Sapienza
Università di Roma

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

anno accademico 2008-2009

Relazione finale su progetto interno

Gestione Centri per Anziani

Candidato:
Giovanni Murru
799913

Relatore:
Prof. Paolo Liberatore

Alla mia famiglia

Indice

Introduzione

La possibilità di rappresentare la realtà che ci circonda attraverso un linguaggio ben definito, di riuscire a plasmarla attraverso la programmazione, ha donato all'essere umano l'abilità di creare una realtà parallela a quella reale, che oggi conosciamo con il nome di realtà virtuale.

Nel parallelo universo di informazioni digitali la necessità di gestire grandi moli di dati è in continuo aumento e, dal punto di vista della memorizzazione e della interrogazione, acquisiscono sempre più importanza i sistemi di gestione di basi di dati *(i.e. DBMS)*, che permettono un *data management* più efficiente e più economico.

Quello che in origine era considerato un lusso esclusivo dei sistemi server, oggi assume un ruolo determinante in vari campi dello sviluppo software, dati i suoi notevoli vantaggi nell'amministrazione, nell'accesso concorrente, nell'integrità e nella sicurezza. Un'ampia fetta di applicazioni di utilizzo quotidiano: dal riproduttore musicale (*e.g. amarok* in *KDE®* utilizza *MySQL®* per memorizzare le informazioni della libreria) alle applicazioni di gestione per le fotografie digitali (*e.g. iPhoto®* della società californiana *Apple*), usa sistemi software di questo tipo. Quindi, risulta evidente che lo *storage* delle informazioni in una base di dati piuttosto che in un file system, è considerato *più robusto, affidabile e soprattutto efficiente*. Quest'ultima qualità risulta irrefutabile data la particolare natura relazionale dei DBMS di maggiore diffusione. Dalla possibilità di mettere in relazione le informazioni scaturisce infatti la capacità di effettuare interrogazioni complesse sui dati.

L'importanza e l'attualità, accompagnati dall'interesse personale per questi argomenti hanno determinato la scelta della materia di progettazione. Tale argomento, come il lettore attento avrà già intuito, riguarda le basi di dati, nella fattispecie la realizzazione di un'applicazione scritta nel linguaggio multi-piattaforma Java, che s'interfaccia, tramite il driver JDBC, con il DBMS relazionale MySQL. Nella stesura di questa relazione vengono illustrati gradualmente i passaggi che hanno guidato la realizzazione di tale applicazione, cercando di esporre in modo sintetico i punti cardine del lavoro svolto.

Si inizia da una fase di analisi in cui si raccolgono i requisiti dell'applicazione studiandone la fattibilità, per poi passare alla realizzazione di uno schema concettuale astratto (*entity-relationship model*). Approvato tale modello si parte con la progettazione logica dell'applicazione tramite la realizzazione di uno schema concettuale che descrive in modo accurato le relazioni. In seguito si attua una fase intermedia di ottimizzazione in cui, con

accorpamenti, eliminazione o introduzione di ridondanze si ristruttura lo schema logico. Sulla base di questo inizia la scrittura del codice struttura, che comprende la creazione del database MySQL e il vincolamento di tale database tramite l'aggiunta di *foreign key* e *stored procedure*, queste ultime associate ad opportuni *trigger*.

Dopo una attenta analisi delle responsabilità delle entità sulle relazioni, si procede ad una fase di progettazione del software secondo le regole dell'omonimo corso. Il dialogo tra l'applicativo java e il sottostrato sql viene gestito tramite l'utilizzo del driver JDBC per MySQL e di opportune transazioni. Viene infine completato il lavoro attraverso l'implementazione di una semplice interfaccia grafica per l'interazione con l'utente.

Parte I

Analisi

Capitolo 1

Raccolta dei requisiti

Di seguito alla decisione di sviluppare un programma di utilità sociale, e in accordo sul fatto che tale software dovesse riguardare in primo luogo il ruolo degli anziani nella città di Roma, si è pensato di valorizzare la figura dei centri anziani che svolgono un importante azione nella Città, realizzando un software per facilitare la gestione delle informazioni a loro associate.

Secondo i principi dell'ingegneria del software si è studiata la fattibilità del progetto e si è poi stilato un documento sui requisiti del programma per meglio identificare le parti salienti. Procediamo quindi con una schematizzazione delle entità d'interesse sottolineandone gli attributi e le relazioni.

1.1 La realtà d'interesse

S'intende sviluppare un'applicazione relativa alla gestione dei centri per anziani situati nel comune di Roma. La gestione riguarderà le informazioni relative ad ogni centro: i medici dell'ambulatorio con cui collabora, i volontari che ci lavorano, gli anziani iscritti e gli eventi organizzati. Di ogni centro per anziani interessa il nome, il municipio di appartenenza e l'indirizzo nel quale risiede. I municipi di Roma sono diciannove, identificati dalla numerazione ordinale romana.

Di ogni anziano interessa il nome, il cognome, il codice fiscale e la data di nascita, e al momento dell'iscrizione egli fornisce i suoi dati oltre alle informazioni sul centro anziani nel quale intende iscriversi (nella fattispecie il nome e il municipio nel quale risiede). Un anziano può iscriversi a più centri anziani ma non due volte nello stesso centro.

Di ogni volontario interessa il nome, il cognome e il numero identificativo all'interno del centro anziani in cui lavora. All'interno dell'ente, i volontari offrono determinati servizi usufruibili dagli anziani. Di ogni servizio interessa il codice identificativo e il volontario che lo offre in una determinata data.

I servizi disponibili possono essere esclusivamente di due tipi: accompagnamento e assistenza domiciliare. Per ogni accompagnamento interessa il mezzo utilizzato, la distanza percorsa e l'indirizzo della persona accompagnata. Per ogni assistenza domiciliare interessa l'indirizzo del domicilio dell'anziano e le ore di assistenza effettuate dal volontario. I due servizi si distinguono in base al codice identificativo. Nella fattispecie, per l'accompagnamento, il codice sarà pari mentre per l'assistenza domiciliare sarà dispari.

Ogni settimana un centro può organizzare alcuni eventi ricreativi coordinati dai volontari a cui possono partecipare le persone iscritte. Di ogni evento interessa il nome, la data in cui si svolge e il volontario che lo coordina. Oltre che partecipare ad un evento, un anziano può proporsi per organizzarlo, rispettando il vincolo che non può organizzarne più di uno a settimana. L'applicazione offre al centro la possibilità di conoscere i servizi e gli eventi in cui sono coinvolti anziani e volontari. Inoltre gli anziani possono scegliere di impegnarsi in qualche tipo di attività sociale. Ogni attività sociale è caratterizzata dal nome (vigilanza in scuole o musei, manutenzione del verde, etc.) e dalla data in cui viene svolta.

I centri per anziani forniscono assistenza sanitaria ai propri iscritti, tramite la collaborazione con un ambulatorio del municipio nel quale risiedono. Di ogni ambulatorio interessa il nome, il municipio di appartenenza e l'indirizzo di residenza. Il personale dell'ambulatorio è costituito da medici. Un medico può esercitare la propria professione in un solo ambulatorio. Di ogni medico interessa il nome, la specializzazione e il numero di tesserino unico nell'ambulatorio in cui lavora. L'ambulatorio ha un primario che dirige la struttura e che lavora necessariamente all'interno di essa. Tramite l'applicazione si può prenotare una visita e conoscere quale medico del personale visiterà l'anziano nella data e ora prevista. Di ogni visita interessa il medico che la effettuerà, la data e l'ora prevista. Tuttavia il medico può accettare un massimo di dieci visite al giorno.

1.2 7 aree interattive

Per facilitare l'uso, il livello di presentazione è organizzato in ***7 aree interattive***:

- **Area Centri Anziani:** vengono fornite le funzionalità per la gestione del personale, degli anziani e dei medici che collaborano con i centri per anziani.
- **Area Comunale:** viene data la possibilità di gestire tutte le informazioni riguardanti i centri anziani e gli ambulatori situati nel comune di Roma.
- **Area Anziani:** le funzioni principali in quest'area, sono quelle di registrazione di attività sociali e partecipazione ad eventi ricreativi da parte degli anziani.
- **Area Volontari:** questa sezione riassume le funzionalità di offerta di servizi ed organizzazione di eventi ricreativi da parte dei volontari.
- **Assistenza Sanitaria:** l'applicazione permette agli iscritti dei centri anziani di usufruire dei servizi offerti, di prenotare visite ambulatorie e ottenere informazioni riguardanti l'assistenza sanitaria.
- **Info Anziani e Volontari:** quest'area permette di ottenere una serie di informazioni sui servizi, gli eventi ricreativi, le visite ambulatoriali e tutto ciò che può riguardare gli anziani e i volontari dei centri anziani.
- **Today & Week:** le funzionalità offerte in questa sezione, permettono di conoscere tutti i servizi e gli eventi offerti dai centri durante la settimana.

Capitolo 2

Progettazione concettuale

L'analisi dei dati inizia con la progettazione concettuale della realtà di interesse dell'applicazione, attraverso un'astrazione ad alto livello dei dati. Lo strumento migliore per questa analisi è il modello semantico entità-relazione, noto anche come schema ER.

Nelle pagine seguenti troveremo tale rappresentazione grafica, che ci permetterà di fornire in modo semplice e schematico una descrizione relazionale.

Ora concentriamoci sui vincoli che non è possibile rappresentare attraverso tale modello. Possiamo elencarli in modo formale qui di seguito:

- I servizi accompagnamento e assistenza domiciliare sono distinguibili in base al codiceID che sarà rispettivamente pari per il primo e dispari per il secondo.
- Un medico può accettare un massimo di 10 visite al giorno.
- Il codice fiscale è una stringa di esattamente 16 caratteri.
- Il nome del municipio è espresso in numeri romani e appartiene ad un insieme predefinito di stringhe: {'I', 'II', 'III', 'IV', 'V', 'VI', 'VII', 'VIII', 'IX', 'X', 'XI', 'XII', 'XIII', 'XV', 'XVI', 'XVII', 'XVIII', 'XIX', 'XX'}
- Un anziano non può organizzare più di un evento a settimana.
- Un anziano non può iscriversi al centro anziani se non ha almeno 60 anni
- L'anziano non può partecipare ad un evento ricreativo se non è iscritto nel centro anziani dove si organizza questo evento.
- Un medico può esercitare la propria professione in un solo ambulatorio.
- Il nome dell'evento ricreativo appartiene ad un insieme predefinito di stringhe: {'torneo carte','ballo','torneo bocce'}

- Il mezzo di accompagnamento appartiene ad un insieme predefinito di stringhe: {'auto','piedi','mezzo pubblico'}
- Il municipio di residenza del centro anziani e il municipio di residenza dell'ambulatorio con cui il centro anziani collabora devono essere uguali.
- Ogni servizio di assistenza domiciliare non può avere un valore dell'attributo ore maggiore di 8.
- Un volontario non può coordinare più eventi ricreativi nella stessa data.
- Un anziano non può svolgere più di un'attività sociale al giorno.
- Non ci possono essere più di 5 eventi ricreativi al giorno nello stesso centro anziani.
- Un anziano non può prenotare una visita in una data e ora già prevista da un'altra sua visita.
- La data dei servizi offerti da un volontario non può precedere la data della sua assunzione al centro anziani.
- Le date di una visita, di svolgimento di un'attività sociale e di eventi ricreativi a cui partecipa un anziano, devono essere successive a quella di iscrizione al centro anziani.
- In Accompagnamento, se la DistanzaKM è maggiore di 9, allora il Mezzo non può assumere il valore *Piedi*, mentre se è minore di 5, non può assumere il valore *Auto*

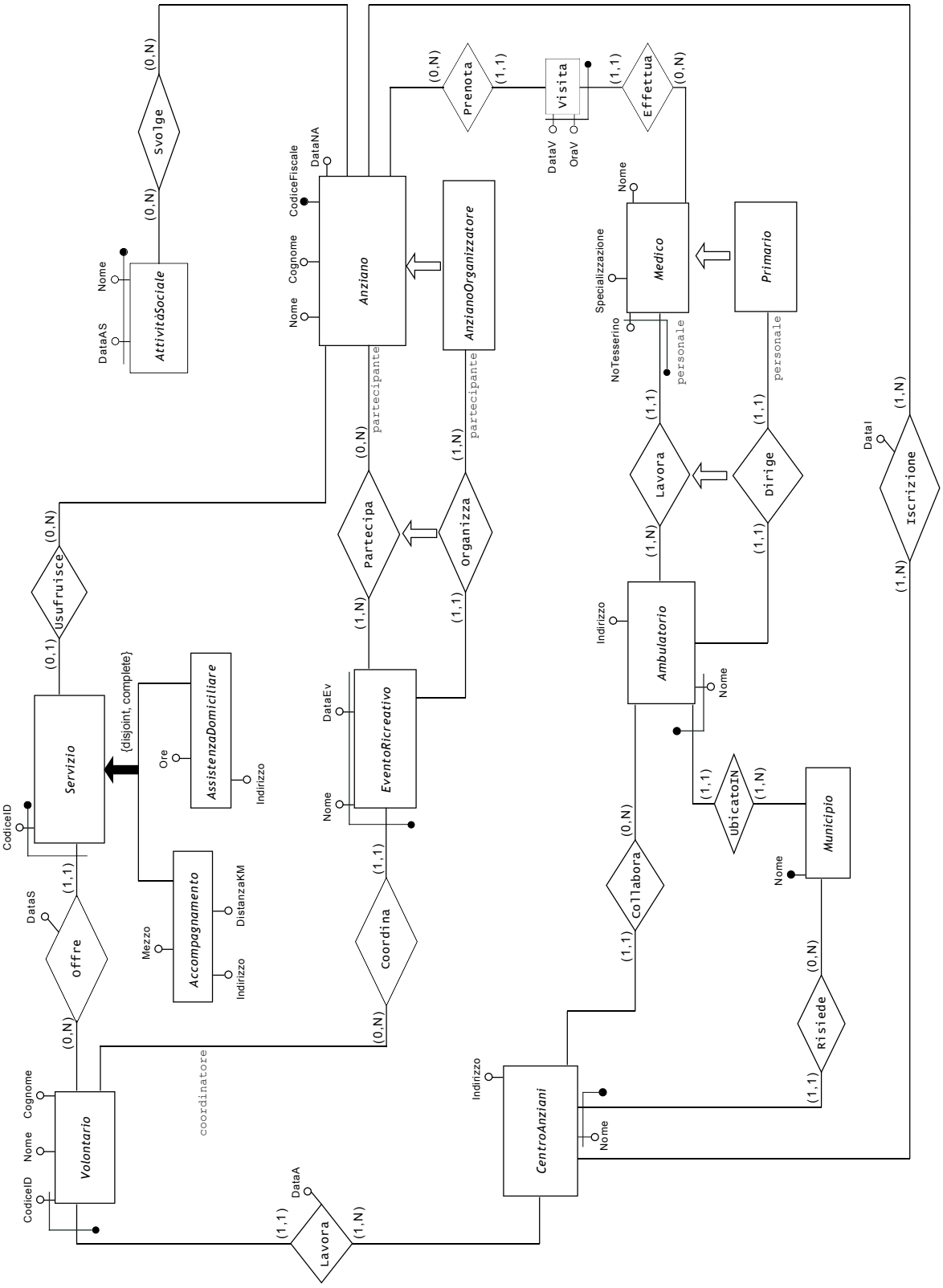

Figura 2.1: *Schema ER non ristrutturato*

2.1 Le funzionalità e i loro limiti

Il programma permette una gestione ottimale dei dati che ruotano attorno al mondo dei centri anziani di Roma prevedendo la possibilità di inserimento, cancellazione e consultazione di tali informazioni attraverso l'ausilio di un potente DBMS quale MySQL. L'utenza a cui è rivolto il programma è composta dal comune e dai centri anziani di Roma, con i volontari e gli anziani iscritti nei centri.

Le funzioni *Today&Week* sono le uniche funzioni accessibili da tutti i tipi di utenza, mentre le altre sono raggiungibili in base ad opportuni diritti di accesso controllati tramite un semplice sistema di login.

Gli anziani possono utilizzare il programma con funzionalità limitate alla registrazione dello svolgimento di un'attività sociale o all'adesione ad un evento ricreativo. I volontari, oltre a poter reperire alcune informazioni riguardanti gli anziani come le visite e le attività sociali svolte da questi, hanno la possibilità di offrire dei servizi e organizzare degli eventi ricreativi. Un addetto all'amministrazione del centro anziani può registrare o cancellare un anziano dalla base di dati e può accedere, così come il volontario, a funzioni di assistenza sanitaria. Il Comune, accedendo ad un area riservata, ha l'opportunità di gestire le informazioni su medici, centri anziani e ambulatori tramite funzioni di inserimento, cancellazione e interrogazione.

La scelta di tali opzioni d'uso è stata guidata da un lavoro di analisi e ricerca, valutando le varie possibilità che probabilmente sarebbero state richieste dall'utenza del programma. Nelle pagine seguenti viene proposta una schematizzazione delle funzionalità attraverso la visione grafica degli use case.

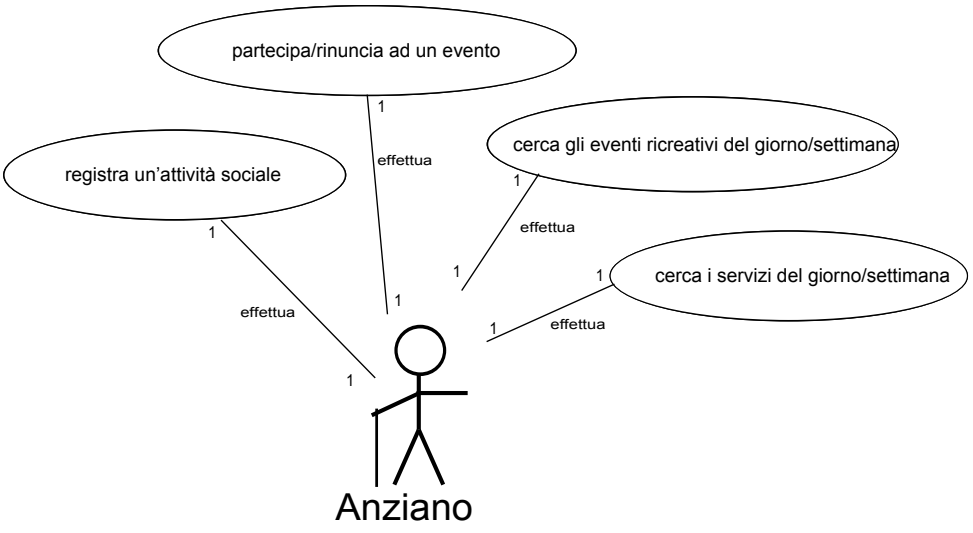

Figura 2.2: *Usecase per l'utente Anziano*

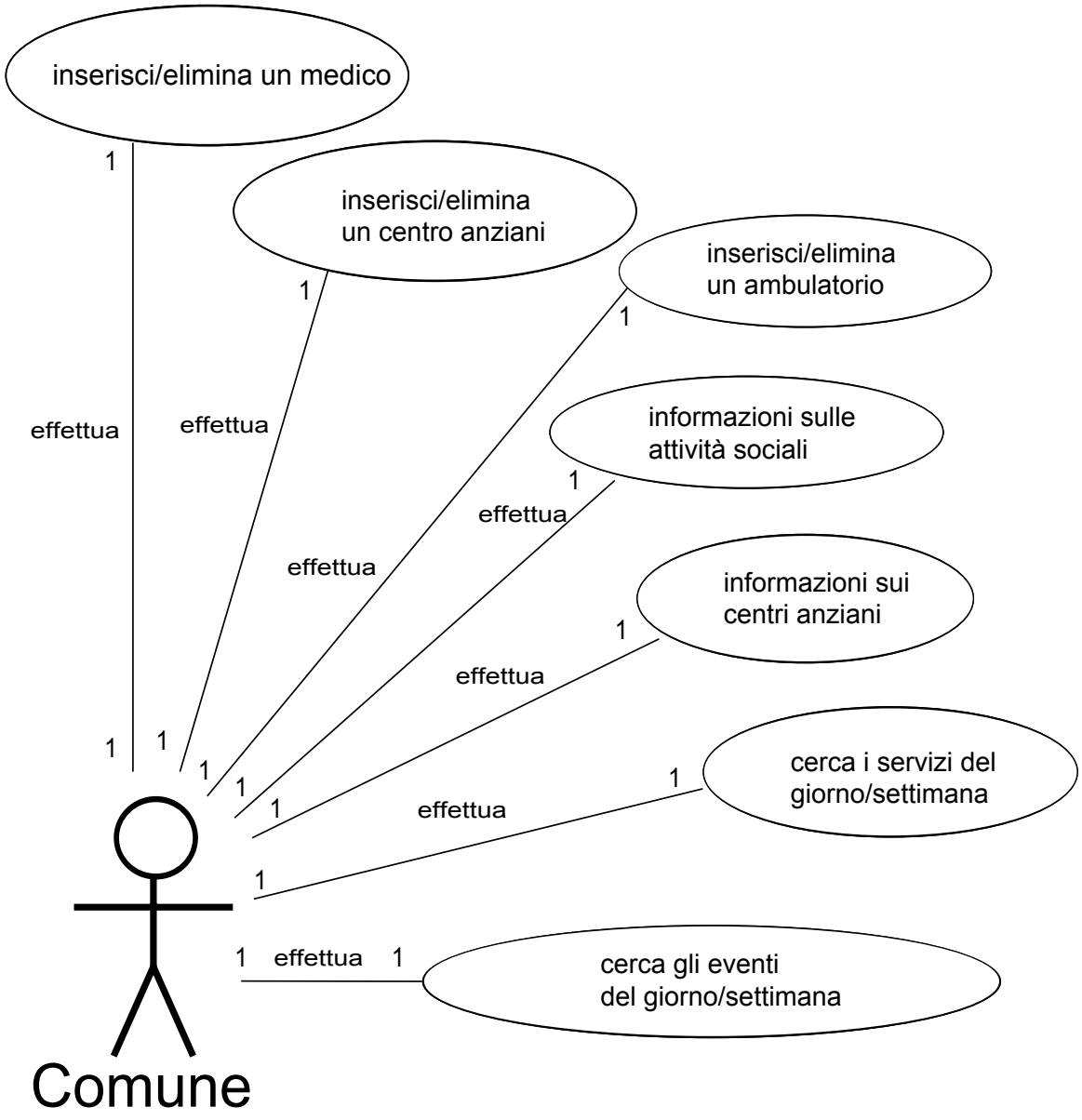

Figura 2.3: *Usecase per l'utente Comune*

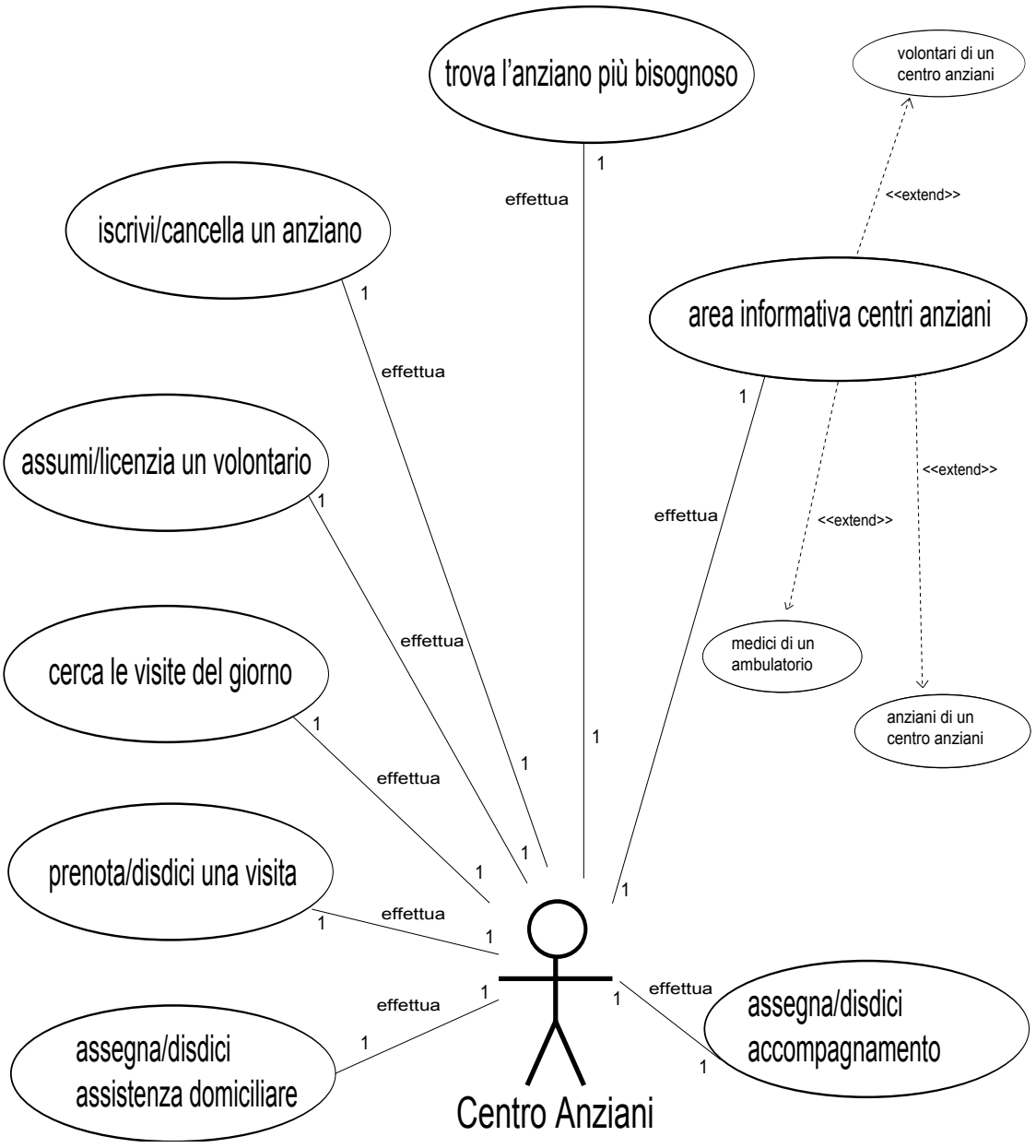

Figura 2.4: *Usecase per l'utente CentroAnziani*

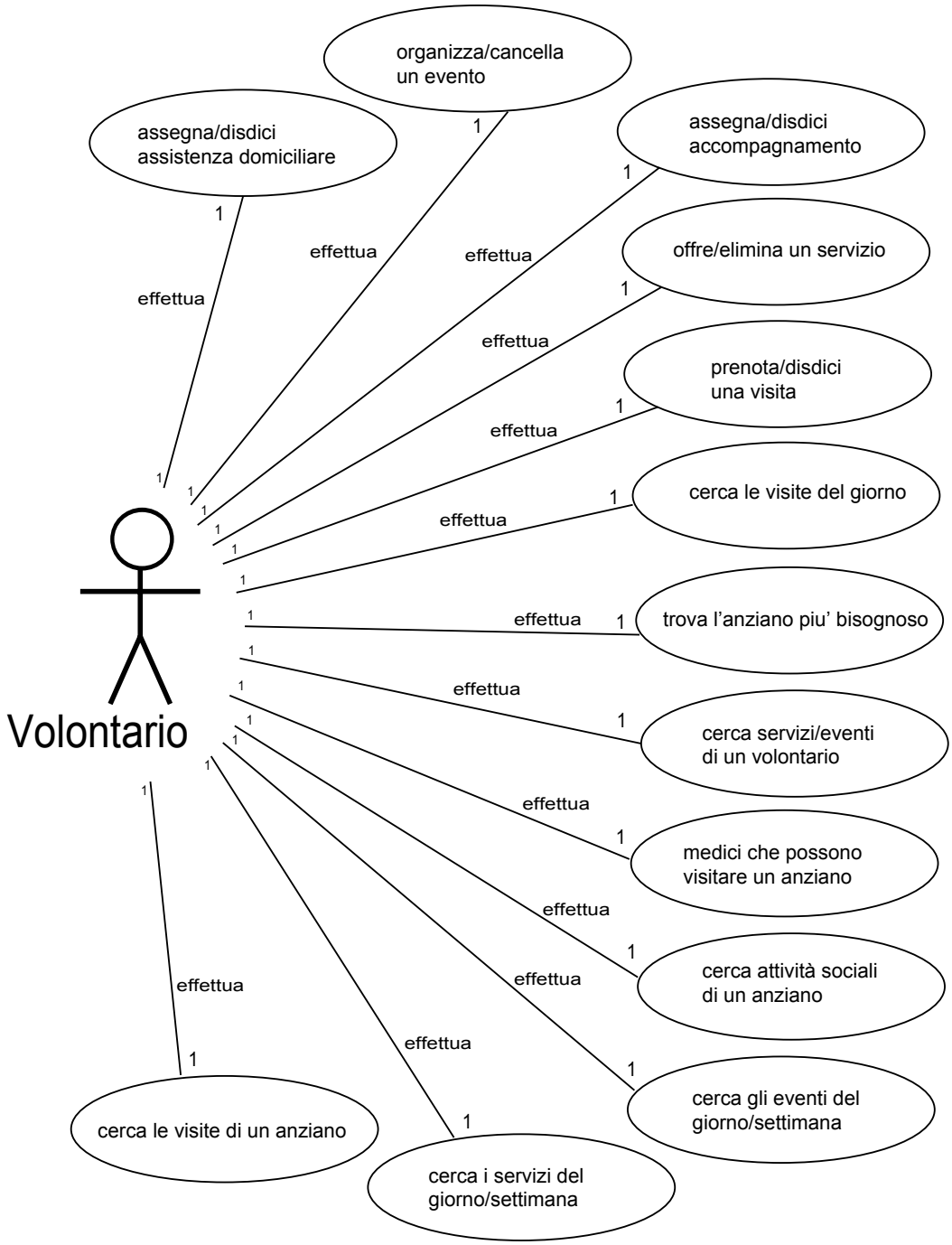

Figura 2.5: *Usecase per l'utente Volontario*

2.2 Tra software e hardware

Non si possono certo trascurare i mezzi attraverso i quali lo sviluppo è stato portato avanti. Il software è ciò che anima l'hardware, che gli da vita e lo rende operativo. Per questo motivo la scelta del sistema operativo, delle tecniche e dei programmi utilizzati per la creazione di altro software è d'importanza cruciale, e determina qualità di diversa natura come la multi-operabilità, la velocità di esecuzione, la versatilità, la modularità, la riusabilità e numerose altre proprietà. E' dunque ragionevole riportare, seppur in brevità, le scelte adottate in materia.

Lo sviluppo dell'applicazione è stato realizzato grazie alla *suite* di sviluppo ***Java 6*** aggiornata all'*update 7*, che comprende l'ottimo e ormai maturo *Netbeans 6.1*. Sfruttando gli strumenti messi a disposizione da tale *IDE* e appoggiandosi al *framework Java Swing*, si è realizzata una semplice GUI.

Il programma prevede l'interfacciamento con un *DBMS* relazionale per la memorizzazione e gestione delle informazioni attraverso l'*API* ***JDBC***. La scelta del *DBMS* utilizzato è ricaduta inevitabilmente su ***MySQL***, già studiato nel corso di basi di dati. La versione utilizzata è la 5.0, l'ultima *release* stabile disponibile al momento dell'inizio del progetto.

L'applicazione, essendo realizzata in Java, è naturalmente portabile in qualsiasi sistema operativo in grado di far girare il *JRE 6* e *MySQL* server. Si riportano tuttavia per correttezza i sistemi operativi utilizzati: *Microsoft Windows XP SP3, Microsoft Windows Vista, Fedora Linux Core 9*.

Inoltre sono stati utilizzati tre computer per lo sviluppo e il test dell'applicazione, di cui elenchiamo le caratteristiche hardware in tabella.

	PC1	PC2	PC3
CPU	Intel Core(TM)2 Duo T7250	Intel Pentium IV	Intel Pentium III M
FRQ CPU	2.0 GHz	2.8 GHz	1.2 Ghz
CPU CACHE	2MB	512KB	512KB
RAM	2048 MB DDR2	3072 MB DDR	512 MB PC133
HD	2.5" 5400rpm SATA	3.5" 7200rpm ATA	2.5" 4600rpm ATA
GPU	NVIDIA GeForce 8400M GS	ATI Radeon X800GT	Integrated Intel 82830 CGC

Tabella 2.1: *Tabella delle caratteristiche HW*

Il linguaggio LaTeX, che permette in campo scientifico una migliore espressione grafica dei concetti, è stato utile per la realizzazione della documentazione del programma e di questa relazione finale sul progetto. Inoltre è stato utilizzato un programma di grafica vettoriale per creare schemi ER, usecase, class e package diagram.

Parte II

Progettazione

Capitolo 3

La Progettazione Logica

3.1 Schema ER ristrutturato

Mostriamo nelle prossime pagine il risultato dello schema ER dopo la ristrutturazione. La metodologia adottata, è stata suggerita dalle slide presentate durante il corso di Basi di Dati [2].

3.1.1 Vincoli non esprimibili tramite l'ER ristrutturato

- Ogni istanza di Servizio partecipa ad ISA-A-S oppure ISA-D-S, ma non ad entrambi.
- Per ogni istanza (Personale:p, Ambulatorio:a) di Dirige, sia m l'istanza di Medico tale che (Primario:p, Medico:m) è un'istanza di ISA-P-M (si noti che m esiste sempre ed è unica); allora (Personale:m, Ambulatorio:a) deve essere un'istanza di Lavora.
- Per ogni istanza (Partecipante:ao, EventoRicreativo:er) di Organizza, sia a l'istanza di Anziano tale che (AnzianoOrganizzatore:ao, Anziano:a) è un'istanza di ISA-AO-A (si noti che a esiste sempre ed è unica); allora (Partecipante:a, EventoRicreativo:er) deve essere un'istanza di Partecipa.
- I servizi accompagnamento e assistenza domiciliare sono distinguibili in base al codiceID che sarà rispettivamente pari per il primo e dispari per il secondo.
- Un medico può accettare un massimo di 10 visite al giorno.
- Il codice fiscale è una stringa di esattamente 16 caratteri.
- Il nome del municipio è espresso in numeri romani e appartiene ad un insieme predefinito di stringhe: {'I', 'II', 'III', 'IV', 'V', 'VI', 'VII', 'VIII', 'IX', 'X', 'XI', 'XII', 'XIII', 'XV', 'XVI', 'XVII', 'XVIII', 'XIX', 'XX'}

- Un anziano non può organizzare più di un evento a settimana.
- Un anziano non può iscriversi al centro anziani se non ha almeno 60 anni
- Un medico può esercitare la propria professione in un solo ambulatorio.
- L'anziano non può partecipare ad un evento ricreativo se non è iscritto nel centro anziani dove si organizza questo evento.
- Il nome dell'evento ricreativo appartiene ad un insieme predefinito di stringhe: {'torneo carte','ballo','torneo bocce'}
- Il mezzo di accompagnamento appartiene ad un insieme predefinito di stringhe: {'auto','piedi','mezzo pubblico'}
- Il municipio di residenza del centro anziani e il municipio di residenza dell'ambulatorio con cui il centro anziani collabora devono essere uguali.
- Ogni servizio di assistenza domiciliare non può avere un valore dell'attributo ore maggiore di 8.
- Un volontario non può coordinare più eventi ricreativi nella stessa data.
- Un anziano non può svolgere più di un'attività sociale al giorno.
- Non ci possono essere più di 5 eventi ricreativi al giorno nello stesso centro anziani.
- Un anziano non può prenotare una visita in una data e ora già prevista da un'altra sua visita.
- La data dei servizi offerti da un volontario non può precedere la data della sua assunzione al centro anziani.
- Le date di una visita, di svolgimento di un'attività sociale e di eventi ricreativi a cui partecipa un anziano, devono essere successive a quella di iscrizione al centro anziani.
- In Accompagnamento, se la DistanzaKM è maggiore di 9, allora il Mezzo non può assumere il valore *Piedi*, mentre se è minore di 5, non può assumere il valore *Auto*

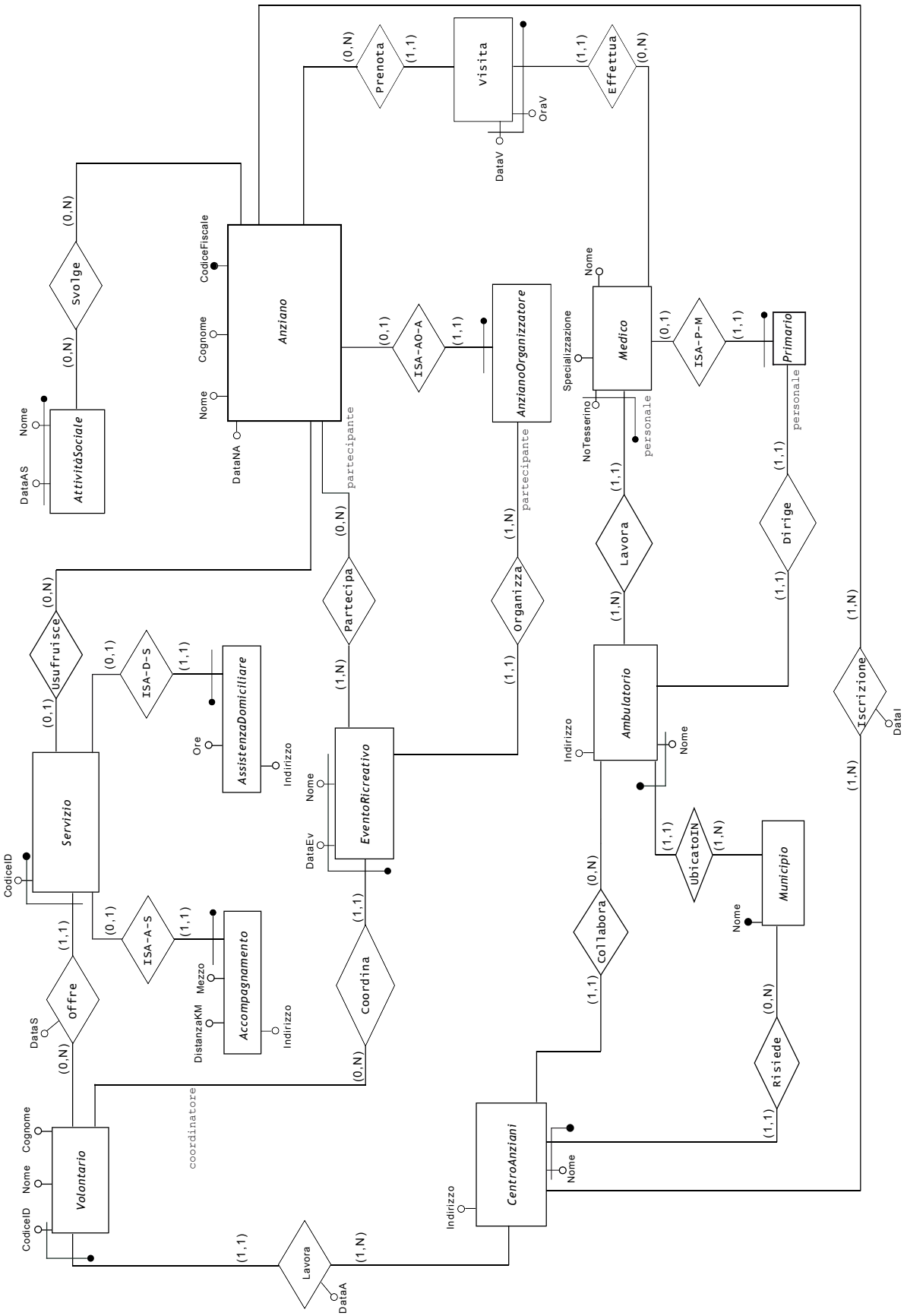

Figura 3.1: *Schema ER ristrutturato*

3.2 Schema logico relazionale dei dati

Presentiamo in questa sezione, la traduzione dello schema ER ristrutturato nello Schema Logico. Come strumenti di riferimento sono state usate le slide del corso di Basi di Dati [2].

Volontario(CodiceID, CentroAnziani, Municipio, Nome, Cognome, DataA)
foreign key: Volontario[CentroAnziani, Municipio] ⊆ CentroAnziani[Nome, Municipio]

Servizio(CodiceID, Volontario, CentroAnziani, Municipio, DataS)
foreign key: Servizio[Volontario, CentroAnziani, Municipio] ⊆ Volontario[CodiceID, CentroAnziani, Municipio]

Accompagnamento(CodiceID, Volontario, CentroAnziani, Municipio, Mezzo, DistanzaKM, Indirizzo)
foreign key: Accompagnamento[CodiceID, Volontario, CentroAnziani, Municipio] ⊆ Servizio[CodiceID, Volontario, CentroAnziani, Municipio]

AssistenzaDomiciliare(CodiceID, Volontario, CentroAnziani, Municipio, Ore, Indirizzo)
foreign key: AssistenzaDomiciliare[CodiceID, Volontario, CentroAnziani, Municipio] ⊆ Servizio[CodiceID, Volontario, CentroAnziani, Municipio]

Usufruisce(Servizio, Volontario, CentroAnziani, Municipio, Anziano)
foreign key: Usufruisce[Anziano] ⊆ Anziano[CodiceFiscale]
foreign key: Usufruisce[Servizio, Volontario, CentroAnziani, Municipio] ⊆ Servizio[CodiceID, Volontario, CentroAnziani, Municipio]

CentroAnziani(Nome, Municipio, Indirizzo)
inclusione: CentroAnziani[Nome, Municipio] ⊆ Volontario[CentroAnziani, Municipio]
inclusione: CentroAnziani[Nome, Municipio] ⊆ Iscrizione[CentroAnziani, Municipio]
foreign key: CentroAnziani[Nome, Municipio] ⊆ Collabora[CentroAnziani, Municipio]
foreign key: CentroAnziani[Municipio] ⊆ Municipio[Nome]

Collabora(CentroAnziani, Municipio, Ambulatorio, MunicipioAmb)
foreign key: Collabora[Ambulatorio, MunicipioAmb] ⊆ Ambulatorio[Nome, Municipio]
foreign key: Collabora[CentroAnziani, Municipio] ⊆ CentroAnziani[Nome, Municipio]

Ambulatorio(Nome, Municipio, Indirizzo)
inclusione: Ambulatorio[Nome, Municipio] ⊆ Medico[Ambulatorio, Municipio]
foreign key: Ambulatorio[Municipio] ⊆ Municipio[Nome]
foreign key: Ambulatorio[Nome, Municipio] ⊆ Dirige[AmbulatorioAmb, MunicipioAmb]

Municipio(Nome)
inclusione: Municipio[Nome] ⊆ Ambulatorio[Municipio]

Medico(NoTesserino Ambulatorio Municipio, Nome, Specializzazione)
foreign key: Medico[Ambulatorio, Municipio] ⊆ Ambulatorio[Nome, Municipio]

Primario(NoTesserino, Ambulatorio, Municipio)
foreign key: Primario[NoTesserino,Ambulatorio,Municipio] ⊆ Medico[NoTesserino, Ambulatorio, Municipio]
foreign key: Primario[NoTesserino, Ambulatorio, Municipio] ⊆ Dirige[Primario, AmbulatorioPrim, MunicipioPrim]

Dirige(Primario,AmbulatorioPrim, MunicipioPrim, AmbulatorioAmb, MunicipioAmb)
foreign key: Dirige[Primario, AmbulatorioPrim, MunicipioPrim] ⊆ Primario[NoTesserino, Ambulatorio, Municipio]
foreign key: Dirige[Primario, AmbulatorioPrim, MunicipioPrima] ⊆ Medico[NoTesserino, Ambulatorio, Municipio]

EventoRicreativo(Nome, DataEv, Volontario, CentroAnziani, Municipio)
inclusione: EventoRicreativo[Nome, DataEv, Volontario, CentroAnziani, Municipio] ⊆ Partecipa[EventoRicreativo, DataEventoRicreativo, Volontario, CentroAnziani, Municipio]
foreign key: EventoRicreativo[Nome, DataEv, Volontario, CentroAnziani, Municipio] ⊆ Organizza[EventoRicreativo, DataEventoRicreativo, Volontario, CentroAnziani, Municipio]
foreign key: EventoRicreativo[Volontario, CentroAnziani, Municipio] ⊆ Volontario[CodiceID, CentroAnziani, Municipio]

Partecipa(Anziano,EventoRicreativo, DataEventoRicreativo, Volontario, CentroAnziani, Municipio)
foreign key: Partecipa[Anziano] ⊆ Anziano[CodiceFiscale]
foreign key: Partecipa[EventoRicreativo, DataEventoRicreativo, Volontario, CentroAnziani, Municipio] ⊆ EventoRicreativo[Nome, DataEv, Volontario, CentroAnziani, Municipio]

Anziano(CodiceFiscale, Nome, Cognome, DataNA)
inclusione: Anziano[CodiceFiscale] ⊆ Iscrizione[Anziano]

AnzianoOrganizzatore(CodiceFiscale)
inclusione: AnzianoOrganizzatore[CodiceFiscale] ⊆ Organizza[Organizzatore]
foreign key: AnzianoOrganizzatore[CodiceFiscale] ⊆ Anziano[CodiceFiscale]

Organizza(EventoRicreativo, DataEventoRicreativo, Volontario, CentroAnziani, Municipio, Organizzatore)
foreign key: Organizza[Organizzatore] ⊆ AnzianoOrganizzatore[CodiceFiscale]
foreign key: Organizza[EventoRicreativo, DataEventoRicreativo, Volontario, CentroAnziani, Municipio, Organizzatore] ⊆ Partecipa[EventoRicreativo, DataEventoRicreativo, Volontario, CentroAnziani, Municipio, Anziano]

AttivitàSociale(Nome, DataAS)

Svolge(AttivitàSociale, DataAttivitaSociale, Anziano)
foreign key: Svolge[AttivitàSociale, DataAttivitaSociale] ⊆ AttivitàSociale[Nome, DataAS]
foreign key: Svolge[Anziano] ⊆ Anziano[CodiceFiscale]

Iscrizione(CentroAnziani, Municipio, Anziano, DataI)
foreign key: Iscrizione[CentroAnziani, Municipio] ⊆ CentroAnziani[Nome, Municipio]
foreign key: Iscrizione[Anziano] ⊆ Anziano[CodiceFiscale]

Visita(Medico, Ambulatorio, Municipio, DataV, OraV)
foreign key: Visita[Medico, Ambulatorio, Municipio, DataV, OraV] ⊆ Prenota[Medico, Ambulatorio, Municipio, DataVisita, OraVisita]
foreign key: Visita[Medico,Ambulatorio,Municipio] ⊆ Medico[NoTesserino, Ambulatorio, Municipio]

Prenota(Medico, Ambulatorio, Municipio, DataVisita, OraVisita, Anziano)
foreign key: Prenota[Anziano] ⊆ Anziano[CodiceFiscale]
foreign key: Prenota[Medico, Ambulatorio, Municipio, DataVisita, OraVisita] ⊆ Visita[Medico, Ambulatorio, Municipio, DataV, OraV]

Vincolo di generalizzazione completa su Servizio:
Accompagnamento[CodiceID, Volontario, CentroAnziani, Municipio] ∩ AssistenzaDomiciliare[CodiceID, Volontario, CentroAnziani, Municipio] = ⊘
Servizio[CodiceID, Volontario, CentroAnziani, Municipio] ⊆ Accompagnamento[CodiceID, Volontario, CentroAnziani, Municipio] ∪ AssistenzaDomiciliare[CodiceID, Volontario, CentroAnziani, Municipio]

Capitolo 4

Ottimizzazione logica

4.1 Ristrutturazione dello schema logico

In questa sezione presentiamo le scelte progettuali relative alla ristrutturazione dello schema logico.

Per quanto riguarda la relazione:
Collabora(CentroAnziani, Municipio, Ambulatorio, MunicipioAmb)

Manteniamo la ridondanza, in quanto l'attributo MunicipioAmb è settato a NULL durante l'eliminazione di un ambulatorio. Infatti, come si vedrà più avanti, Collabora viene accorpato a CentroAnziani, il quale acquisisce l'attributo MunicipioAmb. In questo caso se eliminassimo la ridondanza, avremmo che il Municipio non potrebbe essere settato a NULL, perché, come sappiamo, Municipio è anche chiave primaria di CentroAnziani. Per tale motivo la ridondanza di Municipio permane, nonostante valga il vincolo, gestito a livello applicativo, che codesti attributi debbano avere identico valore.

Per quanto riguarda la relazione:
Dirige(Primario, AmbulatorioPrim, MunicipioPrim, AmbulatorioAmb, MunicipioAmb)

Dalla specifica traspare che l'ambulatorio che dirige il primario è necessariamente quello in cui questi in qualità di medico lavora. Trattandosi dello stesso ambulatorio deve anche necessariamente avere lo stesso municipio di residenza. Quindi eliminiamo le ridondanze mantenendo due soli attributi Ambulatorio e Municipio.

La relazione diventa quindi:
Dirige(Primario, Ambulatorio, Municipio)

4.1.1 Accorpamenti

Di seguito riportiamo gli accorpamenti realizzati per migliorare l'accesso ai dati in determinate funzioni:

ACCORPAMENTO RISULTANTE	RELAZIONE	ENTITA'
CentroAnziani	Collabora	CentroAnziani
PrimarioDirigente	Dirige	Primario
PrenotaVisita	Visita	Prenota

Tabella 4.1: *Tabella degli accorpamenti*

Vista l'intenzione di creare una funzione che stampasse le informazioni del centro anziani tra cui il nome dell'ambulatorio collaborante, per evitare che la richiesta di tali informazioni necessitasse l'accesso a più tabelle, si è optato per l'accorpamento dell'entità CentroAnziani con la relazione Collabora.

CentroAnziani(Nome, Municipio, Indirizzo, Ambulatorio, MunicipioAmb)
inclusione: CentroAnziani[Nome, Municipio] $\subseteq$ Volontario[CentroAnziani, Municipio]
inclusione: CentroAnziani[Nome, Municipio] $\subseteq$ Iscrizione[CentroAnziani, Municipio]
foreign key: CentroAnziani[Municipio] $\subseteq$ Municipio[Nome]
foreign key: CentroAnziani[Ambulatorio, MunicipioAmb] $\subseteq$ Ambulatorio[Nome, Municipio]

Evidente è risultata la necessità di accorpare Primario con Dirige per eliminare la ridondanza di informazione nel database. Come si osserva nella pagina precedente, la tabella risultante dopo l'eliminazione della ridondanza degli attributi municipio e ambulatorio dalla relazione Dirige, coincide perfettamente con la tabella risultante dall'entità Primario. Questa osservazione ci ha fatto propendere per un accorpamento certo tra Dirige e Primario.

PrimarioDirigente(NoTesserino,Ambulatorio,Municipio)
foreign key: PrimarioDirigente[NoTesserino, Ambulatorio, Municipio] $\subseteq$ Medico[NoTesserino, Ambulatorio, Municipio]

Viste le informazioni contenute in Visita e quelle contenute in Prenota, e considerato il fatto che nel programma accediamo alle informazioni delle visite solo in relazione all'anziano che le ha prenotate, si è scelto di accorpare Prenota con Visita ottenendo come risultato la sotto esposta PrenotaVisita.

PrenotaVisita(Medico, Ambulatorio, Municipio, DataV, OraV, Anziano)
foreign key: PrenotaVisita[Medico, Ambulatorio, Municipio] $\subseteq$ Medico[NoTesserino, Ambulatorio, Municipio]
foreign key: PrenotaVisita[Anziano] $\subseteq$ Anziano[CodiceFiscale]

4.1.2 Schema logico ristrutturato

Volontario(CodiceID, CentroAnziani, Municipio, Nome, Cognome, DataA)
foreign key: Volontario[CentroAnziani, Municipio] ⊆ CentroAnziani[Nome, Municipio]

Servizio(CodiceID, Volontario, CentroAnziani, Municipio, DataS)
foreign key: Servizio[Volontario, CentroAnziani, Municipio] ⊆ Volontario[CodiceID, CentroAnziani, Municipio]

Accompagnamento(CodiceID, Volontario, CentroAnziani, Municipio, Mezzo, DistanzaKM, Indirizzo)
foreign key: Accompagnamento[CodiceID, Volontario, CentroAnziani, Municipio] ⊆ Servizio[CodiceID, Volontario, CentroAnziani, Municipio]

AssistenzaDomiciliare(CodiceID, Volontario, CentroAnziani, Municipio, Ore, Indirizzo)
foreign key: AssistenzaDomiciliare[CodiceID, Volontario, CentroAnziani, Municipio] ⊆ Servizio[CodiceID, Volontario, CentroAnziani, Municipio]

Usufruisce(Servizio, Volontario, CentroAnziani, Municipio, Anziano)
foreign key: Usufruisce[Anziano] ⊆ Anziano[CodiceFiscale]
foreign key: Usufruisce[Servizio, Volontario, CentroAnziani, Municipio] ⊆ Servizio[CodiceID, Volontario, CentroAnziani, Municipio]

CentroAnziani(Nome, Municipio, Indirizzo, Ambulatorio, MunicipioAmb)
inclusione: CentroAnziani[Nome, Municipio] ⊆ Volontario[CentroAnziani, Municipio]
inclusione: CentroAnziani[Nome, Municipio] ⊆ Iscrizione[CentroAnziani, Municipio]
foreign key: CentroAnziani[Municipio] ⊆ Municipio[Nome]
foreign key: CentroAnziani[Ambulatorio, MunicipioAmb] ⊆ Ambulatorio[Nome, Municipio]

Ambulatorio(Nome, Municipio, Indirizzo)
inclusione: Ambulatorio[Nome, Municipio] ⊆ Medico[Ambulatorio, Municipio]
foreign key: Ambulatorio[Municipio] ⊆ Municipio[Nome]
foreign key: Ambulatorio[Nome, Municipio] ⊆ PrimarioDirigente[Ambulatorio, Municipio]

Municipio(Nome)
inclusione: Municipio[Nome] ⊆ Ambulatorio[Municipio]

Medico(NoTesserino Ambulatorio Municipio, Nome, Specializzazione)
foreign key: Medico[Ambulatorio, Municipio] ⊆ Ambulatorio[Nome, Municipio]

PrimarioDirigente(NoTesserino,Ambulatorio,Municipio)
foreign key: PrimarioDirigente[NoTesserino, Ambulatorio, Municipio] ⊆ Medico[NoTesserino, Ambulatorio, Municipio]

EventoRicreativo(Nome, DataEv, Volontario, CentroAnziani, Municipio)
inclusione: EventoRicreativo[Nome, DataEv, Volontario, CentroAnziani, Municipio] ⊆ Partecipa[EventoRicreativo, DataEventoRicreativo, Volontario, CentroAnziani, Municipio]
foreign key: EventoRicreativo[Nome, DataEv, Volontario, CentroAnziani, Municipio] ⊆ Organizza[EventoRicreativo, DataEventoRicreativo, Volontario, CentroAnziani, Municipio]
foreign key: EventoRicreativo[Volontario, CentroAnziani, Municipio] ⊆ Volontario[CodiceID, CentroAnziani, Municipio]

Partecipa(Anziano,EventoRicreativo, DataEventoRicreativo, Volontario, CentroAnziani, Municipio)
foreign key: Partecipa[Anziano] ⊆ Anziano[CodiceFiscale]
foreign key: Partecipa[EventoRicreativo, DataEventoRicreativo, Volontario, CentroAnziani, Municipio] ⊆ EventoRicreativo[Nome, DataEv, Volontario, CentroAnziani, Municipio]

Anziano(CodiceFiscale, Nome, Cognome, DataNA)
inclusione: Anziano[CodiceFiscale] ⊆ Iscrizione[Anziano]

AnzianoOrganizzatore(CodiceFiscale)
inclusione: AnzianoOrganizzatore[CodiceFiscale] ⊆ Organizza[Organizzatore]
foreign key: AnzianoOrganizzatore[CodiceFiscale] ⊆ Anziano[CodiceFiscale]

Organizza(EventoRicreativo, DataEventoRicreativo, Volontario, CentroAnziani, Municipio, Organizzatore)
foreign key: Organizza[Organizzatore] ⊆ AnzianoOrganizzatore[CodiceFiscale]
foreign key: Organizza[EventoRicreativo, DataEventoRicreativo, Volontario, CentroAnziani, Municipio, Organizzatore] ⊆ Partecipa[EventoRicreativo, DataEventoRicreativo, Volontario, CentroAnziani, Municipio, Anziano]

AttivitàSociale(Nome, DataAS)

Svolge(AttivitàSociale, DataAttivitaSociale, Anziano)
foreign key: Svolge[AttivitàSociale, DataAttivitaSociale] ⊆ AttivitàSociale[Nome, DataAS]
foreign key: Svolge[Anziano] ⊆ Anziano[CodiceFiscale]

Iscrizione(CentroAnziani, Municipio, Anziano, DataI)
foreign key: Iscrizione[CentroAnziani, Municipio] ⊆ CentroAnziani[Nome, Municipio]
foreign key: Iscrizione[Anziano] ⊆ Anziano[CodiceFiscale]

PrenotaVisita(Medico, Ambulatorio, Municipio, DataV, OraV, Anziano)
foreign key: PrenotaVisita[Medico, Ambulatorio, Municipio] ⊆ Medico[NoTesserino, Ambulatorio, Municipio]
foreign key: PrenotaVisita[Anziano] ⊆ Anziano[CodiceFiscale]

Vincolo di generalizzazione completa su Servizio:
Accompagnamento[CodiceID, Volontario, CentroAnziani, Municipio] ∩ AssistenzaDomiciliare[CodiceID, Volontario, CentroAnziani, Municipio] = ⊘
Servizio[CodiceID, Volontario, CentroAnziani, Municipio] ⊆ Accompagnamento[CodiceID, Volontario, CentroAnziani, Municipio] ∪ AssistenzaDomiciliare[CodiceID, Volontario, CentroAnziani, Municipio]

4.2 Elenco dei vincoli sullo schema logico

La classificazione dei vincoli avviene secondo i seguenti criteri [3]:

- Vincoli di tupla (riga): si riferiscono alle singole righe della tabella
- Vincoli di tabella: si riferiscono all'insieme delle righe di una tabella
- Vincoli inter-tabella: si riferiscono a più di una tabella

Osservazione 1. *Definiamo l'insieme* Σ *come l'insieme dei municipi di Roma. Possiamo quindi dire che:* $\Sigma = \{I, II, III, IV, V, VI, VII, VIII, IX, X, XI, XII, XIII, XV, XVI, XVII, XVIII, XIX, XX\}$

4.2.1 Vincoli di tupla

Volontario

- CodiceID: Intero > 0
- CentroAnziani: Stringa di massimo 20 caratteri
- Municipio: Stringa $\in \Sigma$
- Nome: Stringa di massimo 20 caratteri
- Cognome: Stringa di massimo 20 caratteri
- DataA: Deve rispettare la forma del tipo sql Date

Servizio

- CodiceID: Intero > 0
- Volontario: Intero > 0
- CentroAnziani: Stringa di massimo 20 caratteri

- Municipio: Stringa $\in \Sigma$
- DataS: Deve rispettare la forma del tipo sql Date

Accompagnamento

- CodiceID: Intero > 0 e Pari
- Volontario: Intero > 0
- CentroAnziani: Stringa di massimo 20 caratteri
- Municipio: Stringa $\in \Sigma$
- Mezzo: Stringa $\in$ {Auto, Piedi, Mezzo Pubblico} e inoltre, se la DistanzaKM è maggiore di 9, allora il Mezzo non può assumere il valore *Piedi*, mentre se è minore di 5, non può assumere il valore *Auto*
- DistanzaKM: Intero > 0
- Indirizzo: Stringa di massimo 40 caratteri

AssistenzaDomiciliare

- CodiceID: Intero > 0 e Dispari
- Volontario: Intero > 0
- CentroAnziani: Stringa di massimo 20 caratteri
- Municipio: Stringa $\in \Sigma$
- Ore: Intero > 0
- Indirizzo: Stringa di massimo 40 caratteri

Usufruisce

- Servizio: Intero > 0
- Volontario: Intero > 0
- CentroAnziani: Stringa di massimo 20 caratteri
- Municipio: Stringa $\in \Sigma$
- Anziano: Stringa di 16 caratteri

CentroAnziani

- Nome: Stringa di massimo 20 caratteri
- Municipio: Stringa $\in \Sigma$
- Indirizzo: Stringa di massimo 40 caratteri

Collabora

- Centro Anziani: Stringa di massimo 20 caratteri
- Municipio: Stringa $\in \Sigma$
- Ambulatorio: Stringa di massimo 20 caratteri
- MunicipioAmb: Stringa $\in \Sigma$

Ambulatorio

- Nome: Stringa di massimo 20 caratteri
- Municipio: Stringa $\in \Sigma$
- Indirizzo: Stringa di massimo 40 caratteri

Municipio

- Nome: Stringa $\in \Sigma$

Medico

- NoTesserino: Intero > 0
- Ambulatorio: Stringa di massimo 20 caratteri
- Municipio: Stringa $\in \Sigma$
- Nome: Stringa di massimo 20 caratteri
- Specializzazione: Stringa di massimo 25 caratteri

Primario

- NoTesserino: Intero > 0
- Ambulatorio: Stringa di massimo 20 caratteri

- Municipio: Stringa $\in \Sigma$

Dirige

- Primario: Intero > 0
- AmbulatorioPrim: Stringa di massimo 20 caratteri
- MunicipioPrim: Stringa $\in \Sigma$
- AmbulatorioAmb: Stringa di massimo 20 caratteri
- MunicipioAmb: Stringa $\in \Sigma$

EventoRicreativo

- Nome: Stringa $\in$ {torneocarte, ballo, torneobocce}
- DataEv: Deve rispettare la forma del tipo sql Date
- Volontario: Intero > 0
- CentroAnziani: Stringa di massimo 20 caratteri
- Municipio: Stringa $\in \Sigma$

Partecipa

- Anziano: Stringa di 16 caratteri
- EventoRicreativo: Stringa $\in$ {torneocarte, ballo, torneobocce}
- DataEventoRicreativo: Deve rispettare la forma del tipo sql Date
- Volontario: Intero > 0
- CentroAnziani: Stringa di massimo 20 caratteri
- Municipio: Stringa $\in \Sigma$

Organizza

- AnzianoOrganizzatore: Stringa di 16 caratteri
- EventoRicreativo: Stringa $\in$ {torneocarte, ballo, torneobocce}
- DataEventoRicreativo: Deve rispettare la forma del tipo sql Date

- Volontario: Intero > 0
- CentroAnziani: Stringa di massimo 20 caratteri
- Municipio: Stringa $\in \Sigma$

Anziano

- CodiceFiscale: Stringa di 16 caratteri
- Nome: Stringa di massimo 20 caratteri
- Cognome: Stringa di massimo 20 caratteri
- DataNA: Deve rispettare la forma del tipo sql Date e deve essere tale che l'anziano risulti avere almeno 60 anni

AnzianoOrganizzatore

- CodiceFiscale: Stringa di 16 caratteri

AttivitaSociale

- Nome: Stringa di massimo 25 caratteri
- DataAS: Deve rispettare la forma del tipo sql Date

Svolge

- AttivitaSociale: Stringa di massimo 25 caratteri
- DataAS: Deve rispettare la forma del tipo sql Date
- Anziano: Stringa di 16 caratteri

Iscrizione

- Anziano: Stringa di 16 caratteri
- CentroAnziani: Stringa di massimo 20 caratteri
- Municipio: Stringa $\in \Sigma$
- DataI: Deve rispettare la forma del tipo sql Date

Prenota

- Anziano: Stringa di 16 caratteri
- Medico: Intero > 0
- Ambulatorio: Stringa di massimo 20 caratteri
- Municipio: Stringa $\in \Sigma$
- DataV: Deve rispettare la forma del tipo sql Date
- OraV: Deve rispettare la forma del tipo sql Time

Visita

- Medico: Intero > 0
- Ambulatorio: Stringa di massimo 20 caratteri
- Municipio: Stringa $\in \Sigma$
- DataV: Deve rispettare la forma del tipo sql Date
- OraV: Deve rispettare la forma del tipo sql Time

In generale

1. *Tutti gli attributi di ogni tabella devono essere diversi da NULL.*
2. *Tutte le date devono rispettare il formato Date di tipo sql*

4.2.2 Vincoli di tabella

Includono i vincoli di chiave primaria di ogni tabella

- Un volontario non può coordinare più eventi ricreativi nella stessa data (*nella tabella eventoricreativo non può esistere più di una tupla con lo stesso valore per le colonne volontario, dataev, municipio, centroanziani*)
- Un anziano non può svolgere più di un'attività sociale al giorno (*nella tabella svolge non può esistere più di una tupla con lo stesso valore per le colonne anziano, dataattivitasociale*)
- Non ci possono essere più di 5 eventi ricreativi al giorno nello stesso centro anziani (*nella tabella eventoricreativo non possono esistere più di 5 tuple con lo stesso valore per le colonne dataev, municipio, centroanziani*)

- Un anziano non può prenotare una visita in una data e ora già prevista da un'altra sua visita. (*non può esistere nella tabella prenotavisita più di una tupla con le colonne anziano, datav e orav uguali*)

- Un anziano non può organizzare più di un evento alla settimana. (*non possono esistere due tuple appartenenti alla tabella organizza in cui per una stesso valore della colonna anziano, le date siano nella stessa settimana dello stesso anno*).

- Un medico può esercitare la propria professione in un solo ambulatorio. (*non possono esistere due tuple nella tabella medico che abbiano gli stessi campi Nome, Specializzazione, NoTesserino*)

- Un medico non può avere più di 10 visite al giorno (*non possono esistere nella tabella prenotavisita più di 10 entry con lo stesso valore delle colonne medico, ambulatorio e datav*)

4.2.3 Vincoli inter-tabella

Includono i vincoli d'integrità referenziale di ogni tabella e i vincoli d'inclusione.

- L'anziano non può partecipare ad un evento ricreativo se non è iscritto al centro anziani dove si organizza l'evento (*Al momento della partecipazione all'evento, controllo che l'anziano sia iscritto in quel centro anziani, sede dell'evento a cui intende partecipare.*)

- La data dei servizi offerti da un volontario non può precedere la data della sua assunzione al centro anziani. (*Data l'esistenza di una tupla nella tabella volontario con valori delle colonne codiceid: C, municipio: M, centroanziani: CA, dataa: D, allora non può esistere una tupla in servizio avente valori delle colonne tali che volontario è uguale a C, centroanziani è CA, municipio è M e datas è minore di D*)

- Le date di una visita, di svolgimento di un'attività sociale e di eventi ricreativi a cui partecipa un anziano, devono essere successive a quella di iscrizione al centro anziani. (*come sopra*)

- Il municipio di residenza del centro anziani e il municipio di residenza dell'ambulatorio devono essere uguali. (*presa una qualunque tupla della tabella centroanziani in cui il municipio è M, il nome è N, e l'ambulatorio è A, allora deve esistere una tupla nella tabella ambulatorio tale che la colonna municipio in corrispondenza della colonna nome = A, è uguale a M*)

4.3 La politica di cancellazione dei dati

La conseguente traduzione in SQL dello schema logico ristrutturato, pone di fronte alla decisione di quali clausole scegliere per la cancellazione e l'aggiornamento delle tabelle. In questa sezione ci soffermeremo sulle scelte riguardanti la politica di cancellazione dei dati.

Per le tabelle:

- volontario
- servizio
- accompagnamento
- assistenzadomiciliare
- medico
- primariodirigente
- eventoricreativo
- svolge
- prenotavisita

si fa uso della clausola *ON DELETE CASCADE* nei vincoli di foreign key associati. Le righe delle tabelle in questione, vengono eliminate ogni qualvolta una tupla di una tabella esterna, identificata dalla chiave primaria, è cancellata.

Per il vincolo di integrità referenziale che lega centroanziani ad ambulatorio, è stato scelto di adottare una clausola *ON DELETE SET NULL* . In questa maniera, una volta eliminato un ambulatorio, gli attributi ambulatorio e municipioamb del centro anziani afferente vengono posti a *NULL* , senza eliminare la corrispondente tupla.

È stato descritto l'uso della clausola *ON DELETE* utilizzata nella definizione dei vincoli di foreign key. Parliamo adesso delle scelte compiute per la cancellazione dei dati attraverso funzioni che impongono transazioni.

Gli oggetti interessati da una eliminazione transazionale gestita a livello applicativo sono la cancellazione di un iscrizione di un anziano, l'eliminazione di un centro per anziani, di una prenotazione per un servizio offerto da un volontario, di un evento ricreativo, di un ambulatorio e infine di un centro anziani.

Per quanto riguarda il centro anziani, nel rispetto del vincolo di inclusione verso iscrizione, prima di procedere all'eliminazione della tupla dalla sua tabella, si eliminano tutte le righe afferenti ad esso dalla tabella iscrizione.

Inoltre vengono cancellati gli eventi registrati con le relative informazioni sui partecipanti e gli organizzatori.

In ultima istanza vengono fatti dei controlli per verificare le attività degli anziani dentro Roma, per meglio decidere l'eliminazione di questi dalle tabelle anziano e anzianoorganizzatore.

Non viene gestita transazionalmente l'eliminazione dei volontari afferenti al centro in quanto viene gestita dalla clausola *ON DELETE CASCADE* , impostata nel vincolo di integrità

referenziale che lega volontario a centroanziani. Tuttavia il volontario può essere eliminato solo nel caso sia registrato per errore; vale a dire che nell'ipotesi in cui un volontario offra un servizio e questo venga usufrutto da un anziano, o che coordini un evento ricreativo, la cancellazione dei suoi dati dal database è impedita.

Per quanto riguarda la gestione del vincolo di integrità referenziale verso municipio, nell'applicazione non è prevista la sua eliminazione, assunto per ipotesi che i municipi di Roma rimangano gli stessi per tutto il ciclo di vita dell'applicazione. Per brevità viene omessa una spiegazione dettagliata delle altre cancellazioni transazionali, ma si può certamente osservare la similitudine con la metodologia adottata per l'eliminazione del centro anziani.

L'eliminazione di un anziano dal database avviene automaticamente nel momento in cui non è più iscritto ad alcun centro anziani. Dato che interessa mantenere uno storico di tutte le attività, servizi ed eventi svolti dalla persona che si iscrive ad uno dei centri per anziani distribuiti sul territorio del comune di Roma, alcune funzioni di cancellazione come quella dei servizi sono impedite in determinate circostanze (i.e. la cancellazione di un servizio già usufrutto).

4.4 Gli attori del sistema

Il programma si avvale di un semplice sistema di login che permette a quattro utenti di usufruire delle sue funzioni in base ai diritti di accesso assegnati all'utente. Gli utenti, come definito negli use case potranno accedere a diverse funzioni.

Il **volontario** può chiaramente operare con i dati dell'anziano, assegnando i servizi di assistenza domiciliare e di accompagnamento, e prenotando le visite per questi. Inoltre può accedere a funzioni di query come le informazioni sui medici che possono visitare un anziano, le attività sociali svolte da un anziano, gli eventi ricreativi e i servizi del giorno e della settimana, etc.

L'utente **anziano** è invece quello più limitato, in quanto si riferirà al volontario e all'addetto del centro anziani per risolvere alcune sue funzioni. L'anziano può con i suoi privilegi accedere al sistema e registrare lo svolgimento di una attività sociale, partecipare o ritirarsi da un evento ricreativo, ed infine, come il volontario, può richiedere le informazioni sugli eventi e i servizi del giorno o della settimana.

L'utente **comune** di Roma potrà invece accedere alle informazioni di gestione dei centri anziani, quali l'inserimento e l'eliminazione di un medico, di un ambulatorio o dello stesso centro anziani. Inoltre potrà accedere ad informazioni generiche come l'elenco dei centri anziani di Roma con le relative informazioni associate, e l'elenco delle attività sociali svolte dagli anziani, senza però una possibile identificazione dell'anziano per una questione di privacy.

Il quarto utente, il **Centro Anziani**, che potrebbe essere rappresentato da un segretario, può assegnare o disdire servizi di assistenza domiciliare e di accompagnamento, e prenotare o disdire una visita come il volontario. Il centro anziani può però accedere ad alcune funzioni riservate come quelle di assunzione di un volontario, di iscrizione di un anziano al centro e naturalmente alle opposte funzioni rispettivamente di cancellazione di un volontario e di un anziano. Il centro anziani può anche richiedere informazioni riguardanti i volontari, gli anziani e i medici.

Un utente introdotto al solo scopo di testare l'applicazione e di non dover effettuare ripetutamente il login, è l'utente **admin** che può accedere a tutte le funzioni del programma.

Parte III

Realizzazione

Capitolo 5

Lo strato SQL

Una volta ultimata la ristrutturazione dello schema logico si può procedere alla realizzazione del livello fisico della base di dati, alla traduzione della logica in codice sorgente.

5.1 Il database e le tabelle

Per prima cosa viene creato il database con le tabelle associate, facendo attenzione a rispecchiare la struttura dello schema logico. Di seguito vengono fornite porzioni del codice che rappresentano parte del lavoro svolto.

Listing 5.1: *Parte del codice sql necessario alla creazione del database e delle tabelle*

```
1  DROP DATABASE IF EXISTS anzianidb;
2
3  CREATE DATABASE anzianidb;
4
5  ---------- parte omessa per brevità ----------
6
7  CREATE TABLE centroanziani (
8      nome VARCHAR(20) NOT NULL,
9      municipio ENUM ('I','II','III','IV','V','VI','VII','VIII','IX','X','XI','XII','XIII',
10     'XV','XVI','XVII','XVIII','XIX','XX') NOT NULL,
11     indirizzo VARCHAR(40) NOT NULL,
12     ambulatorio VARCHAR(20),
13     municipioamb ENUM ('I','II','III','IV','V','VI','VII','VIII','IX','X','XI','XII',
14     'XIII','XV','XVI','XVII','XVIII','XIX','XX'),
15     PRIMARY KEY (nome,municipio)
16     );
17
18 ---------- parte omessa per brevità ----------
```

5.2 I vincoli di foreign key

I vincoli di foreign key vengono aggiunti successivamente con un secondo script. Questi servono a creare dei riferimenti tra una coppia tabelle: riferimenti che devono essere rispettati in modo tassativo. In particolare esiste una relazione referenziata e una referente. Quest'ultima relazione possiede la chiave esterna che si riferisce appunto alla chiave primaria della relazione referenziata. In questo caso alcuni vincoli sono stati eliminati dal codice per spezzare i cicli di foreign key: nella fattispecie quelli afferenti ai vincoli fk_ambulatorioPR, fk_eventoricreativoOR. Naturalmente tali vincoli saranno gestiti opportunamente a livello applicativo.

Listing 5.2: *Parte del codice necessario all'inserimento dei vincoli esterni*

```
1  USE anzianidb;
2
3  ---------- parte omessa per brevità ----------
4
5  ALTER TABLE centroanziani ADD CONSTRAINT fk_centroanzianiMU
6  FOREIGN KEY (municipio)
7  REFERENCES municipio (nome)
8  -- ON DELETE RESTRICT      //è la default action quindi la possiamo togliere
9  ON UPDATE CASCADE;
10
11 ALTER TABLE centroanziani ADD CONSTRAINT fk_centroanzianiAM
12 FOREIGN KEY (ambulatorio, municipioamb)
13 REFERENCES ambulatorio (nome, municipio)
14 ON DELETE SET NULL
15 ON UPDATE CASCADE;
16
17 ---------- parte omessa per brevità ----------
```

5.3 Le procedure e i trigger associati

Grazie alle procedure si possono eseguire delle istruzioni sql in modo atomico e tali funzioni possono essere associate ad opportuni trigger, che possono essere immaginati come delle molle contenenti istruzioni che scattano al verificarsi di opportuni eventi. Di solito la procedura viene fatta partire *BEFORE INSERT ON*, ossia prima dell'inserimento di una nuova informazione in tabella. Nelle righe seguenti riportiamo alcune linee di codice che riportano un esempio di coppia procedura-trigger del programma gestione centri anziani.

Listing 5.3: *Esempio di procedura e trigger*

```
1  USE anzianiDB;
2
3  ---------- parte omessa per brevità ----------
4
5  DROP PROCEDURE IF EXISTS procedura_check_eta_anziano;
6
7  ---------- parte omessa per brevità ----------
8
```

```
9  DROP TRIGGER IF EXISTS trg_datana_anziano;
10
11 DELIMITER //
12
13 -- un anziano non può iscriversi al centro anziani se non ha almeno 60 anni
14 -- La procedura controlla se la data di nascita passata in input e' precedente
15 -- la data di oggi meno un intervallo di 60 anni, in caso contrario lancia un
16 -- eccezione.
17 -- La procedura e' usata dal trigger trg_datana_anziano
18 CREATE PROCEDURE procedura_check_eta_anziano (IN datanascita DATE)
19 BEGIN
20     IF((NOW() - INTERVAL 60 YEAR) < datanascita)
21     THEN CALL eccezione();
22     END IF;
23 END //
24
25 ---------- parte omessa per brevità ----------
26
27 -- Di seguito vengono dichiarati tutti i trigger per le procedure che devono essere
28 -- eseguite al verificarsi di un evento
29
30 ---------- parte omessa per brevità ----------
31
32 CREATE TRIGGER trg_datana_anziano BEFORE INSERT ON anziano
33 FOR EACH ROW
34 CALL procedura_check_eta_anziano (new.datana)//
35
36 DELIMITER ;
```

5.4 Scelta di implementazione dei vincoli

I vincoli che il programma deve rispettare sono stati implementati in vari modi. Dove possibile si è preferito usare il linguaggio SQL, utilizzando per quanto possibile i costrutti permessi da MySQL.

Per prima cosa sono stati dichiarati tutti i vincoli di riga che riguardavano il tipo di dato, quale poteva essere *INT* o *VARCHAR*. Poi è stato utilizzato il costrutto *ENUM* per alcuni attributi come il municipio e il nome degli eventi ricreativi, poiché risultavano insiemi finiti. Ogni attributo è stato settato a *NOT NULL* perché tutti i dati sono stati ritenuti necessari; eccezion fatta per l'ambulatorio e il suo municipio di residenza (nella tabella *CentroAnziani*), dove questi attributi vengono temporaneamente settati a *NULL* durante l'eliminazione di una tupla da ambulatorio.

Non è stato usato il costrutto CHECK in quanto incompatibile con MySQL e si è gestito il vincolo di positività dei vari *codici id* e degli altri attributi di tipo *INT* tramite dei controlli a livello applicativo.

Per quasi tutti i vincoli di tupla (lunghezza di una stringa, positività di un intero, appartenza ad un insieme predefinito di dati ...) è stato ritenuto opportuno un controllo sia a livello applicativo che a livello SQL. Questa scelta è stata adottata per rafforzare il controllo e non appesantire inutilmente il server MySQL.

Nel servizio di accompagnamento sono stati pensati dei *vincoli ecologici* riguardanti la relazione tra il mezzo utilizzato (auto, piedi, mezzo pubblico) e la distanza percorsa dal centro anziani al luogo di accompagnamento (espressa in km). Il vincolo, che esprime l'impossibilità di assumere il valore *piedi* per distanze superiori ai 9 km e *auto* per percorsi inferiori ai 5 km, viene espresso nella classe *AccompagnamentoController*.

```
1  //VINCOLI ECOLOGICI :) */
2  if(dist>9 && mezzo.equalsIgnoreCase("piedi")) {
3     throw new ApplicationException("Distanza troppo elevata per andare a piedi");
4  }
5  if(dist<5 && mezzo.equalsIgnoreCase("auto")) {
6     throw new ApplicationException("Distanza troppo piccola per utilizzare l'auto,
7  scegliere un altro mezzo di trasporto");
8  }
```

Durante la fase di testing introducendo stringhe vuote nei campi di attributi stringa, queste venivano viste dall'interprete MySQL come NOT NULL e quindi venivano inserite ugualmente. Per evitare questi inserimenti inutili sono stati introdotti dei controlli su tutte le stringhe a livello delle classi di tipo controller.

```
1  if(stringa == null || stringa.equals(""))
2     throw new ApplicationException("stringa obbligatoria");
```

I vincoli di foreign key sono tutti stati implementati a livello sql con il costrutto:

```
1  ALTER [ONLINE | OFFLINE] [IGNORE] TABLE tbl_name
2      alter_specification [, alter_specification] ...
```

I vincoli di foreign key,

```
1  ALTER TABLE ambulatorio ADD CONSTRAINT fk_ambulatorioPR
2  FOREIGN KEY (nome, municipio)
3  REFERENCES primariodirigente (ambulatorio, municipio);
4
5  ALTER TABLE eventoricreativo ADD CONSTRAINT  fk_eventoricreativoOR
6  FOREIGN KEY (nome, dataev, volontario, centroanziani, municipio)
7  REFERENCES organizza(eventoricreativo, dataeventoricreativo, volontario,
8  centroanziani, municipio)
9  ON DELETE CASCADE
10 ON UPDATE CASCADE;
```

sono stati eliminati per spezzare dei cicli. In ogni caso tali vincoli sono stati implementati a livello applicativo utilizzando transazioni.

L'implementazione dei vincoli di tabella e inter-tabella è stata effettuata tramite degli opportuni trigger che scattano al momento dell'inserimento nelle tabelle interessate dai vincoli, qualora le procedure associate verifichino determinate condizioni.

Capitolo 6

L'architettura dell'applicazione

L'ultimo passo della fase di realizzazione prevede il completamento dello sviluppo del programma attraverso la realizzazione dello strato applicativo. L'architettura software utilizzata per lo sviluppo del programma è chiamata ***three-tier***, un modello ingegneristico in cui i *presentation layer*, *application logic layer*, e *resource management layer* risultano disaccoppiati; in altre parole il programma è diviso in tre moduli ben distinti, dedicati rispettivamente all'interfaccia grafica, alla gestione delle funzioni e a quella dei dati persistenti.

6.1 Il diagramma delle classi

Nella pagina seguente verrà illustrato il diagramma delle classi UML, rappresentante la modellazione astratta della struttura delle classi Java per la costruzione del nostro programma.

Premettiamo che le celle rettangolari rappresentano le *classi*, le rette di collegamento tra una classe e l'altra indicano un'associazione di relazione, e le frecce segnalano l'uso di una gerarchia tra le classi, strutturata secondo uno dei tre principi fondamentali dell'*Object Oriented Programming*: l'ereditarietà. Ogni classe ha al suo interno una descrizione sotto forma di elenco di attributi, specificati in base al tipo di dato.

La realizzazione di questo schema ci aiuterà nella progettazione delle classi di dominio *DC*, secondo i principi impartiti durante il corso di Progettazione del Software.

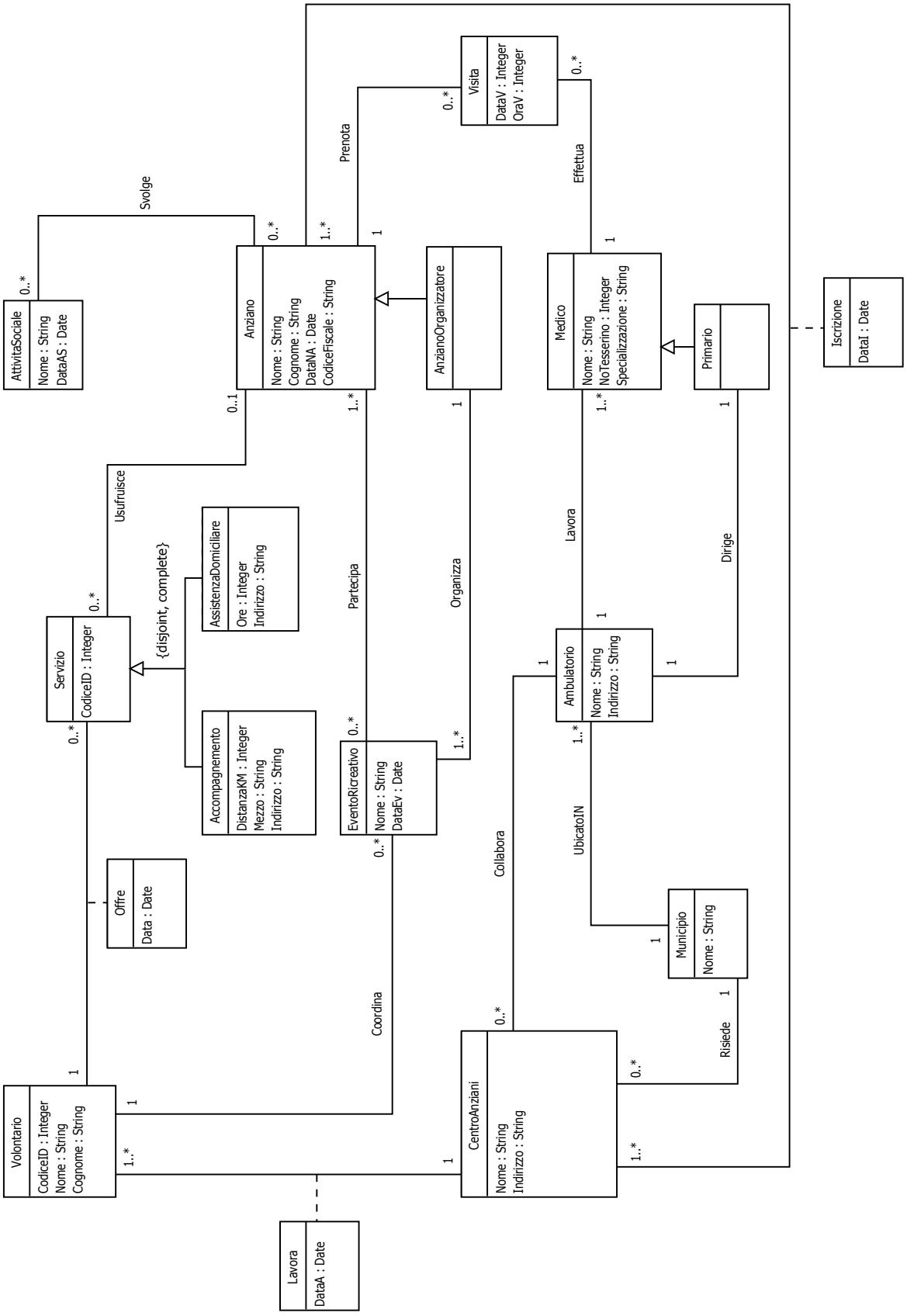

Figura 6.1: *Diagramma delle classi dell'applicazione*

6.2 Tabella delle responsabilità

Associazione	**Classi**	**Responsabilità**
Offre	Volontario	SI_2
	Servizio	$SI_{1,3}$
Usufruisce	Anziano	NO
	Servizio	SI_3
Svolge	Anziano	SI_2
	AttivitaSociale	NO
Coordina	Volontario	SI_2
	EventoRicreativo	$SI_{1,3}$
Partecipa	Anziano	NO
	EventoRicreativo	SI_3
Organizza	AnzianoOrganizzatore	SI_3
	EventoRicreativo	SI_3
Iscrizione	Anziano	SI_3
	CentroAnziani	SI_3
Prenota	Visita	SI_3
	Anziano	SI_2
Effettua	Visita	$SI_{1,3}$
	Medico	NO
Lavora	Ambulatorio	SI_3
	Medico	$SI_{1,3}$
Dirige	Ambulatorio	SI_3
	Primario	SI_3
Collabora	CentroAnziani	SI_3
	Ambulatorio	NO
Risiede	CentroAnziani	$SI_{1,3}$
	Municipio	NO
UbicatoIn	Ambulatorio	$SI_{1,3}$
	Municipio	SI_3
Lavora	Volontario	$SI_{1,3}$
	CentroAnziani	SI_3

Tabella 6.1: *Tabella delle responsabilità*

Legenda

Per ogni associazione A, deve esserci almeno una delle classi coinvolte che ha responsabilità su A. I criteri [1] per comprendere se una classe C ha responsabilità sull'associazione A sono i seguenti:

1. Esiste un riferimento nel documento dei requisiti, da cui si capisce che per ogni oggetto x che è istanza della classe C vogliamo poter eseguire almeno una delle operazioni di load, insert, delete o update.
2. Esiste una funzione negli usecase che necessita la responsabilità della classe C per essere implementata.
3. La responsabilità è logicamente implementata per i vincoli di molteplicità dell'associazione.

6.3 Il package diagram dell'applicazione

Qui sotto la è riportata struttura gerarchica del programma con enfasi ai collegamenti comunicativi tra le sue directory, rappresentati attraverso delle linee tratteggiate.

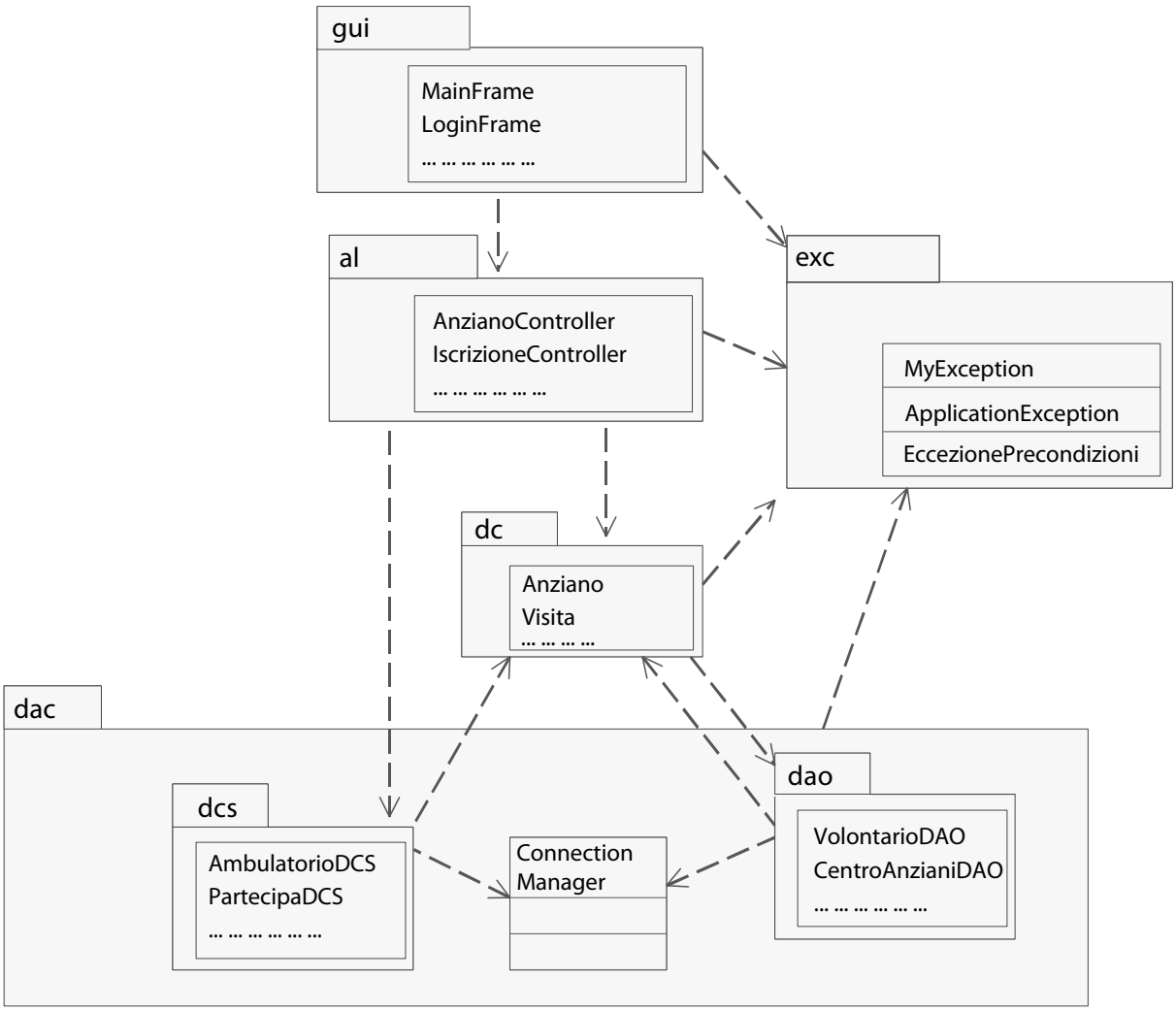

Figura 6.2: *Diagramma dei package*

6.4 Ottimizzazione dell'interazione con l'utente

L'interfaccia grafica assume un ruolo sempre più determinante nella riuscita di un buon software. Essa determina infatti il modo in cui l'utente interagisce con il programma.

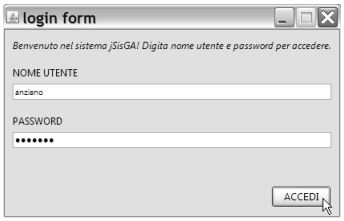

Figura 6.3: *screenshot della schermata di accesso*

Una buona interfaccia grafica è essenziale per l'usabilità del programma. In questa pagina riportiamo la cattura di una schermata del software realizzato.

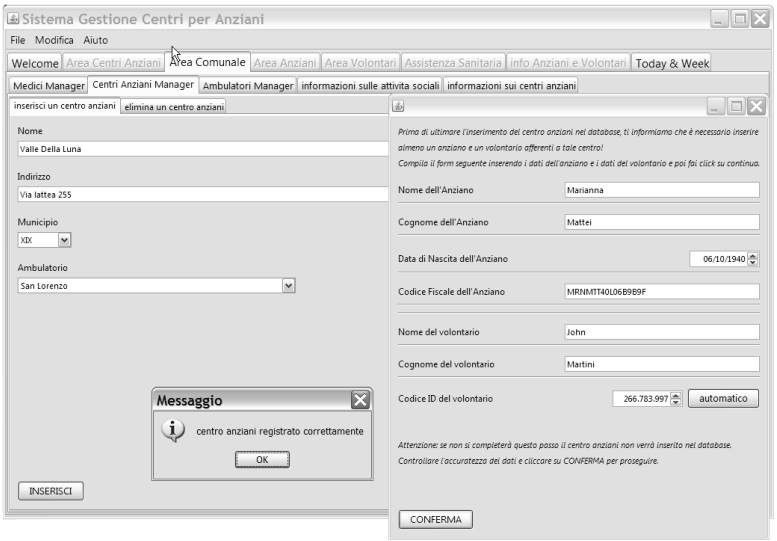

Figura 6.4: *screenshot del programma*

Data la complessa fase di configurazione si è pensato di svincolare l'utente dal dover digitare tutti i comandi di creazione e popolamento del database, attraverso la scrittura di un programma che permettesse queste operazioni in modo automatizzato.

Figura 6.5: *screenshot prima configurazione*

Inoltre si è usato IzPACK, basato su tecnologia Java, per creare un installer multi-piattaforma.

Figura 6.6: *screenshot dell'installatore automatizzato*

Grazie a poche linee di codice Java e XML è stato possibile rendere alla portata di tutti la configurazione e l'installazione del programma.

Conclusioni

Concludo la trattazione con alcune idee per uno sviluppo futuro del programma.

Una possibile variante del programma prevede una sua versione utilizzabile attraverso un browser web, secondo i principi del cloud computing, rendendo disponibili dati e funzionalità indipendentemente dalla postazione di accesso. Questo approccio eliminerebbe totalmente l'onere di installazione e configurazione a carico dell'utente, semplificando e sincronizzando il lavoro dei centri anziani distribuiti nel territorio di Roma.

Alcune funzionalità utili si potrebbero aggiungere grazie alle API di servizi come google maps (così come live maps o yahoo maps), per rendere facilmente individuabili le aree geografiche degli indirizzi di residenza degli anziani, in merito ai servizi di assistenza domiciliare.

Bibliografia

[1] Lembo De Giacomo. Dispense di progettazione del software 1. Sapienza, Roma.

[2] Lenzerini De Giacomo. Dispense di basi di dati. Sapienza, Roma.

[3] Antonella Poggi. Dispense del progetto di basi di dati. Sapienza, Roma.

[4] Gehrke Ramakrishnan. *Sistemi di basi di dati*. McGraw-Hill, italian edition, 2004.