Invocació explícita versus invocació implícita: Anàlisi comparativa de dos enfocaments de disseny de Sistemes d'Informació.

Cristina Gómez[1] Luis Felipe Fernández[1] (*) Juan Ramón López[2] Antoni Olivé[1]

(1) LSI - Facultat d'Informàtica - Universitat Politécnica de Catalunya
C/Jordi Girona Salgado, 1-3, Edif. C6. 08034 Barcelona - Catalunya.
{ luisf, cristina, olive }@lsi.upc.es

(2) Dpto. Computación - Facultade de Informática - Universidade da Coruña
Campus de Elviña s/n 15071 A Coruña - Galicia
mon@rosalia.dc.fi.udc.es

* Becario de la Universidad Autónoma de Ciudad Juárez (UACJ) con fondos PROMEP. (lfernand@uacj.mx)

Resum

L’objectiu principal d’aquest treball és desenvolupar una caracterització inicial d’un enfocament de disseny de sistemes d’informació, que anomenarem *d’invocació implícita*, i analitzar els seus avantatges i inconvenients potencials en relació a l’enfocament convencional, que anomenem *d’invocació explícita*. El nostre treball pretén contribuir a aquest enfocament a l’àmbit específic del disseny orientat a objectes de sistemes d’informació. Al nostre anàlisi usem un exemple consistent en algunes operacions, simplificades, del sector bancari. Concretament, és una extensió de l’estàndard conegut al món de las bases de dades com Debit-Credit [Ano85], que utilitzem com a base de comparació pel disseny alternatiu que explorem. Es caracteritza el que podria ser la invocació implícita en el disseny de sistemes d’informació, apliquem aquesta caracterització a l’exemple, i es presenta una anàlisi comparativa dels dos dissenys.

Índex

1. Introducció.

L'objectiu principal d'aquest report és el de desenvolupar una caracterització inicial d'un enfocament de disseny de sistemes d'informació, que anomenem d'invocació implícita, i analitzar-ne els seus avantatges i inconvenients potencials en relació a l'enfocament convencional, que anomenem d'invocació explícita.

L'enfocament d'invocació implícita no és nou en el disseny de sistemes software. Des de fa temps, de quan en quan apareixen treballs, usant –com acostuma a passar- noms diversos, que l'usen en algun sistema concret o que l'avaluen. D'entre aquests treballs mencionem [Sul92], que l'anomena invocació implícita/mitjancer, i que l'usa per al disseny i construcció d'entorns integrats d'eines software; [Sha96] que l'anomena invocació implícita, integració reactiva o difusió selectiva, i fa una proposta detallada de com es podria incorporar en els llenguatges de programació d'ús general; i [Lar98] que l'anomena disseny basat en notificació d'esdeveniments, i en descriu algunes de les seves qualitats distintives. La invocació implícita també està present en alguns patrons de disseny populars, com ara l'Observador [Gam95], el Model-Vista-Controlador o l'Agent [Bus96], i el de Notificació d'esdeveniments [Rie96].

La idea de la invocació implícita s'utilitza també, a vegades, fora de l'àmbit del disseny de sistemes. Es pot considerar semblant a les regles causa-efecte de ROSES [Cos97], que és un llenguatge per a la modelització conceptual de sistemes d'informació. També es pot fer correspondre a les regles de les bases de dades actives [Wid96], on l'ocurrència d'un esdeveniment invoca, implícitament, accions definides en regles.

El nostre treball preten contribuir a aquest enfocament en l'àmbit específic del disseny orientat a objectes de sistemes d'informació. Pretenem caracteritzar la invocació implícita en aquest àmbit concret, i analitzar-ne els avantatges i inconvenients en relació al disseny convencional dels sistemes d'informació.

L'objectiu de fons del nostre treball és millorar el disseny de sistemes d'informació. En principi, aquesta millora podria intentar-se en diverses propietats d'un disseny, però nosaltres volem centrar-nos en la millora de la propietat no-funcional anomenada "canviabilitat", que inclou els aspectes de mantenibilitat, extensibilitat, reestructuració i portabilitat [Bus96]. Considerem que aquesta propietat és crítica per a la majoria dels sistemes d'informació i que encara calen més esforços per a millorar-la. En aquest sentit, volem explorar fins a quin punt un ús enraonat de la invocació implícita podria augmentar la canviabilitat del software.

Aquest report s'estructura en sis seccions, que giren al voltant d'un exemple concret. En la secció 2 fem una breu descripció d'aquest exemple. En la secció 3 presentem un disseny convencional, usant la invocació explícita, del sistema. Usarem aquest disseny com a base de comparació per al disseny alternatiu que explorem. A la secció 4 tractem de caracteritzar el que podria ser la invocació implícita en el disseny de sistemes d'informació. A la secció 5 apliquem aquesta caracterització a l'exemple. En la secció 6 fem una anàlisi comparativa dels dos dissenys obtinguts. Les conclusions del nostre treball, i les línies de recerca futures, les exposem en la secció 7.

2. Presentació de l'exemple.

L'exemple que usarem per a la nostra anàlisi consisteix en algunes operacions, simplificades, del sector bancari. Concretament, és una extensió de l'estàndard conegut en el món de les bases de dades com a Debit-Credit [Ano85]. En aquesta secció en farem una breu presentació general. Els detalls els anirem explicant en la secció següent, conjuntament amb el primer disseny que fem del sistema.

Suposem un banc que disposa d'un conjunt d'oficines repartides per un cert territori. Els clients del banc tenen comptes oberts en les oficines. Hi ha dues menes de comptes: d'estalvi i corrent. La diferència principal entre un i altre està en el càlcul dels interessos, i en les operacions que s'hi poden fer, però en aquest exemple no aprofundirem gaire en les diferències. Un compte corrent pot estar vinculat a un compte d'estalvi.

Cada oficina té un conjunt de caixers automàtics, en els quals poden operar els clients amb els seus comptes (ingressos i reintegraments). Els clients poden fer operacions en les dues menes de comptes. Si es fa un reintegrament d'un compte corrent, i no hi ha prou saldo i el compte està associat a un compte d'estalvi amb saldo suficient, es fa un traspàs automàtic d'un compte a l'altre, per l'import necessari.

Els reintegraments es poden fer sempre que en el caixer on es demanen hi hagi diners suficients. Tenen un efecte immediat sobre el saldo dels comptes corresponents. Els ingressos, en canvi, s'accepten provisionalment, fins que són confirmats per un operador. Un cop confirmats, s'incrementa el saldo amb la quantitat ingressada. Tots els moviments dels comptes s'han de guardar en un arxiu històric.

Tots els comptes estan assignats a una oficina. Per cada oficina es manté un atribut saldo, que té un valor igual a la suma dels saldos dels comptes que té assignats.

3. Disseny del sistema: Invocació explícita.

Com és ben conegut, es poden fer diversos dissenys diferents d'un mateix sistema amb un mateix estil de disseny, i per a un mateix entorn d'execució. Nosaltres estem interessats en l'estil orientat a objectes i, per als nostres propòsits, aquí en tenim prou amb suposar un entorn d'execució centralitzat, ignorant els aspectes de concurrència i persistència.

Per fer el disseny, usarem el mètode descrit a [Mar95], tant perquè és prou representatiu dels mètodes existents, com perquè es basa en la consideració dels esdeveniments externs (també anomenats transaccions) com a objectes (normalment transitoris), aspecte aquest que creiem molt rellevant en els sistemes d'informació. Mètodes que es poden considerar semblants a aquest són els descrits a [Coa95, Lar98]. Com a notació, usarem el llenguatge UML.

La Figura 3.1 mostra el diagrama de classes del sistema. No s'inclouen els tipus dels atributs ni els paràmetres de les operacions. La classe *Compte* és abstracta, i defineix els atributs i les operacions comuns a les dues subclasses. L'operació *saldo?():Integer* es pot usar per conèixer el saldo d'un compte. L'operació *ingressar(i:Integer)* s'usarà per comunicar al sistema que s'ha produït un ingrés al compte corresponent, per un import *i*. L'operació *reintegrar(i:Integer)* també es defineix en Compte.

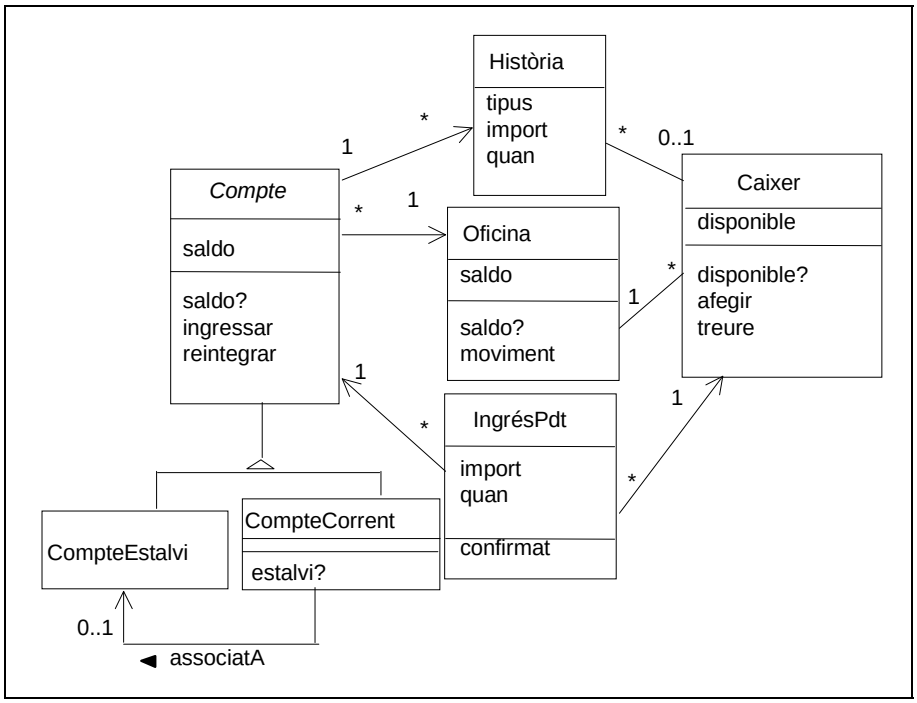

Figura 3.1: Diagrama de classes d'objectes

Com hem indicat, en aquest disseny els esdeveniments externs es consideren com a objectes, de les classes corresponents. Definim una classe abstracta esdeveniment extern (*EsdevExt*), que només té definida una operació virtual *executar*(), amb tres subclasses, també abstractes, per diferenciar els esdeveniments externs que són de consulta (*EsdExtCons*), d'actualització (*EsdExtAct*) i de petició d'acció (*EsdExtPetAcc*). L'operació *executar()* s'ha de definir en cada esdeveniment extern: el seu mètode ha de realitzar les accions corresponents. En la Figura 3.2 mostrem alguns d'aquests esdeveniments externs.

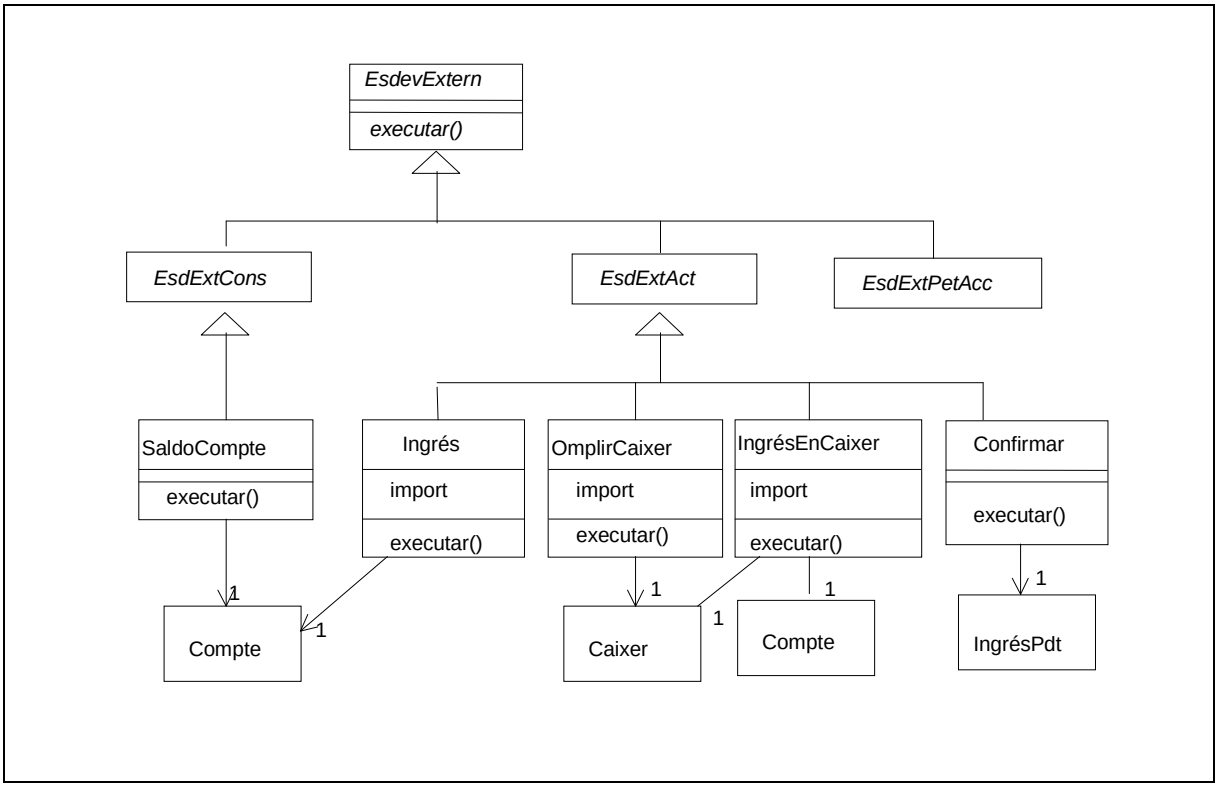

Figura 3.2: Diagrama de classes d'esdeveniments

L'esdeveniment extern *SaldoCompte* té només un paràmetre, que és el compte que es vol consultar. El diagrama de seqüència de la Figura 3.3 mostra l'efecte de la seva execució: és simplement invocar l'operació *saldo?()* del compte.

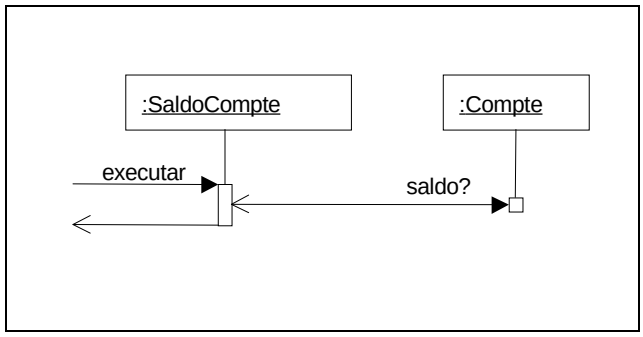

Figura 3.3: Diagrama seqüència SaldoCompte

L'esdeveniment extern *OmplirCaixer* té dos paràmetres: el caixer al qual s'afegeix diners i la quantitat que s'afegeix. La seva execució invoca l'operació *afegir(i:Integer)*, que tindrà l'efecte de sumar la quantitat *i* a l'atribut *disponible* del caixer. La Figura 3.4 en mostra el diagrama de seqüència.

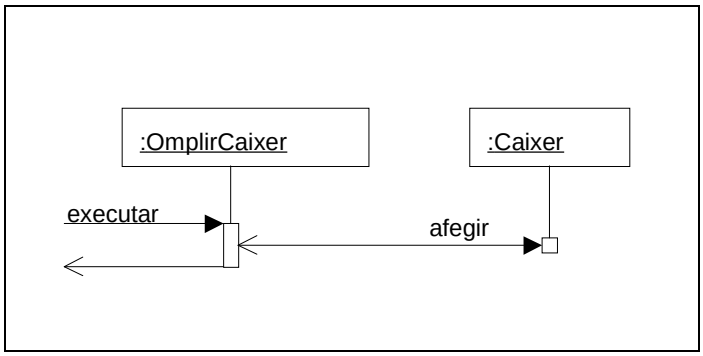

Figura 3.4: Diagrama seqüència OmplirCaixer

Malgrat que aquests dos esdeveniments externs són molt simples, ja es pot veure l'essència del disseny amb invocació explícita: l'operació *executar* ha de conèixer les operacions existents dels altres objectes, i invocar explícitament aquelles que produiran l'efecte desitjat. En el cas de *SaldoCompte*, l'efecte d'*executar()* ha de ser retornar el saldo del compte, i això ho pot fer *saldo?()*. En el cas de *OmplirCaixer*, l'efecte d'*executar()* ha de ser que el caixer incrementi el valor de *disponible*. Qui ho pot fer és l'operació *afegir(i:Import)*, per tant s'ha d'invocar explícitament aquesta operació per tal que ho faci.

L'esdeveniment extern *IngrésEnCaixer* (que es produeix quan un client introdueix diners en un caixer, per ser ingressats en un cert compte) té tres paràmetres: El compte al qual es vol ingressar, el caixer on es fa l'ingrés, i l'import. L'ingrés no té un efecte immediat sobre el compte, sinó que requereix una confirmació posterior. Per aquest motiu, l'únic que cal fer es guardar l'ingrés en un objecte de la classe *IngrésPdt*. La Figura 3.5 mostra el diagrama de seqüència.

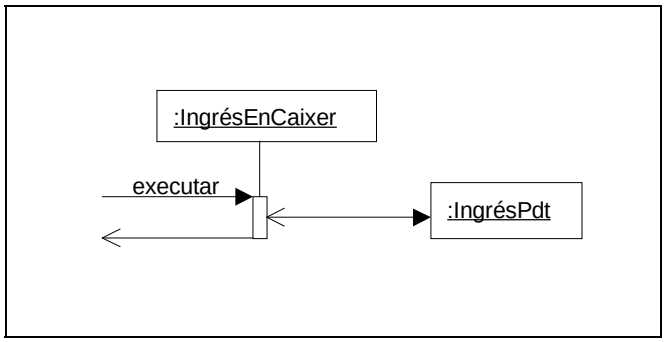

Figura 3.5: Diagrama seqüència IngrésEnCaixer

Per confirmar un ingrés, un operador del banc usa l'esdeveniment extern *Confirmar*. La Figura 3.6 en mostra el diagrama de seqüència. L'execució de l'esdeveniment extern invoca l'operació *confirmat*, la qual ha de garantir que el saldo del compte corresponent s'incrementi, així com el de l'oficina a la qual està assignat i, alhora, que s'enregistri l'ingrés en la història de moviments. L'operació *ingressar(i:Integer)* de *compte* proporciona tota aquesta funcionalitat. Li cal invocar l'operació *moviment(i:Integer)* d'*oficina* i crear un nou objecte de la classe *Història*.

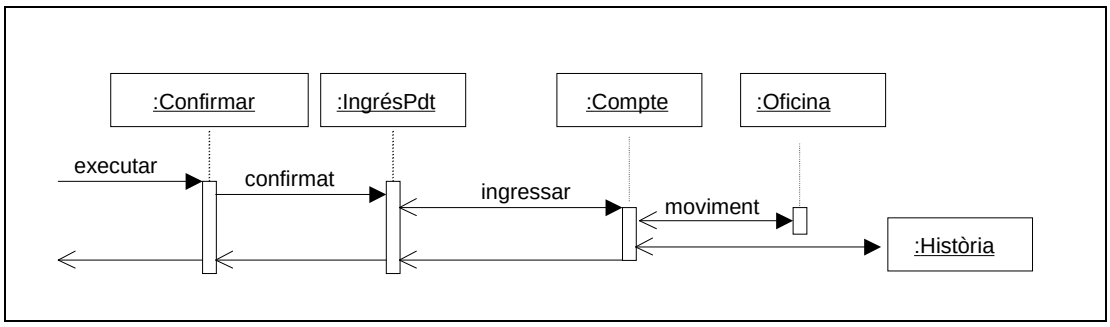

Figura 3.6: Diagrama seqüència Confirmar

També es poden fer ingressos en comptes directament (transferències, etc.). Per aquest propòsit s'usa l'esdeveniment extern *Ingrés*, amb l'efecte documentat a la Figura 3.7. L'execució d'aquest esdeveniment ha d'implicar que el saldo del compte corresponent s'incrementi, així com el de l'oficina a la qual està assignat i, alhora, que s'enregistri l'ingrés en la història de moviments. Com hem vist abans (Figura 3.6) l'operació *ingressar(i:Integer)* de compte proporciona tota aquesta funcionalitat.

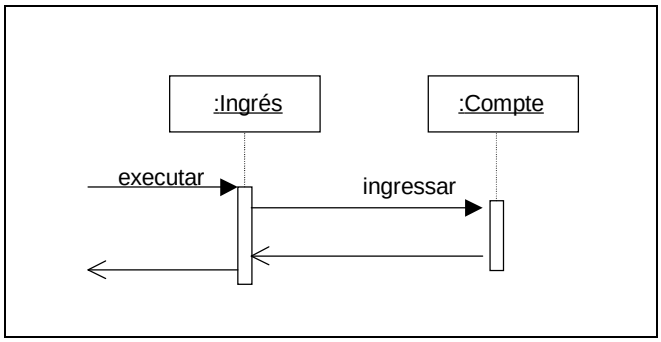

Figura 3.7: Diagrama seqüència Ingrés

Queda per definir els dos esdeveniments externs de reintegrament. La Figura 3.8 descriu la seva estructura.

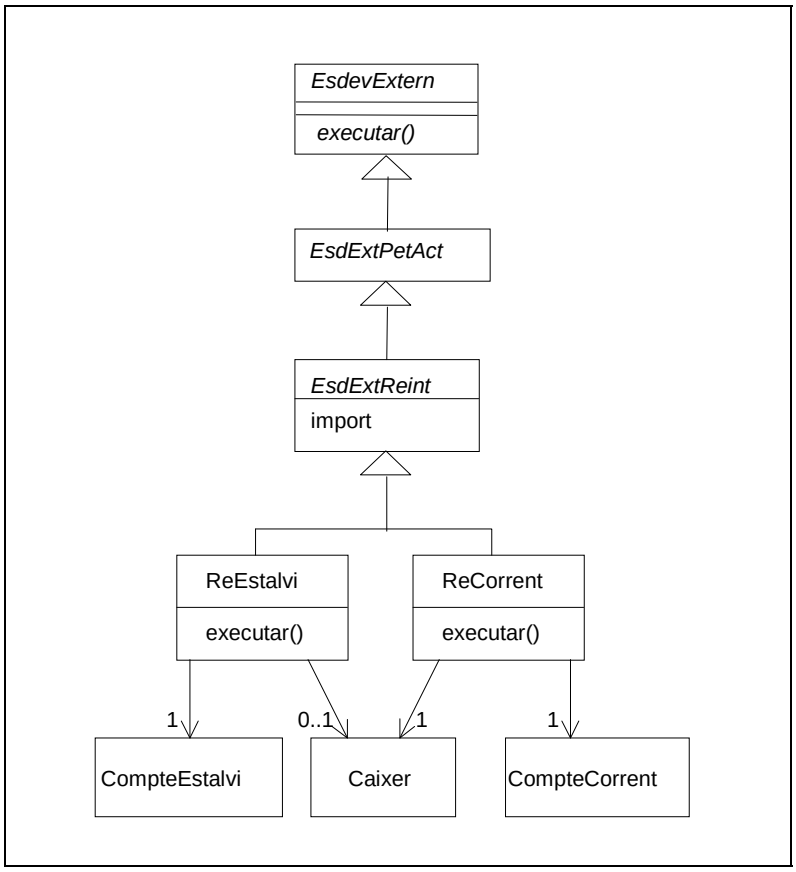

Figura 3.8: Diagrama de classes d'esdeveniments externs

L'esdeveniment extern ReEstalvi extreu diners d'un compte d'estalvi, sempre i quan hi hagi saldo suficient. L'esdeveniment es pot fer, o no, des d'un caixer. En el primer cas, cal que el caixer tingui prou diners disponibles. La Figura 3.9 ens mostra el diagrama de seqüència, per l'escenari de reintegrament des d'un caixer on hi ha diners al caixer i saldo suficient. Observi's que s'invoca l'operació *reintegrar(i:Import)*, definida a nivell compte.

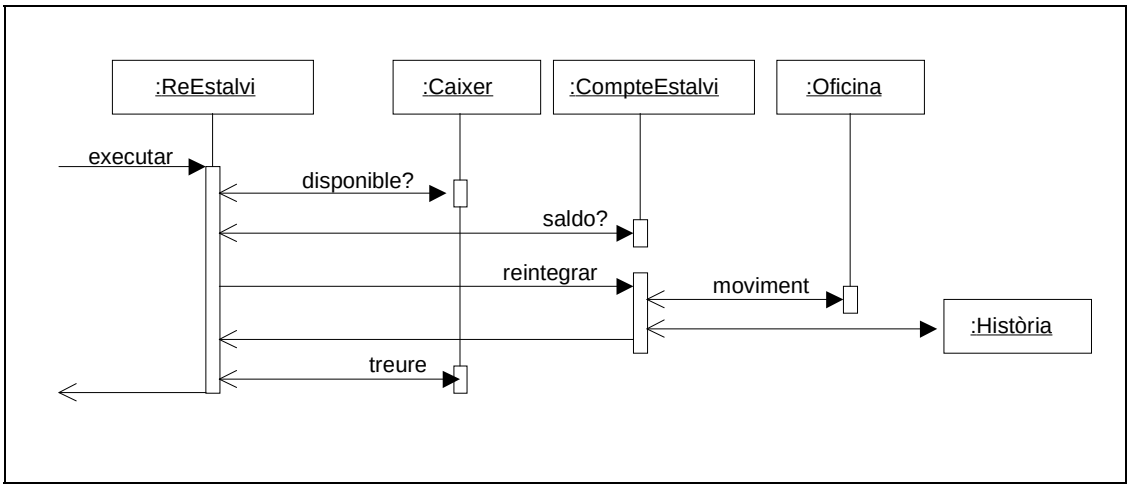

Figura 3.9: Diagrama seqüència ReEstalvi

L'esdeveniment extern ReCorrent és una mica més complex, perquè si no hi ha saldo suficient en el compte corrent, però hi ha un compte d'estalvi associat, amb el saldo mínim necessari, es pot fer un traspàs d'aquest a aquell. La Figura 3.10 mostra el diagrama de seqüència d'aquest esdeveniment extern per l'escenari:

- Hi ha diners suficients al Caixer.
- No hi ha saldo suficient al compte corrent.
- Hi ha compte d'estalvi associat.
- Es pot fer un traspàs des del compte d'estalvi associat.

El diagrama mostra que es genera un esdeveniment extern de reintegrament de compte d'estalvi, s'executa, es genera un altre esdeveniment extern d'ingrés en compte corrent (per l'import extret en l'anterior) i s'invoca l'operació *reintegrar* (l'efecte de la qual s'ha descrit en la figura anterior).

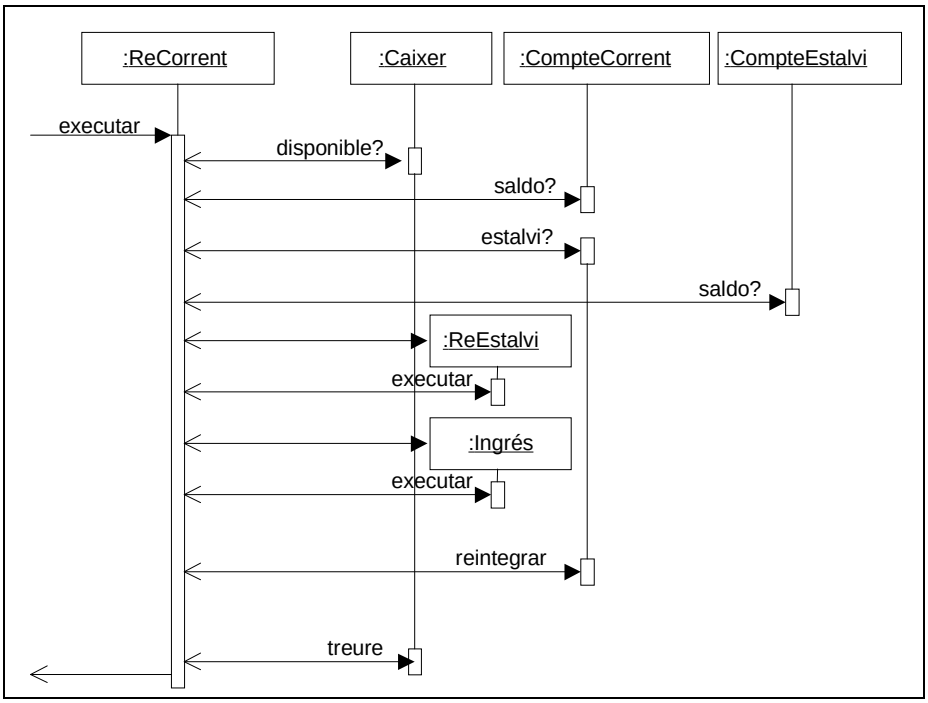

Figura 3.10: Diagrama seqüència ReCorrent

4. Caracterització de la invocació implícita en sistemes d'informació.

Per desenvolupar una caracterització inicial de la invocació implícita es necessari tenir una idea clara del que entenem per invocació implícita.

La idea fonamental de la invocació implícita és eliminar la interacció entre objectes substituint la invocació directa per l'anunci d'un o més *esdeveniments*. Quan es produeix una ocurrència d'un esdeveniment i el seu anunci per part d'un objecte, altres objectes interessats en aquest esdeveniment reaccionen.

Aquesta idea d'invocació implícita, al nostre treball, implica l'existència d'un component mitjançant el qual es produirà la comunicació entre els objectes. Aquest component l'anomenarem **Gestor d'Esdeveniments** (GE).

Una vegada establerta l'existència del GE, definirem tres **premisses fonamentals**:

> **1.** Els objectes anuncien al GE l'ocurrència d'un o més esdeveniments. Un objecte, en comptes d'invocar explícitament una operació d'altre objecte, anuncia al GE l'ocurrència d'un esdeveniment.
>
> **2.** Els objectes comuniquen al GE el seu interès per un determinat esdeveniment, associant com a mínim una de les seves operacions.
>
> **3.** El GE accepta la comunicació de l'interès d'un objecte per un esdeveniment i les operacions associades. A més a més, rep la comunicació d'una ocurrència d'un esdeveniment. El GE invoca les operacions associades a l'esdeveniment.

La Figura 4.1 mostra gràficament els conceptes explicats anteriorment.

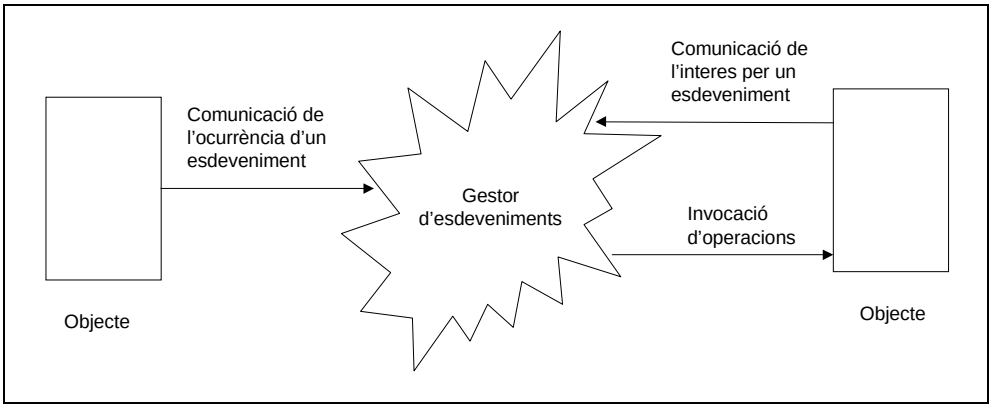

Figura 4.1: Esquema general de funcionament

Hem definit ja els conceptes fonamentals que utilitzarem a la invocació implícita. A continuació descriurem amb més detall aquests conceptes.

4.1 Esdeveniments

A la nostra caracterització considerem un nou tipus d'esdeveniment, a part dels esdeveniments externs:

Esdeveniments generats: són successos o accions que es produeixen internament al sistema en resposta a altres esdeveniments (externs o generats). Els esdeveniments generats poden ser de dos tipus: de *consulta* i *d'actualització*.

	Esdeveniments Externs		
	Consulta	**Actualització**	**Petició d'acció**
Esdeveniments generats	Consulta	Actualització	Consulta/ Actualització

Taula 4.1: Correspondència entre tipus d'esdeveniments

En aquest treball els esdeveniments són classes d'objectes i les ocurrències d'un esdeveniment són instàncies de les classes. La Figura 4.2 mostra l'estructura general dels tipus

d’esdeveniments existents.

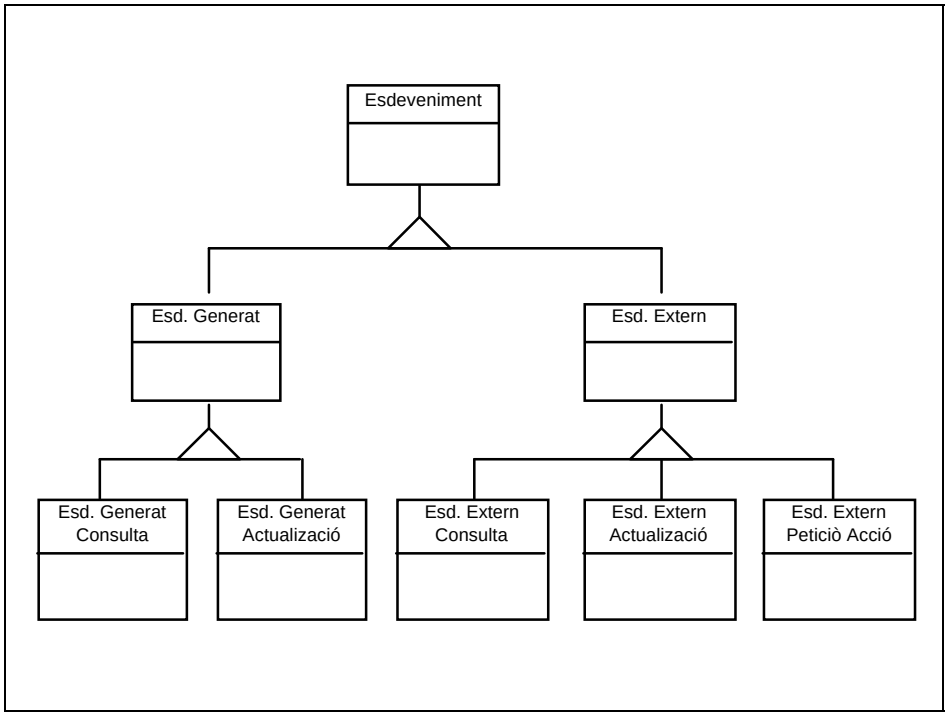

Figura 4.2: Diagrama de classes d’esdeveniments

4.2 Anunci d’esdeveniments

Els objectes comuniquen al GE l’ocurrència d’un o més esdeveniments. Aquesta comunicació es produeix via creació d’una instància d’esdeveniment generat i invocació d’una operació del GE que anomenarem *anunci*.

Considerarem dos tipus d’anuncis diferents:

1. Anunci síncron: Aquest anunci es refereix únicament als esdeveniments generats de consulta. L’objecte que genera l’esdeveniment invoca l’operació d’anunci de forma síncrona, per tant s’ha d’esperar al resultat. La sintaxi de l’operació és la següent:

Sintaxi:

anunciSíncron(Esdeveniment e):any

on:

e: és l’esdeveniment generat que s’anuncia.

any: és el tipus del resultat que retorna l’anunci.

2. Anunci asíncron: Aquest tipus d’anunci pot ser utilitzat tant pel cas dels esdeveniments generats de consulta com pels d’actualització. L’objecte que genera l’esdeveniment invoca l’operació d’anunci de forma asíncrona, per tant no s’espera al resultat i continua amb el seu processament.

- Si l'esdeveniment generat que s'anuncia és de consulta, s'obtindrà un resultat. En aquesta modalitat d'anunci, es possible enviar el resultat a un objecte qualsevol, especificant-lo mitjançant els arguments de l'operació.

  Sintaxi:

  anunciAsíncron(Esdeveniment e, Objecte o, Operació op)

  on:

  e: és l'esdeveniment generat que s'anuncia.

  o: és l'objecte on es rep la resposta.

  op: operació que s'invoca a l'objecte *o* per comunicar el resultat.

- Si l'esdeveniment generat és d'actualització, en aquest cas no s'espera resposta, per tant l'operació d'anunci només necessita com argument l'esdeveniment a comunicar.

anunciAsíncron(Esdeveniment e)

La Figura 4.3 mostra gràficament com es fa la comunicació d'una ocurrència d'esdeveniment.

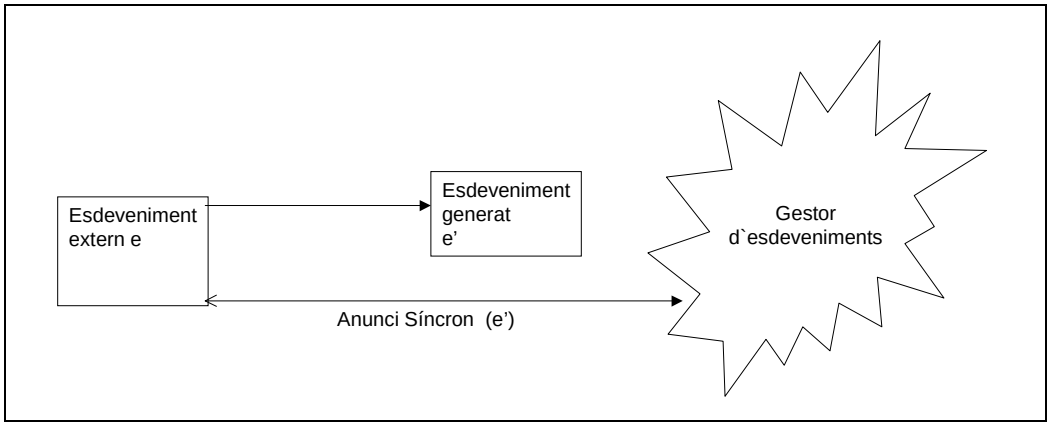

Figura 4.3: Comunicació d'esdeveniment

Els esdeveniments poden tenir associats arguments. Aquests arguments els representarem com atributs de la classe.

4.3 Subscripcions

Els objectes comuniquen al GE el seu interès per un determinat esdeveniment, associant com a mínim una de les seves operacions. A aquesta acció l'anomenarem *subscripció*: Un objecte es subscriu a un esdeveniment amb una o més operacions.

Considerarem dos tipus de subscripcions diferents:

1. *Subscripcions genèriques:* són subscripcions que es refereixen a classes i no a instàncies concretes d'objectes. Les subscripcions genèriques fan referència a totes les instàncies d'una classe. És en el moment de fer l'anunci quan es coneix, a partir dels arguments de l'esdeveniment, a quina instància concreta fa referència la invocació de l'operació.

2. *Subscripcions particulars:* són subscripcions que es refereixen a una instància concreta d'una classe d'objectes, que subscriu una de les seves operacions.

Per les subscripcions proposem una estructura similar a les regles ECA de BD actives. Per tots els tipus de subscripcions definim la següent sintaxi:

```
when esdeveniment e
if condició
tellTo objecte.operació(llista_arguments)
where var1=expressió1
      var2=expressió2
      .........
      varM=expressióM  M≥0
```

Figura 4.4: Sintaxi d'una subscripció

on:

- La clàusula *when* permet especificar el tipus concret d'esdeveniment al qual fa referència la subscripció.
- La clàusula *if* permet especificar una condició que ha de ser certa perquè es faci la invocació de l'operació de l'objecte subscrit, quan es produeix l'esdeveniment. L'especificació d'aquesta condició serà opcional.
- La clàusula *tellTo* permet especificar quina operació de l'objecte ha de ser invocada i els seus arguments.
- La clàusula *where* permet relacionar variables i expressions. Aquestes expressions han de ser operacions que retornen un valor.

A qualsevol subscripció només es pot utilitzar la següent informació:

- La proporcionada per l'esdeveniment mitjançant els seus atributs.
- La proporcionada per operacions de consulta aplicada a aquests atributs.
- La proporcionada per operacions de consulta aplicada a l'objecte que es subscriu.
- Les variables definides al *where*.

Per simplificar, quan diverses subscripcions fan referència al mateix esdeveniment i la mateixa condició proposem aquesta variant de la sintaxi.

```
when esdeveniment e
if condició
tellTo objecte1.operació1(llista1_arguments)
tellTo objecte2.operació2(llista2_arguments)
        .....
tellTo objecteN.operacióN(llistaN_arguments)  N≥1
where var1=expressió1
        var2=expressió2
        .....
        varM=expressióM  M≥0
```

Figura 4.5: Sintaxi abreviada d'una subscripció

4.4 Gestor d'esdeveniments

Com ja hem dit anteriorment el GE és el component que ens permetrà substituir les invocacions explícites per les invocacions implícites. Fins ara hem definit el concepte d'anunci i subscripció. Ara explicarem aquests conceptes des del punt de vista del gestor.

- El GE accepta la subscripció d'una operació d'un objecte a un esdeveniment. El GE pot conèixer subscripcions des d'un començament i també pot acceptar noves subscripcions a mesura que siguin comunicades.

- El GE rep l'anunci de l'ocurrència d'un determinat esdeveniment.

- El GE decideix quan ha de gestionar l'anunci d'un esdeveniment determinat d'acord amb una política prèviament definida. El dissenyador *pot suposar* que els esdeveniments es processen *en l'ordre en que es reben* i que un esdeveniment es processarà *completament* abans que qualsevol altre esdeveniment anunciat amb posterioritat.

  El primer que farà el GE serà localitzar totes les subscripcions associades a l'esdeveniment i per cada subscripció comprova en el cas d'existir, si es compleix la seva condició. Han de ser invocades totes les operacions per les quals la condició associada sigui certa. L'ordre d'invocació és indeterminat.

- Les invocacions es realitzen a operacions que poden ser de dos tipus:

  1. *De consulta:* aquestes operacions retornen un resultat. Quan el GE decideix invocar (tellTo) una operació d'un objecte i aquesta operació és de consulta, ho farà de forma síncrona. El GE espera el resultat, el rep i el torna a l'objecte receptor del resultat, que serà el que ha generat l'esdeveniment en el cas de l'anunci síncron o el que s'especifica a l'operació d'anunci en el cas de l'anunci asíncron. Per aquesta raó només es permet associar una única operació de consulta a cada esdeveniment generat.

  2. *D'actualització:* aquestes operacions no retornen resultat. Quan el GE decideix invocar (tellTo) una operació d'un objecte i aquesta operació és d'actualització, el GE no espera resultat i per tant continua amb el seu processament.

La Figura 4.6 mostra gràficament l'anunci d'un esdeveniment, la subscripció a un esdeveniment i la invocació de l'operació o operacions associades a aquest esdeveniment.

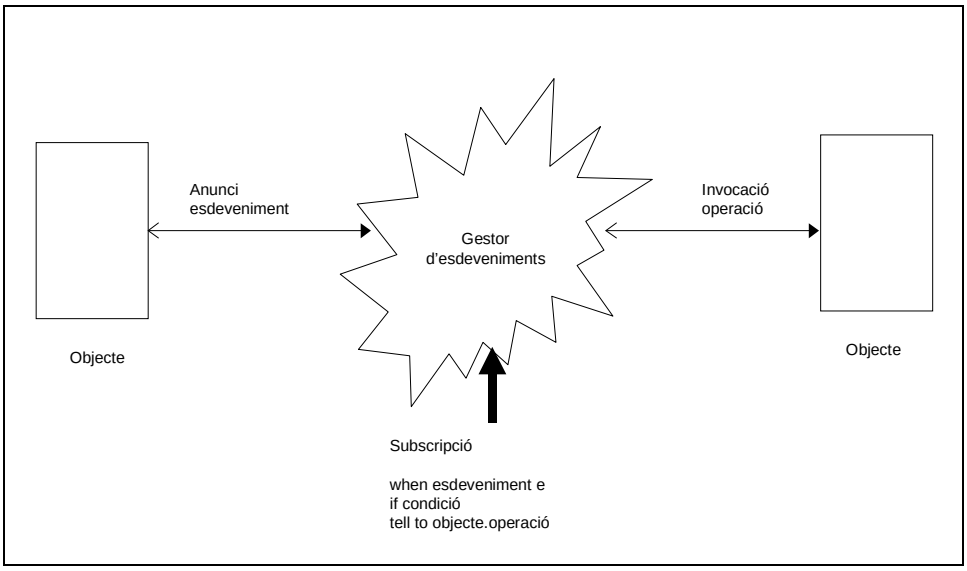

Figura 4.6: Esquema general des del punt de vista del GE

5. Disseny del sistema: Invocació implícita.

En aquest punt mostrarem el disseny amb invocació implícita de l'exemple descrit al punt 3. Utilitzarem en aquest disseny el GE i les tres premisses fonamentals definides al punt 4. Veurem així que és possible construir un disseny on no hi ha invocacions explícites entre objectes.

Per fer aquest nou disseny definim un nou diagrama de classes semblant a l'anterior però amb algunes diferències. La Figura 5.1 mostra el diagrama de classes del sistema:

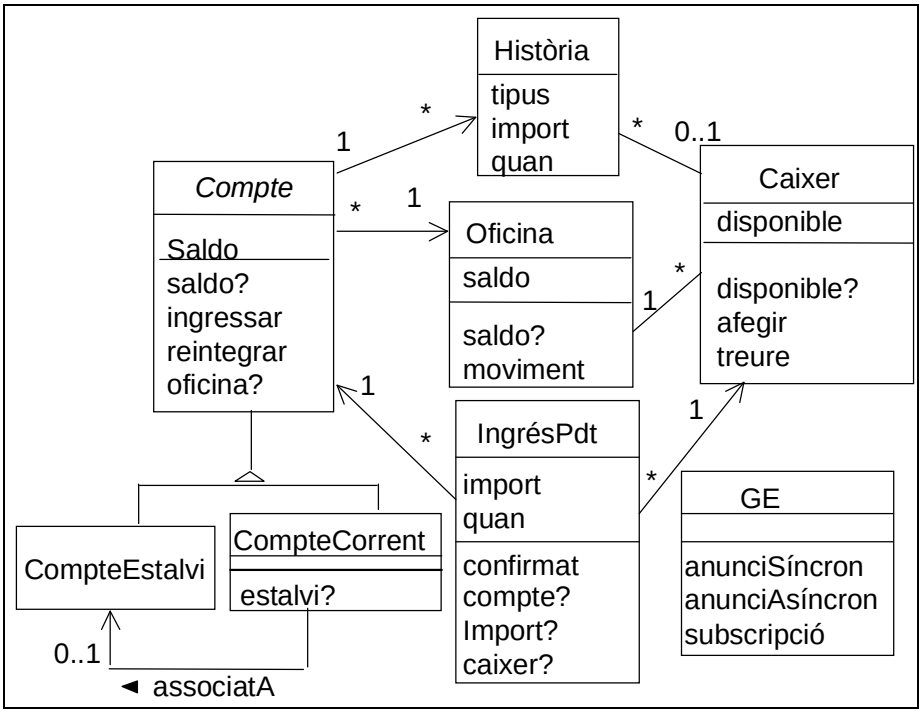

Figura 5.1: Diagrama de classes d'objectes

Com es pot veure en aquesta figura, les diferències fonamentals amb el diagrama de classes de

la Figura 3.1 són les noves operacions de consulta que s'han d'afegir a alguns objectes per ser utilitzades a la clàusula where d'algunes subscripcions. S'afegeix també una classe per al GE amb operacions de anunci i subscripció.

A les Figura 5.2, Figura 5.3 i Figura 5.4 es mostra el diagrama de classes d'esdeveniments.

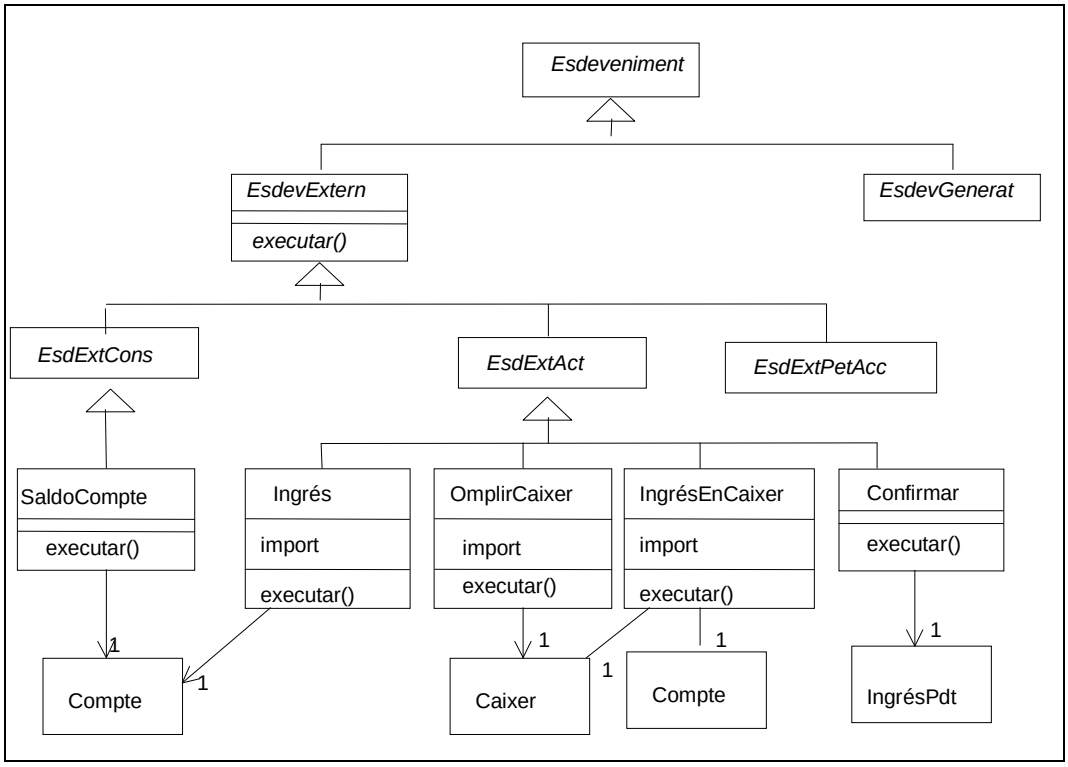

Figura 5.2: Diagrama de classes d'esdeveniments

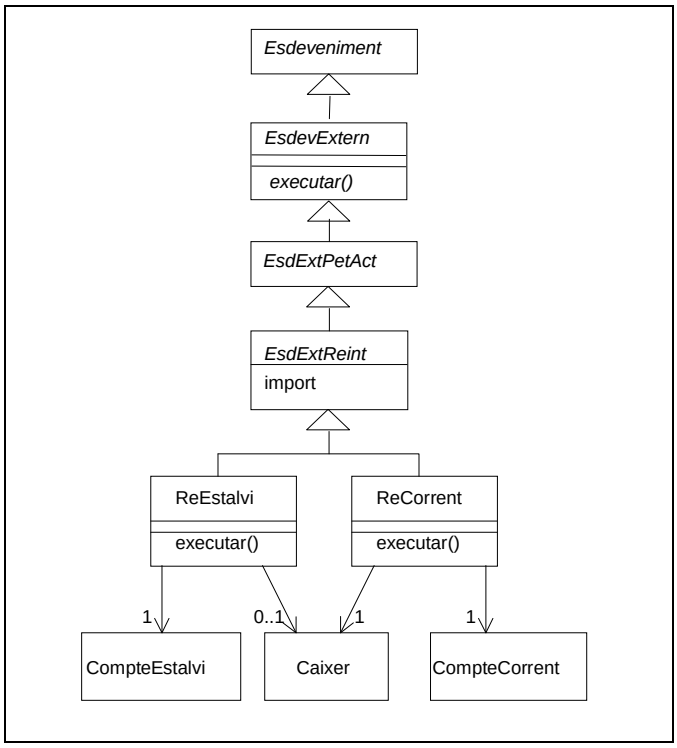

Figura 5.3: Diagrama de classes d'esdeveniments externs

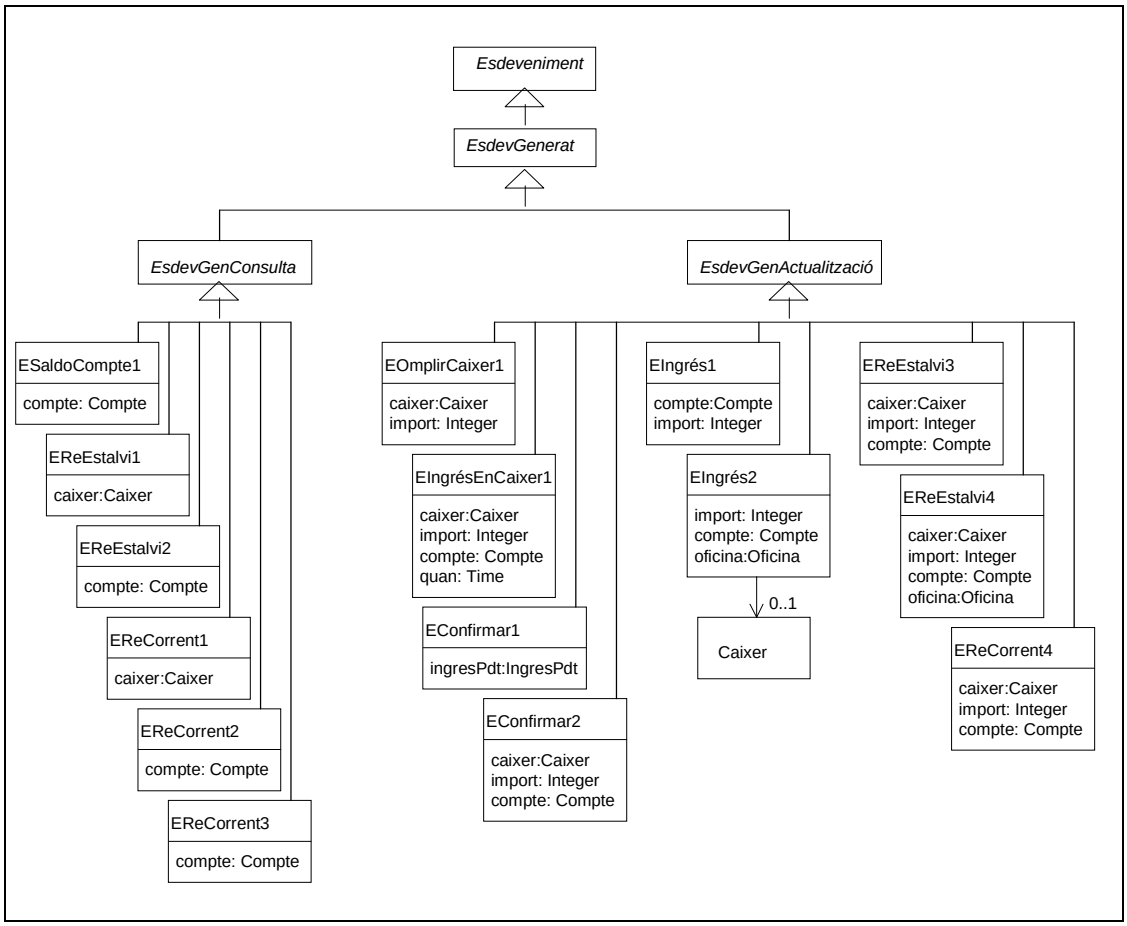

Figura 5.4: Diagrama de classes d'esdeveniments generats

Com es pot veure a la figura anterior, afegim al diagrama de classes d'esdeveniments una nova classe abstracta que anomenarem *esdeveniment generat*. Aquesta classe tindrà dos subclasses també abstractes per representar els esdeveniments generats de consulta i actualització. Com a subclasses d'aquestes classes tindrem tots els esdeveniments que el nou disseny exigeix generar. Pel moment, proposem la següent notació pels esdeveniments generats:

E<*nom_esd_extern_que_el_genera*><*num_seqüència*>

A continuació mostrarem els diagrames de seqüència de tots els esdeveniments externs. Per claredat, agruparem els diagrames de seqüència segons el tipus d'esdeveniment extern (consulta, actualització i petició d'acció).

5.1 Esdeveniments externs de consulta.

Cada vegada que es produeix un esdeveniment extern de consulta, aquest donarà lloc a un esdeveniment generat de consulta i a un anunci al GE. Cada esdeveniment generat de consulta ha de tenir associat com a mínim una subscripció que impliqui una operació de consulta.

En el nostre disseny tenim un cas d'esdeveniment extern de consulta: *SaldoCompte*. El

diagrama de seqüència per aquest cas es mostra a continuació.

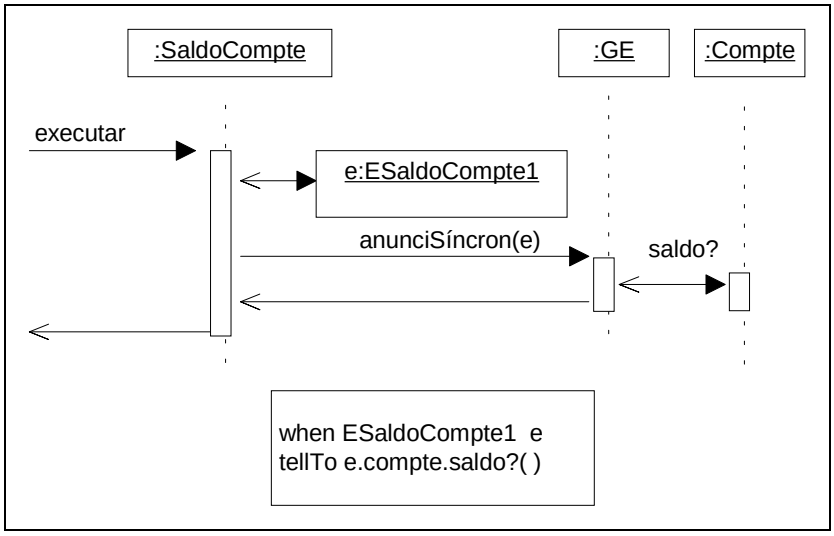

Figura 5.5: Diagrama de seqüència SaldoCompte

L'esdeveniment extern *SaldoCompte* dóna lloc a un esdeveniment generat *ESaldoCompte1* (d'acord amb la notació proposada), que tindrà com argument el compte. Aquest esdeveniment s'anuncia al GE mitjançant l'operació *anunciSíncron*. El GE invoca l'operació saldo sobre l'objecte compte. El GE coneix la subscripció que es mostra a la Figura 5.5.

5.2 Esdeveniments externs d'actualització

Cada vegada que es produeix un esdeveniment extern d'actualització, aquest donarà lloc a un esdeveniment generat d'actualització i a un anunci al GE. Cada esdeveniment generat d'actualització ha de tenir associat com a mínim una subscripció i aquesta pot implicar més d'una operació d'actualització.

En el nostre disseny tenim quatre esdeveniments externs d'actualització: *OmplirCaixer*, *IngrésEnCaixer*, *Confirmar*, *Ingrés*. El diagrama de seqüència per *OmplirCaixer* es mostra a continuació.

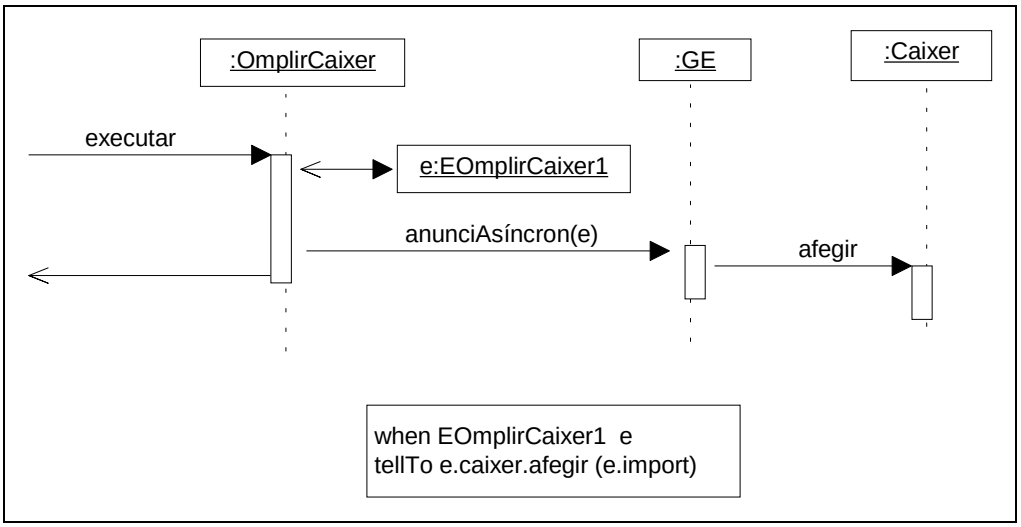

Figura 5.6: Diagrama de seqüència OmplirCaixer

En aquest cas es produeix un esdeveniment generat *EOmplirCaixer1* que té com a argument el caixer. Aquest esdeveniment s'anuncia al GE. El gestor invoca l'operació *afegir* sobre l'objecte *Caixer*. Cal remarcar que aquest esdeveniment extern no requereix resposta. Al igual que en el cas anterior el GE té una subscripció que es mostra a la Figura 5.6.

En el següent diagrama (*IngrésEnCaixer*) seguim el mateix procediment:

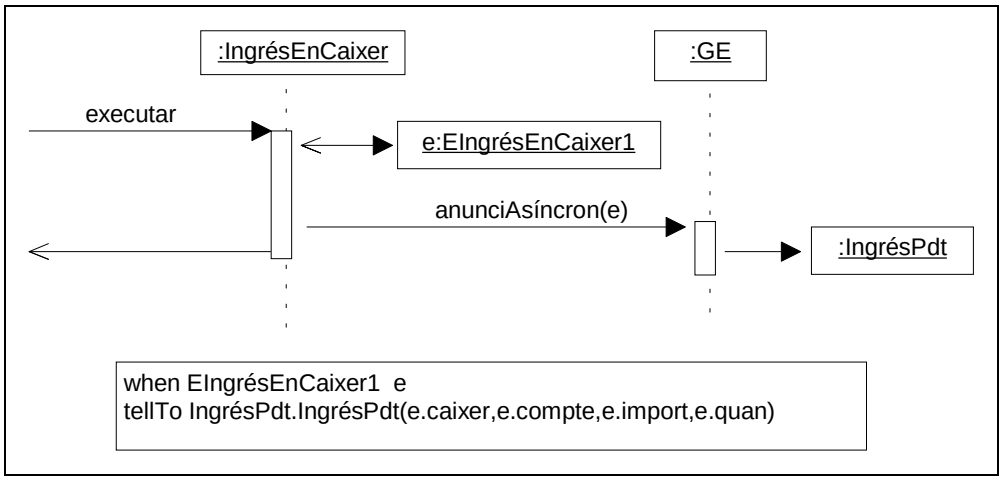

Figura 5.7: Diagrama de seqüència IngrésEnCaixer

El següent diagrama correspon a l'esdeveniment extern *Confirmar*:

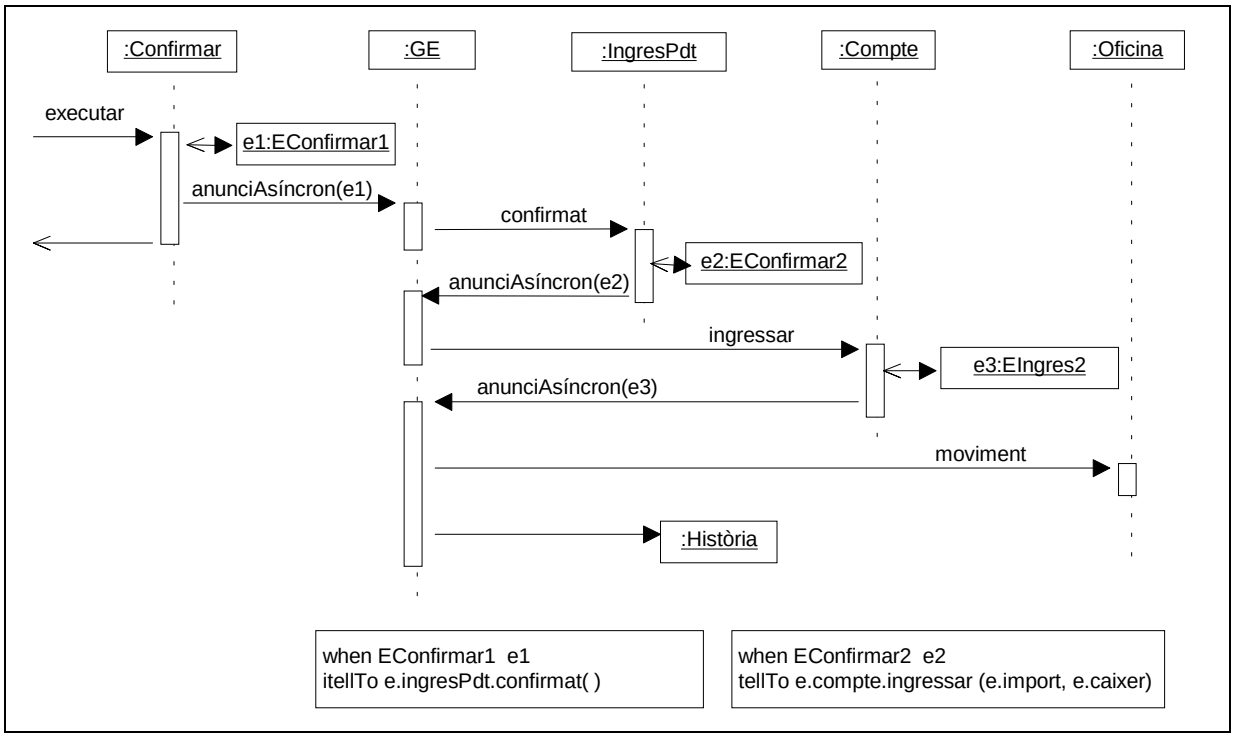

Figura 5.8: Diagrama de seqüència Confirmar (primera versió)

En aquest cas hem utilitzat el mateix procediment de disseny que als anteriors. Cal destacar que en aquest diagrama es produeix una situació nova: la invocació d'una operació per part del GE, *IngresPdt::confirmat*, dóna lloc a un esdeveniment generat d'actualització, *EConfirmar2*, ja que hi ha una certa dependència entre l'operació invocada i les operacions

associades al nou esdeveniment que es genera. El mateix succeeix amb l'operació *Compte::ingressar* i l'esdeveniment *EIngrés2*. Aquest esdeveniment no segueix la nomeclatura que hem proposat, ja que sempre que utilitzem l'operació ingressar es genera el mateix esdeveniment amb el mateix nom *EIngrés2* A la Figura 5.8 també podem veure, per primera vegada, una subscripció (l'associada a *EIngrés2*) on apareix la invocació de vàries operacions pel mateix esdeveniment.

Per una altra banda, en el cas de poder suposar una *independència* entre les operacions a invocar llavors tindríem la possibilitat d'associar *totes* les operacions a un mateix (i *únic*) esdeveniment generat d'actualització. Això ens donaria un disseny alternatiu tal i com es mostra a la Figura 5.9.

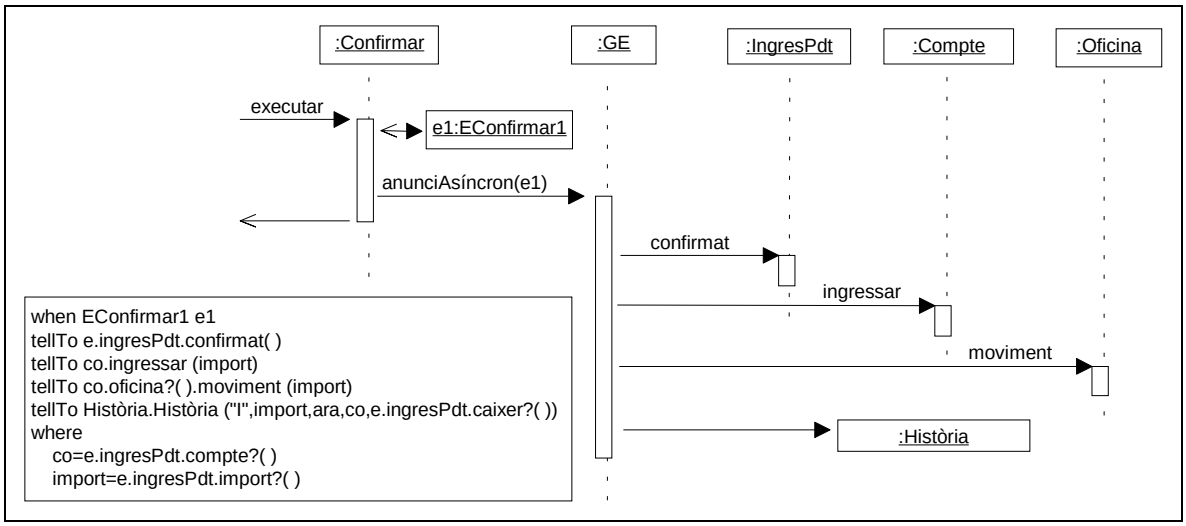

Figura 5.9: Diagrama seqüència Confirmar (segona versió)

A la subscripció de l'anterior figura, com a l'esdeveniment generat d'actualització no tenim tota la informació necessària per invocar les operacions, fem servir la clàusula where. En concret, a l'esdeveniment *EConfirmar1* només tenim la informació d'*IngrésPdt* i l'import. Per conèixer el compte i l'oficina associades, fem ús de les operacions de consulta (*IngrésPdt::compte?*, *Compte::oficina?*).

Existeix un últim esdeveniment extern d'actualització: *Ingrés*. Donat que es semblant als anteriors només mostrarem els diagrames corresponents a les dues alternatives de disseny possibles.

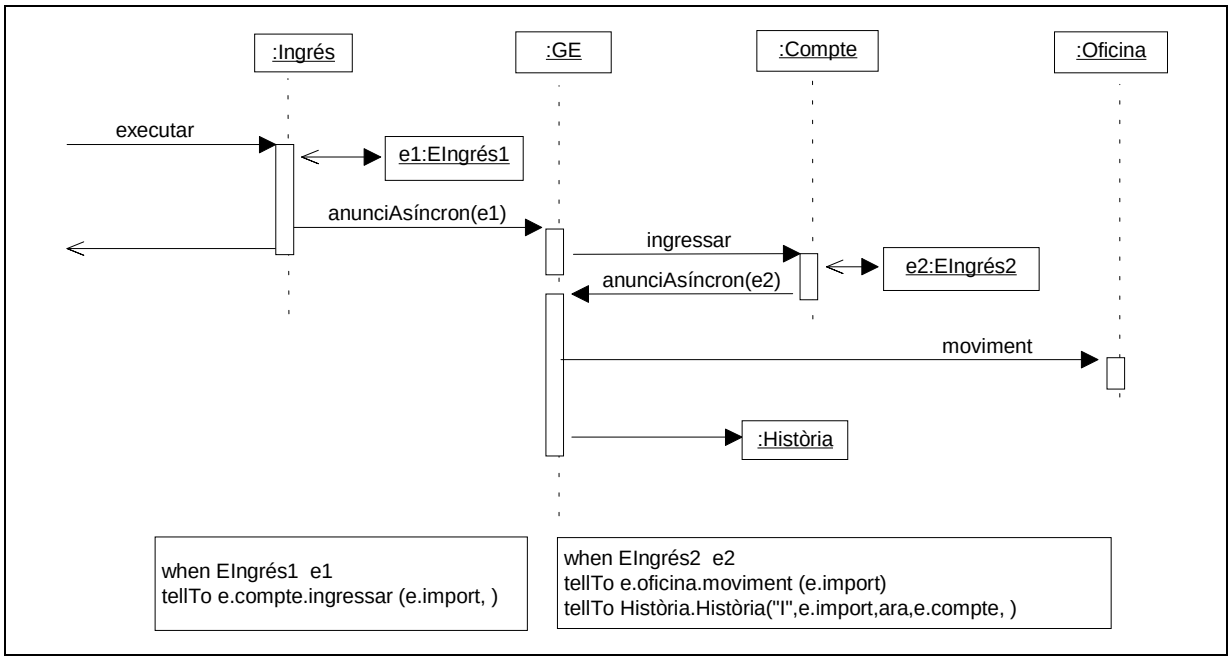

Figura 5.10: Diagrama seqüència Ingrés (primera versió)

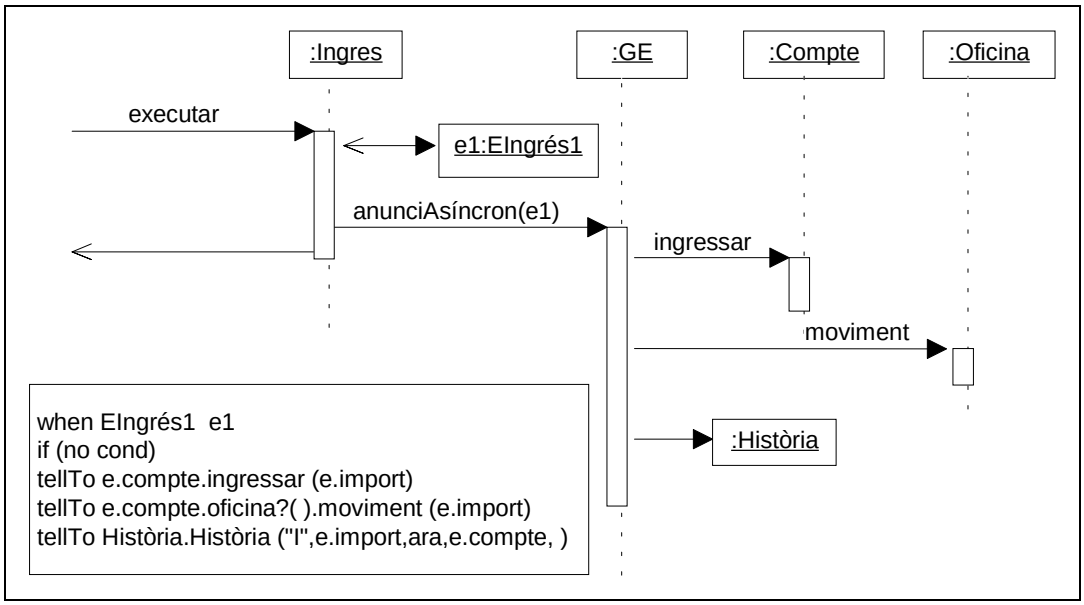

Figura 5.11: Diagrama seqüència Ingrés (segona versió)

5.3 Esdeveniments externs de petició d'acció

Cada vegada que es produeix un esdeveniment extern de petició d'acció, aquest donarà lloc a un o més esdeveniments generats de consulta i a un o més esdeveniments generats d'actualització. En el nostre exemple tenim dos esdeveniments externs de petició d'acció: *ReEstalvi* i *ReCorrent*. A la Figura 5.12 es mostra el diagrama de seqüència de *ReEstalvi*.

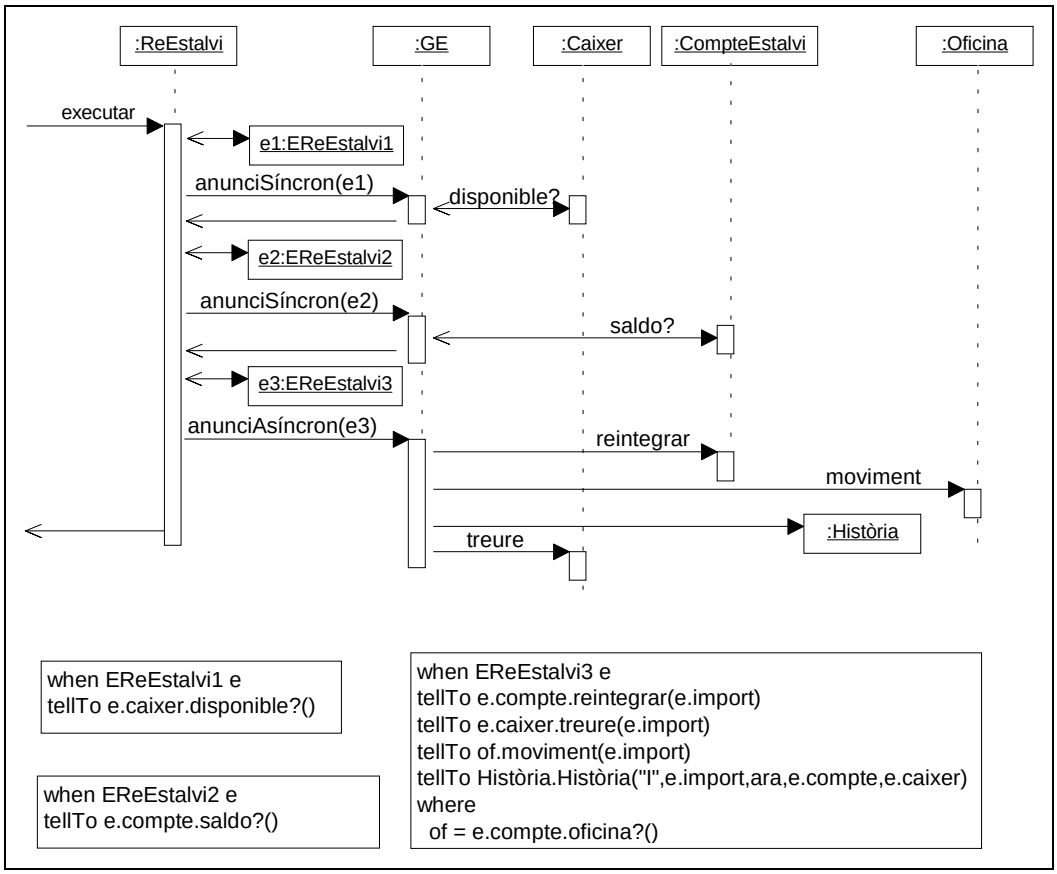

Figura 5.12: Diagrama seqüència ReEstalvi (cas d'independència)

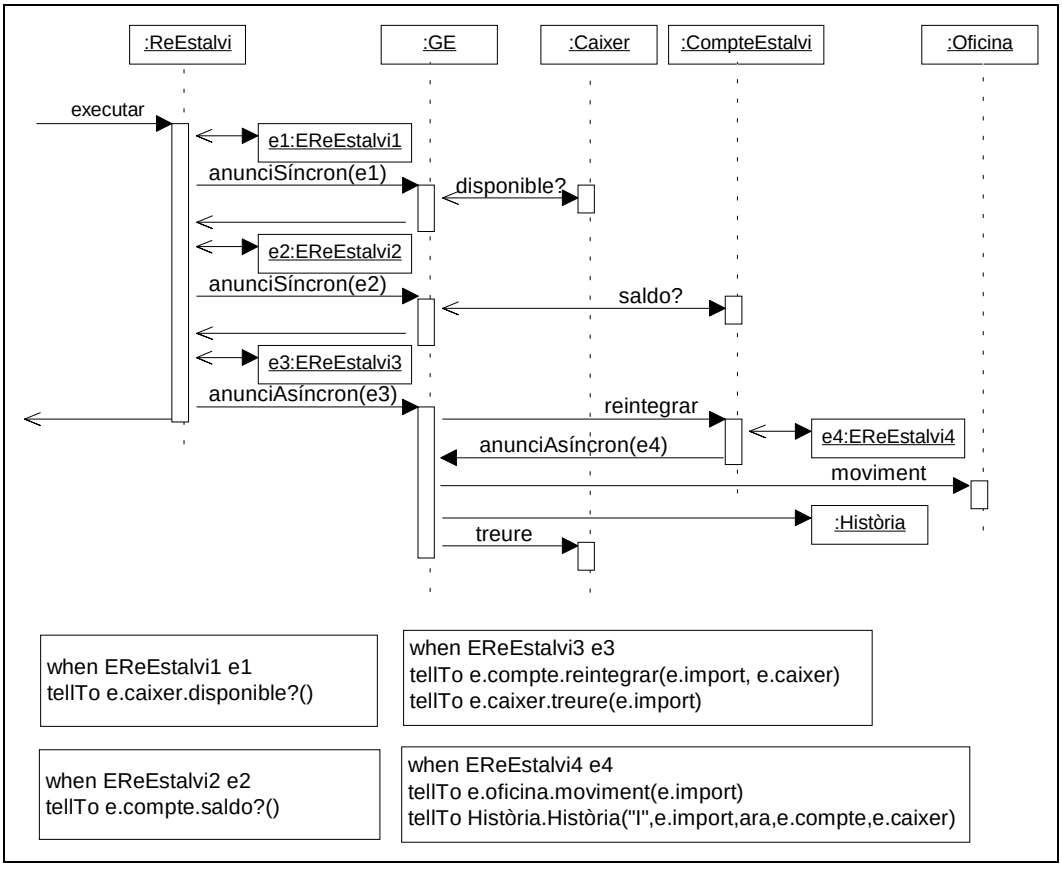

Figura 5.13: Diagrama seqüència ReEstalvi (cas de dependència)

El processament d'aquest esdeveniment extern exigeix l'avaluació d'una sèrie de condicions, a partir del resultat d'una sèrie de consultes. Per això es produeixen una sèrie d'esdeveniments generats de consulta. Una vegada que s'han avaluat les condicions s'hauran d'efectuar les actualitzacions corresponents. En aquest cas suposarem que totes les operacions a invocar són *independents* i per tant les associarem totes a un mateix esdeveniment generat d'actualització: *EReintegramentEstalvi3*.

Pel cas de depèndencia entre les operacions, la Figura 5.13 mostra el diagrama de seqüència de l'esdeveniment extern ReEstalvi.

La Figura 5.14 mostra el diagrama de seqüència de l'esdeveniment extern *ReCorrent*.

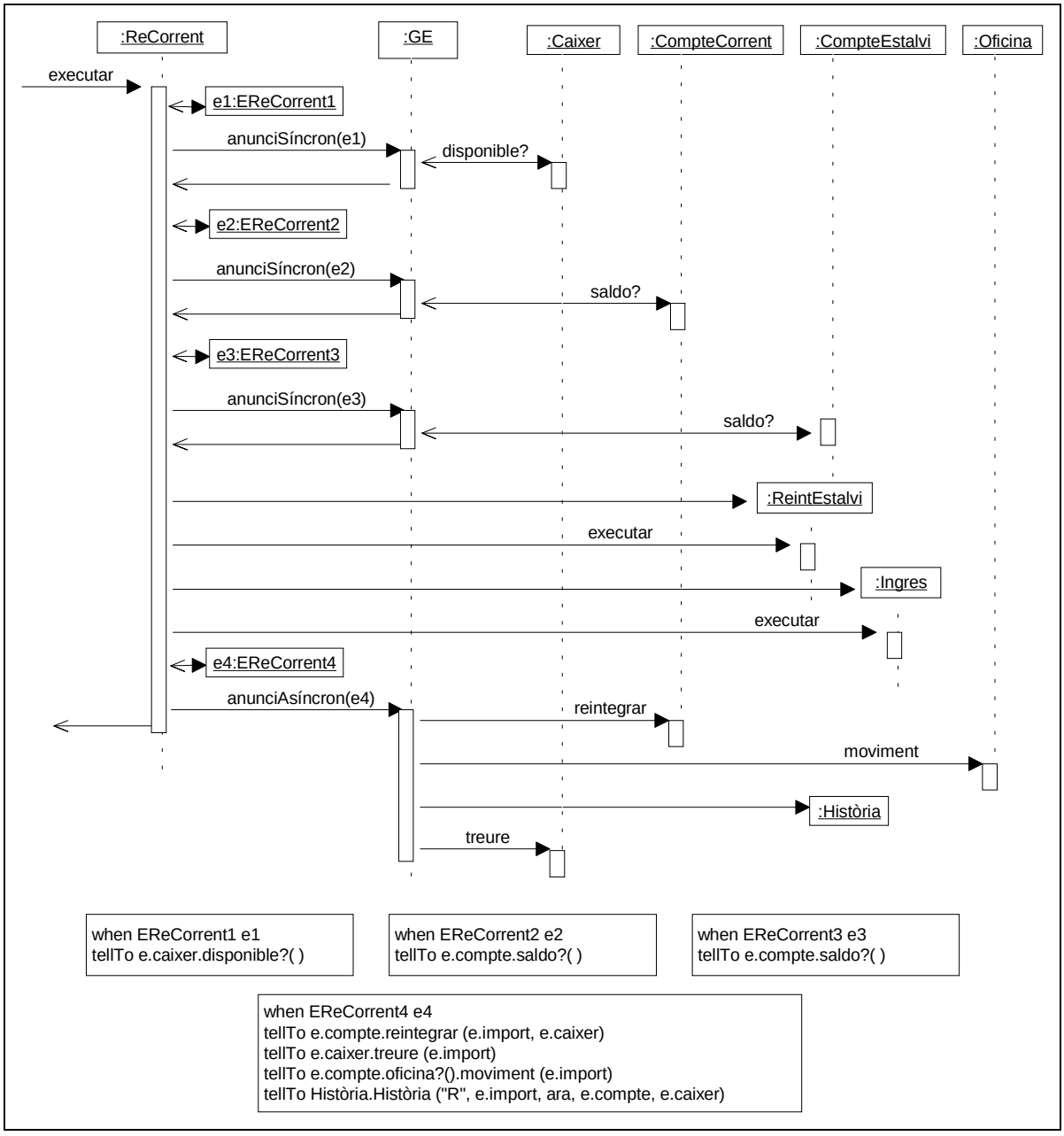

Figura 5.14: Diagrama seqüència ReCorrent (cas d'independència)

Aquest diagrama és el resultat d'aplicar el mecanisme utilitzat al cas anterior a l'esdeveniment extern *ReCorrent*. Part del processament d'aquest esdeveniment extern inclou el processament

associat als esdeveniments externs *Ingrés* i *ReEstalvi*. Per reutilitzar el codi d'aquests dos esdeveniments permetrem que *ReCorrent* faci la creació dels dos esdeveniments externs *Ingrés* i *ReEstalvi*, i invoqui a les seves operacions d'executar. Aquesta serà ***l'única*** excepció que permetrem d'invocacions no fetes ***des*** del GE.

Com hem pogut observar en els exemples anteriors en cap cas hem fet una reutilització dels esdeveniments generats, per poder associar després conducta específica a cada un d'ells, en el cas de canvis posteriors al disseny.

6. Anàlisi comparativa dels dos dissenys.

6.1 Introducció

Com ja hem mencionat, l'objectiu principal d'aquest treball és la comparació de dues aproximacions de disseny de sistemes d'informació. A més a més volem orientar aquesta comparació a la propietat no-funcional de *canviabilitat* dels dissenys obtinguts mitjançant cada aproximació.

No resulta fàcil determinar quina és la tècnica més adequada que ens permet l'obtenció de conclusions realment fiables. És per tant interessant l'estudi de les solucions adoptades en altres treballs davant de situacions semblants. Aquí mencionem [Olivé1], [Olivé2], [Sullivan1], [Sullivan2].

En [Olivé1] i [Olivé2] es fa la comparació de dues aproximacions del modelat conceptual de sistemes d'informació (operacional i deductiu), des de diferents punts de vista: entre ells la propietat de canviabilitat dels models obtinguts amb cada aproximació. Les dues aproximacions (caracteritzades amb un mateix marc de treball) s'apliquen a un SI d'exemple i s'analitza la repercussió, en els models obtinguts, d'una sèrie d'exemples de canvis típics en els requeriments d'un SI. La diferència en els dos treballs radica, fonamentalment, en la complexitat de l'exemple considerat i en la selecció dels tipus de canvis a realitzar.

[Sullivan1] presenta una nova aproximació al disseny d'entorns integrats d'eines software. Per demostrar que els dissenys obtinguts a partir d'aquesta aproximació suporten millor els canvis que els produïts per aproximacions més convencionals (entre elles una basada en la invocació explícita) es presenta un exemple simple d'un entorn a dissenyar. Com en el cas anterior, s'observa l'impacte, sobre els dissenys generats per l'exemple, d'un parell de canvis representatius.

Aquesta aproximació es torna a avaluar a [Sullivan2], partint de l'experiència de la seva aplicació al disseny d'un entorn integrat real. Pel que fa referència a la propietat de canviabilitat aquesta es justifica amb un exemple real de com es va fer l'adaptació de l'entorn de forma senzilla a un canvi imprevist en la seva funcionalitat. Es pren en consideració també l'opinió de diversos experts sobre les dificultats d'evolució de sistemes desenvolupats a partir de diferents tipus d'aproximacions.

Tots aquests són exemples de comparacions basades en un anàlisi *qualitatiu*. Si tractem d'enfocar la comparació a aspectes *quantitatius*, un treball interessant és [ChSKeC94], on es presenten un conjunt de mètriques aplicables a dissenys orientats a objectes. Part d'aquestes mètriques permeten quantificar aspectes relacionats amb la canviabilitat del disseny.

6.2 Anàlisi quantitativa

L'anàlisi quantitativa que mostrem a continuació es realitza en base a les mètriques proposades per [ChSKeC94]. Els valors que cada disseny ens proporciona per aquestes mètriques són comparats i valorats en funció de la filosofia que hi ha darrera de cada mètrica.

Per presentar aquestes mètriques es mostren unes taules on apareixen les classes i subclasses, al costat dret dues columnes amb el valor mesurat per cada classe, primer per la invocació implícita (I) i després per la explícita (E). Els valors que apareixen en negreta indiquen que són més alts que els que apareixen a l'altre disseny.

6.2.1 WMC.

La Taula 6.1 compara els valors per la mètrica *Weighted Methods Per Class* (WMC). L'objectiu d'aquesta mètrica és mesurar la complexitat de les classes del disseny en funció del número i complexitat dels seus mètodes.

Per aquest cas l'augment dels valors per les classes *CompteEstalvi* i *CompteCorrent* es deu a que el disseny proposat en aquest treball exigeix afegir una operació *Compte::oficina?* a la classe *Compte* i aquesta es heretada per les seves subclasses. A la classe *IngrésPdt* també s'han afegit les operacions *IngrésPdt::compte?*, *IngrésPdt::import?* i *IngrésPdt::caixer?*. Cal remarcar que el disseny amb invocació implícita al qual hem aplicat les mètriques correspon a la suposició d'independència d'operacions. En el cas de dependència tant a la classe Compte com a les seves subclasses no cal afegir cap operació tal i com s'ha explicat a l'apartat de disseny.

Pot observar-se que en el cas de les operacions *d'oficina?*, *compte?*, *import?* i *caixer?* es tracta d'operacions que no afegeixen complexitat significativa, ja que són simples consultes dels atributs de la classe.

Les operacions afegides a la resta de casos no són més que una distribució en diversos mètodes d'un processament ja existent en la classe equivalent definida amb la invocació explícita. Per tant la complexitat de cada classe no varia substancialment.

Com a resultat de l'aplicació d'aquestes mètriques podem destacar que l'esforç que es requereix per desenvolupar i mantenir aquestes classes no augmenta significativament d'un disseny a l'altre. L'augment més significatiu d'aquestes mètriques es produeix en classes que no tenen fills i per tant la seva complexitat no s'hereta. El grau de reusabilitat de les classes en els dos casos és semblant.

6.2.2 DIT.

La comparació de la mètrica *Depth of Inheritance Tree* (DIT) es mostra a la Taula 6.2. Aquesta mètrica tracta de mesurar com determinades superclasses poden afectar potencialment a una classe donada.

En aquesta mètrica la diferència es deu a la nova jerarquia que hem proposat pels esdeveniments en el disseny amb invocació implícita, ja que aquest disseny implica la generació d'esdeveniments. Normalment quan més gran és la profunditat més gran és la profunditat del disseny ja que més classes i mètodes involucra. Com en aquest cas, la introducció de noves classes es fa amb la intenció de diferenciació semàntica, les classes

introduïdes són totes abstractes i no afecten a les seves subclasses.

6.2.3 NOC

La comparació de la mètrica *Number of Children* (NOC) es mostra a la Taula 6.3. L'augment en aquesta mètrica en relació al disseny amb invocació explícita no té una implicació gran, ja que proporciona informació sobre quantes subclasses hereten els mètodes de les classes pares i a més indica l'esforç que requereix fer proves de canvis als mètodes d'una classe amb un gran nombre de fills. De nou ens referirem a la nova jerarquia proposada pels esdeveniments que implica l'aparició d'esdeveniments generats i que aquests no tenen mètodes, per tant l'augment de la mètrica no és significatiu.

6.2.4 CBO

La comparació de la mètrica *Coupling between object classes* (CBO) es mostra a la Taula 6.4. Aquesta mètrica mesura el grau d'acoblament entre un objecte i altres. La filosofia proposada per aquest treball parteix de que no es permet la invocació explícita, el que es tradueix en aquesta mètrica a obtenir un valor 0 o 1 en el cas de que es consideri acoblament per la invocació de l'operació d'anunci sobre el GE. En el disseny amb invocació implícita no hi ha dependència entre les classes d'objectes. Això ens permet més fàcilment la reutilització d'aquests objectes en altres aplicacions, així com menys dificultats per mantenir l'aplicació. En aquest sentit, això ens indica que la propietat de canviabilitat es presenta en un grau més gran al nou disseny.

6.2.5 RFC

La comparació de la mètrica *Response for a class* (RFC) es mostra a la Taula 6.5. Aquesta mètrica ens proporciona el grau de resposta d'una classe a un esdeveniment rebut, és a dir, la resposta dels mètodes de la classe, així com aquells que poden ser invocats en altres classes. Recordem que ja havíem vist a la mètrica anterior que en el diseny amb invocació implícita disminueix l'acoblament i per tant desapareixen les invocacions que des dels mètodes d'una classe es fan a mètodes d'altres. L'esforç per provar les parts de l'aplicació disminueix considerablement, ja que el nombre d'invocacions en resposta a un esdeveniment es redueix.

6.2.6 LCOM.

La comparació de la mètrica Lack of Cohesion in Methods (LCOM) es mostra a la Taula 6.6. Aquesta mètrica no varia pels dos dissenys proposats. Això ens indica que les noves classes no són menys cohesives en el nou disseny, encara que hem introduït noves operacions.

Weighted Methods Per Class (WMC) **$WMC = \Sigma_i^n c_i \quad (c=1)$**					
Classes				**Mètrica**	
				I	**E**
Oficina				2	2
Caixer				3	3
Compte				**4**	3
	CompteEstalvi			**4**	3
	CompteCorrent			**5**	4
IngrésPdt				**4**	1
Història				0	0
Esdeveniment				0	
	Esd. Generat			0	
		Consulta		0	
			ESaldoCompte1	0	
			EReEstalvi1	0	
			EReEstalvi2	0	
			EReEstalvi3	0	
			EReCorrent1	0	
			EReCorrent2	0	
			EReCorrent3	0	
			EReCorrent4	0	
		Actualització		0	
			EOmplirCaixer1	0	
			EIngrésEnCaixer1	0	
			EConfirmar1	0	
			EConfirmar2	0	
			EConfirmar3	0	
			EIngrés1	0	
			EIngrés2	0	
	Esd. Extern			1	1
		Consulta		1	1
			SaldoCompte	1	1
		Actualització		1	1
			Ingrés	1	1
			OmplirCaixer	1	1
			IngrésEnCaixer	1	1
			Confirmar	1	1
		PetícióAcció		1	1
			ReEstalvi	**3**	1
			ReCorrent	**3**	1

Taula 6.1: Resultats de la mètrica WMC

Depth of Inheritance Tree (DIT)					
Classes				**Mètrica**	
				I	**E**
Oficina				0	0
Caixer				0	0
Compte				0	0
	CompteEstalvi			1	1
	CompteCorrent			1	1
IngrésPdt				0	0
Història				0	0
Esdeveniment				0	0
	Esd. Generat			1	
		Consulta		2	
			ESaldoCompte1	3	
			EReEstalvi1	3	
			EReEstalvi2	3	
			EReEstalvi3	3	
			EReCorrent1	3	
			EReCorrent2	3	
			EReCorrent3	3	
			EReCorrent4	3	
		Actualització		2	
			EOmplirCaixer1	3	
			EIngrésEnCaixer1	3	
			EConfirmar1	3	
			EConfirmar2	3	
			EConfirmar3	3	
			EIngrés1	3	
			EIngrés2	3	
	Esd. Extern			1	1
		Consulta		**2**	1
			SaldoCompte	**3**	2
		Actualització		**2**	2
			Ingrés	**3**	2
			OmplirCaixer	**3**	2
			IngrésEnCaixer	**3**	2
			Confirmar	**3**	2
		PetícióAcció		2	**3**
			ReEstalvi	3	3
			ReCorrent	3	3

Taula 6.2: Resultats de la mètrica DIT

Number of Children (NOC)					
Classes				**Mètrica**	
				I	**E**
Oficina				0	0
Caixer				0	0
Compte				2	2
	CompteEstalvi			0	0
	CompteCorrent			0	0
IngrésPdt				0	0
Història				0	0
Esdeveniment				2	
	Esd. Generat			2	
		Consulta		8	
			ESaldoCompte1	0	
			EReEstalvi1	0	
			EReEstalvi2	0	
			EReEstalvi3	0	
			EReCorrent1	0	
			EReCorrent2	0	
			EReCorrent3	0	
			EReCorrent4	0	
		Actualització		7	
			EOmplirCaixer1	0	
			EIngrésEnCaixer1	0	
			EConfirmar1	0	
			EConfirmar2	0	
			EConfirmar3	0	
			EIngrés1	0	
			EIngrés2	0	
	Esd. Extern			**3**	2
		Consulta		1	1
			SaldoCompte	0	0
		Actualització		4	**5**
			Ingrés	0	0
			OmplirCaixer	0	0
			IngrésEnCaixer	0	0
			Confirmar	0	0
		Petició Acció		2	2
			ReEstalvi	0	0
			ReCorrent	0	0

Taula 6.3: Resultats de la mètrica NOC

Coupling between object classes (CBO)					
Classes				**Mètrica**	
				I	**E**
Oficina				0	0
Caixer				0	0
Compte				0	**2**
	CompteEstalvi			0	0
	CompteCorrent			0	0
IngrésPdt				0	**1**
Història				0	0
Esdeveniment				0	
	Esd. Generat			0	
		Consulta		0	
			ESaldoCompte1	0	
			EReEstalvi1	0	
			EReEstalvi2	0	
			EReEstalvi3	0	
			EReCorrent1	0	
			EReCorrent2	0	
			EReCorrent3	0	
			EReCorrent4	0	
		Actualització		0	
			EOmplirCaixer1	0	
			EIngrésEnCaixer1	0	
			EConfirmar1	0	
			EConfirmar2	0	
			EConfirmar3	0	
			EIngrés1	0	
			EIngrés2	0	
	Esd. Extern			0	
		Consulta		0	
			SaldoCompte	0	**1**
		Actualització		0	
			Ingrés	0	**1**
			OmplirCaixer	0	**1**
			IngrésEnCaixer	0	**1**
			Confirmar	0	**1**
		PetícióAcció		0	
			ReEstalvi	0	**2**
			ReCorrent	0	**5**

Taula 6.4: Resultats de la mètrica CBO

Response for a Class (RFC)					
Classes				**Mètrica** **I**	**E**
Oficina				2	2
Caixer				3	3
Compte				**4**	3
	CompteEstalvi			**4**	3
	CompteCorrent			**5**	4
IngrésPdt				**4**	1
Història				0	0
Esdeveniment				0	
	Esd. Generat			0	
		Consulta		0	
			ESaldoCompte1	0	
			EReEstalvi1	0	
			EReEstalvi2	0	
			EReEstalvi3	0	
			EReCorrent1	0	
			EReCorrent2	0	
			EReCorrent3	0	
			EReCorrent4	0	
		Actualització		0	
			EOmplirCaixer1	0	
			EIngrésEnCaixer1	0	
			EConfirmar1	0	
			EConfirmar2	0	
			EConfirmar3	0	
			EIngrés1	0	
			EIngrés2	0	
	Esd. Extern			1	1
		Consulta		1	1
			SaldoCompte	**3**	2
		Actualització		1	1
			Ingrés	**3**	2
			OmplirCaixer	**3**	2
			IngrésEnCaixer	**3**	2
			Confirmar	**3**	2
		Petició Acció		1	1
			ReEstalvi	**6**	5
			ReCorrent	**11**	10

Taula 6.5: Resultats de la mètrica RFC

Lack of Cohesion in Methods (LCOM)					
Classes				**Mètrica**	
				I	**E**
Oficina				0	0
Caixer				0	0
Compte				0	0
	CompteEstalvi			0	0
	CompteCorrent			0	0
IngrésPdt				0	0
Història				0	0
Esdeveniment				0	0
	Esd. Generat			0	
		Consulta		0	
			ESaldoCompte1	0	
			EReEstalvi1	0	
			EReEstalvi2	0	
			EReEstalvi3	0	
			EReCorrent1	0	
			EReCorrent2	0	
			EReCorrent3	0	
			EReCorrent4	0	
		Actualització		0	
			EOmplirCaixer1	0	
			EIngrésEnCaixer1	0	
			EConfirmar1	0	
			EConfirmar2	0	
			EConfirmar3	0	
			EIngrés1	0	
			EIngrés2	0	
	Esd. Extern			0	0
		Consulta		0	0
			SaldoCompte	0	0
		Actualització		0	0
			Ingrés	0	0
			OmplirCaixer	0	0
			IngrésEnCaixer	0	0
			Confirmar	0	0
		PetícióAcció		0	0
			ReEstalvi	0	0
			ReCorrent	0	0

Taula 6.6: Resultats de la mètrica LCOM

6.3 Anàlisi qualitativa.

L'objectiu d'aquest punt és realitzar una comparació qualitativa dels dissenys presentats en aquest treball, per poder avaluar fins a quin punt un ús enraonat de la invocació implícita podria augmentar la canviabilitat del software.

Per fer aquesta comparació considerarem dos tipus de canvis habituals que es produeixen en els sistemes d'informació. Aquests canvis consistiran en:

- Afegir un nou atribut, el valor del qual s'obté amb els esdeveniments ja existents.
- Afegir un nou esdeveniment que afecta a classes o atributs ja existents.

6.3.1 Afegir un nou atribut, el valor del qual s'obté amb els esdeveniments ja existents

Anem a examinar quines són les modificacions que calen en els dos enfocaments, si ens afegeixen un nou requeriment:

> *"Volem saber el número d'ingressos que fan els comptes de cada oficina."*

Afegir aquest requeriment en el disseny amb invocació explícita suposa els següents canvis:

- Afegir a la classe *Oficina* un atribut *Oficina::numIngressos*.
- Afegir a la classe *Oficina* una operació *Oficina::incrementarNumIngr*.
- Modificar la operació *Compte::ingressar* per que invoqui a *Oficina::incrementarNumIngr*

Es mostra a continuació el diagrama de seqüència on hi apareix la nova operació *Compte::ingressar*.

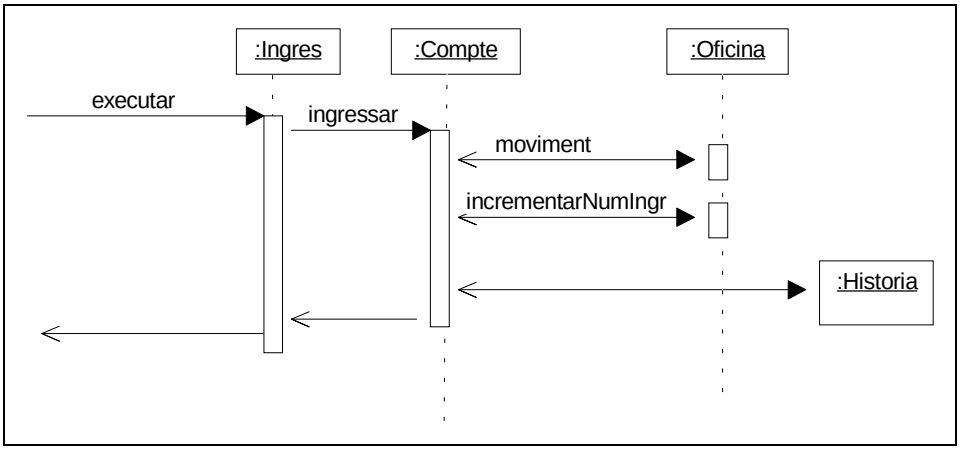

Figura 6.1: Diagrama seqüència Ingrés (invocació explícita)

Anem a veure ara com afecta aquesta modificació pel cas de la invocació implícita. Els canvis seran els següents:

- Afegir a la classe oficina un atribut *Oficina::numIngressos*.
- Afegir a la classe oficina una operació *Oficina::incrementarNumIngr*.
- A la subscripció on tenim l'esdeveniment que genera l'operació *Compte::ingressar* cal afegir una nova invocació a l'operació *Oficina::incrementarNumIngr*.

Es mostra a continuació el diagrama de seqüència on hi apareix la modificació.

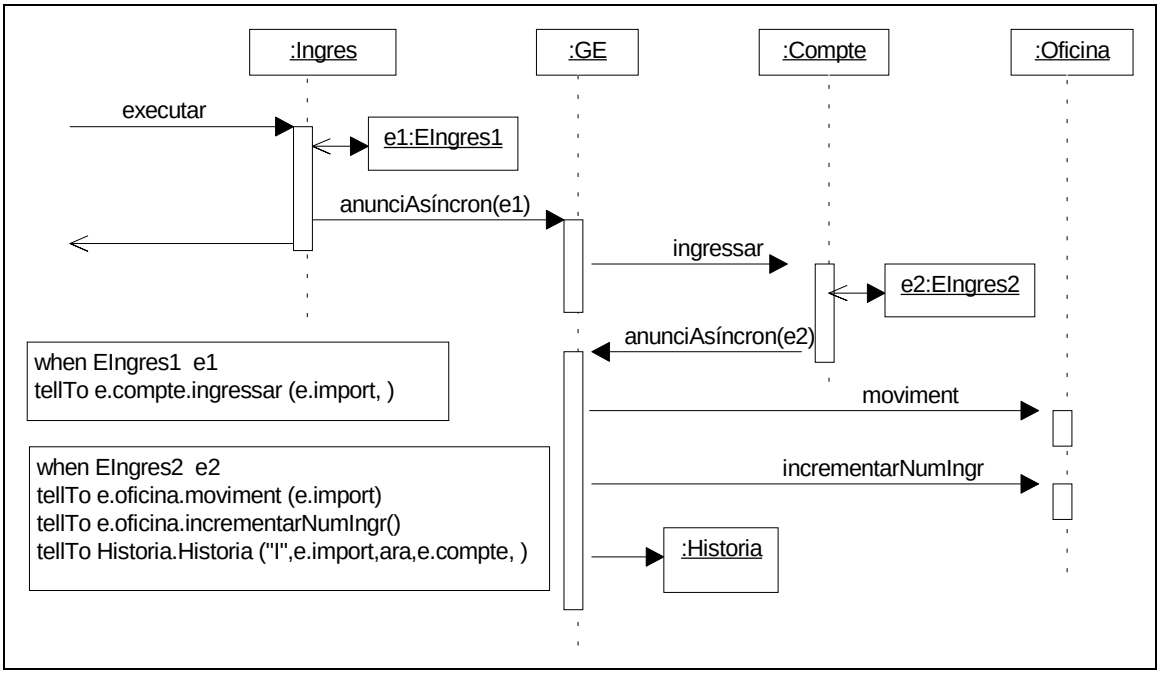

Figura 6.2: Diagrama seqüència Ingrés (invocació implícita)

Com hem vist en els dos dissenys cal afegir el nou atribut i la nova operació de modificació. En canvi en el disseny amb invocació explícita cal modificar el codi de l'operació *Compte::ingressar* afegint una crida a l'operació *Oficina::incrementarNumIngr,* mentre que a l'altre disseny cal afegir una invocació en una subscripció.

Tot i que sembla que els canvis en els dos dissenys són semblants hi ha diferències. En el primer cas s'ha de fer la modificació del codi d'una operació. En el segon cas s'ha d'afegir una invocació en una subscripció. Es important destacar que requereix més esforç modificar el codi d'una operació que inserir una nova invocació en una subscripció.

En aquest cas afegir un nou requeriment només implica la modificació d'una operació, però es pot donar el cas d'afegir nous requeriments que impliquin canvis en el codi d'operacions més complicats, per exemple que es facin en funció d'unes determinades condicions. Amb la invocació explícita s'han d'inserir aquests canvis en el codi de les operacions mentre que amb la invocació implícita afegirem la invocació i si cal afegirem o modificarem la condició de la subscripció.

Tal i com hem afegit aquest nou requeriment al nostre exemple podem observar que hi ha una dependència entre l'operació *Compte::ingressar* i les operacions *Oficina::incrementarNumIngr, Oficina::moviment* i el constructor d'*Història*. Potser en aquest cas seria més real considerar que aquestes operacions no tenen aquesta dependència i que és possible que no totes les transaccions que fan la invocació a l'operació *Compte::ingressar* volen incrementar el número d'ingressos de la seva oficina. En aquest cas parlem de que hi ha una independència de les operacions. També en aquest cas la modificació que estem considerant tindria el mateix efecte que el cas anterior. En el primer disseny hauríem de cridar a aquesta nova operació des de totes les transaccions que vulguin registrar els ingressos que es fan. En el segon hauríem d'afegir les noves invocacions en les subscripcions. També en aquest cas es requereix més esforç per modificar el codi de totes les transaccions que vulguin registrar el nou requeriment que inserir una nova invocació en les

subscripcions que es vegin afectades.

6.3.2 Afegir un nou esdeveniment que afecta a classes o atributs ja existents.

En aquest punt, anem a examinar quines són les modificacions que calen en els dos enfocaments si ens afegeixen un nou requeriment:

> *"Volem afegir un nou esdeveniment extern RegalBanc. Aquest esdeveniment tindrà com a paràmetre un compte. El banc farà un ingrés de 500 ptes al compte si aquest té un saldo més gran que 100.000 ptes."*

Afegir aquest nou requeriment en el disseny amb invocació explícita suposa els següents canvis:

- Construir aquest nou esdeveniment extern.

A continuació es mostra el diagrama de seqüència on hi apareix aquesta nova transacció.

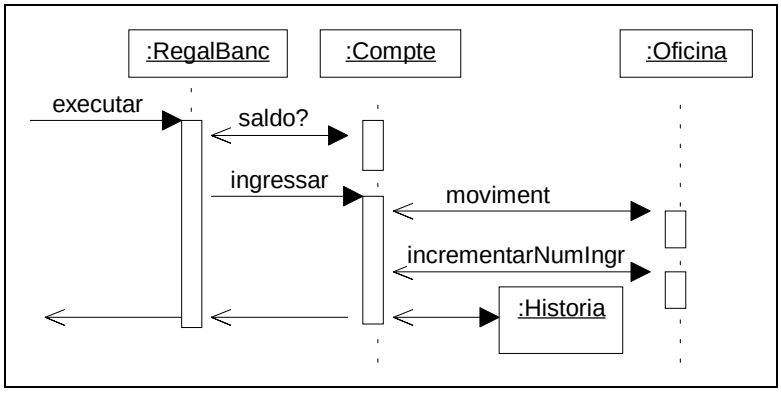

Figura 6.3: Diagrama seqüència RegalBanc (invocació explícita)

Anem a veure com afecta aquest canvi en el cas de la invocació implícita. Els canvis seran els següents:

- Construir part del nou esdeveniment.

Es mostra a continuació el diagrama de seqüència on apareix la modificació.

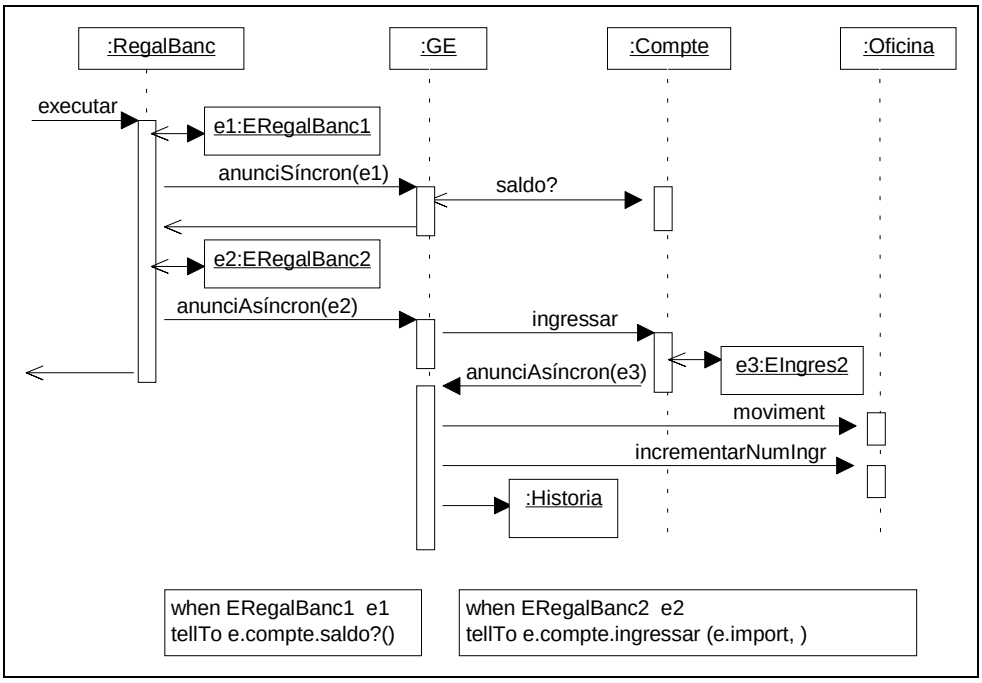

Figura 6.4: Diagrama seqüència RegalBanc (invocació implícita)

Com hem vist en els dos dissenys cal afegir un nou esdeveniment. En el cas del disseny amb invocació explícita cal construir tota la transacció des del començament. En canvi en el cas del disseny amb invocació implícita podem observar que es poden afegir les subscripcions associades als esdeveniments que ha de generar el nou esdeveniment. Per tant el codi del nostre esdeveniment extern s'encarregarà de generar els esdeveniments i anunciar-los al GE. El gestor ja té les subscripcions i per tant ja sap a quines operacions s'han d'invocar.

7. Conclusions.

La contribució d'aquest treball es centra en tres aspectes:

- La proposta d'un GE i tres premisses fonaments com a base de disseny de SI amb invocació implícita.
- Un segon aspecte és l'aplicació de la visió anterior a un exemple concret.
- Per últim, els avantatges que proporciona un disseny basat en aquest enfocament analitzades des d'un punt de vista quantitatiu i qualitatiu.

El treball futur té dues vessants: profunditzar en l'estudi del disseny de SI amb invocació implícita i en la implementació del GE. Això inclou:

- Definició d'una nova nomenclatura, més adient, pels esdeveniments generats.
- Possibilitat de la reutilització d'esdeveniments
- Gestió de concurrència.
- Aspectes de gestió d'esdeveniments (anunci, invocació i subscripció)
- Tractament d'excepcions produïdes per operacions invocades pel GE.
- Coexistència de la invocació implícita i explícita en un mateix disseny. Delimitació dels casos en que resulta més adient una o l'altra.

8. Referències.

[Ano85] Anon et al. “A measure of Transaction Processing Power”, Datamation, April.

[Bub86] Bubenko, J.A.; Olivé, A. “Dynamic or Temporal modeling? - an illustrative comparison” SYSLAB Working Paper Nr. 117, 21 November

[Bus96] Buschmann,F.; Meunier,R.; Rohnert,H.; Sommerlad,P. Stal,M. “Pattern-Oriented Software Architecture. A System of Patterns”, John Wiley.

[Coa95] Coad, P.; North, D.; Mayfield, M. “Object Models. Strategies, Patterns and Applications”, Yourdon Press.

[Cos97] Costal, D. Et al. “The Cause-Effect Rules of ROSES”, Proc. ADBIS’97, St. Petersburg, pp. 399-405.

[Chi94] Chidamber, S.R., Kemerer, C.F. “A metrics suite for object oriented design”, IEEE Transactions on Software Engineering, Vol. 20, No. 6, June, pp. 476-493

[Gam95] Gamma,E.; Helm,R.; Johnson,R.; Vlisides,J. “Design Patterns”. Elements of Reusable Object-Oriented Software”, Addison-Wesley.

[Lar98] Larman, C. “Applying UML and Patterns”, Prentice Hall.

[Mar95] Martin, R.C. “Designing Object-Oriented Applications using the Booch Method”, Prentice-Hall.

[Oli86] Olivé, A. “A comparison of the operational and deductive approaches to conceptual information systems modelling” , Information Processing 86, H.-J. Kugler (ed.), pp 91-96

[Rie96] Riehle, D. “The Event Notification Pattern - Integrating Implicit Invocation with Object Orientation”. TPOS, Vol. 2(1), pp. 43-52.

[Sha96] Shaw, M.; Garlan, D. “Software Architecture. Perspectives on an emerging discipline”, Prentice Hall.

[Sul92] Sullivan, K.J.; Notkin, D. “Reconciling Environment Integration and Software Evolution”, ACM Trans. Softw. Eng. and Methodologies, Vol. 1, No. 3, July, pp. 229-268.

[Sul96] Sullivan, K.J.; Kalet, I.J.; Notkin, D. “Evaluating the mediator method: Prism as a case study”, IEEE Transactions on Software Engineering, Vol. 22, No. 8, August, pp. 563-579

[Wid96] Widom J.; Ceri, S. “Active Database Systems”, Morgan-Kaufmann.

Índex de Figures.

Índex de Taules.