Relazione progetto di Ingegneria del software 2019

Edoardo Zorzi, Elia Piccoli, Marian Statache

Luglio 2019

1 Ingegneria e sviluppo

1.1 Organizzazione iniziale del processo di sviluppo

Decisioni organizzative La dimensione del team di sviluppo — tre persone — e i diversi livelli di interesse relativi agli 'ambiti' di programmazione concernenti tale progetto, espressi inizialmente dai soggetti del team, hanno portato alla decisione di suddividere in modo moderatamente netto i compiti assegnati alle diverse persone, perlomeno nella fase iniziale, pre-design.

Specificatamente, la decisione è stata quella di assegnare a Marian il compito di creare e sviluppare la interfacce grafiche di tutti i componenti costituenti il sistema nel suo complesso e quello di ideare e sviluppare la GUI generale, e in particolar modo di considerare i modi con cui gli utenti si aspettano di interagire con il sistema e quindi di sviluppare accordatamente le relative interfacce grafiche.

Ad Elia ed Edoardo invece è stato assegnato il compito di progettare e sviluppare la back-end, l'architettura del sistema generale e le interfacce di comunicazione e interazione dei diversi componenti. Pur non avendo definito inizialmente la suddivisione ulteriore di questi compiti, nel corso del processo di sviluppo e programmazione del sistema, a seguito della fase di progettazione in cui si sono prese decisioni relative a quali design patterns usare, la progettazione dell'architettura è stata a grandi linee divisa in due aree: lo sviluppo dei controllori e dei modelli dei componenti, secondo il pattern MVC, assegnata ad Elia, e lo sviluppo e gestione dello stato centralizzato, propagato poi seguendo il pattern Observer, assegnata invece ad Edoardo. Data la fondamentale connessione tra tali due aree da un certo punto in poi lo sviluppo dell'architettura, e in particolar modo dell'interfaccia di comunicazione tra stato e controllori dei componenti, è stata eseguita in modo unico dai i due membri del team che hanno seguito quasi esclusivamente la tecnica del *dual programming*, che ha portato a scrivere gran parte del codice insieme, a turno, uno con l'input dell'altro: questo sia per permettere di migliorare la comprensione dell'architettura nel suo complesso e delle interazioni tra le parti, sia per permettere di sviluppare codice che sin da subito rispettasse i due diversi approcci di programmazione e che unisse nel modo più chiaro possibile le interfacce tra le due aree.

Gestione del codice Per permettere, soprattutto inizialmente, a tutti i membri del team di contribuire al progetto in modo personale senza creare conflitti nel codice si è deciso sin da subito di utilizzare un sistema di versionamento: la scelta è ricaduta subito su *Git*, data la sua ubiquità e portabilità, installato localmente sulle macchine dei membri del team e riferente una repository remota privata localizzata su *Github*. Oltre che a permettere di evitare conflitti e di mantenere coerente lo stato del progetto per tutti i membri questa scelta è stata molto importante soprattutto per risolvere diversi problemi relativi all'installazione e utilizzo di un programma per la gestione del database: l'uso di diverse *branches*, utilizzate anche per lo sviluppo di parti sperimentali del sistema, ha permesso lo sviluppo in contemporanea, almeno fino alla risoluzione di tutti i problemi

di installazione e configurazione, del codice concernente la gestione del database e di quello relativo alle interfacce grafiche e i loro controllori.

Gestione dei dati La chiara natura relazionale dei dati necessari per il funzionamento del sistema ha subito portato alla decisione di affidarsi ad un RDBMS per la loro gestione, e in particolare a PostgreSQL; tale scelta è stata motivata principalmente dalla previa esperienza di utilizzo di un membro del team. Oltre che ad utilizzare un RBDMS si è inoltre deciso di affidarsi ad un ORM — Hibernate — per il mapping tra entità nel database e oggetti in memoria: questa decisione è derivata soprattutto dalla volontà di mantenere semplice la logica del sistema che, per sua natura e limitata dimensione, non ha mai richiesto la maggior efficienza derivante dall'utilizzo di pure query SQL.

1.2 Ingegneria dei requisiti

Data la natura didattica del progetto non si sono seguite le classiche tecniche di elicitazione dei requisiti ma essi sono stati ricavati e studiati a partire dalla specifica di consegna, considerata, almeno in parte, come documento dei requisiti. La stesura iniziale delle principali funzionalità, sotto forma di un provvisorio schema dei casi d'uso, è stata eseguita nella fase di pre-design da parte di tutti i membri del team basandoci appunto su tale documento dei requisiti.

Nel corso dello sviluppo del progetto, a seconda delle diverse funzionalità da implementare, sono anche stati stilati diversi casi d'uso per i vari utenti finali secondo un processo simile a quello adottato dai modelli agili: prima di implementare funzionalità specifiche del sistema si è pensato ad almeno un caso d'uso reale di tali funzionalità per permettere poi un'implementazione quanto più semplice, naturale e rispettante le aspettative e necessità degli utenti.

Di seguito parte dei casi d'uso utilizzati, aggregati per chiarezza nei vari utenti finali.

Primario

Caso d'uso	**Compilazione lettera di dimissioni**
Utente:	Primario
Precondizioni:	Il primario deve essersi autenticato
Passi	– Nella propria dashboard seleziona la finestra per compilare le lettere di dimissioni – Seleziona da un menù a tendina i pazienti ricoverati in attesa di essere dimessi – Visualizza nella schermata i dati sommari del ricovero e compila la lettera di dimissione, e conferma la dimissione premendo il tasto di conferma
Postcondizioni:	Il paziente viene dimesso e il suo ricovero non compare più nel menù

Caso d'uso	**Visualizzazione e stampa dei reports dei ricoveri**
Utente:	Primario
Precondizioni:	Il primario deve essersi autenticato
Passi	– Nella propria dashboard seleziona la finestra per visualizzare i reports dei ricoveri settimanali – Seleziona da un menù a tendina, o tramite ricerca, i ricoveri settimanali, attivi o non attivi – Visualizza nella schermata i dati sommari dei ricoveri e dei pazienti. Eventualmente può stampare tali report cliccando su un pulsante che genera un documento pdf
Postcondizioni:	Vengono visualizzati i reports dei ricoveri settimanali. Se il primario ha premuto il bottone per stamparli, viene generato un documento pdf con i dati richiesti

Caso d'uso	**Visualizzazione dati dei pazienti ricoverati (ultime due ore)**
Utente:	Primario, medico, infermiere
Precondizioni:	L'utente deve essersi autenticato
Passi	– Nella propria dashboard l'utente seleziona la finestra per visualizzare i dati dei pazienti ricoverati – L'utente visualizza, in una tabella, i dati aggiornati in tempo reale, ed esclusivi delle ultime due ore, di tutti i pazienti ricoverati
Postcondizioni:	L'utente visualizza le informazioni, non più vecchie di due ore, sui parametri vitali dei pazienti ricoverati

Medico

Caso d'uso	**Spegnimento allarme**
Utente:	Medico

Precondizioni:	Deve essere partito un allarme
Passi	– In un finestra compaiono informazioni circa il paziente e la condizione segnalata – Indica le attività svolte per portare il paziente alla normalità – 1. se è gia autenticato, spegne l'allarme 2. se non è autenticato, spegne l'allarme indicando le proprie credenziali
Postcondizioni:	L'allarme viene spento. Se il medico non era autenticato e ha inserito correttamente le sue credenziali rimarrà autenticato per un certo periodo di tempo

Caso d'uso	**Ammissione dei pazienti in attesa**
Utente:	Medico
Precondizioni:	Il medico deve essersi autenticato; un infermiere deve avere aggiunto i dati anagrafici dei pazienti in attesa di ricovero
Passi	– Nella propria dashboard seleziona la finestra per visualizzare i pazienti in attesa di ricovero – Da un menù a tendina sceglie tra i diversi pazienti in attesa, visualizzando informazioni sommarie – Se accetta di ammetterlo, da un campo di testo inserisce la diagnosi e preme un bottone di conferma 1. se il numero di pazienti attualmente ricoverati è minore di dieci, viene ammesso 2. altrimenti compare una finestra di errore
Postcondizioni:	Se il numero di pazienti attualmente ricoverati è minore di dieci e ha accettato di ammetterne uno inserendo la diagnosi, allora il nome del paziente scompare dal menù a tendina ed esso sarà compreso tra quelli ricoverati

Caso d'uso	**Aggiunta di prescrizioni**
Utente:	Medico
Precondizioni:	Il medico deve essersi autenticato; il paziente a cui si aggiunge una prescrizione deve essere ricoverato
Passi	– Nella propria dashboard seleziona la finestra per aggiungere una prescrizione – Da un menù a tendina sceglie tra i diversi pazienti attualmente ricoverati, visualizzando, a seconda della selezione, informazioni sommarie sul paziente – In diversi campi di testo aggiunge il nome del medicinale, la dose giornaliera, la quantità per dose e la durata della terapi e preme il pulsante di conferma
Postcondizioni:	La prescrizione viene aggiunta a quelle associate al paziente ricoverato.

Infermiere

Caso d'uso	**Compilazione dati anagrafici paziente**
Utente:	Infermiere
Precondizioni:	L'infermiere deve essersi autenticato
Passi	– Nella propria dashboard seleziona la finestra per inserire i dati anagrafici dei pazienti – In diversi campi di testo aggiunge i dati anagrafici del paziente, tra cui nome, congnome, luogo e data di nascita. Premendo un pulsante può generare, sulla base di tali dati, il codice fiscale corretto del paziente
Postcondizioni:	Il paziente è inserito in stato di attesa

Caso d'uso	**Aggiunta di somministrazioni**
Utente:	Infermiere

Precondizioni:	L'infermiere deve essersi autenticato; il paziente a cui viene aggiunta una somministrazione deve essere ricoverato e avere almeno una prescrizione registrata
Passi	– Nella propria dashboard seleziona la finestra per aggiungere una somministrazione – Seleziona da un menù a tendina il paziente e il medicinale prescritto per cui può aggiungere una somministrazione – Visualizza nella schermata i dati sommari della prescrizione, tra cui le massime dosi giornaliere; in un campo di testo può inserire note circa la somministrazione e la conferma premendo un pulsante
Postcondizioni:	La somministrazione è aggiunta al paziente, associata alla data prescrizione

1.3 Design del sistema

La scelta di design iniziale del sistema è stata fatta a tavolino tra tutti i membri del team prima di implementare qualsiasi funzionalità. Dopo aver letto attentamente la specifica e aver redatto lo schema dei casi d'uso, provvisorio, si sono discussi diversi possibili approcci e in particolare si è cercato di determinare se e quali *design patterns* fossero i più adatti da usare al fine di costruire il sistema generale.

Dopo diverse discussioni e lavoro di gruppo e individuale si è deciso a grandi passi di adottare una architettura generalmente simile a quella adottata da alcuni frameworks di sviluppo web: un sistema centralizzato (*repository*) di gestione e salvataggio dello stato che ad ogni modifica è propagato a tutta l'applicazione in ascolto e che si aggiorna in modo asincrono (*observer pattern*); componenti decentralizzati, creati da *factories* adoperate dal sistema centrale, che rappresentano singole finestre e che comprendono sia la *view*, cioè quello che vede l'utente, sia la business logic (*controller*) che determina come si reagisce alle diverse interazioni e modifiche dello stato centrale, oltre che ai dati (*model*) ottenuti e aggiornati centralmente dal gestore dello stato — *pattern MVC*.

Oltre a questo è stato spesso fatto uso della *dependency injection* per garantire una maggiore modularità dei componenti e limitare la dipendenza da classi statiche e stato nascosto, spesso rischioso e prono ad errori. ***TODO (spiegazione del design dello stato, completa patterns)***

1.4 Implementazione

Heila ***TODO***

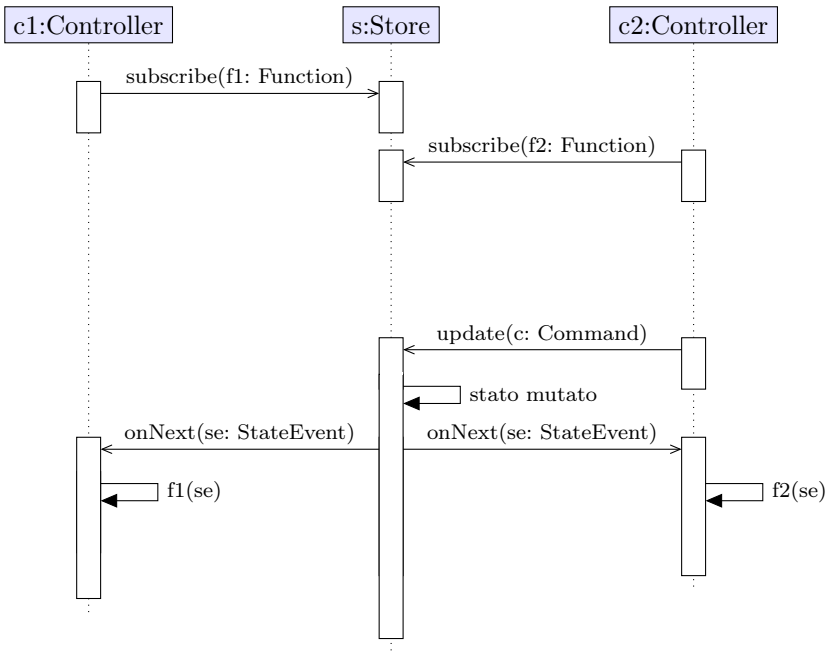

Figura 1 *Sequence diagram* dell'interazione di due generici controllers con lo stato centralizzato gestito dallo Store. Dopo aver invocato in modo non bloccante *subscribe()*, passandogli una funzione, qualsiasi *update()* chiamato sullo store comporta una mutazione interna dello stato, sulla base del comando passato: il nuovo stato viene quindi propagato a tutti i controllori 'sottoscritti' tramite *onNext()*; la funzione precedentemente passata in *subscribe* sarà quindi chiamata, all'interno di tutti i controllori, sul nuovo stato ricevuto dallo Store.

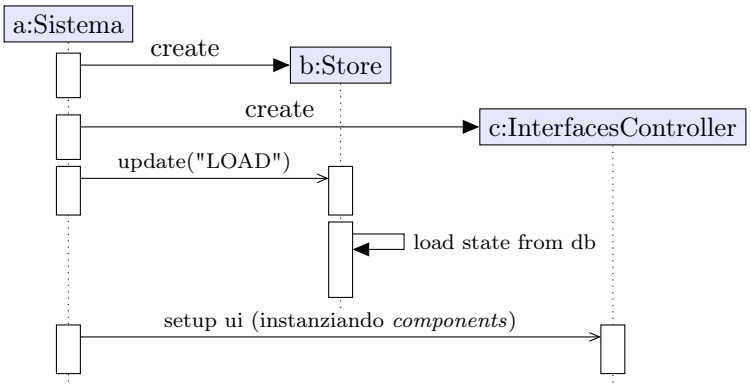

Figura 2 *Sequence diagram* delle operazioni eseguite durante il bootstrapping del programma.

1.5 Validazione e verifica

Le operazioni di validazione e verifica sono state fondamentali per garantire che il progetto rispettasse la specifica e i requisiti, e che si comportasse in modo coerente e prevedibile, senza errori, a tutti gli input previsti e operazioni eseguibili degli utenti.

Validazione Le operazioni di validazione sono state eseguite facendo un check periodico, dopo ogni implementazione di funzionalità non triviali, del rispetto della specifica per verificare sia che tutti i vincoli temporali e di caratteristiche dei dati fossero rispettati (come ad esempio i tipi di ricovero da visualizzare o le informazioni sui parametri vitali dei pazienti, limitati a dati intervalli temporali), sia che tali funzionalità corrispondessero a quello richiesto dal software finale. Questo è stato soprattutto importante dopo l'implementazione delle funzionalità di ogni utente: data la non

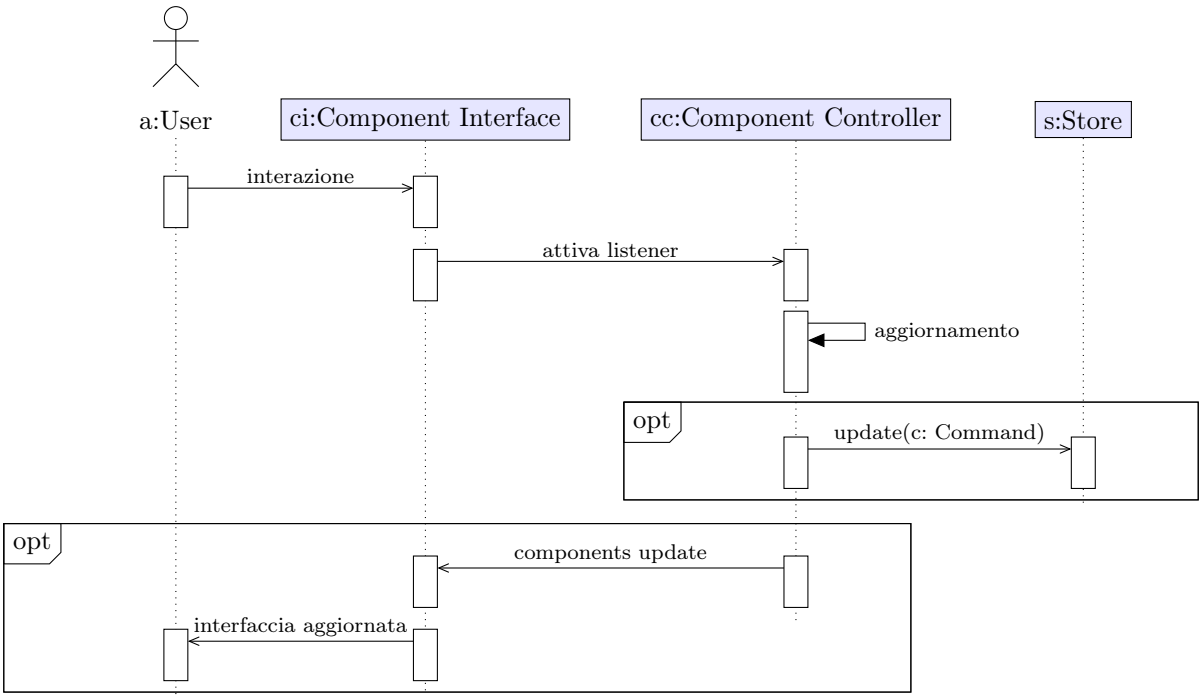

Figura 3 *Sequence diagram* che mostra le operazioni che avvengono a seguito di una generica interazione di un utente con il software. Premendo qualche pulsante, inserendo dati, cliccando tasti della tastiera un utente interagisce con l'interfaccia a cui è associato un *controller*: a seconda dell'azione vengono eseguite certe operazioni interne che aggiornano lo stato del controllore (ed eventualmente un *update* è inviato allo store.) A seguito di questo è possibile che ci sia un aggiornamento dell'interfaccia, come mostrata all'utente.

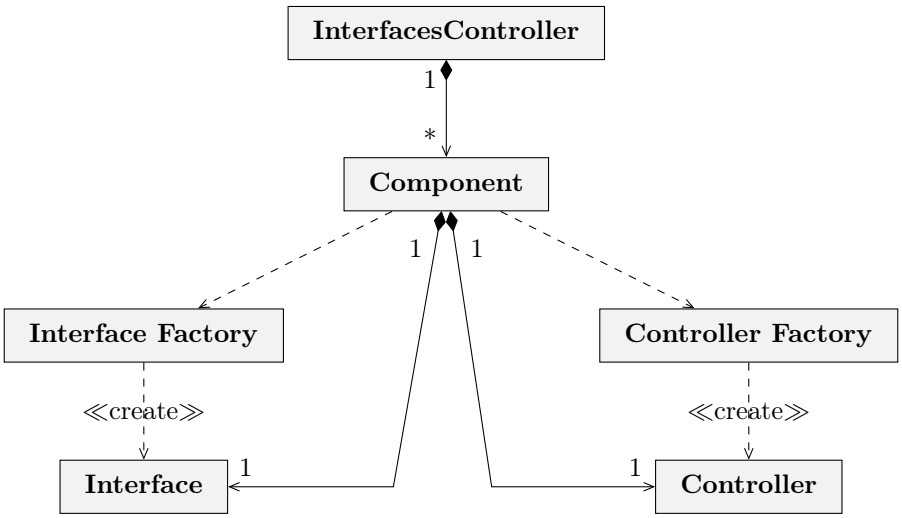

Figura 4 *Class diagram* che mostra la relazione tra InterfaceController e i vari oggetti che costituiscono la parte View-Controller dell'architettura *MVC* dell'applicazione.

modesta quantità di funzioni da mettere a disposizione e dati da considerare/manipolare in molti casi il check di verifica eseguito successivamente ha messo alla luce rilevanti criticità e importanti modifiche da implementare. Oltre a queste verifiche periodiche si è anche eseguito un controllo finale, al ridosso della scadenza, per validare il sistema nel suo complesso e cercare di individuare eventuali omissioni e elementi non completamente rispettosi della specifica e dei requisiti.

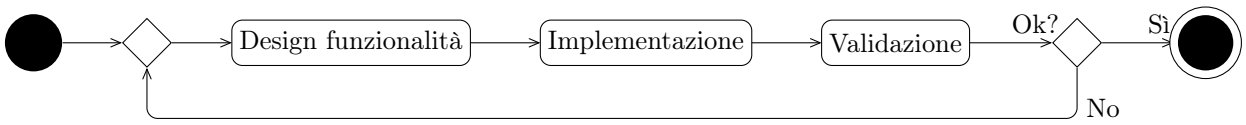

Figura 5 *Activity diagram* della fase di validazione per le funzionalità dei componenti utente.

Verifica Molto importante è stata anche la verifica della correttezza del programma che data la sua non banale complessità ha mostrato spesso comportamenti sbagliati in fase di programmazione. A tal fine si è deciso di operare nel seguente modo:

- una fase di *development testing* utilizzando *unit tests* per tutte quelle classi e aree di più facile testabilità automatica: quelle meno dipendenti dagli input specifici dell'utente, eseguite dall'interfaccia grafica, e dall'integrazione con altri oggetti del sistema. Essi sono stati eseguiti affindandosi ad una libreria esterna, *JUnit4*, e sono stati principalmente usati per verificare il comportamento della classe *Sistema*, il 'bootstrapper' del programma, del gestore centralizzato dello stato, di implementazione interna complessa, e delle diverse entità (vedi figura 14) che lo costituiscono le quali hanno richiesto particolare attenzione soprattutto in termini di gestione del loro stato interno e delle sue mutazioni, che se avvenisse in modo sbagliato provocherebbe errori di alta severità: pazienti con più ricoveri attivi, in uno stato non ben definito (non *in attesa*/*ricoverato*/*dimesso*), somministrazioni non associate a prescrizioni etc.

- una fase di *release testing* dove si è cercato di testare tutti i comportamenti possibili degli utenti finali simulando diverse casistiche e flow di lavoro per verificare che il sistema gestisse correttamente gli input. In particolare si è testato il corretto comportamento di tutti i pulsanti e dei menù e si è verificato che le diverse entità, consistenti lo stato del sistema, fossero correttamente gestite, manipolate, e visualizzate nelle diverse view degli utenti. Questo tipo di test è stato eseguito a parte da tutti i membri del team cercando di simulare il comportamento di un soggetto non tecnico.

- una fase di *user testing* dove si è fatto testare il software ad un soggetto di media/moderata abilità informatica per verificare l'usabilità del programma di fronte ad input reali e sostanzialmente conformi a quelli che ci si aspetta da soggetti normali che utilizzerebbero in modo coerente il programma. Questa fase è stata molto utile per implementare piccole migliorie all'usabilità del programma come ad esempio un ritocco del comportamento dei menù a tendina, modifiche ad alcune tabelle — al fine di includere dati differenti o formattati diversamente — e anche lievi cambiamenti alla struttura e disposizione degli elementi dell'interfaccia grafica (tipo rendere alcuni pulsanti non attivi, cambiare alcune grafiche etc.).

2 Specifica

2.1 Casi d'uso

2.2 Diagrammi di classe

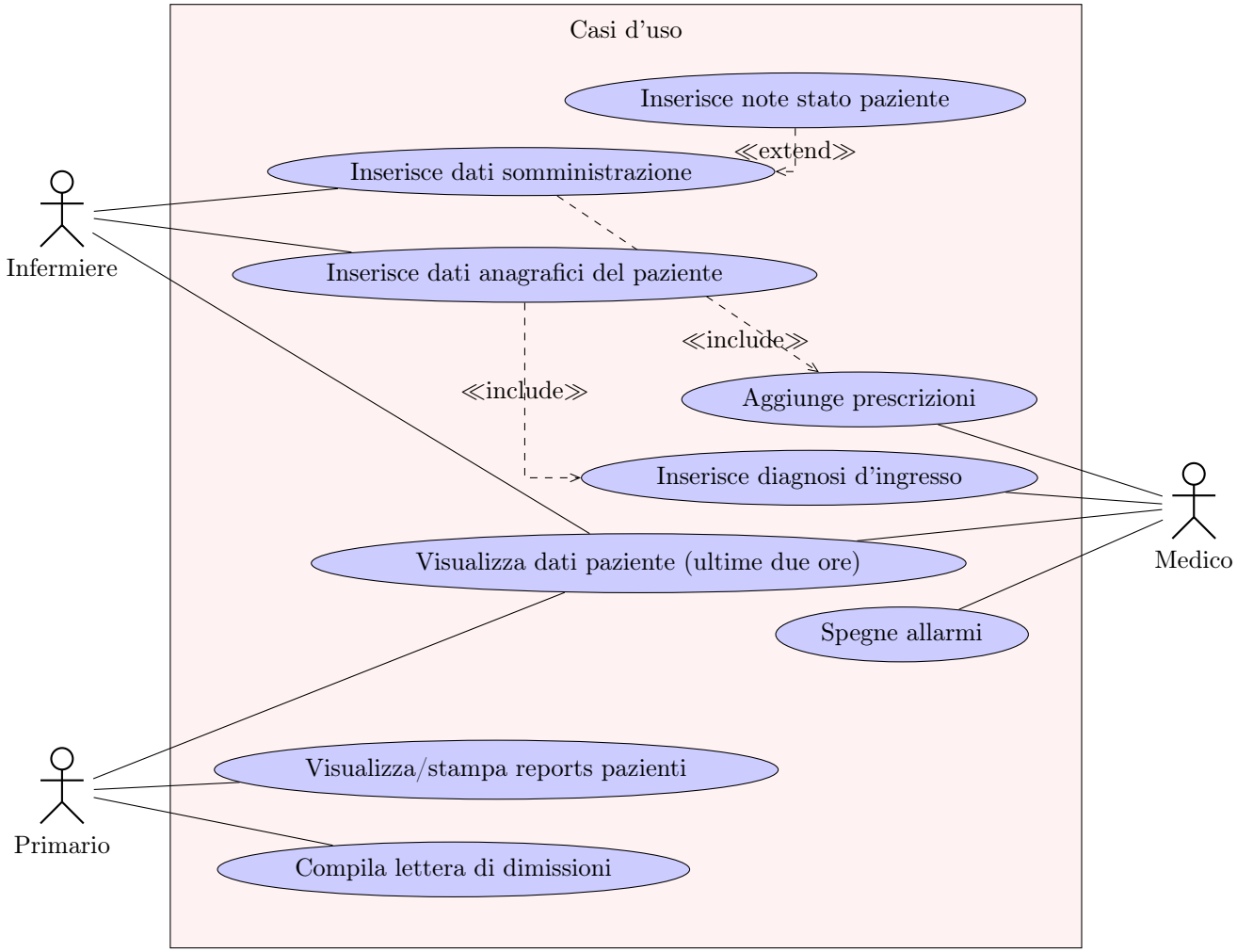

Figura 6 Diagramma UML dei casi d'uso del sistema.

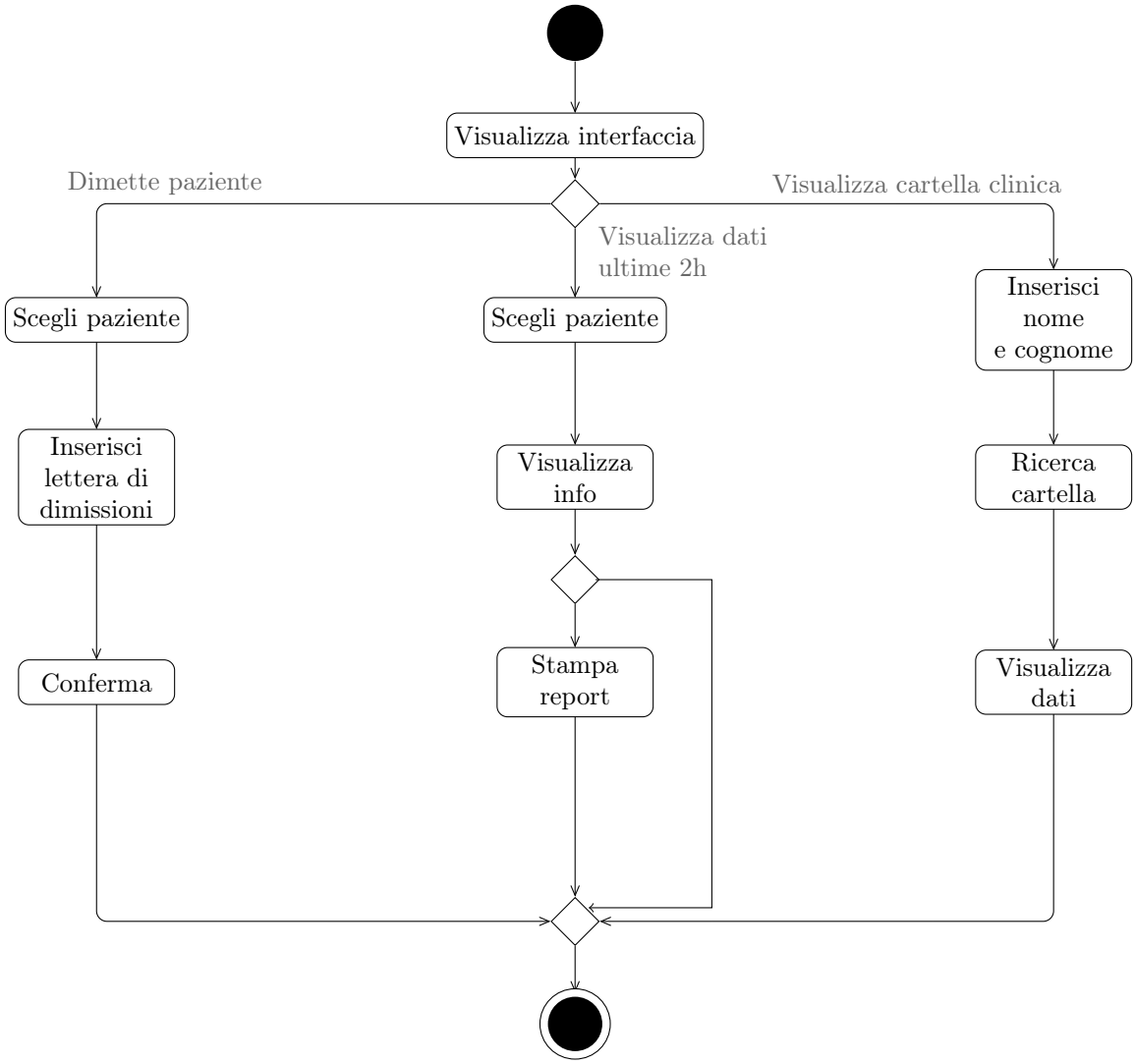

Figura 7 *Activity diagram* delle diverse azioni che possono essere compiute dal primario.

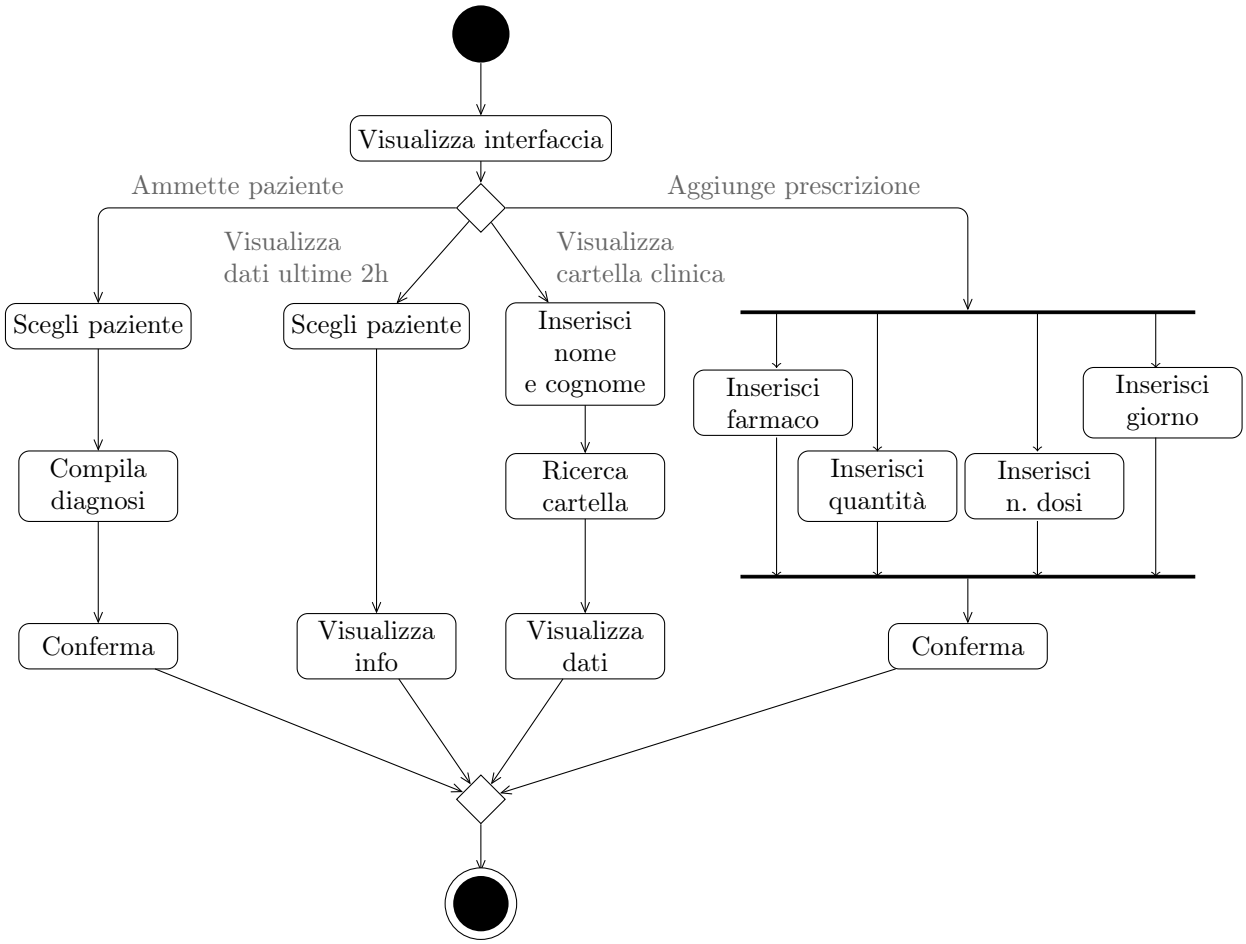

Figura 8 *Activity diagram* delle diverse azioni che possono essere compiute dal medico.

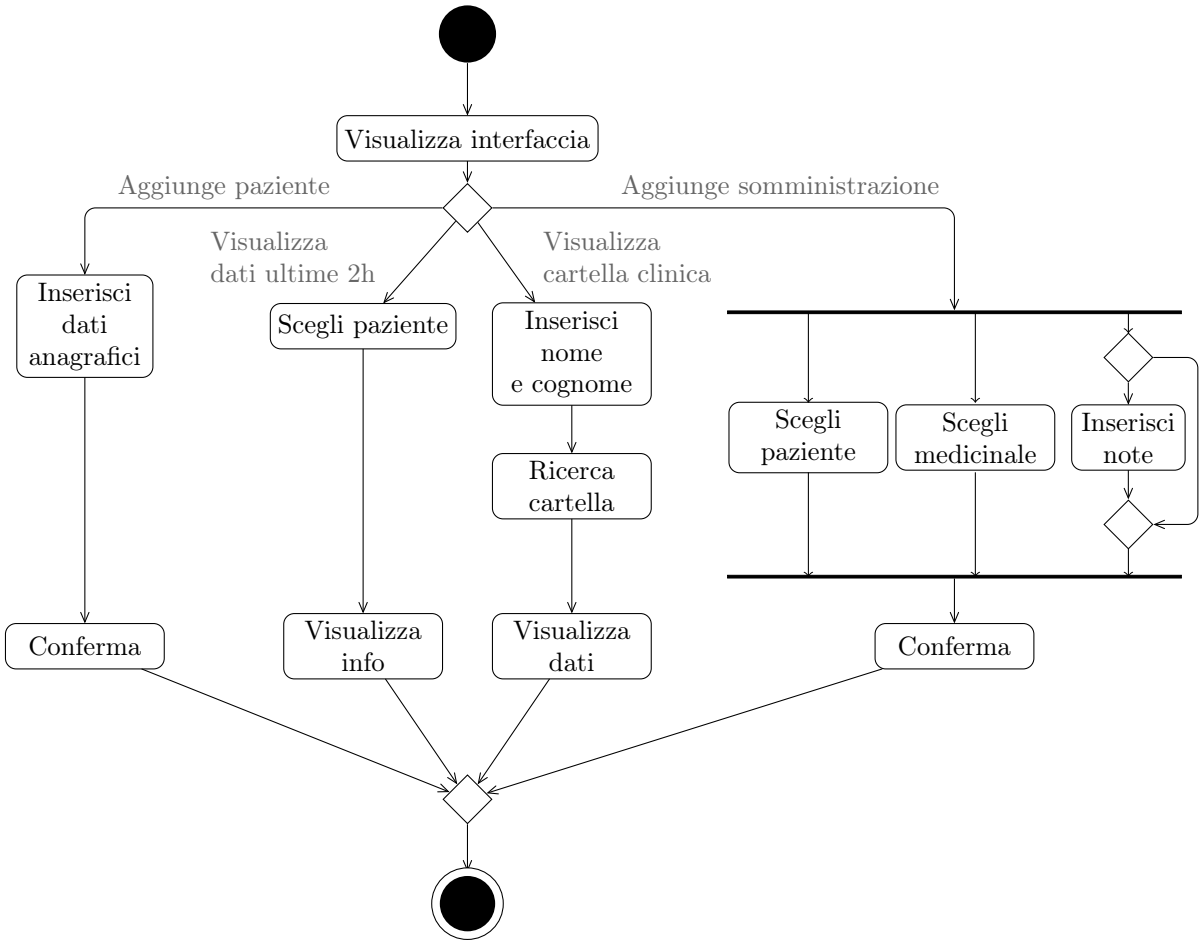

Figura 9 *Activity diagram* delle diverse azioni che possono essere compiute dall'infermiere.

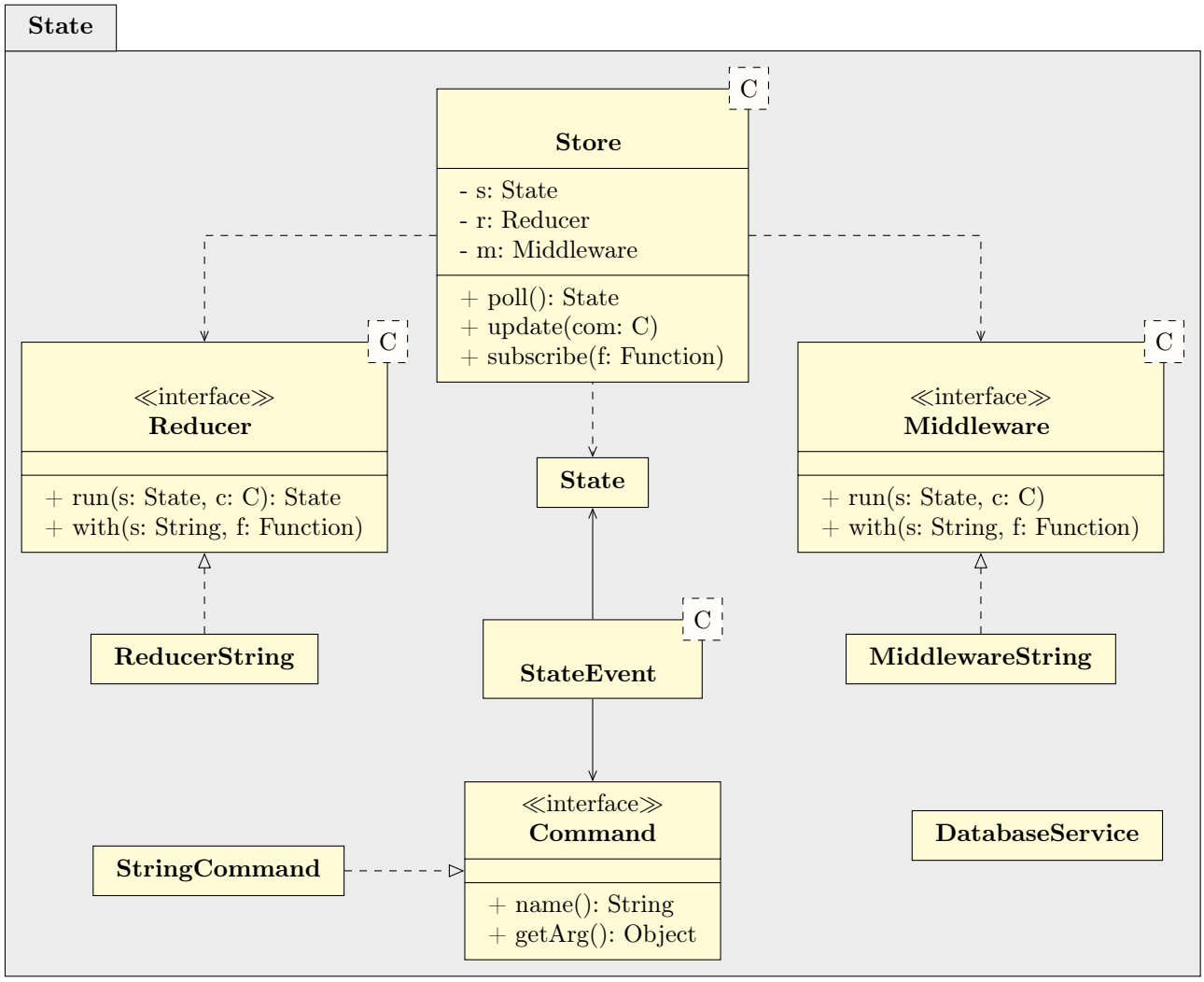

Figura 10 Diagramma di classe del *package* State.

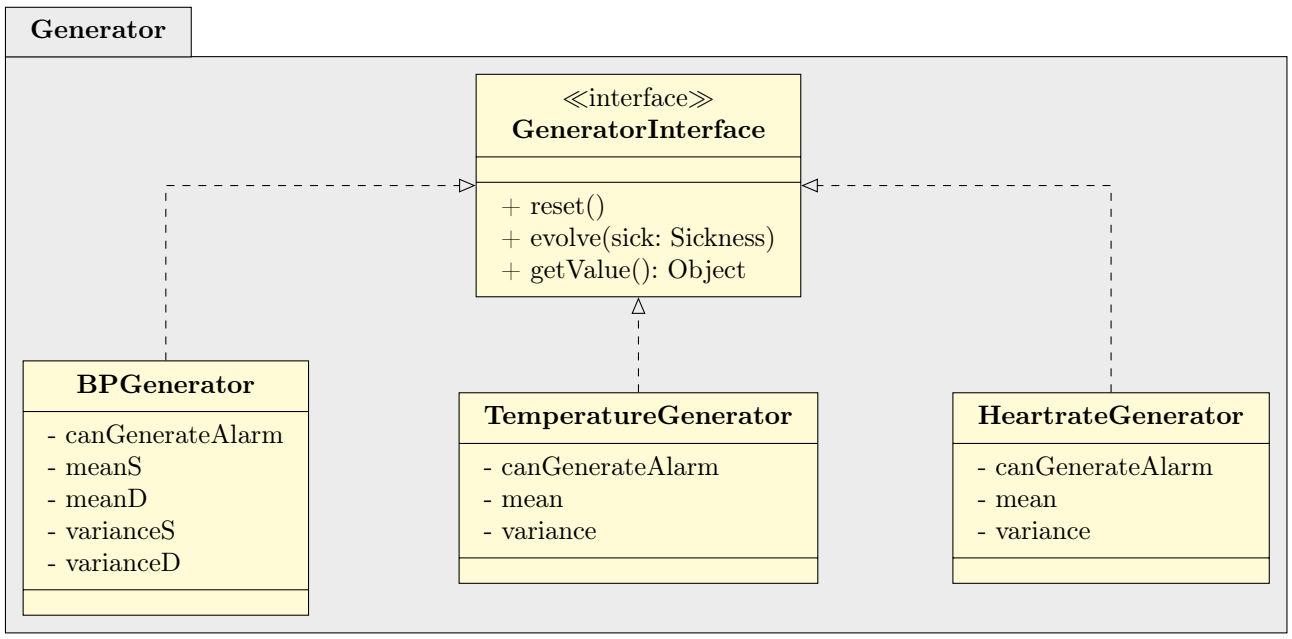

Figura 11 Diagramma di classe del *package* Generator.

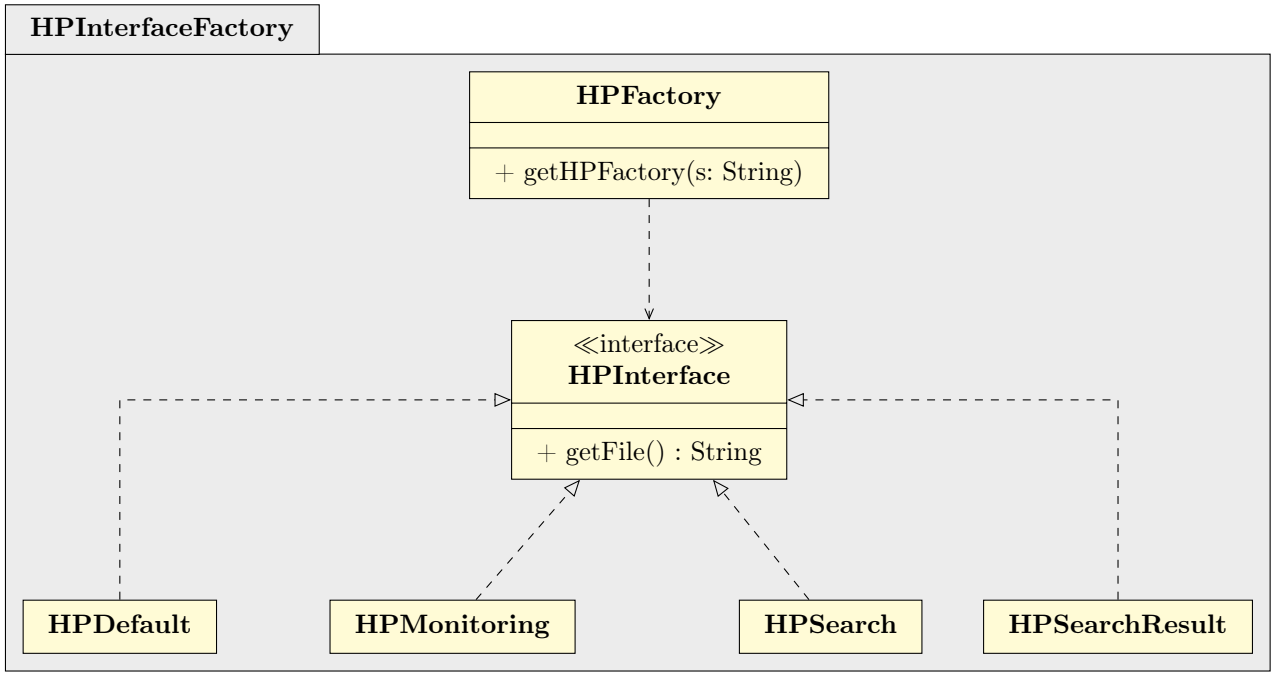

Figura 12 Diagramma di classe del *package* HPInterfaceFactory.

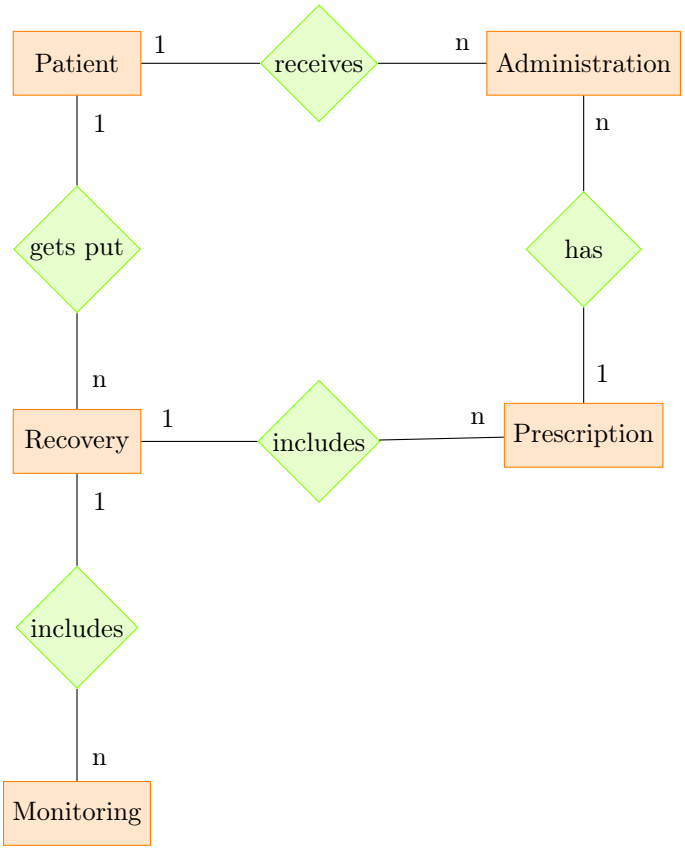

Figura 13 Schema *Entity-Relationship* dei dati del sistema.

Entità	Campi
Patient	Id, Name, Surname, Fiscal code, Place of birth, Date of birth, Patient state
Administration	Id, Date, Hour, Dose, Notes, *Patient*, *Prescription*
Prescription	Id, Drug, Duration, Daily dose, Number of doses, Doctor, *Recovery*
Recovery	Id, Start date, End date, Diagnosis, Active, Discharge summary, Recovery state, *Patient*
Monitoring	Id, Date, Diastolic pressure, Systolic pressure, Heart rate, Temperature, *Recovery*

Figura 14 Campi e relazioni delle entità del database. I campi sottolineati indicano le *primary keys*, quelli in corsivo le *foreign keys* (mappate sempre ai campi Id della rispettiva entità).

Figura 15 *Class diagram* dell'intero programma.