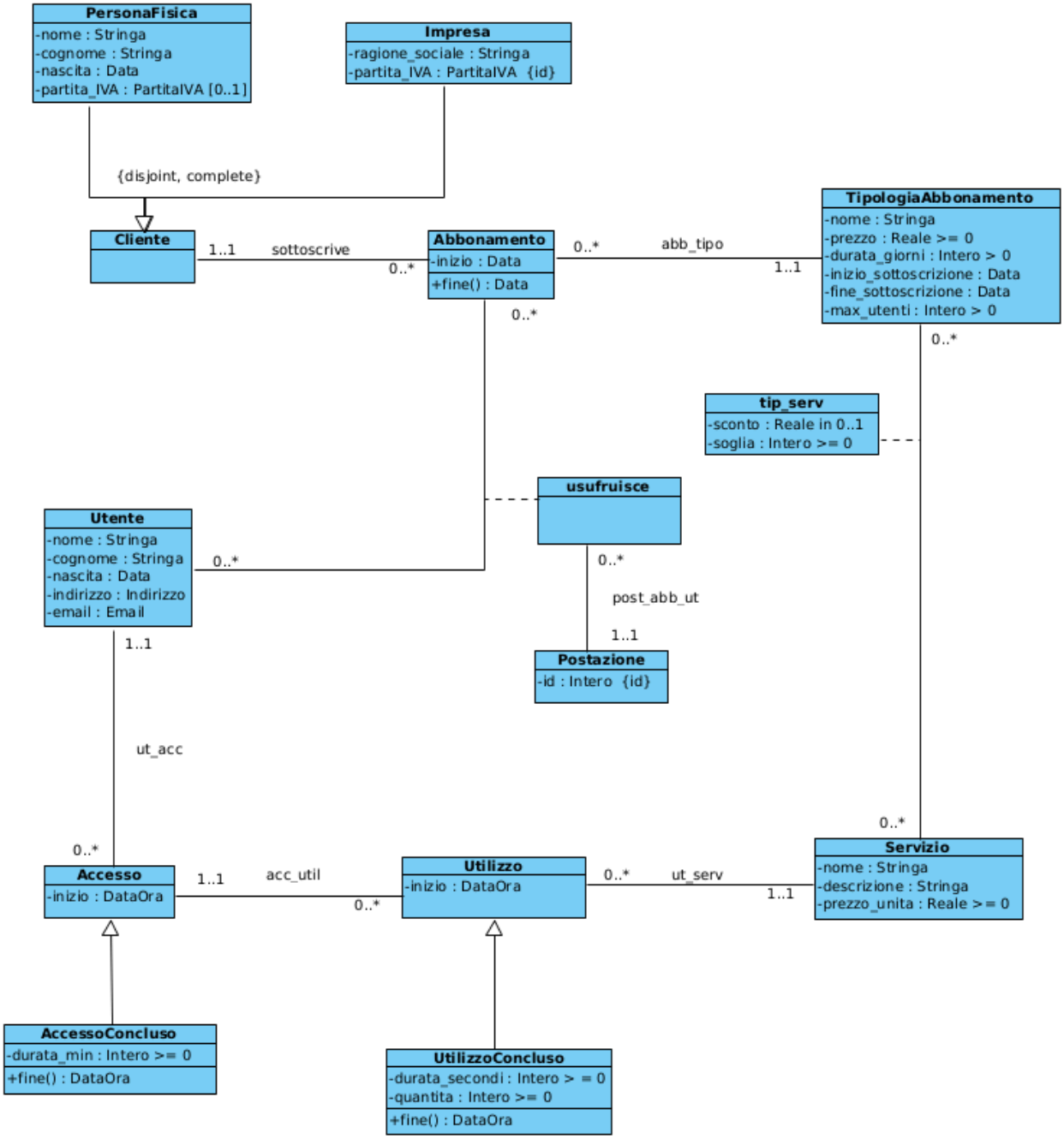
Progetto CoLab

Assunzioni

1. Ogni abbonamento può essere utilizzato da più utenti, anche se è stato acquistato da un cliente privato.
2. Un utente non può usufruire di due abbonamenti che si sovrappongono nel tempo.

1. Analisi Concettuale

1.1. Diagramma UML delle classi Concettuale



1.2. Specifica dei tipi di dato

- Indirizzo: tipo record con campi [...]
- Email: stringa secondo standard
- PartitaIVA: Stringa numerica secondo standard
- FasciaOraria = (
    - inizio: Ora
    - fine: Ora
    - ALL f, inizio, fine FasciaOraria(f) and inizio(f, inizio) and fine(f, fine) → inizio <= fine

d, o
Data(d) and Ora(o)

i tale che DataOra(i) and data(i, d) and ora(i, o)

https://pastebin.com/grRf29yd

1.3. Specifiche delle classi

1.3.1. Specifica della classe *Abbonamento*:

fine(): Data
Precondizioni: nessuna
Postcondizioni:

Siano *ta, d, i*
tali da soddisfare la seguente formula
abb_tipo(this, ta) AND durata(ta, d) AND inizio(this, i)

Il risultato *result* è definito come *i + d*

1.3.2. Specifica della classe Accesso

[V.Accesso.stesso_utente_no_sovrapposti]
NOT EXISTS a1, a2, u, t, i1, i2
Accesso(a1) AND Accesso(a2) AND acc_ut(a1, u) AND acc_ut(a2, u) AND
inizio(a1, i1) AND inizio(a2, i2) and DataOra(t) AND
AND t >= i1 AND t >= i2 AND
AND [ALL f1 fine(a1, f1) → t <=f1] AND
AND [ALL f2 fine(a1, f2) → t <=f2]

1.3.3. Specifica della classe UtilizzoConcluso

quantita_non_gratuita(): Intero >= 0
Precondizioni: nessuna
Postcondizioni:

Siano acc, ute, abb, serv, q, f
Tali da soddisfare
Acc è l'accesso di this, serv è l'accesso di this, ute è l'utente di acc, f è la fine di thisabb è l'unico abbonamento di ute in this.fine e q è la quantità di this

Result = max(0, q - gratis_rimanenti(abb, serv, f))

1.4. Vincoli Esterni

Operazioni ausiliarie

non_sovrapposti(i_1: DataOra, f_1: DataOra, i_2: DataOra, f_2: DataOra): Booleano
Precondizioni: $i_1 <= f_1$ AND $i_{2 <=} f_2$
Postcondizioni:
Result = True ⇔[$i_1 > f_2$ OR $i_2 > f_1$]

gratis_rimanente(abb: Abbonamento, s: Servizio, i: DataOra): Intero >= 0
Precondizioni:
ALL ia, fa inizio(abb, ia) AND fine(abb, ai) → ia <= i <= fa
AND
ALL ta tip_abb(abb, ta) → tip_serv(ta, s)

Postcondizioni:
Sia m tale da soddisfare mese(i, m)
Sia sg tale da soddisfare la seguente formula
ALL ta tip_abb(abb, ta) → tip_serv(ta, s) AND soglia(ta, s, sg)

U = { (u, q) | UtilizzoConcluso(u) AND quantita(u, q) AND EXISTS fu fine(u, fu)
AND fu < i AND mese(fu, m) AND ut_serv(u, s) }

Tot_q = $\sum_{(u,\,q)\,\in\,U} q$

result = max(0, sg - tot_q)

Specifica realizzativa:

Q =

```
SELECT ts.tipologia as tip,
          ts.soglia as soglia,
          ta.sum(uc.q) as q_tot

FROM abbonamento ab, servizio serv,
     utilizzoconcluso uc, accesso acc,
     utente ut, usufruisce us, tip_serv ts

WHERE Uc.accesso = acc.id and acc.utente = ut.id
     and us.utente = ut.id and us.abbonamento=:abb
     and uc.serv = :s
     and (extract 'month' from uc.fine)
          = (extract 'month' from :i) and uc.fine < i
     and abb.tipologia = ts.tipologia and ts.servizio = :s

GROUP BY ts.tipologia, ts.soglia
```

Sia (tip, soglia, q_tot) l'unica ennupla di Q

Return max(0, soglia - q_tot)

[V.Abbonamento.numero_usufruisce_minore_max]
Il numero di utenti che usufruiscono di un abbonamento è non superiore al numero massimo previsto dalla tipologia dell'abbonamento
ALL a, ta, m
Abbonamento(a) AND tip_abb(a, ta) AND max_utenti(ta, m) =>
| { u | Utente(u) and usufruisce(u, a) } | <= m

[V.Abbonamento.Utente.usufruisce.non_sovrapposti]
ALL u, a_1, a_2, i_1, f_1, i_2, f_2

Utente(u) AND Abbonamento(a_1) AND Abbonamento(a_2) AND a_1=/= a_2 AND usufruisce(u, a_1) AND usufruisce(u, a_2) AND inizio(a_1, i_1) AND fine(a_1, f_1) AND inizio(a_2, i_2) AND fine(a_2, f_2)
–> non_sovrapposti(i_1, f_1, i_2, f_2, True)

[V.Postazione.usufruisce.non_sovrapposti]
ALL p, u1, u2, a1, a2, i_1, f_1, i_2, f_2
Postazione(p) AND usufruisce(u1, a1) AND usufruisce(u2, a2) AND
AND post_abb_ut((u1, a1), p) AND post_abb_ut((u2, a2), p) AND inizio(a_1, i_1) AND fine(a_1, f_1) AND inizio(a_2, i_2) AND fine(a_2, f_2) → non_sovrapposti(i_1, f_1, i_2, f_2, True)

[V.Accesso.Solo_con_abbonamento_attivo]
ALL a, u, i, ia, fa
ut_acc(u, a) AND inizio(a, i) =>
EXISTS ab
usufruisce(u, ab) AND inizio(ab, ia) AND
AND fine(ab, fa) AND i >= ia AND i <= fa

[V.Utilizzo.Servizio.incluso_in_tipologia_abbonamento]
ALL uti, s, acc, ute, ini_uti, abb, i_abb, f_abb, ta
ut_serv(uti, s) AND acc_util(acc, uti) AND ut_acc(ute, acc) AND
AND inizio(uti, ini_uti) AND usufruisce(ute, abb) AND inizio(abb, i_abb) AND fine(abb, i_abb) AND i_abb <= ini_uti <= f_abb AND tip_abb(abb, ta) =>
tip_serv(ta, s)

[V.Accesso.Utente.dopo_nascita]
ALL acc, ut, i, na
    ut_acc(ut, acc) AND inizio(acc, i) AND nascita(ut, na) => na < i

[V.Utilizzo.all_interno_di_accesso]
ALL uti, acc, i_uti, i_acc
    acc_util(acc, uti) AND inizio(uti, i_uti) AND inizio(acc, i_acc)
    –> i_uti >= i_acc AND [ALL f_acc fine(acc, f_acc) –> i_uti <= f_acc]

1.5. Specifiche concettuali delle operazioni di Use-Case

1) media_giornaliera_accessi(F: FasciaOraria [1..*]): (FasciaOraria, Reale >= 0) [1..*]
Lasciata come esercizio

2) estratto_conto(imp: Impresa, inizio: Data, fine: Data): Reale >= 0
Il sistema deve poter calcolare, data un'impresa e un lasso di tempo, il costo totale 50 dei servizi utilizzati oltre le soglie gratuite dagli utenti a essa associati, nel periodo Indicato.

Precondizioni: inizio < fine
Postcondizioni:

- L'operazione non modifica il livello estensionale
- Il risultato *result* è così definito:

A = {a | Abbonamento(a) AND sottoscrive(imp, a) AND EXISTS ia, fa inizio(a, ia) AND fine(a, fa) and non_sovrapposti(inizio, fine, ia, fa, False) }

U = { (uc, sc, pu) | EXISTS s, acc, ute, abb, ta, fu
UtilizzoConcluso(uc) AND ut_serv(uc, s) and acc_util(uc, acc) AND ut_acc(acc, ute) and usufruisce(ute, abb) AND abb € A AND tip_abb(abb, ta) AND sconto(ta, s, sc) and prezzo_unita(s, pu) AND fine(uc, fu) and inizio <= fu <= fine }

Result = $\sum_{(uc,\, sc,\, pu) \in U} pu$ * uc.quantita_non_gratuita() * (1-sc)

4) Il servizio clienti vuole poter calcolare, dato un periodo, gli utenti associati ad un abbonamento valido che non hanno effettuato alcun accesso nel periodo dato.

utenti_no_accesso(inizio: DataOra, fine: DataOra): Utente [0..*]
Precondizioni: inizio < fine
Postcondizioni:

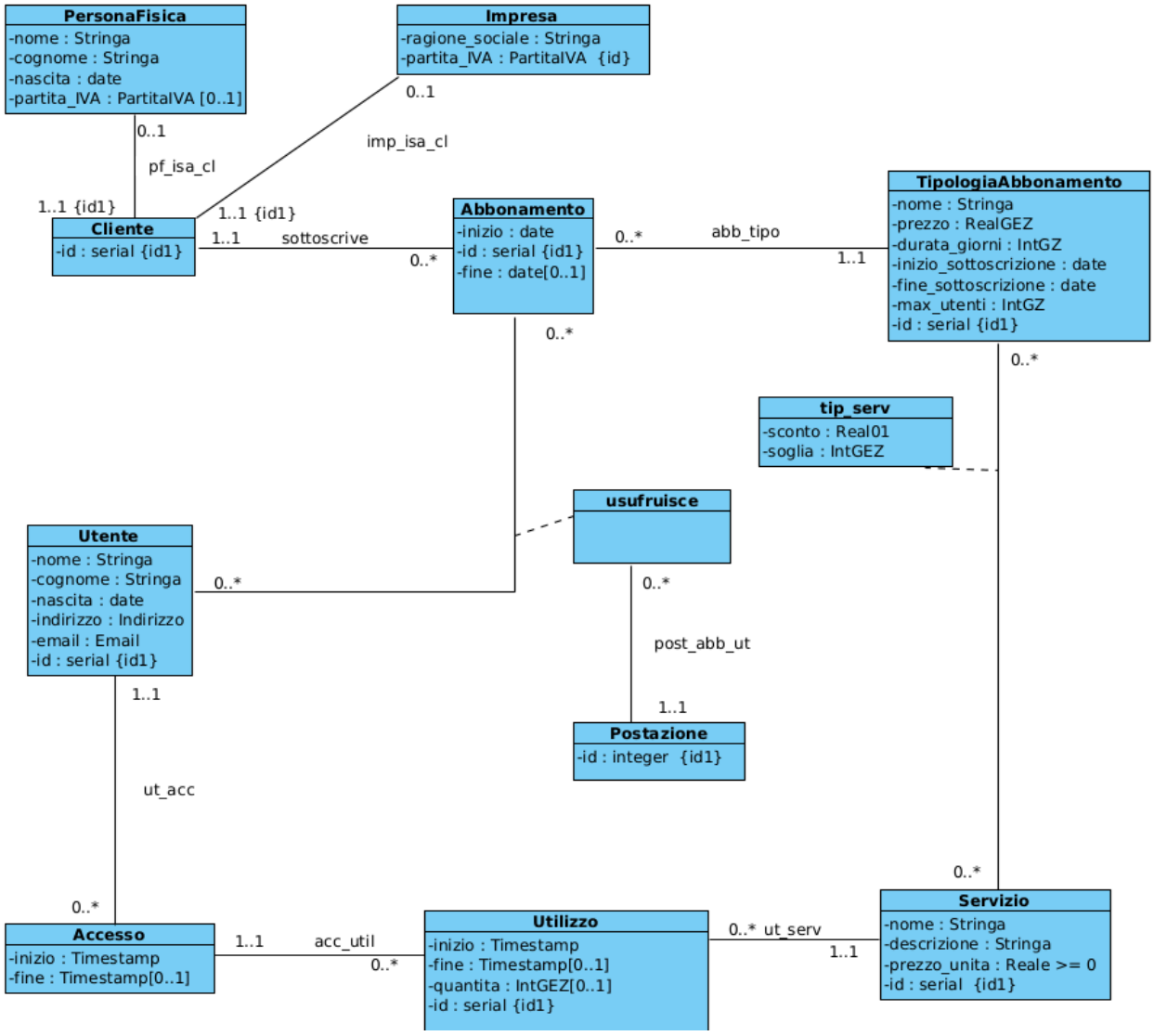
- L'operazione non modifica il livello estensionale
- Il risultato *result* è così definito:

result = { u | ∃ a, ia, fa usufruisce(u, a) AND inizio(a, ia) AND fine(a, fa) and
non_sovrapposti(inizio, fine, ia, fa, False) AND
NOT EXISTS acc, iacc, facc
ut_acc(u, acc) and inizio(acc, iacc) and fine(acc, facc) and
non_sovrapposti(inizio, fine, iacc, facc, False)
}
Alternativa equivalente
U = { u | EXISTS a, ia, fa usufruisce(u, a) AND inizio(a, ia) AND fine(a, fa) and
non_sovrapposti(inizio, fine, ia, fa, False) }

result = { u | u € U AND OT EXISTS acc, iacc, facc
ut_acc(u, acc) and inizio(acc, iacc) and fine(acc, facc) and
non_sovrapposti(inizio, fine, iacc, facc, False) }

2. Progettazione

2.1. Diagramma UML delle classi ristrutturato



3. Realizzazione

3.1. Schema Relazionale

Cliente(id: serial)

PersonaFisica(id:serial, nome: Stringa, …)
FK (id) ref Cliente(id)

Impresa(id:serial, ragione_sociale: Stringa, …)
FK (id) ref Cliente(id)

Abbonamento(id: serial, inizio: date, fine*: date, tipologia: integer) – accorpa att_tipo
FK (tipologia) ref TipologiaAbbonamento(id)
Check (fine is null or fine > inizio)

TipologiaAbbonamento(id: serial, …, max_utenti: IntGZ, …)

sottoscrive(cliente: integer, abbonamento: integer)
FK (cliente) ref Cliente(id)
FK (abbonamento) ref Abbonamento(id)

Servizio(id: serial, prezzo_unitario: RealGEZ, …)

tip_serv(tipologia: integer, servizio: integer, soglia: IntGEZ, sconto: Real01)
FK (tipologia) ref TipologiaAbbonamento(id)
FK (servizio) ref Servizio(id)

Utente(id: serial, …)

Postazione(id: serial)

usufruisce(utente:integer, abbonamento:integer, postazione: integer)
FK (abbonamento) ref Abbonamento(id)
FK (utente) ref Utente(id)
FK (postazione) ref Postazione(id)

Accesso(id: serial, inizio: timestamp, fine*: timestamp, utente: integer)
FK (utente) ref Utente(id)
Check (fine is null or fine > inizio)

Utilizzo(id: serial, inizio: timestamp, fine*: timestamp, accesso: integer, quantita: IntGEZ, servizio: integer)
FK (accesso) ref Accesso(id)
FK (servizio) ref Servizio(id)
Check (fine is null or fine > inizio)

3.2. Trigger

1) [V.Abbonamento.numero_usufruisce_minore_max]
Il numero di utenti che usufruiscono di un abbonamento è non superiore al numero massimo previsto dalla tipologia dell'abbonamento
ALL a, ta, m
    Abbonamento(a) AND tip_abb(a, ta) AND max_utenti(ta, m) =>
    | { u | Utente(u) and usufruisce(u, a) } | <= m

REVOKE UPDATE ON TIPOLOGIAABBONAMENTO.MAX_UTENTI
…

Evento intercettato: INSERT in *usufruisce*
Quando: AFTER
Nuova ennupla: *new*

Is_error =
SELECT *
FROM usufruisce u, tipologiaabbonamento ta, abbonamento a
WHERE new.abbonamento = u.abbonamento
And u.abbonamento = a.id and a.tipologia = ta.id
Group by ta.id, ta.max_utenti
HAVING Count(*) > ta.max_utenti

If is_error fai ROLLBACK, altrimenti COMMIT

3.3. Specifiche realizzative delle operazioni di use-case

4)
utenti_no_accesso(inizio: timestamp, fine: timestamp): Set<Integer>

```
Q =
      SELECT DISTINCT u.id
      FROM utente u, usufruisce us, abbonamento abb
      WHERE
             us.utente = u.id and us.abbonamento = abb.id
            And overlaps(inizio, fine, abb.inizio, abb.fine)
            And NOT EXISTS (
                  SELECT *
                  FROM accesso acc
                  WHERE acc.utente = u.id
                        and (inizio, fine)
                              overlaps (acc.inizio, acc.fine)
            )
      );

Q =
SELECT DISTINCT u.id
FROM usufruisce us, abbonamento abb,
      utente u LEFT OUTER JOIN
      (
            SELECT *
            FROM accesso acc
            WHERE (inizio, fine) overlaps (acc.inizio, acc.fine
      ) acc_periodo
      on u.id = acc_periodo.utente
WHERE
       us.utente = u.id and us.abbonamento = abb.id
      and (inizio, fine) overlaps (abb.inizio, abb.fine)
      And acc_periodo.id is null;
```