SAPIENZA Università di Roma
Facoltà di Ingegneria - Corso di Laurea in Ingegneria Informatica
Corso di Progettazione del Software A.A. 2008/2009
Prova al calcolatore del 5 giugno 2009

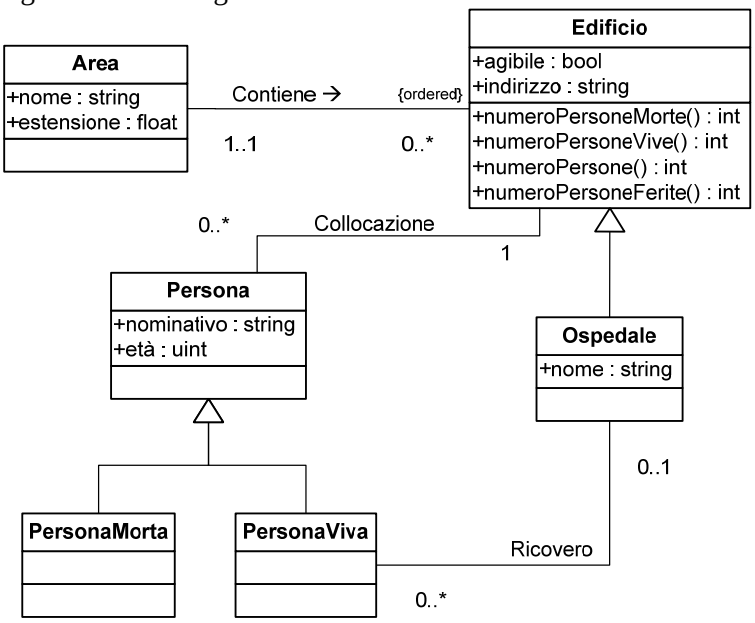
L'applicazione da progettare riguarda la gestione dei terremoti che possono verificarsi in una data zona. L'applicazione ha lo scopo di eseguire un'insieme di attività, in un dato ordine, le quali hanno lo scopo di:

- dare il primo supporto alle popolazioni colpite,
- salvare le persone intrappolate negli edifici,
- dirigere le persone agli ospedali più vicini,
- effettuare una prima analisi sull'agibilità degli edifici

La zona da supervisionare è divisa in aree, ognuna delle quali contiene un insieme di edifici, alcuni dei quali sono ospedali. Per ogni edificio di ogni area, viene fatto un sopralluogo per valutarne l'agibilità e per determinare il numero di persone che vi sono rimaste intrappolate. Le persone rimaste negli edifici possono essere vive oppure defunte. Una persona viva può essere ferita, nel qual caso viene trasportata in un ospedale per una diagnosi.

Il processo di supervisione prevede che un operatore della Protezione Civile inserisca le informazioni sulle aree colpite; tali informazioni vengono successivamente memorizzate nel sistema. Successivamente, viene gestita l'emergenza nelle diverse aree identificate. Per velocizzare le operazioni, le aree possono essere gestite in parallelo da diversi operatori. Inizialmente l'operatore addetto, non appena è in grado di gestire una certa area, inserisce la lista degli edifici da controllare. Successivamente viene effettuato un sopralluogo per ogni edificio, valutandone l'agibilità, determinando le persone defunte e vive ed, eventualmente, decidendo gli ospedali presso cui ricoverare i feriti. Il capo della Protezione Civile è, inoltre, interessato ad effettuare alcune statistiche: la frazione di defunti (il numero di persone defunte diviso quello di persone presenti negli edifici) e la frazione dei feriti (il numero delle persone ferite diviso il numero complessivo). Una persona è considerata ferita se è stata ricoverata in qualche ospedale.

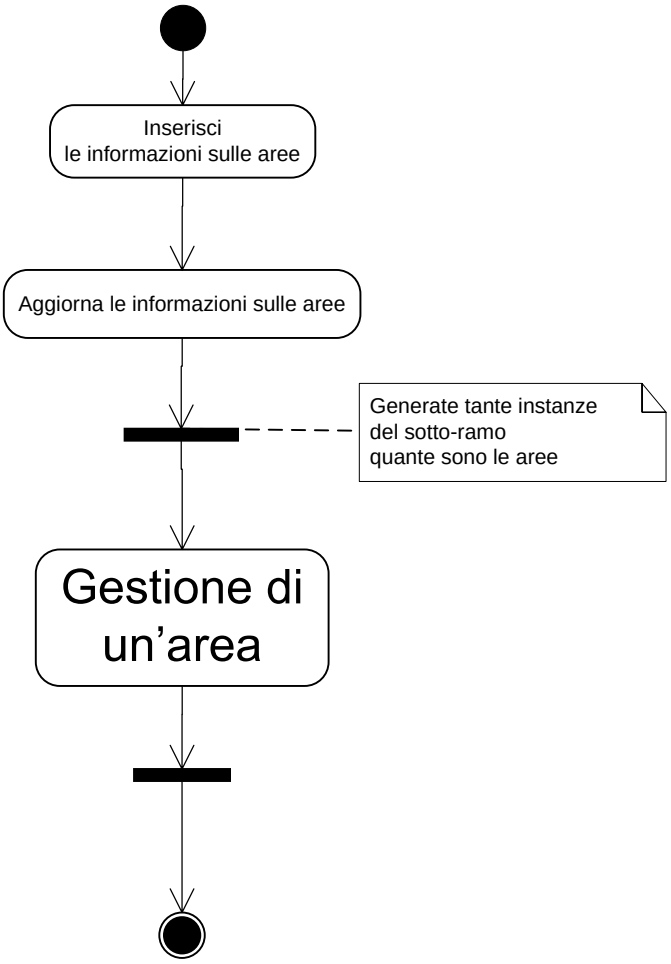
Il diagramma delle classi è raffigurato in Figura 1, insieme alla descrizione della responsabilità sulle associazioni e gli invarianti da garantire



Associazione	Classe	Resp.
Contiene	Area	SI
	Edificio	SI
Collocazione	Edificio	SI
	Persona	SI
Ricovero	PersonaViva	SI
	Ospedale	NO

```
context PersonaViva
inv self.Ricovero->empty() or
  (not self.Ricovero->empty() and
  self.Ricovero.Ospedale.agibile=true)
```

Figura 1: Diagramma delle Classi, Responsabilità ed invarianti




```
VariabiliAttività GestioneEmergenza
    var: aree: Set<RecordArea>
Fine

InizioSpecificaAttivitàIO
    operazione: inserisciInfoAree
    in:  --
    out: areeIO: List<RecordArea>


InizioSpecificaAttività aggiornaInfoAree
                    (areeIO: List<RecordArea>)
    pre:
    post: areeIO->forall(a |
            Area.allInstance()->
              exists(b | a.Nome=b.Nome
              && a.Estensione = b.Estensione))
```

Figura 2: Diagramma dell'attività principale e specifica OCL

Tutte le istanze di classi del diagramma vengono aggiunte al diagramma degli oggetti attraverso i metodi statici della classe **emergenza.Modello**. Tale classe fornisce anche i metodi necessari a reperire le istanze aggiunte in precedenza oppure tutte quelle di una data classe. In particolare, la classe assume che non possano esistere due aree con lo stesso nome, due edifici con lo stesso indirizzo, due persone con lo stesso nominativo oppure due ospedali con lo stesso nome. Maggiori dettagli sulla classe Modello sono disponibili come commenti nel codice stesso.I metodi dichiarati nella classe **emergenza.Edificio** possono essere specificati in OCL come segue:

```
InizioSpecificaOperazioniClasse Edificio
Context Edificio::numeroPersoneVive()
pre: --
post: result = self.Collocazione.Persona->
             select(p | p.oclIsKindOf(PersonaViva))->size()
Context Edificio::numeroPersoneMorte()
pre: --
post: result = self.Collocazione.Persona->
             select(p | p.oclIsKindOf(PersonaMorta))->size()
Context Edificio::numeroPersone ()
pre: --
post: result = self.Collocazione.Persona -> size()
Context Edificio::numeroPersoneFerite()
pre: --
post: result = self.Collocazione.Persona->
             select(p | p.oclIsKindOf(PersonaViva) and p.Ricovero->size()=0)->size()
```

La classe cliente **gui.OperazioniUtente** fornisce due metodi utilizzati per effettuare alcune statistiche:

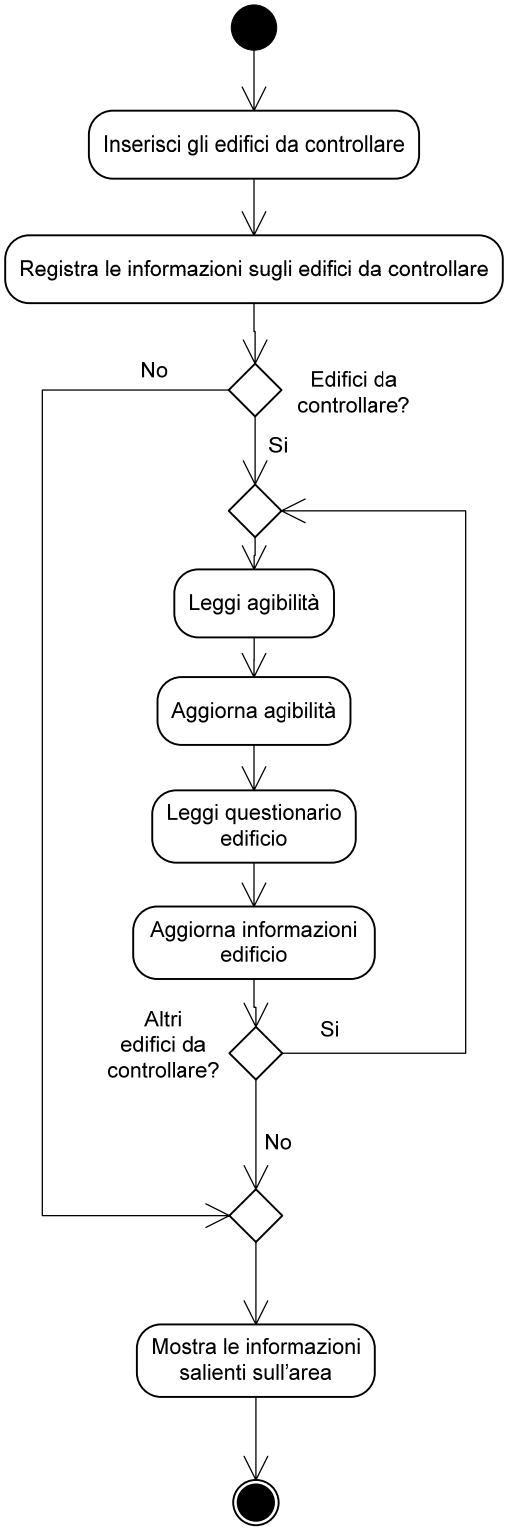
```
InizioSpecificaOperazioniClasse OperazioniUtente
Context OperazioniUtente::percentualiMorti(a: Area)
pre: --
post: result = let NumeroMorti = (a.Contiene.Edificio^numeroPersoneMorte())->sum()
               in
                let NumeroTotale = (a.Contiene.Edificio^numeroPersone())->sum()
                in
               result = NumeroMorti/NumeroTotale
Context OperazioniUtente::percentualiFeriti(a: Area)
pre: --
post: result = let NumeroFeriti = (a.Contiene.Edificio^numeroPersoneFerite())->sum()
               in
               let NumeroVivi = (a.Contiene.Edificio^numeroPersoneVive())->sum()
               in
               result = NumeroFeriti/NumeroVivi
```

Il diagramma delle attività è rappresentato in Figura 2. L'attività **Gestione di un area** è composita e il corrispondente diagramma delle attività è rappresentato in Figura 3. Il diagramma delle attività utilizza alcuni tipi di dato:

- **RecordArea** che definisce due proprietà:
  - **nome: String**
  - **estensione: float**
- **RecordPersona** che definisce due proprietà
  - **nominativo: String**
  - **eta: int**
  - **vivo: boolean**
  - **ospedale: nome**
    - La proprietà **ospedale** può valere anche *null* se la persona in questione è morta oppure è viva ma non ferita. Ovviamente non può accadere che esista una persona per cui `vivo=false` e `ospedale!=null`

Tutte le classi che codificano il diagramma delle attività sono attestate nel package **processo**. Le attività di I/O e i task sono codificate in classi che hanno le stesso nome di quello usato nella specifica OCL. Il diagramma in Figura 2 è codificato nella classe **processo.GestioneEmergenza** e il sotto-diagramma è codificato nella classe **processo.GestioneArea**. Quest'ultimo definisce un costruttore che prende come input un oggetto **RecordArea** che identifica l'area di interesse.

- **Al fine di superare la prova al calcolatore, lo studente deve completare le funzionalità mancanti modificando opportunamente il codice. Il risultato dell'elaborazione degli studenti dovrà essere memorizzato nel disco di rete T: . Qualsiasi codice salvato in posizioni diverse non verrà preso in considerazione. Se il codice non compila, lo studente è automaticamente ritenuto "non idoneo".**
- **Al fine di modificare il codice fornito per implementare le funzionalità, si suggerisce di intervenire esclusivamente sulle seguenti classi:**
  - **emergenza.PersonaViva**
  - **emergenza.Edificio**
  - **emergenza.ManagerContiene**
  - **emergenza.TipoLinkRicovero**
  - **processo.GestioneArea**
  - **processo.AggiornaAgibilita**
  - **processo.AggiornaInfoAree**
  - **gui.PrincipaleListener**
  - **gui.OperazioneUtente**



Ogni instanza dell'attività GestioneArea prende come input un RecordArea che serve ad identificare l'area su cui si agisce.

```
VariabiliAttività GestioneArea
    var: area: RecordArea
    var: edifici: List<String>
         /* La lista degli indirizzi degli
            edifici nell'area*/
Fine

InizioSpecificaAttivitàIO
    operazione: inserisciEdifici
    in:  nomeArea: String
    out: edifIO: List<String>
    operazione: leggiAgibilita
    in:  indirizzo: String
    out: agibile: boolean
    operazione: leggiQuestionario
    in:  indirizzo: String
    out: recordList: List<RecordPersona>
    operazione: mostraInfoSalienti
    in:  recordArea: RecordArea
    out: --
Fine

InizioSpecificaAttività registraInfoEdifici
                      (edifIO: List<String>,
                       area: RecordArea)
    pre:
    post: edifIO->forall(indir |
            Edificio.allInstances()->
              exists(b | indir = b.indirizzo &&
              b.Contiene.Area.nome=area.nome))
Fine

InizioSpecificaAttività aggiornaAgibilita
                      (indirizzo: String,
                       agibilita:boolean)
    pre:
    post: Edificio.allInstances()->exists(e |
            e.indirizzo=indirizzo &&
            e.agibilita=agibilita)
Fine

InizioSpecificaAttività aggiornaInfoEdifici
                      (indirizzo: String)
                      (quest:RecordQuestionario)
    pre:
    post: associa ogni persona ad un edificio
          e, se vivo e ferito, associa tale
          persona all'ospedale
Fine
```

Figura 3: Diagramma dell'attività di **Gestione Area** e specifica OCL