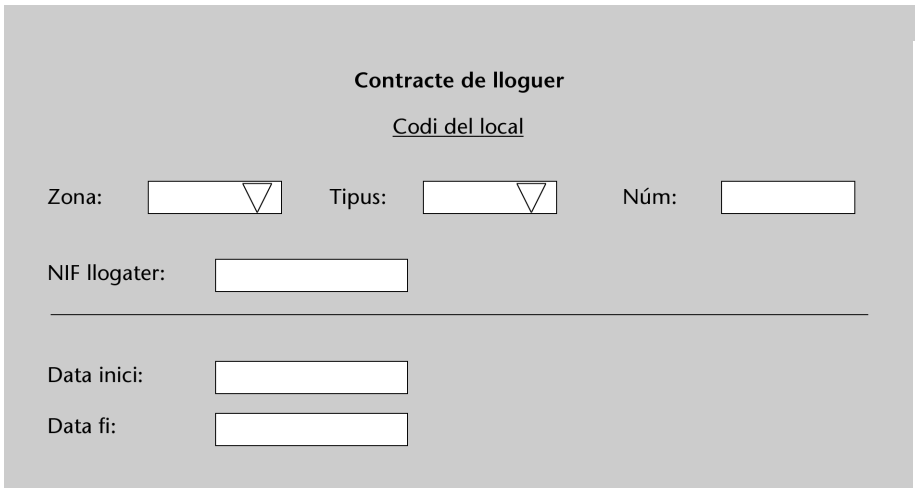
2) Classe *PantallaAfegirPropietari*



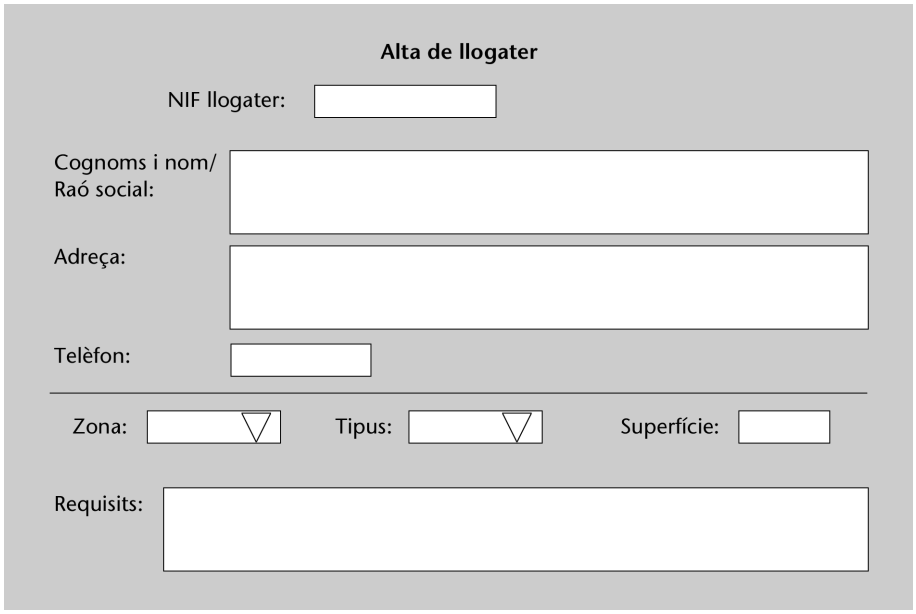
3) Classe *PantallaIntroduirInforme*



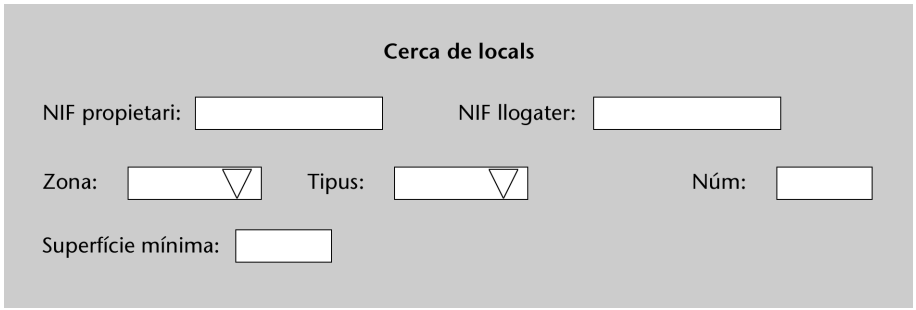
4) Classe *PantallaIntroduirContracte*



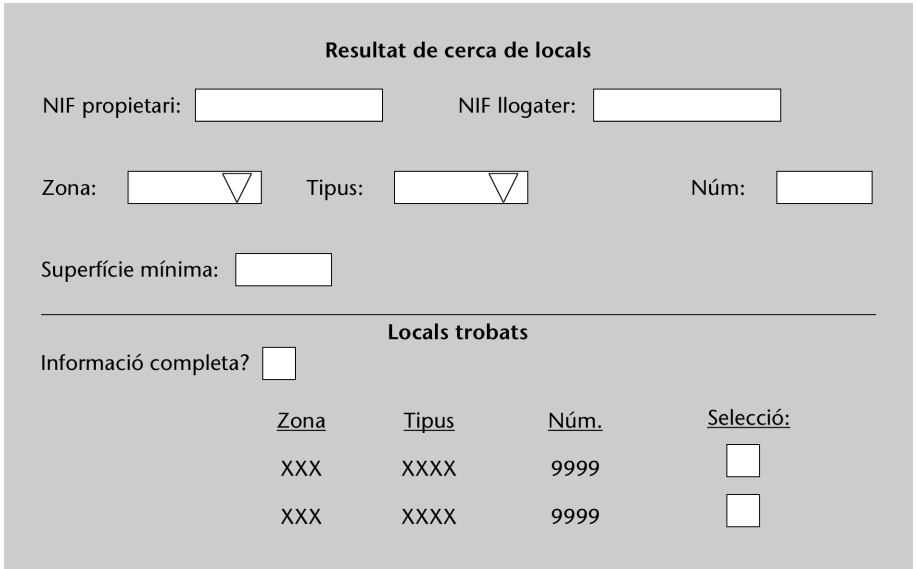
5) Classe *PantallaAfegirLlogater*



6) Classe *PantallaBuscarLocals*



7) Classe *PantallaLlistaLocals*



A la capçalera...

… surten els valors dels arguments de la cerca donats per l'usuari i, a sota, la llista dels codis dels locals que compleixen les condicions corresponents. "Informació completa?" serveix per a indicar si l'usuari vol que apareguin també l'adreça del local i el NIF del propietari. "Selecció" serveix per a demanar les dades d'un local concret.

8) Classe *PantallaDadesLocal*

L'esquema de la finestra és el mateix que per a la classe *PantallaIntroduirInforme* més una opció per a imprimir-ne el contingut.

Resum

Hem vist com es duu a terme l'etapa d'anàlisi segons un mètode basat en el Rational Unified Process. L'objectiu d'aquesta etapa és modelar els requisits d'una manera adient per a servir de base del desenvolupament pròpiament dit del programari.

Es comença per revisar els casos d'ús especificats a l'etapa anterior amb vista a eliminar redundàncies i afegir eventuals casos d'ús no demanats explícitament pels usuaris però tanmateix necessaris per a satisfer els requisits.

Després s'identifiquen les classes d'entitats, llurs atributs i les relacions entre elles, i a continuació es fa un diagrama de col·laboració simplificat per a cada cas d'ús amb vista a identificar les classes de frontera, de control i les operacions d'aquestes i de les classes d'entitats, acabant d'obtenir així la informació necessària per a elaborar el diagrama estàtic d'anàlisi.

Després es fan diagrames d'interacció més detallats per a aquells casos d'ús que calgui, i eventualment es fan també diagrames d'activitats, d'estats i transicions. L'anàlisi de la interfície d'usuari consisteix a descriure el contingut de les finestres associades a les classes de frontera, partint de la llista d'aquestes i de la descripció de les tasques futures. Així s'ha obtingut el que hem anomenat **model de l'anàlisi**.

Activitats

1. Feu la documentació d'anàlisi per al cas de l'assegurança d'automòbils, partint de la documentació sobre els requisits obtinguda com a activitat en el mòdul anterior.

Vegeu l'activitat del mòdul "Recollida i documentació de requisits" d'aquesta assignatura.

Exercicis d'autoavaluació

1. Per què cal revisar els casos d'ús descrits en la documentació de requisits en començar l'anàlisi?

2. A què correspon cada classe de frontera?

3. Què voldria dir que en un cas d'ús hi hagués una classe de frontera i una classe d'entitat i cap classe de control?

4. Com es representaria l'herència múltiple mitjançant agregacions?

Solucionari

Exercicis d'autoavaluació

1. Perquè, d'una banda, hi pot haver redundàncies o contradiccions, i de l'altra, pot ser que alguns requisits hagin quedat implícits.

Vegeu l'apartat 3 d'aquest mòdul didàctic.

2. Per a cada cas d'ús, hi ha almenys una classe de frontera per a cada actor que hi participa; o mirat d'una altra manera, per a cada paper de cada actor hi ha d'haver almenys una classe de frontera. N'hi pot haver més si entre l'actor i el sistema té lloc un diàleg amb diversos passos.

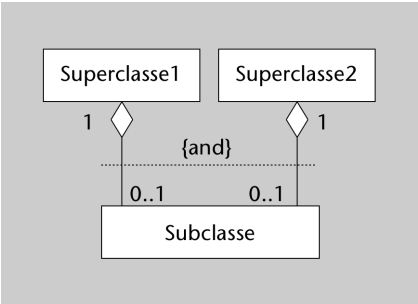
Vegeu el diagrama de col·laboració del cas d'ús 7 en el subapartat 7.6 i el subapartat 4.1 d'aquest mòdul didàctic.

3. Que l'únic que és fa és gravar a la base de dades les dades introduïdes per l'actor i/o presentar a l'actor les dades llegides de la base de dades

Vegeu el subapartat 4.1 d'aquest mòdul didàctic.

4. Posant una classe component que tingués relacions d'agregació amb les dues superclasses alhora; aquesta classe component tindria tots els atributs i operacions específics de la subclasse.

Vegeu el subapartat 4.4.3 d'aquest mòdul didàctic.



Glossari

classe de control
Classe no persistent que implementa tots els algorismes principals d'un cas d'ús o part d'ells.

classe d'entitats
Classe del domini del programari.

classe de frontera
Classe que implementa una part de la interfície d'usuari a nivell d'anàlisi.

diagrama estàtic d'anàlisi
Diagrama estàtic obtingut durant l'etapa d'anàlisi que comprèn les classes d'entitats, de control i de frontera amb llurs atributs, operacions i relacions.

Bibliografia

Booch, G.; Rumbaugh, J.; Jacobson, I. (1999). *UML. El lenguaje unificado de modelado. Guía del usuario*. Addison-Wesley.

Jacobson, I. (1994). *Object-Oriented Software Engineering*: *A Use Case Driven Approach*. Addison-Wesley.

Jacobson, I; Booch, G.; Rumbaugh, J. (2000). *El proceso unificado de desarrollo de software*. Addison-Wesley.

Larman, C. (1998). *Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design*. Upper Saddle River: Prentice-Hall.

Meyer, B. (1997). *Object-Oriented Software Construction* (2a. edició). Prentice-Hall.

Richter, Ch. (1999). *Designing Flexible Object-Oriented Systems with UML*. Indianapolis: MacMillan.

Rosenberg, D.; Scott, K. (1999). *Use Case Driven Object Modeling with UML: A Practical Approach*. Addison-Wesley.

Rumbaugh, J.; Jacobson, I.; Booch, G. (2000). *UML. El lenguaje de modelado unificado. Manual de referencia*. Addison-Wesley.

Weinschenk, S.; Jamar, P.; Yeo, S.C. (1997). *GUI Design Essentials*. John Wiley & Sons.