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 **PC.20080110**

versione del 11 gennaio 2008

Si vuole progettare e realizzare *QuickHospital*, un sistema informatico per la gestione di ricoveri e di visite mediche in un ospedale estremamente efficiente. Il sistema deve permettere la memorizzazione e gestione dei pazienti e dei relativi ricoveri ospedalieri e prenotazioni di visite ambulatoriali, nonché degli itinerari di visita dei medici dell'ospedale.

Si richiede di effettuare le fasi di Analisi, Progetto, e Realizzazione del sistema in JAVA, utilizzando la metodologia illustrata nel corso.

Requisiti

Il sistema *QuickHospital* deve permettere di memorizzare e gestire informazioni circa i pazienti e i medici dell'ospedale nel quale viene installato. In particolare, dei pazienti interessano alcune informazioni anagrafiche (nome, cognome e data di nascita) ed i loro recapiti, distinti in recapiti telefonici, recapito email e postale (questi ultimi unici).

Per quanto riguarda i medici dell'ospedale invece, interessa mantenere informazioni sul loro nome, cognome e data di nascita, ed i pazienti che hanno in cura.

Un paziente può essere ricoverato, in una certa data, solo se una precedente verifica della disponibilità dei posti letto presenti nell'ospedale ha dato esito positivo.

Una volta effettuato il ricovero, il paziente ha assegnato un posto letto nell'ambito di una stanza; una stanza può contenere da un minimo di 1 ad un massimo di 8 posti letto. Le stanze hanno un piano ed un settore (interi positivi).

Il sistema deve inoltre permettere la memorizzazione dello storico di tutti i pazienti che sono stati ricoverati e poi dimessi nel tempo, con le informazioni relative ai posti letto occupati durante i diversi ricoveri.[1]

Sono funzionalità specifiche del sistema la registrazione del ricovero di un paziente e della sua dimissione ad opera del personale di accettazione. Inoltre il sistema deve assistere i medici ottimizzando il loro percorso di visite.

[1] Si assuma per semplicità che durante il periodo di un ricovero il paziente non possa cambiare letto.

In particolare, il sistema deve permettere di calcolare, su richiesta di un medico, il suo itinerario delle visite, ovvero un insieme ordinato delle stanze cui accedere (che sono tutte e sole le stanze che ospitano i pazienti che ha in cura).

L'ordinamento è dato in primo luogo dal piano delle stanze dei pazienti da visitare, ed in secondo luogo dal settore di appartenenza di tali stanze (entrambi in ordine crescente). I settori sono infatti numerati secondo un criterio di vicinanza topologica. Pertanto se un dato medico deve visitare le stanze $\{(7,4),(7,1),(1,3),(1,1),(3,4)\}$ dove la prima componente di ognuna è il piano e la seconda il settore, l'itinerario di visita proposto deve essere $[(1,1),(1,3),(3,4),(7,1),(7,7)]$.

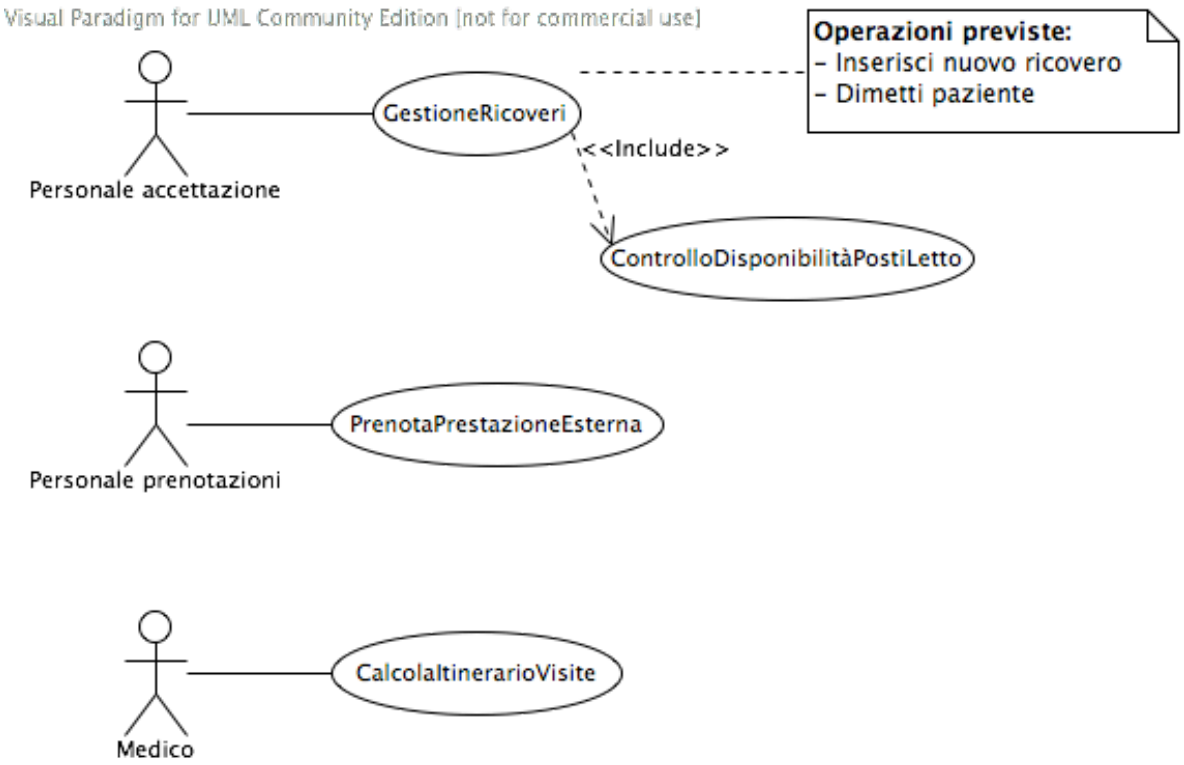
Oltre ai pazienti dell'ospedale, il sistema gestisce anche prestazioni mediche fatte da medici dell'ospedale a pazienti esterni. L'anagrafica di tali pazienti è registrata nel sistema (ad opera del personale addetto alle prenotazioni), con l'informazione aggiuntiva della particolare prestazione medica richiesta al personale ospedaliero (oltre che la data richiesta). Le prestazioni sono caratterizzate da una specializzazione richiesta (ad., ortopedia, dermatologia, ecc.) e una descrizione più estesa.

Di ogni medico il sistema deve conoscere la sua specializzazione primaria e le sue specializzazioni secondarie.

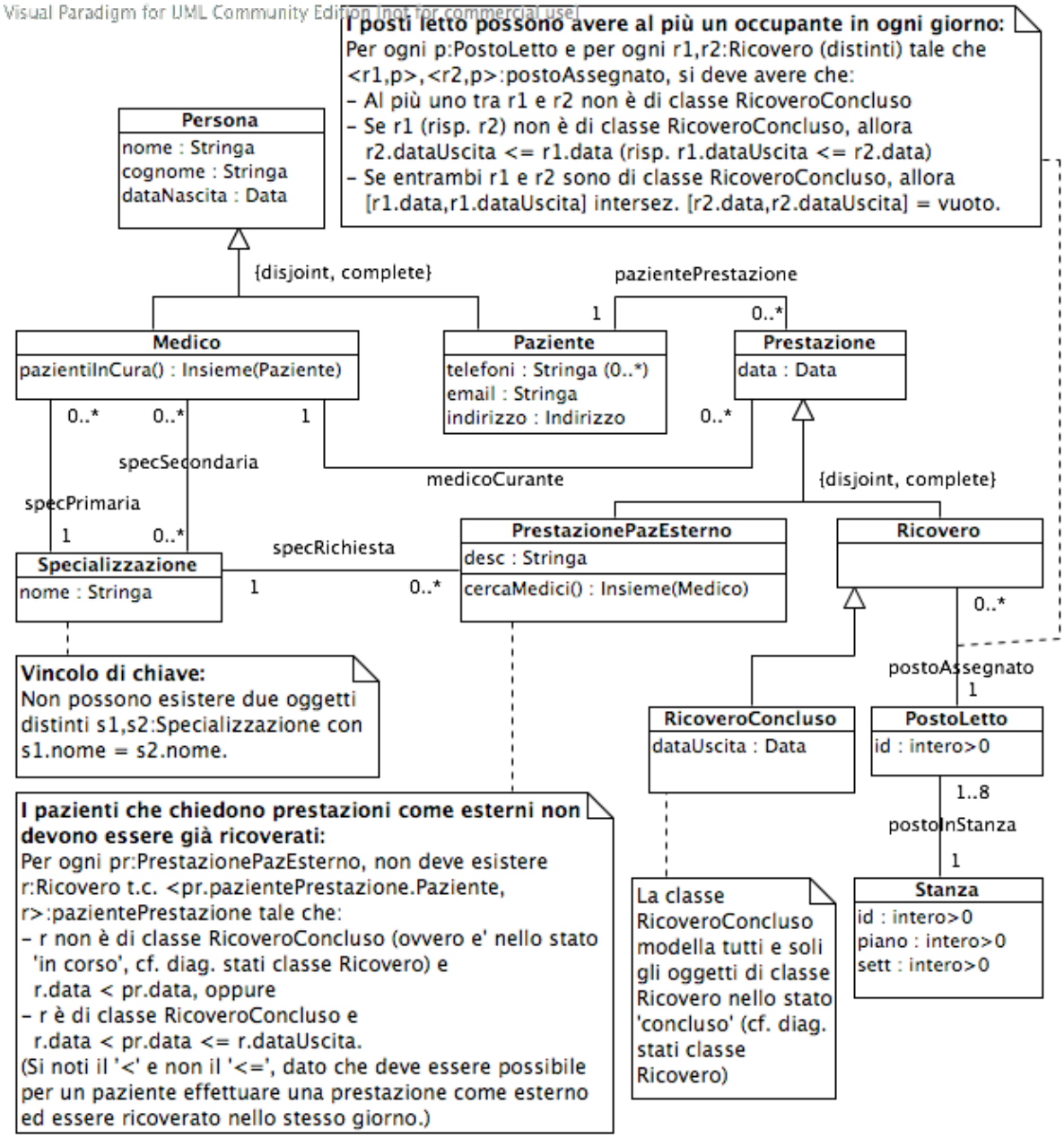
Data una prestazione richiesta da un paziente esterno (per una specializzazione s), il sistema deve restituire l'insieme dei medici maggiormente idonei a soddisfarla. Il criterio di idoneità è il seguente: se esistono medici con specializzazione primaria pari ad s, il risultato è l'insieme di tali medici. Altrimenti, il risultato è l'insieme dei medici che hanno s tra le loro specializzazioni secondarie.

1 Fase di Analisi

1.1 Diagramma degli Use Case



1.2 Diagramma delle classi UML



1.3 Specifica dei tipi di dato

Nessun tipo di dato definito

1.4 Specifica degli use case

```
SpecificaUseCase ControlloDisponibilitaPostiLetto
  postiDisponibili(): Insieme(PostoLetto)
    pre: nessuna
    post: result = { p:PostoLetto t.c. non esiste alcun link
                         <p,r> con r nello stato 'in corso' }.
FineSpecifica

SpecificaUseCase GestioneRicoveri
  inserisciNuovoRicovero(p:Paziente): Ricovero
    pre: p non e' gia' ricoverato, ovvero non esiste alcun oggetto r:Ricovero tale che
           r.pazienteRicovero = p e r e' nello stato 'in corso'.
           Inoltre, ControlloDisponibilitaPostiLetto.postiDisponibili() != vuoto
    post: result e' pari ad un nuovo oggetto di classe piu' specifica Ricovero, con:
             - result.pazienteRicovero = p;
             - result.dataIngresso = oggi
                (istanza del tipo Data relativa alla data corrente)
             - result.postoAssegnato e' pari ad un elemento qualsiasi di
               ControlloDisponibilitaPostiLetto.postiDisponibili().

  dimettiPaziente(p:Paziente)
    pre: esiste un oggetto r:Ricovero tale che r.pazienteRicovero = p e
           r e' nello stato 'in corso'
    post: viene generato l'evento r.registraTerm. Di conseguenza,
             r passa nello stato 'concluso' e all'attributo dataUscita
             viene assegnato il valore 'oggi'.
FineSpecifica

SpecificaUseCase PrenotaPrestazioneEsterna
  prenota(p:Paziente, s:Specializzazione, d:Data, desc:Stringa): PrestazionePazEsterno
    pre: d > oggi;
         inoltre, p non e' gia' ricoverato, ovvero non esiste alcun oggetto
         r:Ricovero in p.pazienteRicovero tale che
         r non e' di classe RicoveroConcluso e r.dataIngresso < d, oppure
```

```
        (cf. vincolo nel diagramma delle classi, e si noti la semplificazione possibile
        grazie all'aggiunta della precondizione 'd > oggi' e alla semantica
        dell'operazione GestioneRicoveri.dimettiPaziente(), che fissa ad 'oggi'
        la data di uscita)

    post: result e' pari ad un nuovo oggetto di classe PrestazionePazEsterno con:
            - result.pazientePrestazione = p
            - result.data = d
            - result.desc = desc
            - result.specRichiesta = s
FineSpecifica

SpecificaUseCase CalcolaItinerarioVisite
  itinerario(m:Medico): Lista(Stanza)
    pre: nessuna
    post: Detto
             stanze = { s:Stanza  |   esiste r:Ricovero tale che:
                                      - r.medicoCurante = m
                                      - r e' nello stato 'in corso'
                                      - r.postoAssegnato.postoInStanza = s }

            l'insieme delle stanze che il medico m deve visitare,
            result e' pari ad una lista ordinata contenente tutti e soli gli elementi
            di 'stanze'.
            L'ordinamento e' tale che detti s1 e s2 due elementi in result, si ha che
            s1 occorre prima di s2 nella lista se e solo se:
            - s1.piano < s2.piano, oppure
            - s1.piano = s2.piano e s1.sett < s2.sett
FineSpecifica
```

1.5 Specifica delle classi e diagrammi degli stati e transizioni

La classe Medico

```
SpecificaClasse Medico
  pazientiInCura(): Insieme(Paziente)
    pre: nessuna
    post: result = { p:Paziente  |   esiste r:Ricovero tale che:
                                       - r.medicoCurante = this
```

```
                                        - r e' nello stato 'in corso'
                                        - r.pazienteRicovero = p }
FineSpecifica
```

La classe PrestazionePazEsterno

```
SpecificaClasse PrestazionePazEsterno
  cercaMedici(): Insieme(Medico)
    pre: nessuna
    post: se this.specRichiesta.specPrimaria != vuoto, allora
               result = {m:Medico  |  <m,s>:specPrimaria }
             altrimenti
               result = {m:Medico |  <m,s>:specSecondaria}
FineSpecifica
```

La classe Ricovero Gli oggetti della classe Ricovero evolvono secondo il seguente diagramma degli stati e transizioni: