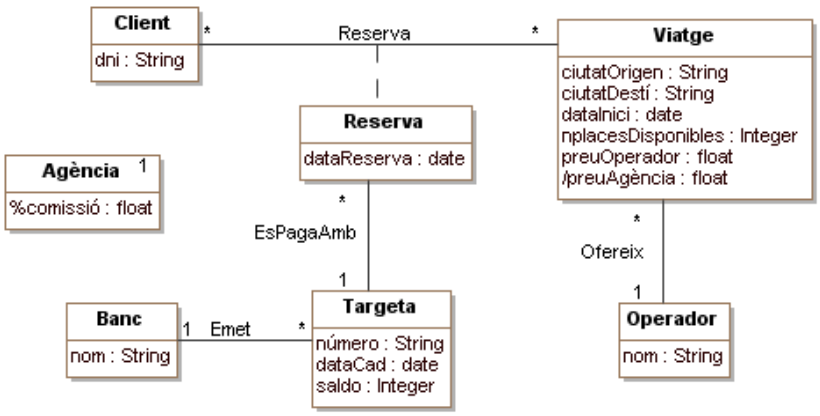
Problema 4 [3 punts]. Volem crear una agència de viatges per Internet. Per tal de posar en marxa l'agència necessitem un sistema software que gestioni el nostre negoci. Cada viatge s'identifica per un origen, un destí, una data d'inici i un operador, i se'n saben les places disponibles. Els operadors venen el viatge a l'agència per un preu, i l'agència els ofereix als clients aplicant-hi una comissió. **Considereu que el preu d'un viatge no pot canviar un cop creat el viatge**. Els clients, que s'identifiquen per dni, faran reserves dels viatges en una data concreta. Es gestiona el pagament de les reserves per tarja de crèdit.



Restriccions textuals:

RT1. Claus: (Client, dni); (Operador, nom);
(Targeta, Banc::nom+número); (Banc, nom)
(Viatge, ciutatOrigen+ciutatDestí+
dataInici+Operador::nom)

RT2. nplacesDisponibles >= 0, preuOperador > 0,
saldo > 0, dataCad >= dataReserva,
dataInici >= dataReserva

Informació derivada:

ID1. preuAgència =
preuOperador+%comissió*preuOperador/100

Hem decidit organitzar el nostre sistema com una arquitectura SOA amb els serveis següents (**que no es poden modificar**):

- Servei de pagament (SvPagament): aquest servei autoritza els pagaments realitzats amb targetes de qualsevol entitat bancària i els ingressa en un compte de la nostra agència.

```
context SvPagament::autorització(nomB: String, numT: String, dataC: Date, imp: Float)
exc targeta-no-vàlida: la targeta no existeix o la data de caducitat no és vàlida
exc no-hi-ha-saldo: la targeta no té saldo suficient
post autoritza: descompta l'import imp del saldo de la targeta
```

- Servei d'operadors (SvOperadors): cada operador disposa d'un servei que enregistra els viatges que ofereix. **Els noms dels operadors poden usar-se per localitzar els serveis** (per exemple, el servei de l'operador Rumbo es diu SvRumbo). Les operacions proporcionades pel servei són:

```
context SvOperador::comprarViatge(orig: String, dest: String, dataI: Date): Float
exc viatge-no-existeix: l'operador no ofereix cap viatge amb orig, dest i dataI
exc no-hi-ha-places: no hi ha places disponibles per al viatge
post compra: decrementa en 1 el nombre de places disponibles del viatge
post resultat: retorna el preu del viatge ofert per l'operador
```

```
context SvOperador::dadesViatge(orig: String, dest: String, dataI: Date):
                <orig: String, dest: String, dataI:Date, nomOp: String,
                 durada: Integer, nplacesDisp: Integer, preuOp: Float>
exc viatge-no-existeix: l'operador no ofereix cap viatge amb orig, dest i dataI
post resultat: retorna les dades del viatge
```

Observació important: tots dos serveis són utilitzats per altres sistemes. Els operadors poden ampliar els viatges que ofereixen en qualsevol moment i, evidentment, la nostra agència no tindrà notificació d'aquest fet. És a dir, l'operador pot oferir més viatges que els que té registrats l'agència.

Considereu que la navegabilitat de l'associació entre Client i Viatge és doble. Considereu que la informació derivada s'ha de materialitzar.

(a) **[0.5 punts]** Mostreu com queda el model conceptual de dades del sistema de la nostra empresa.

(b) **[2.5 punts]** Volem disposar de dos casos d'ús (amb un únic esdeveniment): *operadorEconòmic* que permet saber el preu i el nom de l'operador que proporciona el viatge més econòmic des d'una ciutat origen a una ciutat destí per a una data d'inici i *ferReserva* que permet als nostres clients fer reserves de viatges. Es demana que feu el diagrama de seqüència de les operacions de la capa de domini dels casos d'ús anteriors. A continuació disposeu dels contractes de les operacions a dissenyar:

```
context CapaDeDomini::operadorEconòmic(orig: String, dest: String, dataI: Date):
                                <nomOp: String + preuOp: Float>
exc no-hi-ha-operador: cap operador (de la nostra agència) ofereix el viatge
post operador-econòmic: retorna el nom de l'operador (d'entre els que tenen acords
amb la nostra agència) que ofereix el viatge amb orig, dest i dataI amb el preu més
econòmic. (Noteu que, atesa l'observació anterior, l'operador pot oferir el viatge
sense que la nostra agència el tingui enregistrat.) Es retorna també aquest preu.
```

```
context CapaDeDomini::ferReserva(orig: String, dest: String, dataI: Date,
           nomOp: String, nomB: String, numT: String, dataC: Date, dniC: String)
pre client-existeix: el client amb dniC existeix
pre viatge-existeix: el viatge V amb orig+dest+dataI+nomOp és ofertat per l'agència
                    (és a dir, forma part dels viatges que té enregistrats l'agència)
pre hi-ha-lloc: el viatge V no està ple
exc reserva-existeix: ja existeix una reserva del client dniC per al viatge V
exc targeta-no-vàlida: la targeta no existeix o la data de caducitat no és vàlida
exc no-hi-ha-saldo: la targeta no té saldo suficient
post alta-reserva: es dóna d'alta la reserva en data d'avui pel client dni, pel
                  viatge i per la targeta dels paràmetres
post pagament-viatge: es decrementa el saldo de la targeta en la quantitat indicada
                     pel preu del viatge
```

Useu Domain Model sense operacions de consulta arbitràries a la capa de gestió de dades, Data Mapper i controlador transacció. **Com a criteri de disseny principal en aquest problema volem minimitzar el nombre d'invocacions remotes. Com a criteri secundari subordinat a l'anterior també es vol minimitzar la redundància de les dades.** A banda, com sempre, tenim els criteris habituals de canviabilitat, reusabilitat i llegibilitat.