Università degli Studi di Roma "La Sapienza"
Facoltà di Ingegneria – Corso di Laurea in Ingegneria Gestionale
Corso di Progettazione del Software
Proff. Toni Mancini e Monica Scannapieco

Progetto **PI.20060324**, passo **P.1**

versione del 11 marzo 2007

Si vuole progettare un'applicazione che monitorizzi e gestisca gli orari lavorativi dei dipendenti dell'azienda *Paradise* ed assegni bonus e premi ai dipendenti che si dimostrano particolarmente dediti al lavoro, penalizzando i dipendenti indisciplinati.

Si richiede di svolgere la fase di Progetto del sistema secondo la metodologia vista nel corso. Si consideri come input lo schema concettuale prodotto nei passi A.1 e A.2.

Requisiti

L'azienda *Paradise* distingue due livelli per i dipendenti: il livello I, proprio delle categorie dirigenziali, ed il livello II proprio delle altre categorie. Ciascun dipendente è munito di una matricola, che ne consente l'identificazione nel contesto aziendale, di un indirizzo email, e della sede di lavoro presso cui presta normalmente servizio. L'azienda ha infatti diverse sedi dislocate in punti diversi della città, di ognuna delle quali interessa l'indirizzo.

I dipendenti di I livello sono tutti muniti di un cellulare aziendale (di cui se ne vuole memorizzare il numero), mentre quelli del II livello possono ottenerne uno come premio lavorativo.

Le singole giornate lavorate vengono chiamate *prestazioni lavorative giornaliere* del singolo dipendente. Di esse interessa il giorno a cui fanno riferimento. Durante ogni giornata lavorativa, al fine di monitorare la loro effettiva presenza in sede, è previsto che i singoli dipendenti timbrino il loro badge, sia in ingresso che in uscita. Ciascuna timbratura avviene in una certa ora e presso una delle sedi di lavoro (non obbligatoriamente quella presso cui il dipendente lavora normalmente). Queste informazioni devono essere memorizzate dal sistema.

Si osservi che i dipendenti possono effettuare più timbrature durante ogni giornata lavorativa (ad esempio, per uscire a prendere un caffè, il dipendente deve timbrare in uscita e ritimbrare al suo ingresso). Tuttavia vanno distinte la timbratura di inizio e quella di fine giornata. L'ora della timbratura di inizio non può essere precedente alle 7:45 e l'ora della timbratura di fine non può essere successiva alle 19:00.

Le procedure di timbratura del badge hanno come scopo quello di permettere al sistema di conoscere i dipendenti indisciplinati. Sono considerati indisciplinati i dipendenti di I livello che

hanno una percentuale di prestazioni giornaliere con pause (cioè con timbrature intermedie) maggiore dell'80%. Tale percentuale scende al 50% per i dipendenti di II livello.

I dipendenti di I livello partecipano a progetti aziendali. Di essi (di cui interessa la data di inizio, quella di fine, e il codice identificativo), interessa conoscere l'impegno, in ore/persona, dichiarato dai singoli partecipanti.

Il sistema deve offrire le seguenti funzionalità, utilizzate dall'Ufficio Personale:

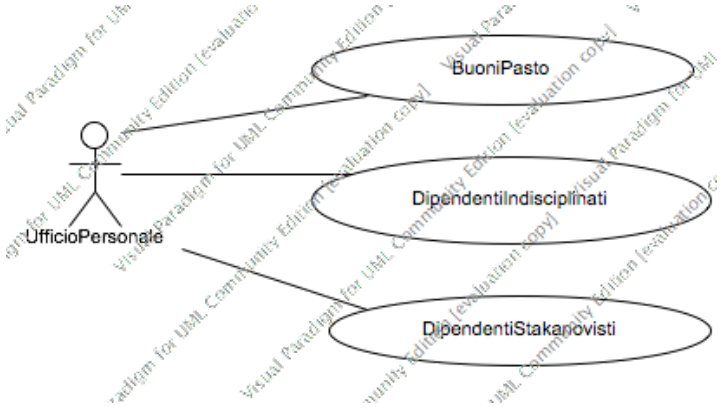
1. Dato un periodo temporale p (nella forma di 2 date) e un insieme di dipendenti S, si vuole restituire il sottoinsieme di S dei dipendenti di I livello che, nel periodo p, hanno lavorato di più a progetti (assumendo che l'impegno dichiarato per ogni progetto sia erogato in maniera uniforme durante il ciclo di vita di quel progetto).

2. Dato un insieme di dipendenti S si vuole calcolare il sottoinsieme di S dei dipendenti indisciplinati, cioè che superano le soglie di disciplina sopra descritte.

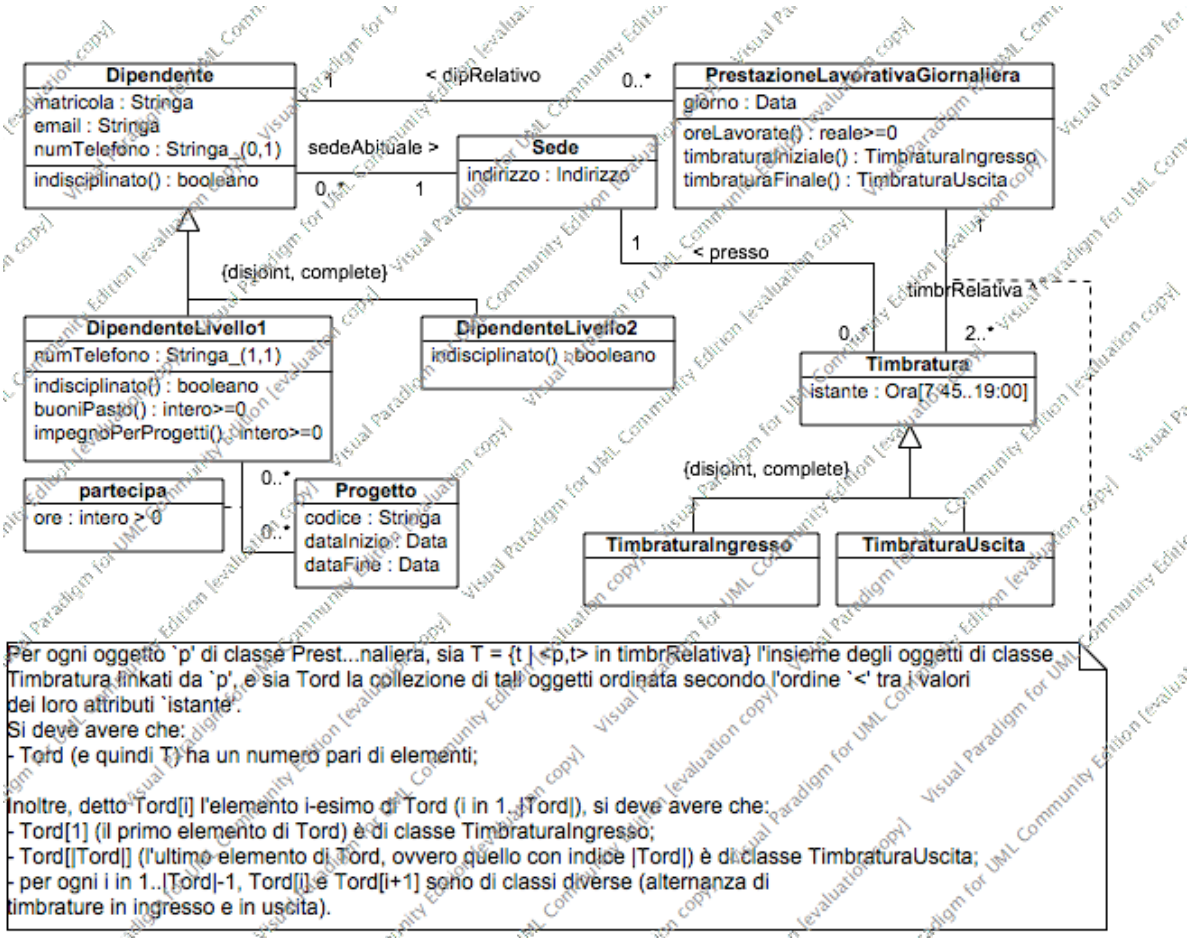
3. Dato un insieme di dipendenti S ed un periodo temporale p (nella forma di 2 date) si vuole restituire il numero totale di buoni pasto da erogare ai dipendenti in S. Si osservi che solo i dipendenti I livello hanno diritto a buoni pasto. Inoltre un buono pasto è erogato esclusivamente per le giornate lavorative di durata (*al netto* delle pause) superiore a 6 ore (ovvero 360 minuti).

1 Fase di Analisi

1.1 Diagramma degli Use Case



1.2 Diagramma delle classi UML



Nota: Il vincolo secondo il quale, per ogni oggetto `p` di classe PrestazioneLavorativaGiornaliera, il numero di link in `p.timbrRelativaA` è pari è una semplificazione (ciò non è ovviamente vero durante una giornata di lavoro). Tuttavia, dato che le funzionalità richieste al sistema sono esclusivamente relative a giornate di lavoro già trascorse, questa assunzione è valida.

Una modellazione più accurata prevederebbe un diagramma degli stati per la classe PrestazioneLavorativaGiornaliera, che regola il ciclo di vita dei suoi oggetti: gli stati sarebbero "DipendenteIn", e "DipendenteOut", con le ovvie transizioni (è un utile esercizio).

1.3 Specifica degli use case

```
SpecificaUseCase BuoniPasto
  totaleBuoniPasto(D: Insieme(Dipendente), dataIn: Data, dataFin:Data): intero >= 0
    pre: dataIn <= dataFin
    post: Detto D1 = { d in D | d e' di classe DipendenteLivello1 },
```

$$\texttt{result} = \Sigma_{\texttt{d}\in\texttt{D1}}\texttt{d.buoniPasto(dataIn, dataFin)}.$$

```
FineSpecifica

SpecificaUseCase  DipendentiIndisciplinati
  dipendentiIndisciplinati(D: Insieme(Dipendente), dataIn: Data, dataFin:Data):
         Insieme(Dipendente)
    pre: dataIn <= dataFin
    post:
       result = { d in D  |  d.indisciplinato(dataIn, dataFin) = true }.
FineSpecifica

SpecificaUseCase  DipendentiStakanovisti
  dipendentiStakanovisti(D: Insieme(Dipendente), dataIn: Data, dataFin:Data):
        Insieme(DipendenteLivello1)
    pre: dataIn <= dataFin
    post:
       result =  { d in D  |  d e' di classe DipendenteLivello1 e
                              per ogni d' in D tale che
                                 d' e' di classe DipendenteLivello1 e d != d',
                                    si ha che d.impegnoPerProgetti(dataIn, dataFin) >=
                                        d'.impegnoPerProgetti(dataIn, dataFin) }.
FineSpecifica
```

1.4 Specifica delle classi

La classe Dipendente

```
SpecificaClasse Dipendente
  indisciplinato(dataIn: Data, dataFin: Data): booleano
    pre: nessuna
    post:
```

```
      Detti
         LinkPrest =
           { lnk in this.dipRelativo  t.c.
                  lnk.PrestazioneLavorativaGiornaliera.giorno in [dataIn, dataFin] }

      e
         LinkPrestConPause =
            { lnk in LinkPrest  t.c.
                 |lnk.PrestazioneLavorativaGiornaliera.timbrRelativa| > 2 }

     result = true se e solo se  |LinkPrestConPause| / |LinkPrest| >= x,
     dove x dipende dalla sottoclasse piu' specifica di this.
```

La classe DipendenteLivello1

```
SpecificaClasse DipendenteLivello1 is-a Dipendente
  indisciplinato(dataIn: Data, dataFin: Data): booleano
    pre: nessuna
    post: quelle di Dipendente.indisciplinato(dataIn, dataFin) con x = 0.8

  buoniPasto(dataIn: Data, dataFin: Data): intero >= 0
    pre: dataIn <= dataFin
    post:
       result = | { prest in PrestazioneLavorativaGiornaliera  t.c.
                           <this, prest> in dipRelativo e
                           prest.giorno in [dataIn, dataFin] e
                           prest.oreLavorate() >= 6 } |

  impegnoPerProgetti(dataIn: Data, dataFin: Data): intero >= 0
    pre: dataIn <= dataFin
    post: Detto
        L = { l in this.partecipa  |
                        [l.Progetto.dataInizio, l.Progetto.dataFine] intersezione
                               [dataIn, dataFin] != insieme vuoto }

        l'insieme dei link di associazione 'partecipa' che coinvolgono this e progetti
        attivi nel periodo [dataIn, dataFin], result e' pari a:
```

$$\Sigma_{l \in L}\Big(\frac{\texttt{l.ore}}{\texttt{l.Progetto.dataFine.differenza(l.Progetto.dataInizio, GIORNI)}} \times$$

$$\texttt{min(l.Progetto.dataFine, dataFin).differenza(max(l.Progetto.dataInizio, dataIn), GIORNI)}\Big)$$

```
   dove il primo fattore e' l'impegno medio giornaliero di this verso il singolo
   progetto l.Progetto (cf. assunzione di impegno uniforme) e il secondo e' pari al numero
   di giorni in cui il singolo progetto l.Progetto e' attivo nel periodo [dataIn, dataFin].
FineSpecifica
```

La classe DipendenteLivello2

```
SpecificaClasse DipendenteLivello2 is-a Dipendente
  indisciplinato(dataIn: Data, dataFin: Data): booleano
    pre: nessuna
    post: quelle di Dipendente.indisciplinato(dataIn, dataFin) con x = 0.5
FineSpecifica
```

La classe PrestazioneLavorativaGiornaliera

```
SpecificaClasse PrestazioneLavorativaGiornaliera
  timbraturaIniziale(): TimbraturaIngresso
    pre: nessuna
    post:
       Detto T = { t in Timbratura t.c. <this, t> in timbrRelativaA }
       l'insieme delle timbrature effettuate nella prestazione lav. giorn. this,
       sia tmin in T l'oggetto tale che per ogni t in T, con t != tmin, si ha:
          t.istante >= tmin.istante.

       result = tmin.

       Si osservi che la garanzia che tmin sia di classe TimbraturaIngresso e' data
       dal vincolo imposto sul diagramma delle classi.

  timbraturaFinale(): TimbraturaUscita
    pre: nessuna
    post:
       Detto T = { t in Timbratura t.c. <this, t> in timbrRelativaA }
       l'insieme delle timbrature effettuate nella prestazione lav. giorn. this,
       sia tmax in T l'oggetto tale che per ogni t in T, con t != tmax, si ha:
```

```
         t.istante <= tmax.istante.

      result = tmax.

      Si osservi che la garanzia che tmax sia di classe TimbraturaUscita e' data
      dal vincolo imposto sul diagramma delle classi.

  oreLavorate(): reale >= 0
    pre: nessuna
    post:
       Detto T = { t in Timbratura t.c. <this, t> in timbrRelativaA }
       l'insieme delle timbrature effettuate nella prestazione lav. giorn. this,
       sia Tord la collezione degli oggetti in T ordinati secondo il valore
       dell'attributo 'istante'.
```

$\texttt{result} = \Sigma_{\texttt{i} \in \texttt{1..|Tord|-1 e dispari}} (\texttt{Tord[i+1].istante.differenza(Tord[i].istante, ORE)})$

```
  dove Tord[i+1] e Tord[i] sono rispettivamente gli elementi (i+1)-esimo e i-esimo della
  collezione ordinata Tord.
  Si osservi inoltre che Tord ha un numero pari di elementi, come previsto dal vincolo
  imposto sul diagramma delle classi.

FineSpecifica
```

2 Fase di Progetto

2.1 Corrispondenza tra tipi UML e tipi Java

Tipo UML	Tipo Java	Note
Stringa	String	–
Indirizzo	Indirizzo	cf. spec. realizzativa
Data	Data	disponibile, usiamo vers. senza side-effect, con condiv.
Ora[7:45..19:00]	Ora	Verifica ammissibilità sul lato server
intero $>/\geq 0$	int	Verifica ammissibilità sul lato server
reale ≥ 0	double	Verifica ammissibilità sul lato server
Insieme(...)	HashSet< ... >	Implementa l'interfaccia Set< ... >

2.2 Specifica realizzativa delle strutture dati

Indirizzo: la solita.

2.3 Specifica realizzativa delle classi

La classe Dipendente

```
SpecificaClasse Dipendente
 +abstract indisciplinato(dataIn: Data, dataFin: Data): boolean;

 #indisciplinato(dataIn: Data, dataFin: Data, x: double): boolean
    pre: nessuna
    algoritmo:
       n = 0;
       m = 0;
       per ogni 'lnk' in this.dipRelativo {
         se lnk.PrestazioneLavorativaGiornaliera.giorno in [dataIn, dataFin]
           allora {
             n++;
             se |lnk.PrestazioneLavorativaGiornaliera.timbrRelativa| > 2
               allora m++;
           }
       }
       ritorna (n/m >= x);
```

La classe DipendenteLivello1

```
SpecificaClasse DipendenteLivello1 is-a Dipendente
 +indisciplinato(dataIn: Data, dataFin: Data): boolean
    pre: nessuna
    algoritmo: ritorna super.indisciplinato(dataIn, dataFin, 0.8);

 +buoniPasto(dataIn: Data, dataFin: Data): int
    pre: dataIn.prima(dataFin) o dataIn = dataFin;
    algoritmo:
       result = 0;
       per ogni 'l' in this.dipRelativo {
         prest = l.PrestazioneLavorativaGiornaliera;
```

```
          se prest.giorno in [dataIn, dataFin] e prest.oreLavorate() >= 6
            allora result++;
       }
       ritorna result;

 +impegnoPerProgetti(dataIn: Data, dataFin: Data): int
    pre: dataIn.prima(dataFin) o dataIn = dataFin;
    algoritmo:
        per ogni 'l' in this.partecipa {
           p = l.Progetto;
           se [p.dataInizio, p.dataFine] intersez [dataIn, dataFin] != ins. vuoto
             allora {
                // il progetto e' attivo nel periodo da considerare
                durataP = p.dataFine.differenza(p.dataInizio, GIORNI);

                inizioLav = max(p.dataInizio, dataIn);
                fineLav = min(p.dataFine, dataFin);
                durataLav = fineLav.differenza(inizioLav, GIORNI);

                result +=  (l.ore / durataP) * durataLav;
           }
      }
FineSpecifica
```

La classe DipendenteLivello2

```
SpecificaClasse DipendenteLivello2 is-a Dipendente
 +indisciplinato(dataIn: Data, dataFin: Data): boolean
    pre: nessuna
    algoritmo: ritorna super.indisciplinato(dataIn, dataFin, 0.5).
FineSpecifica
```

La classe PrestazioneLavorativaGiornaliera

```
SpecificaClasse PrestazioneLavorativaGiornaliera
 +timbraturaIniziale(): TimbraturaIngresso
    pre: nessuna
    algoritmo:
       LTord = Lista che contiene gli elementi di this.timbrRelativaA ordinati secondo
```

```
           i valori dell'attributo 'istante' dell'oggetto linkato;
       ritorna LTord[1].Timbratura;

 +timbraturaFinale(): TimbraturaUscita
    pre: nessuna
    algoritmo:
       LTord = Lista che contiene gli elementi di this.timbrRelativaA ordinati secondo
           i valori dell'attributo 'istante' dell'oggetto linkato;
       ritorna LTord[|LTord|].Timbratura;

 +oreLavorate(): double
    pre: nessuna
    algoritmo:
       LTord = Lista che contiene gli elementi di this.timbrRelativaA ordinati secondo
           i valori dell'attributo 'istante' dell'oggetto linkato;
       result = 0;
       i=1;
       finche' (i < |LTord|) {
            result += LTord[i+1].Timbratura.istante.differe-
                    nza(LTord[i].Timbratura.istante, ORE);
            i = i+2;
       }
       ritorna result;
FineSpecifica
```

2.4 Specifica realizzativa degli use-case

```
SpecificaUseCase BuoniPasto
 +totaleBuoniPasto(D: Set<Dipendente>, dataIn: Data, dataFin:Data): int
    pre: dataIn <= dataFin
    algoritmo:
       result = 0;
       per ogni 'd' in D {
         se d e' di classe DipendenteLivello1
           allora result += d.buoniPasto(dataIn, dataFin);

       ritorna result
FineSpecifica
```

```
SpecificaUseCase  DipendentiIndisciplinati
 +dipendentiIndisciplinati(D: Set<Dipendente>, dataIn: Data, dataFin:Data): Set<Dipendente>
    pre: dataIn <= dataFin
    algoritmo:
       result = { d in D  |  d.indisciplinato(dataIn, dataFin) = true }.
FineSpecifica

SpecificaUseCase  DipendentiStakanovisti
 +dipendentiStakanovisti(D: Set<Dipendente>, dataIn: Data, dataFin:Data):
                                                                    Set<DipendenteLivello1>
    pre: dataIn <= dataFin
    algoritmo:
       result = insieme vuoto di oggetti di classe DipendenteLivello1;
       max = 0;
       per ogni 'd' in D {
         se d e' di classe DipendenteLivello1
           allora {
             imp = d.impegnoPerProgetti(dataIn, dataFin);
             se imp > max {
                result = {d};
                max = imp;
             }
             altrimenti, se imp = max allora result = result U {d};
           }
       }
       ritorna result;
FineSpecifica
```

2.5 Progetto dei diagrammi degli stati

Non sono stati definiti diagrammi degli stati in fase di Analisi.

2.6 Responsabilità sulle associazioni

Dai requisiti, dalla specifica delle operazioni di classi e di use case, e delle molteplicità nel diagramma delle classi emerge che:

Associazione	Classe	Ha resp?	Motivo
dipRelativo	Dipendente	SI	u.c. op. 2 & 3
	Prestaz...	SI	vincolo 1..1
sedeAbituale	Dipendente	SI	vincolo 1..1
	Sede	NO	–
presso	Timbratura	SI	vincolo 1..1
	Sede	NO	–
timbrRelativaA	Prestaz...	SI	vincolo 2..*
	Timbratura	SI	vincolo 1..1
partecipa	DipendenteLivello1	SI	u.c. op. 1
	Progetto	NO	–

2.7 Vincoli sull'evoluzione delle proprietà mutabili

Tutte le proprietà mutabili possono variare arbitrariamente, con le seguenti eccezioni:

- I link di associazione *timbrRelativaA* non possono essere eliminati;
- I link di associazione *dipRelativo* non possono essere eliminati;
- Il valore per l'attributo *numTelefono* degli oggetti di classe *DipendenteLivello2* non è noto alla nascita.

2.8 Diagramma delle classi realizzativo