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.

Consulenze Nella fase progettuale è stato consultato anche un medico, la dott.sa De Carli, per avere indicazioni sui dettagli che rendono utile e in linea con le esigenze dei medici un software di

questo tipo. Tra le tante cose, ha guidato il team nella definizione dei valori dei parametri vitali da monitorare, quelli che normalmente sono nei limiti accettabili, e quelli che sono motivo di allarme. Ha poi aiutato con le informazioni che sono solitamente presenti in una lettera di dimissione, oltre che quelle richieste in alcune delle schermate grafiche del software. E ha infine ricordato che, trattandosi del reparto di terapia intensiva, la lettera di dimissione non dimette realmente un paziente: infatti gli unici due modi per lasciare tale reparto sono grazie ad un miglioramento, e quindi con un cambio di reparto, oppure per via di un decesso.

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 dimissione**
Utente:	Primario
Precondizioni:	Il primario deve essersi autenticato
Passi	– Nella propria dashboard seleziona la finestra per compilare le lettere di dimissione – 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 dati e stampa dei report settimanali**
Utente:	Primario

Precondizioni:	Il primario deve essersi autenticato
Passi	– Nella propria dashboard seleziona la finestra per visualizzare i dati delle ultime due ore – Seleziona da un menù a tendina il paziente – Visualizza nella schermata i dati sui parametri vitali delle ultime due ore e le somministrazioni degli ultimi due giorni. Eventualmente può stampare un report sul paziente cliccando su un pulsante che genera un documento pdf
Postcondizioni:	Vengono visualizzati i dati sui parametri vitali e somministrazioni. Se il primario ha premuto il bottone per stampare il report degli ultimi sette giorni, 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 relativi ai parametri vitali, aggiornati in tempo reale ed esclusivi delle ultime due ore, di tutti i pazienti ricoverati. Visualizza inoltre tutte le somministrazioni non più vecchie di due giorni amministrate al paziente.
Postcondizioni:	L'utente visualizza le informazioni, non più vecchie di due ore, sui parametri vitali dei pazienti ricoverati, oltre che informazioni sulle amministrazioni.

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 una tabella 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, le dosi giornaliere, la quantità per dose e la durata della terapia 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, cognome, sesso, 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.

In generale, scelta è stata guidata e ispirata dai sistemi web dove è presente un gestore centralizzato del software che gestisce le varie componenti e la loro comunicazione con il sistema di storage. Seguendo questa filosofia è stato quindi sviluppato SICURA, un sistema centralizzato che controlla i diversi component, i segnali generati dai loro controllori, il loro accesso con lo stato attuale del

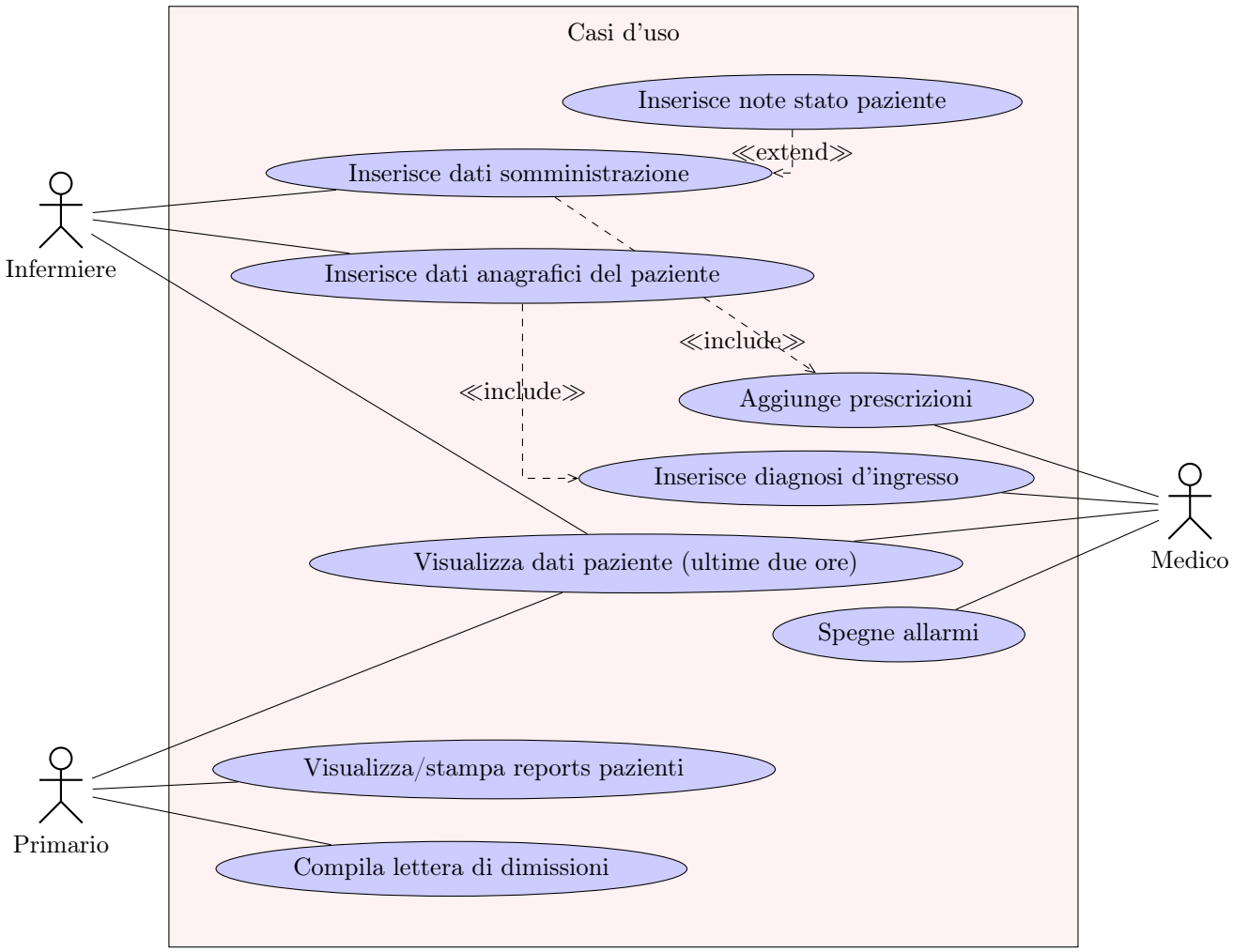

Figura 1 Diagramma UML dei casi d'uso del sistema.

sistema e gestisce la coerenza e update del database, sulla base dei dati che vengono man mano generati dall'applicativo e caricati nello stato.

Scelte di design grafico Preliminarmente si è deciso di seguire un design non troppo informale, sobrio, ma al contempo elegante ed accattivante. Per la realizzazione dell'interfaccia grafica Java FX è sembrata la scelta più comoda. Si è scelto di scrivere l'interfaccia in lingua italiana, dato che gli utenti sono italiani, e che magari in ambito medico è più importante l'immediatezza della lettura. Il software avrebbe da subito dovuto avere una parte dedicata al monitoraggio in tempo reale, comune ed accessibile a tutti, anche senza login. L'altra parte dell'applicazione, invece, sarebbe stata dedicata alle operazioni che i vari utenti avrebbero potuto fare autenticandosi. Arrivando alla progettazione nel dettaglio di ogni singola finestra, quella dedicata al monitoraggio live è stata ideata in modo tale da rendere ben visibili le varie informazioni, grazie all'ausilio di due grafici e alla rappresentazione in grande dei valori dei parametri, posti sopra la tabella dove sono salvati tutti. Inoltre, la volontà di mettere nella parte inferiore i dieci grandi tasti per ogni letto, al posto ad esempio di un menù a tendina, è stata dettata dalla necessità che hanno i medici di effettuare con rapidità certe azioni. All'interno di questa finestra, poi, è possibile accedere alla simulazione dei vari allarmi, per poter verificare il comportamento del software in tali situazioni.

Proprio in merito a tali allarmi sarà possibile notare che è stato aggiunto un segnale acustico durante l'attesa dello spegnimento dell'allarme, oltre ad un segnale luminoso, opportunamente programmato, da parte della tastiera RGB del terminale in uso: tutto questo per assicurarsi l'attenzione del personale nei momenti critici.

La parte dedicata all'utilizzo con autenticazione, invece, individua tre tipi di utenti: primario, medici e infermieri; ognuna delle corrispondenti schermate che si aprono in seguito al login, permette loro 4 azioni, di cui le prime 2 specifiche per la tipologia di utente, e le ultime 2 comuni a tutti. Tutte queste schermate sono ovviamente di pari dimensione. Nel caso del primario, la schermata di visione delle ultime 2 ore di parametri vitali, è presente il tasto con la seconda delle funzioni specifiche del primario: la stampa di un report riassuntivo degli ultimi 7 giorni, che genera un file PDF con tali informazioni, e ne permette, appunto, la stampa. Un altro PDF viene generato e visualizzato quando il primario effettua una dimissione, contenente la relativa lettera. Infine, dentro la sezione di aggiunta di un nuovo paziente da parte degli infermieri, è possibile notare un pulsante in fianco al campo del codice fiscale, che ne permette, così, oltre che l'inserimento manuale, anche il calcolo automatico. Durante tutto il processo un'altra priorità è stata quella di disporre in modo omogeneo sulle schermate i vari componenti, occupando al meglio lo spazio a disposizione, senza sbilanciare e rendere 'pesanti' certe parti dello schermo, tutto ciò per rendere piacevole l'utilizzo dell'applicazione, oltre che chiaro e rapido.

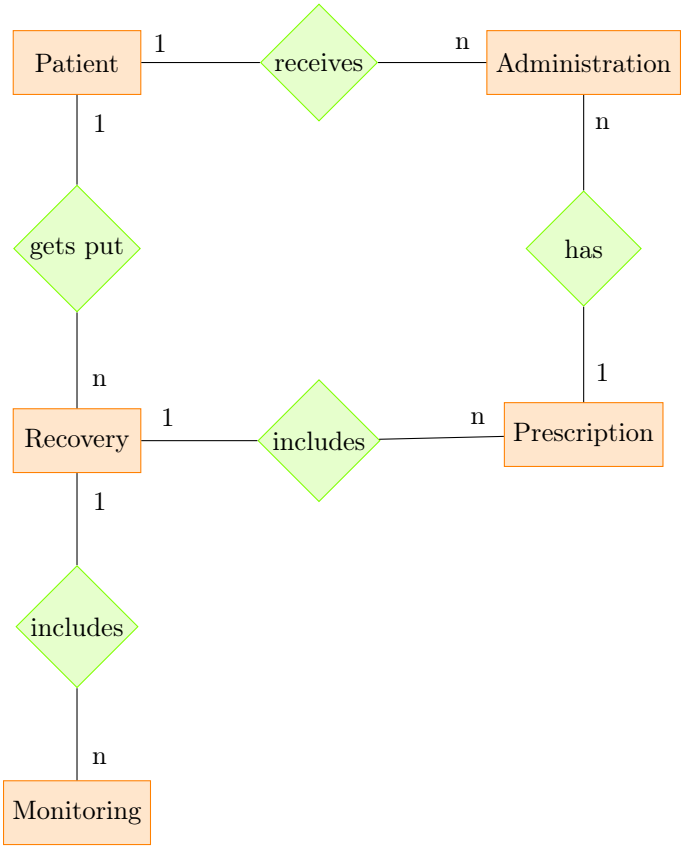

Figura 2 Schema *Entity-Relationship* dei dati del sistema.

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 3 Schema logico, rappresentanti 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).

Design dello schema dati Quando si è passati all'analisi della gestione dei dati presenti nel sistema, come già citato, è subito emerso come i diversi dati fossero strettamente legati tra loro tramite relazioni, che creavano così il 'pacchetto' di informazioni del singolo paziente registrato e gestito dal sistema. Finita questa fase si è passati alla creazione del Diagramma E/R (Entity Relationship), stilando così un primo schema che rappresentava come avrebbe dovuto essere costruito il sistema di storage sulla base delle considerazioni effettuate precedentemente. Una volta corrette alcune imprecisioni, e confermato la bozza, si è passati alla creazione del diagramma E/R ristrutturato, riportato nella figura 2.

Come si può notare Paziente è collegato tramite una relazione 1-N sia con Somministrazioni, che con Ricovero. A primo impatto si nota subito come la prima non sia necessaria in quanto, tramite le diverse relazioni, si riesce a collegare perfettamente il paziente con le sue somministrazioni (Paziente → Ricovero → Prescrizioni → Somministrazioni). Si è però deciso di aggiungere questa relazione tra le due entità in quanto spesso, all'interno del software, viene richiesta l'informazione di un determinato Paziente e le sue Somministrazioni, senza che sia necessario avere informazioni relative al Ricovero e/o Prescrizioni. Il team ha deciso di evitare tre JOIN concatenati, fornendo più semplicemente un rifermento al Paziente all'interno della Somministrazione, in modo tale da ottenere le informazioni necessarie con il minimo uso di risorse. Successivamente è stato creato lo schema Logico sulla base dello schema E/R, rispettando le regole di transizione tra i due schemi. Una volta controllate le forme normali dello schema, si è proseguito con l'implementazione del Database in modo tale che le diverse tabelle rispettassero quanto riportato all'interno della tabella 3.

Generazione di dati e allarmi Per quanto riguarda la generazione dei dati relativi al monitoraggio dei diversi pazienti ricoverati, è stato scelto di creare dei processi attivi in background dedicati a questo compito. Nel software sono presenti tre diversi generatori, uno per ogni gruppo di dati (battito cardiaco, pressione sanguigna, temperatura). Per l'implementazione si è optato di utilizzare delle distribuzioni Gaussiane, dove la media e varianza è stata calibrata rispetto ai dati segnalati dalla Dott.ssa De Carli durante l'incontro.

I generatori, a regime normale, generano dati all'interno di intervalli accettabili per i segnali vitali di un paziente ricoverato in terapia intensiva. Una volta che viene richiesta la generazione di un allarme i generatori evolvono cambiando media e varianza della distribuzione, in questo modo verranno generati dati che non rispettano gli standard di valori accettabili. I valori rimangono fuori dalla norma fintanto che l'allarme è attivo, una volta disattivato o scaduto viene traslata la curva gaussiana entro i range di dati accettabili.

Design dei componenti e controllori Il software ha una classe *singleton* Sistema, la quale rappresenta il core del programma, che gestisce le interfacce del sistema, lo stato centralizzato e fa da ponte con il database. Durante il bootstrapping del software viene creata l'istanza della classe Sistema; in questo passaggio la classe va a settare lo stato iniziale dei diversi elementi che ne sono alla base e permettono l'esecuzione del programma. Nello specifico quando si va a creare l'unica istanza vengono eseguiti diversi passaggi rispettando l'ordine specificato nel sequence diagram in figura 4.

In un primo momento viene creata un'istanza di Store, al quale vengono associati gli handler ai diversi comandi di update dello stato, e una di InterfacesController, al quale verrà dato il compito di gestire l'evoluzione delle interfacce durante la vita del software. Una volta finita questa fase viene inviato il comando di LOAD, quest'ultimo viene gestito dallo Store caricando da database tutti i dati in esso contenuti e inizializzando cosi lo State iniziale del software. Ultimo, non per importanza, viene popolata la pool di interfacce gestite da InterfacesController, associando una stringa a una determinata interfaccia. Tramite questo controllore il sistema può decidere a seconda della sua evoluzione quale interfaccia caricare semplicemente inviando un messaggio di update. Le interfacce, come anche i controllori, per quanto riguarda i tre diversi utenti all'interno del sistema,

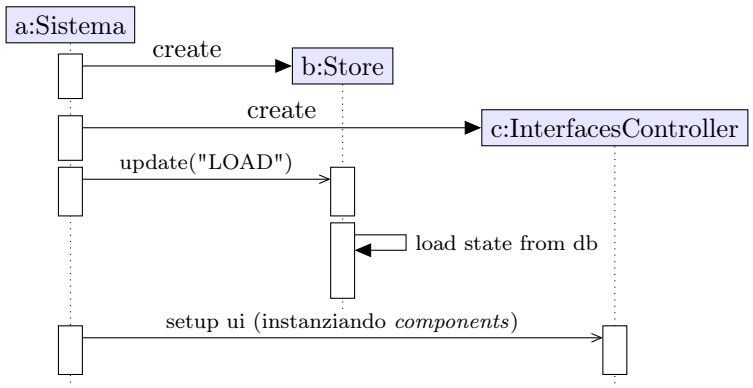

Figura 4 *Sequence diagram* delle operazioni eseguite durante il bootstrapping del programma.

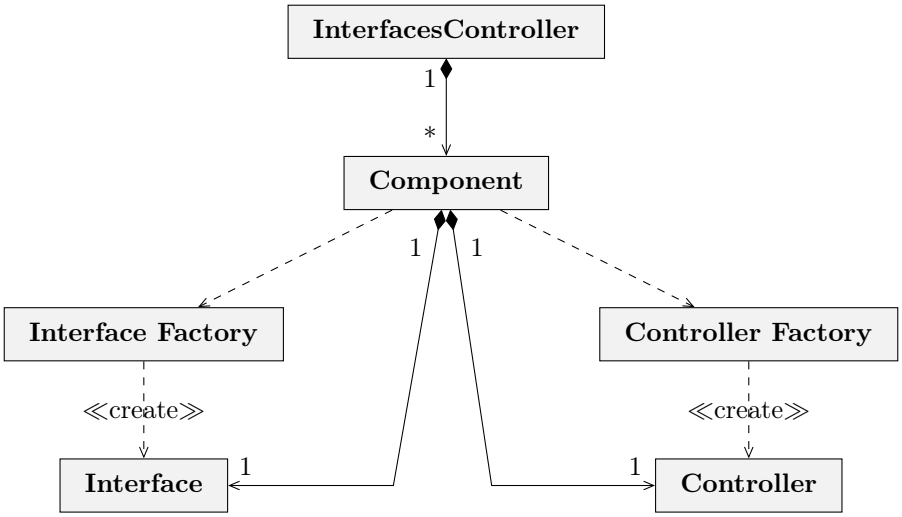

Figura 5 *Class diagram* che mostra la relazione tra InterfaceController e i vari oggetti che costituiscono la parte View-Controller dell'architettura *MVC* dell'applicazione.

vengono istanziate e ottenute tramite le sei factory presenti all'interno del sistema. Infatti nel software sono presenti diverse factory, tre dedicate al caricamento dei component e altrettante dedicate al caricamento dei controllori delle interfacce, seguendo la struttura indicata in figura 5 (semplificato aggregando le diverse factory). Terminato questo processo il Sistema è stato correttamente istanziato e portato allo stato iniziale dal quale potrà poi evolvere a seconda delle richieste da parte dell'utente.

Le istanze di component e controller interagiscono tra loro a seconda degli eventi generati da azioni da parte dello user (figura 6). Nello specifico, i vari componenti dell'interfaccia sono mappati tramite degli id dal controllore; quando lo user interagisce con l'interfaccia attiva un determinato componente che invoca il suo EventHandler. Quest'ultimo una volta aggiorna internamente l'interfaccia evolvendo eventuali campi e informazioni, e successivamente se necessario invia un comando di update allo Store che gli era stato iniettato durante la fase di creazione del component. Il controller dell'interfaccia rimane poi in ascolto di eventuali messaggi di update dello State e/o utilizzi degli elementi presenti nell'interfaccia, che nel caso in cui vanno a variare i dati correntemente attivi e salvati, aggiorna di conseguenza l'interfaccia mostrando all'utente, tramite il pattern Observer, un update in tempo reale.

Stato Uno dei punti di forza del design di tale architettura è la gestione centralizzata dello stato che viene mantenuto all'interno del sistema generale il quale mostra un interfaccia unica per il suo accesso e la sua modifica. In particolare, ogni sua manipolazione può avvenire esclusivamente a

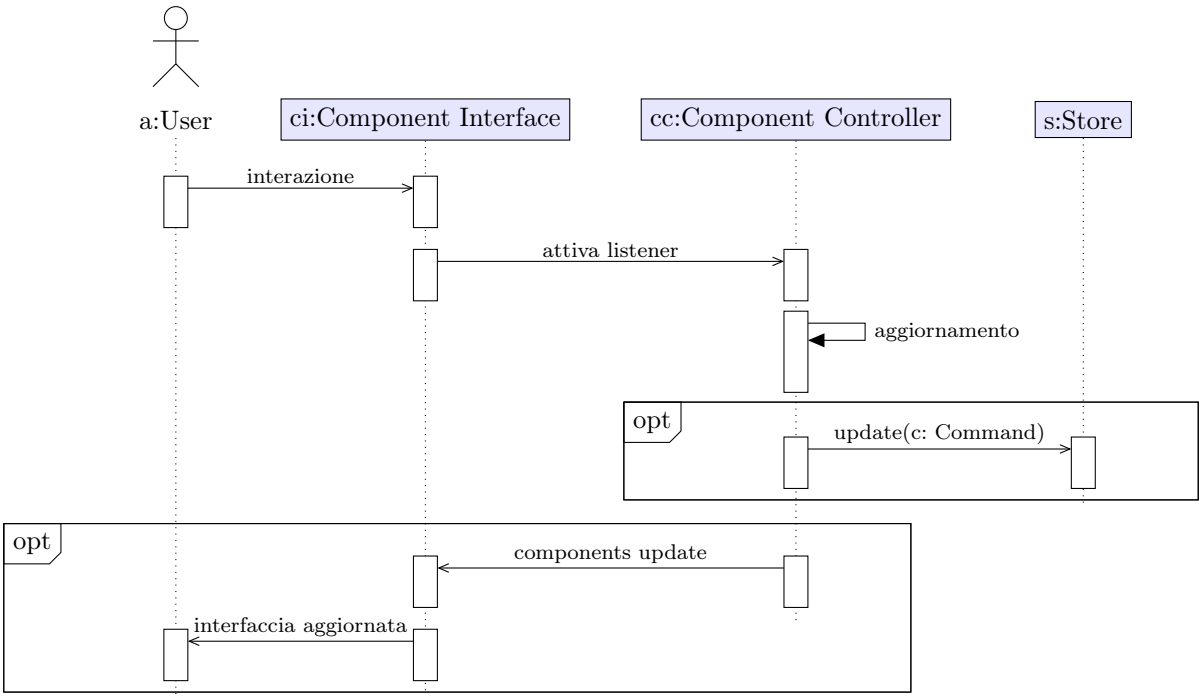

Figura 6 *Sequence diagram* che mostra le operazioni che avvengono a seguito di una generica interazione di un utente con il software. Premendo qualche pulsante, inserendo dati, cliccando tasti della tastiera un utente interagisce con l'interfaccia a cui è associato un *controller*: a seconda dell'azione 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.

seguito dell'invio di un comando al gestore, precedentemente inizializzato, specificando l'insieme di tutti i comandi possibili e le diverse funzioni a loro associate. In questo modo è necessario specificare, durante il design dell'architettura, quali sono le varie modifiche che potrebbero essere eseguite dai diversi componenti del sistema e l'interfaccia unica e comune — i comandi — che deve essere usata a tal fine. In questo modo si evita il problema di modifiche ´nascoste' dello stato che porta spesso ad errori quando diviene difficile determinare da chi, e dove, sono state eseguite modifiche.

Altro punto di forza del design è la propagazione asincrona delle modifiche a tutti i *subscribers*, secondo il pattern *observer*: una volta instanziato lo Store da parte del Sistema i vari controllori si sottoscrivono ad esso, passandogli una funzione, che verrà richiamata asincronicamente ad ogni cambiamento di stato: questo permette, oltre ad una semplificazione della struttura dell'architettura, non dipendente da chiamate bloccanti a metodi specifici dello Store, di creare interfacce reattive che si aggiornano in tempo reale. Questa astrazione inoltre riduce la complessità associata alla gestione dei dati del sistema che altrimenti dovrebbero essere propagati in diversi stadi ai vari componenti: questo approccio è molto fragile e prono ad errori dato che spesso porta a diversi livelli di aggiornamento dei dati consegnati ai vari componenti che quindi si trovano in uno stato inconsistente; utilizzando il sistema centralizzato con interfaccia a priori determinata, si evita questo problema. Si veda figura 7.

1.4 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.

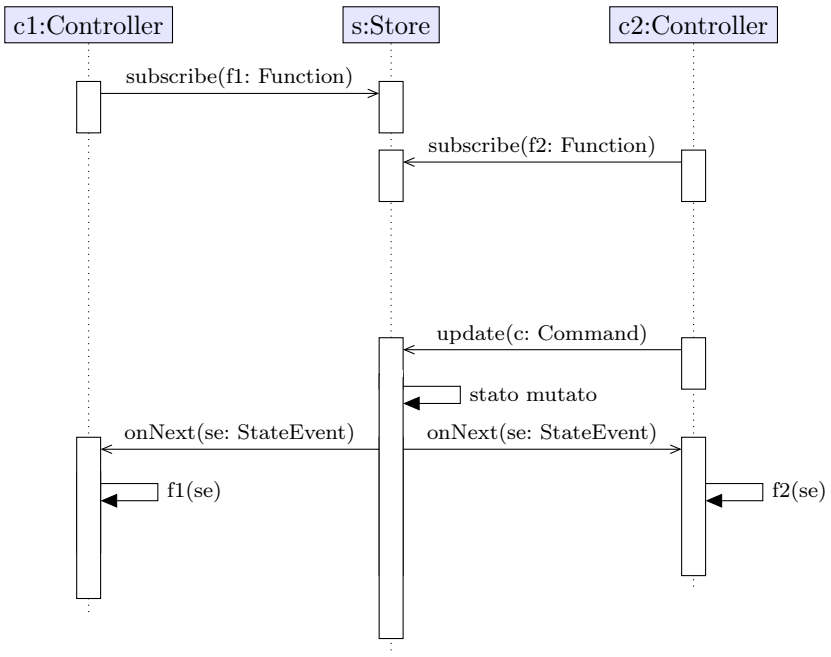

Figura 7 *Sequence diagram* dell'interazione di due generici controllers con lo stato centralizzato gestito dallo Store. Dopo aver invocato in modo non bloccante *subscribe()*, passandogli una funzione, qualsiasi *update()* chiamato sullo store comporta una mutazione interna dello stato, sulla base del 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.

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 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.

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 3) 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.

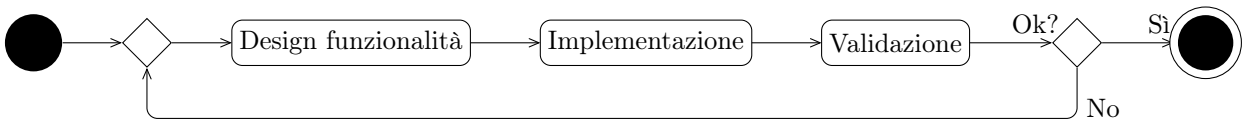

Figura 8 *Activity diagram* della fase di validazione per le funzionalità dei componenti utente.

- 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.).

1.5 Evoluzione

Per quanto riguarda la struttura del codice essa è stata pensata affinché non precludesse l'aggiunta di ulteriori funzionalità, non indicate nella specifica: è cioè stata pensata affinché tali aggiunte non stravolgeresso tutto il lavoro fatto in precedenza, già strutturato in modo modulare ed estendibile. In particolare, il design basato su *MVC* e stato centralizzato ben si presta all'aggiunta di componenti facilmente collegabili con la logica generale. Infatti, sia per quanto riguarda un potenziale completamento del software, per portarlo da demo ad effettiva versione definitiva completa in ogni dettaglio, sia per aggiunte extra, ampliando con nuovi moduli il software per ulteriori specifiche fornite in un secondo momento, l'architettura permette l'integrazione delle componenti con estrema facilità.

Esempi di interventi che dovremo effettuare al fine di rendere operativo il software sul campo, sono il completamento dell'algoritmo del calcolo del codice fiscale, che per ora non tiene conto del luogo di nascita, oppure la stabilizzazione della visualizzazione dei grafici di monitoraggio, o ancora, il perfezionamento della ricerca delle cartelle cliniche. L'organizzazione, quindi, distingue nettamente le parti che si occupano del database, quelle del sistema di controllo, e le componenti prettamente grafiche, tramite opportuna suddivisione in sorgenti e package. Si individua, così, in modo rapido il luogo di intervento per eventuali modifiche o risoluzioni di bug.

2 Specifica

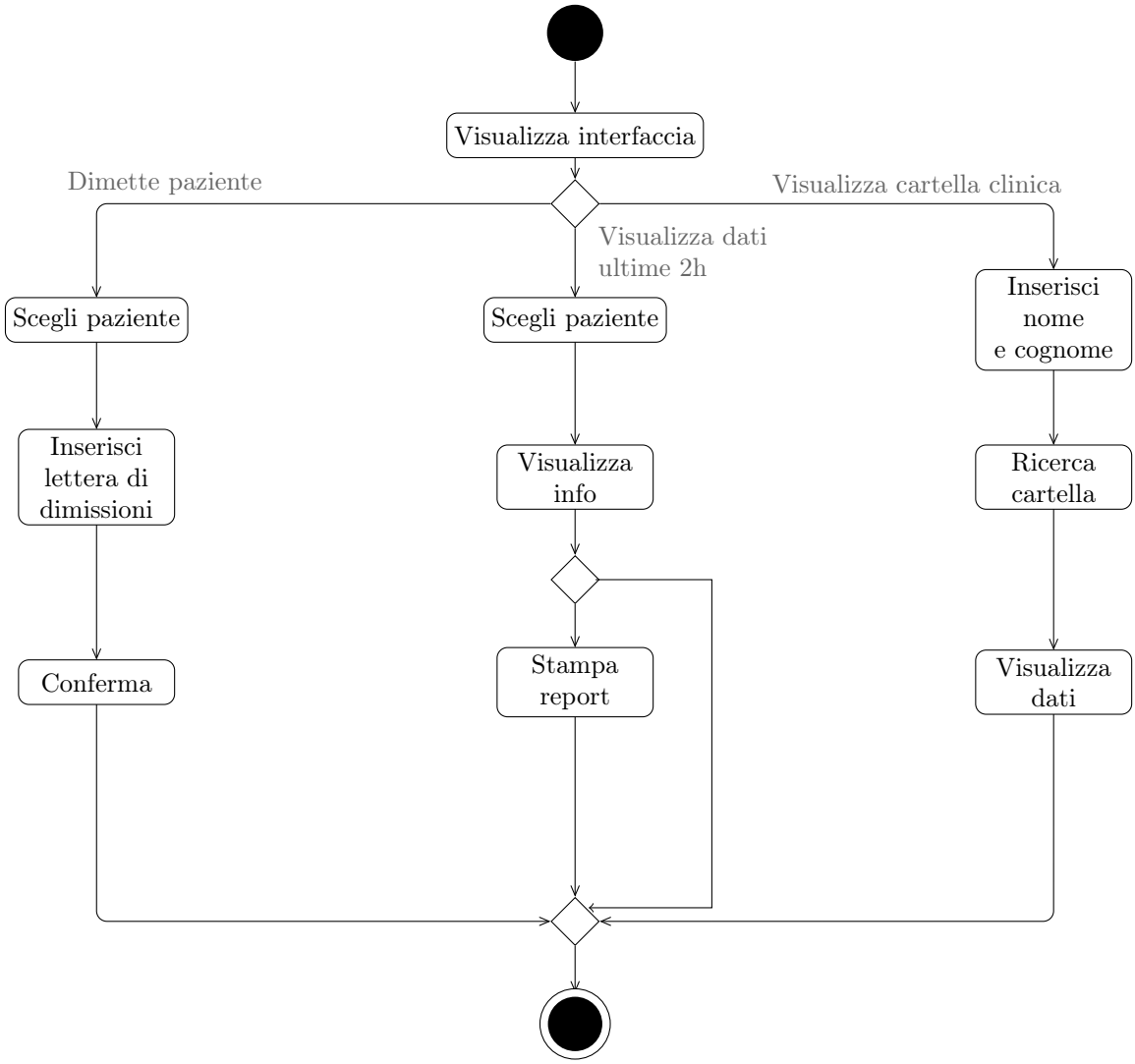

Figura 9 *Activity diagram* delle diverse azioni che possono essere compiute dal primario.

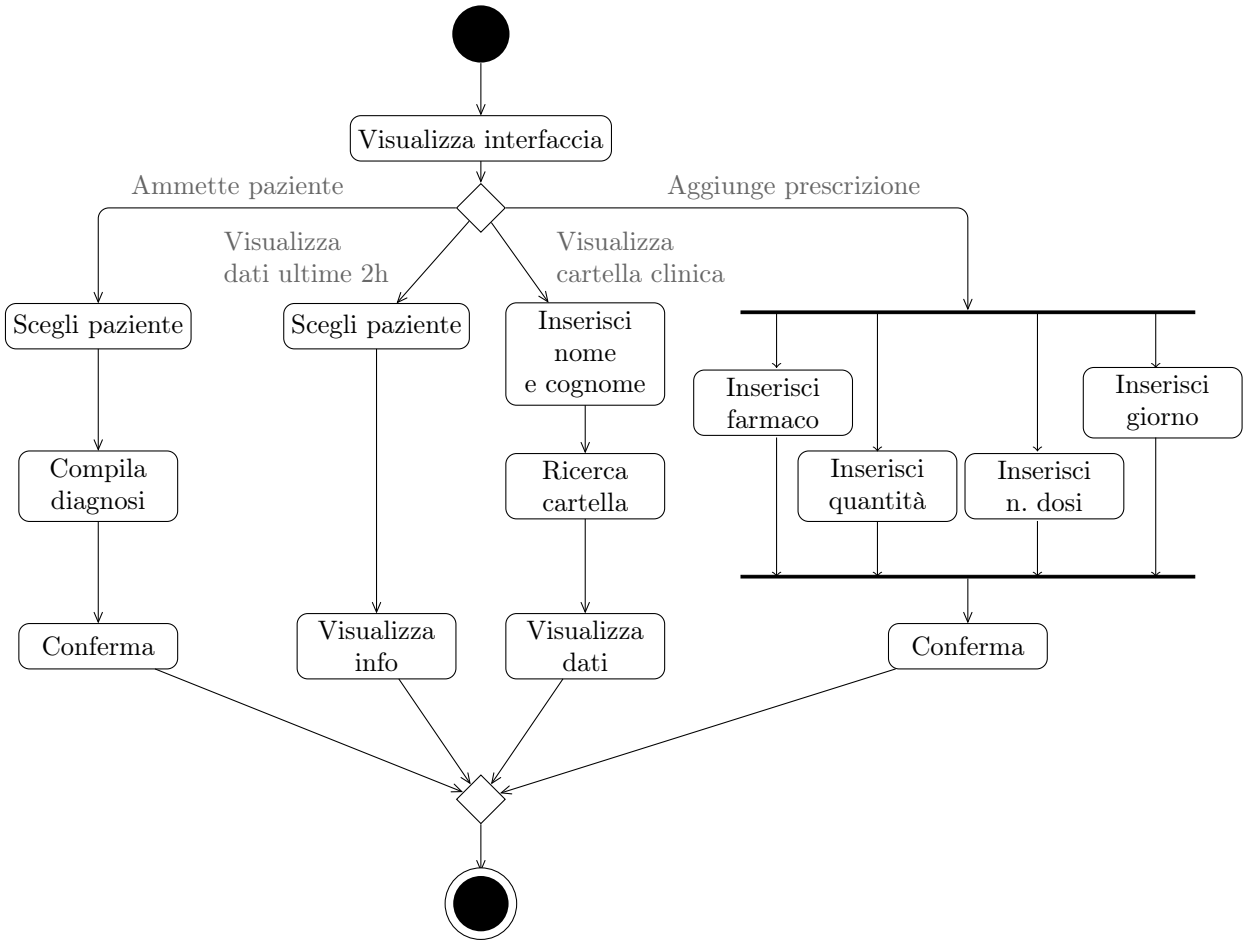

Figura 10 *Activity diagram* delle diverse azioni che possono essere compiute dal medico.

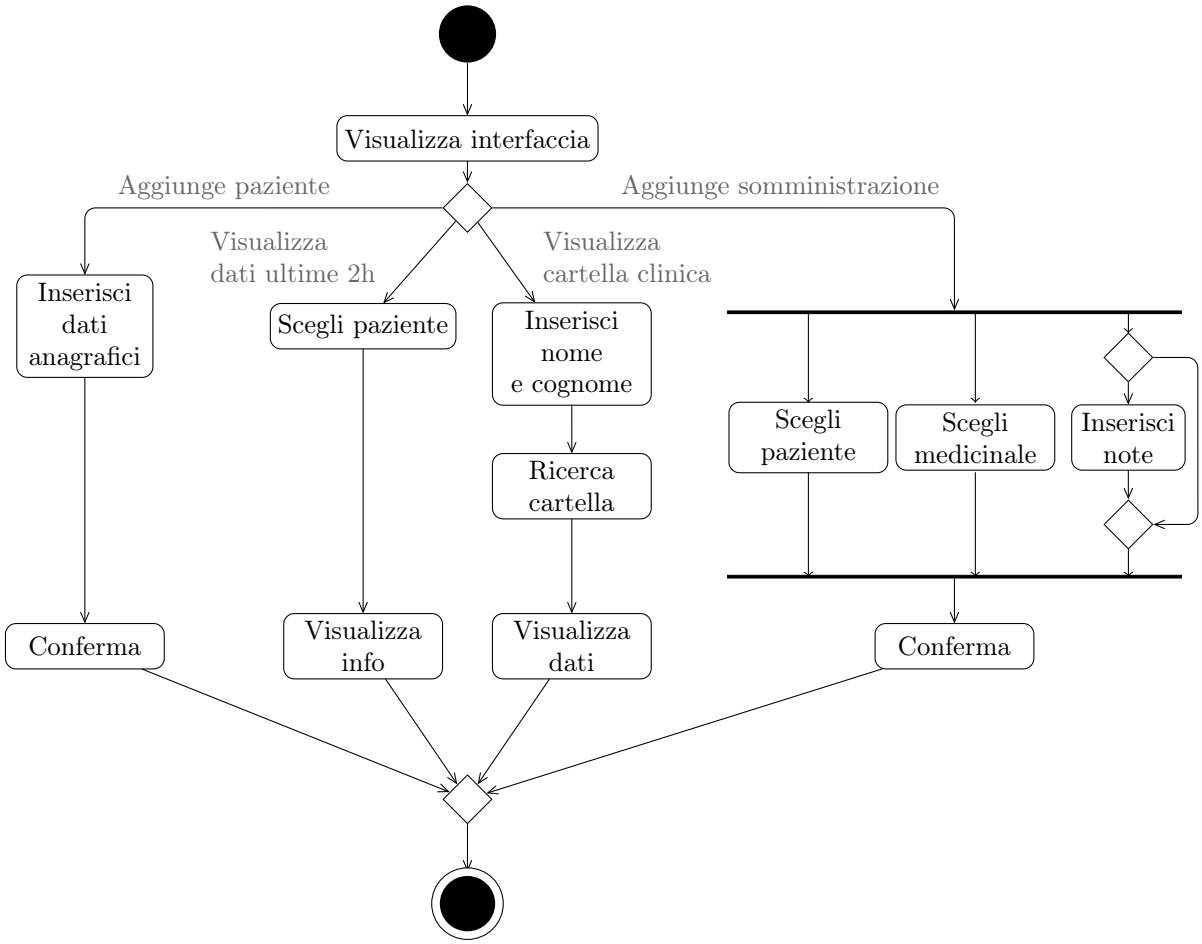

Figura 11 *Activity diagram* delle diverse azioni che possono essere compiute dall'infermiere.

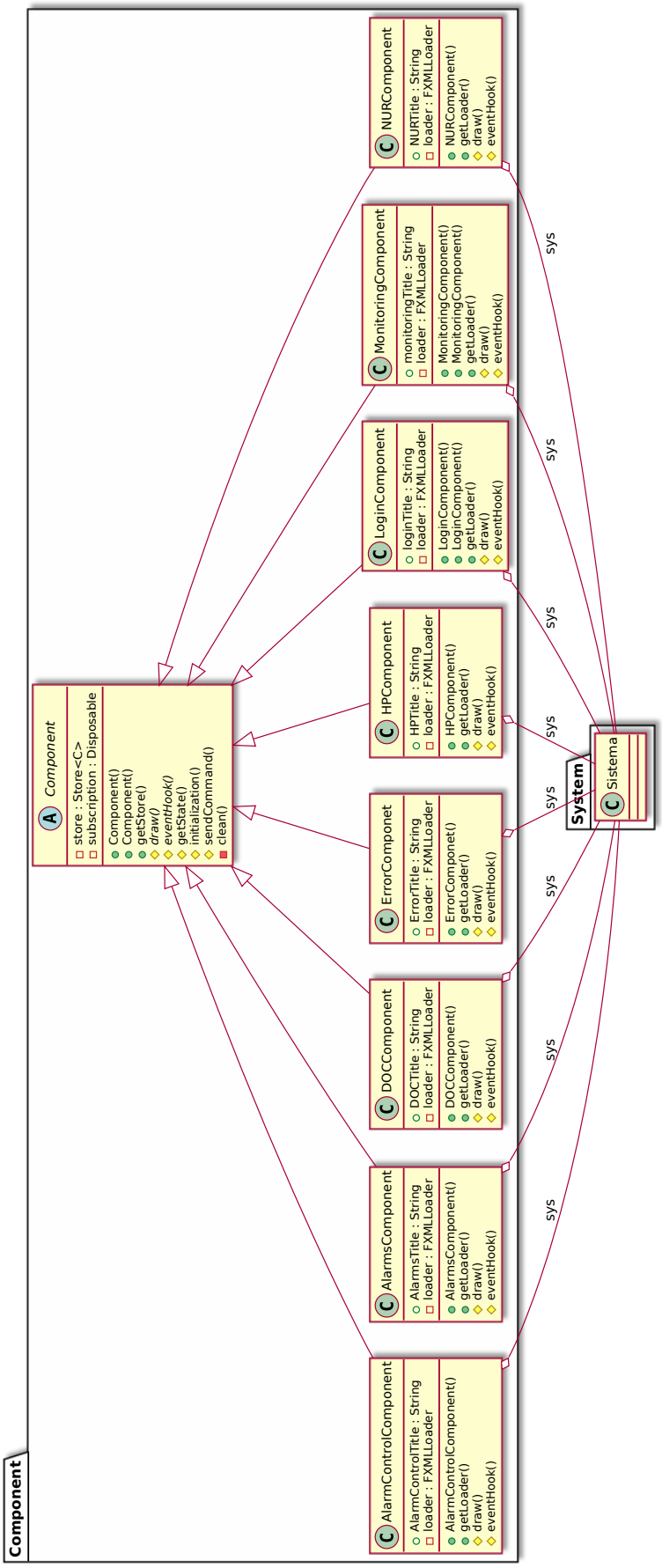

Figura 12 *Class diagram* di Component.

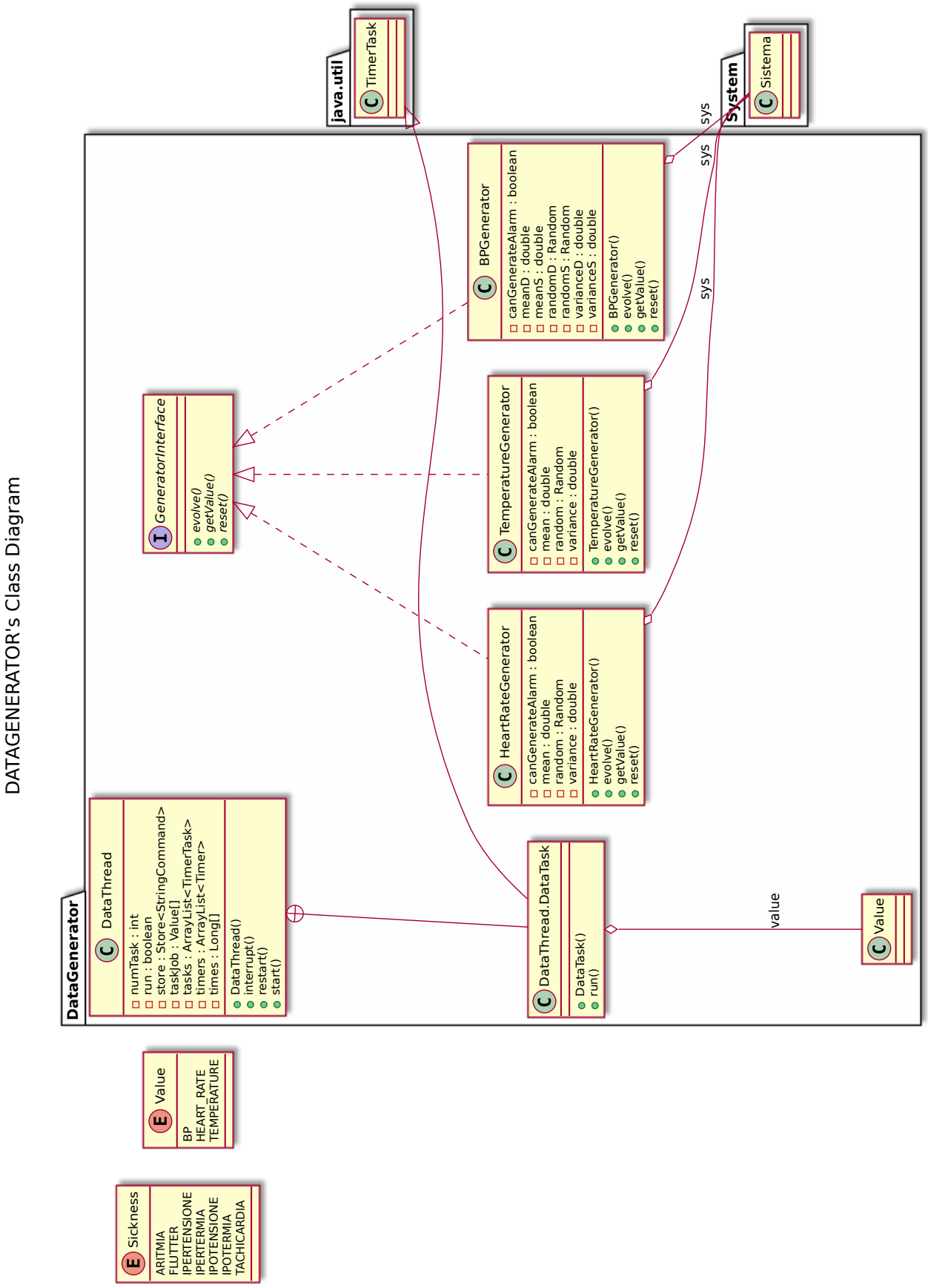

Figura 13 *Class diagram* di DataGenerator.

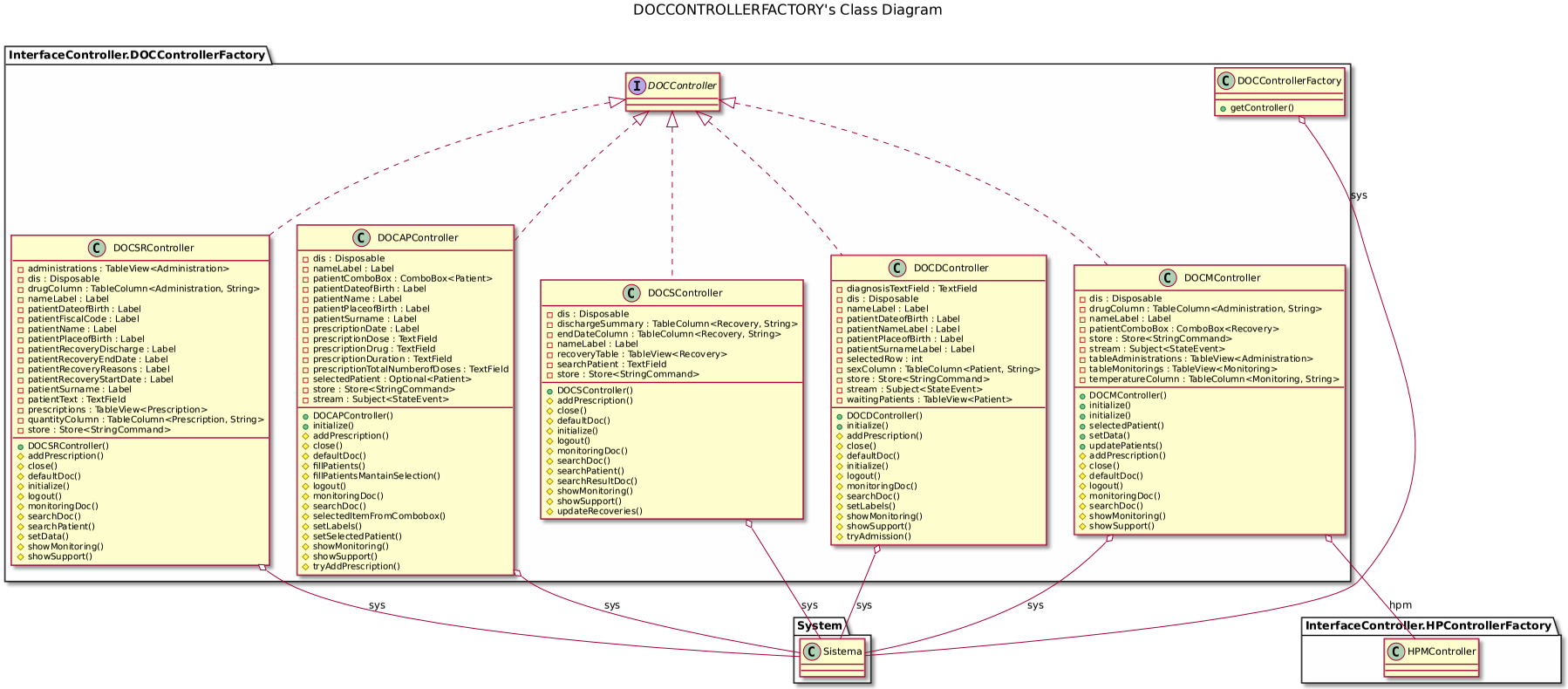

Figura 14 *Class diagram* di DOCController.

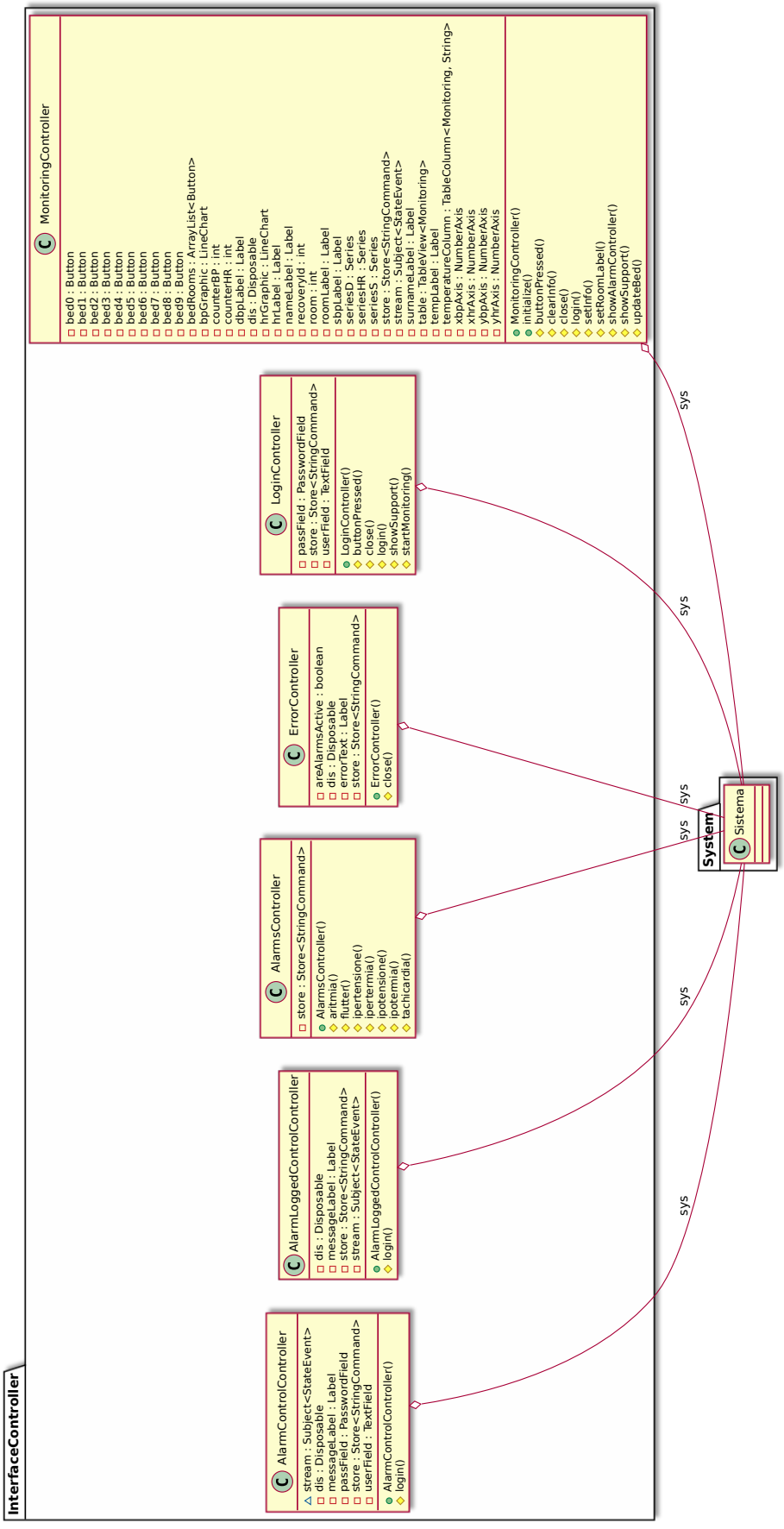

Figura 15 *Class diagram* di InterfacesController.

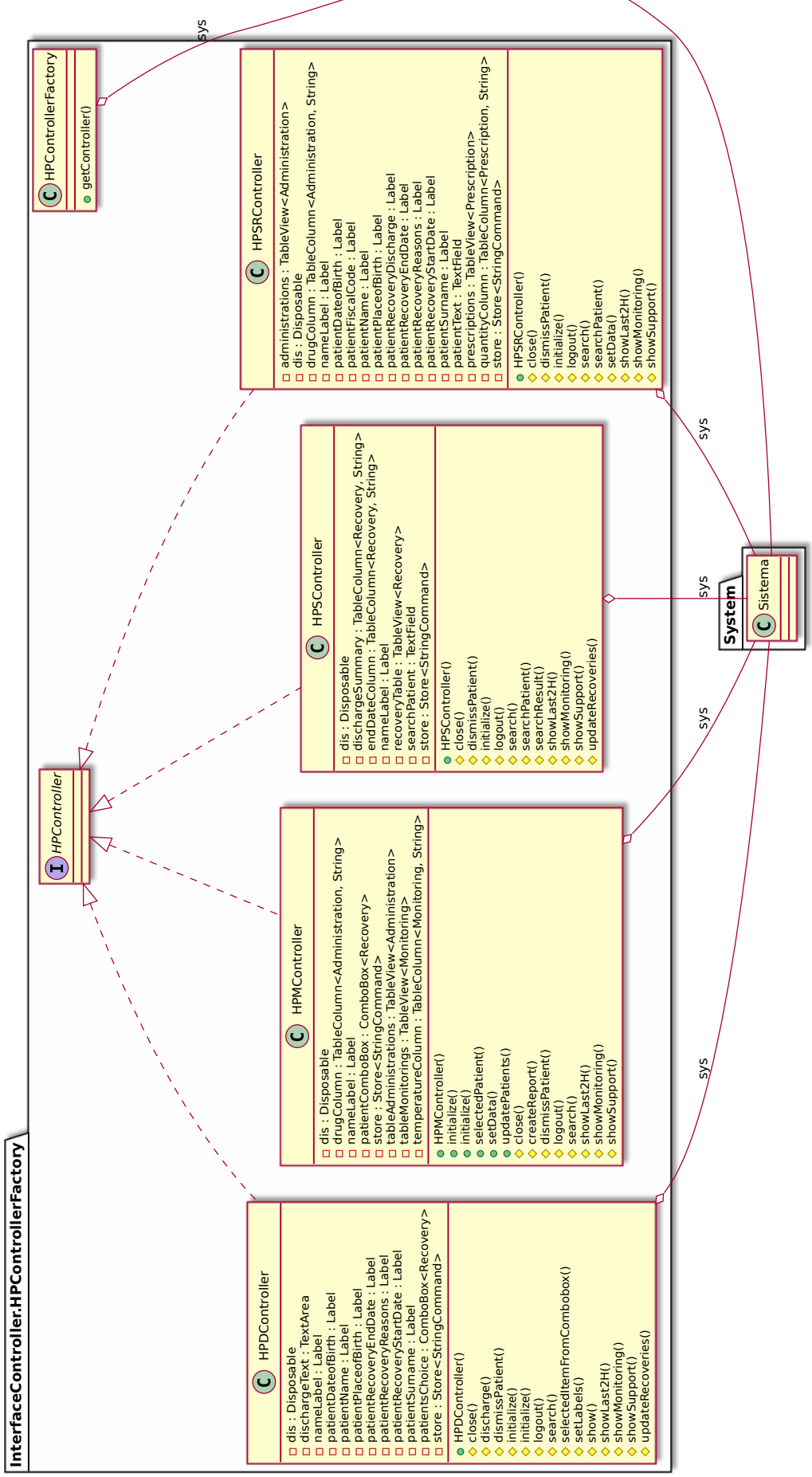

Figura 16 *Class diagram* di HPController.

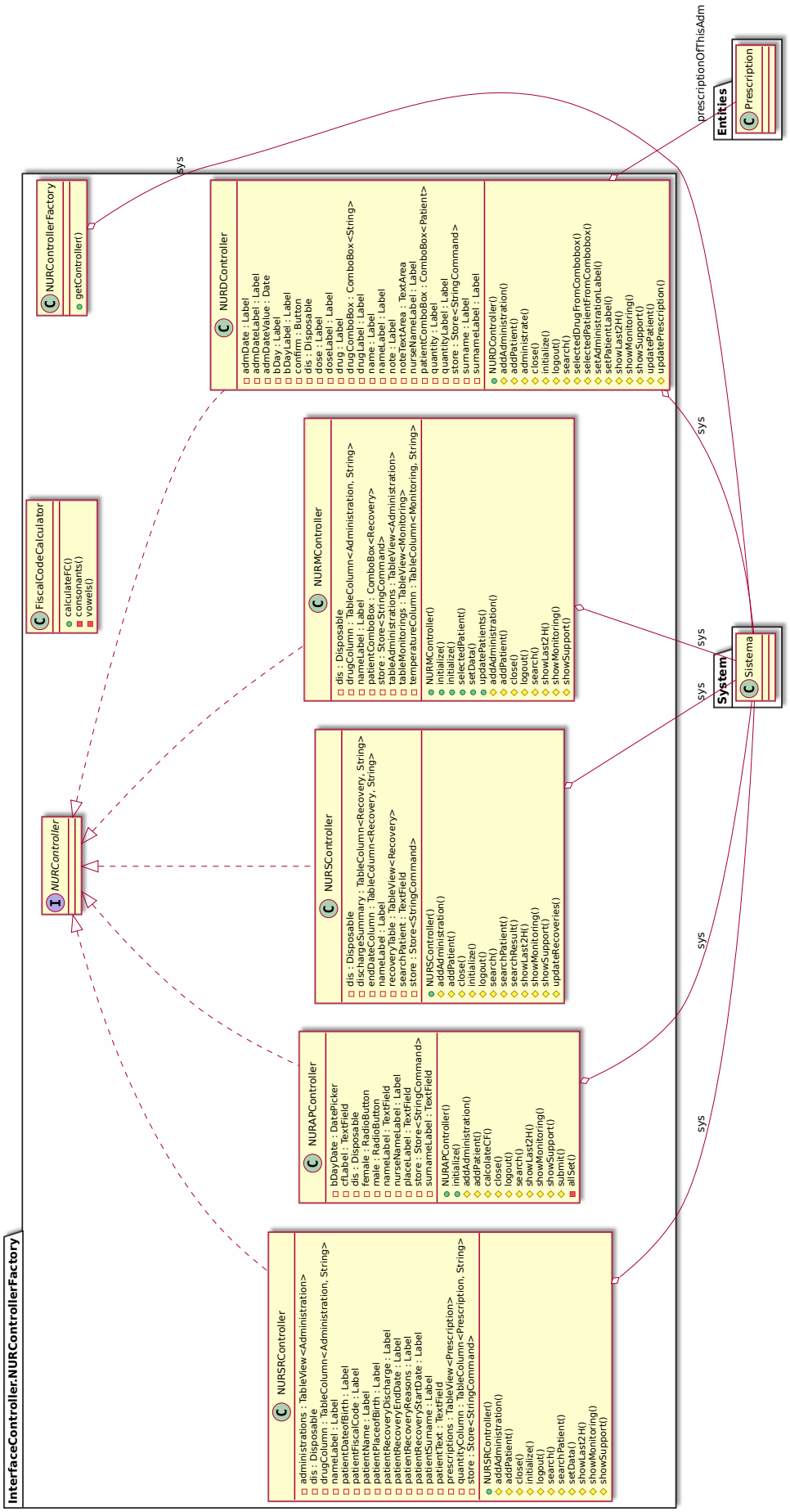

Figura 17 *Class diagram* di NurseController.

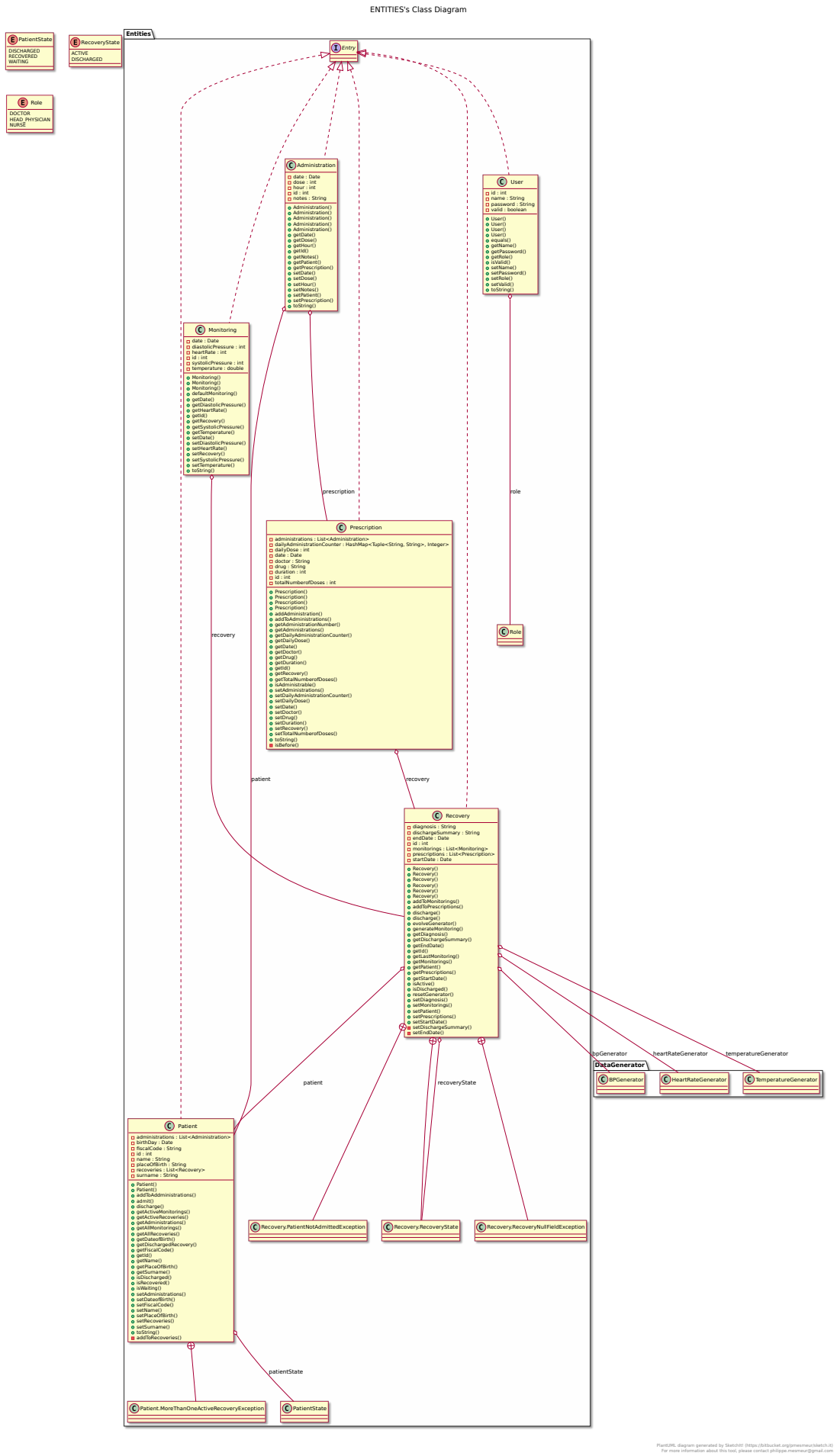

Figura 18 *Class diagram* di Entities.

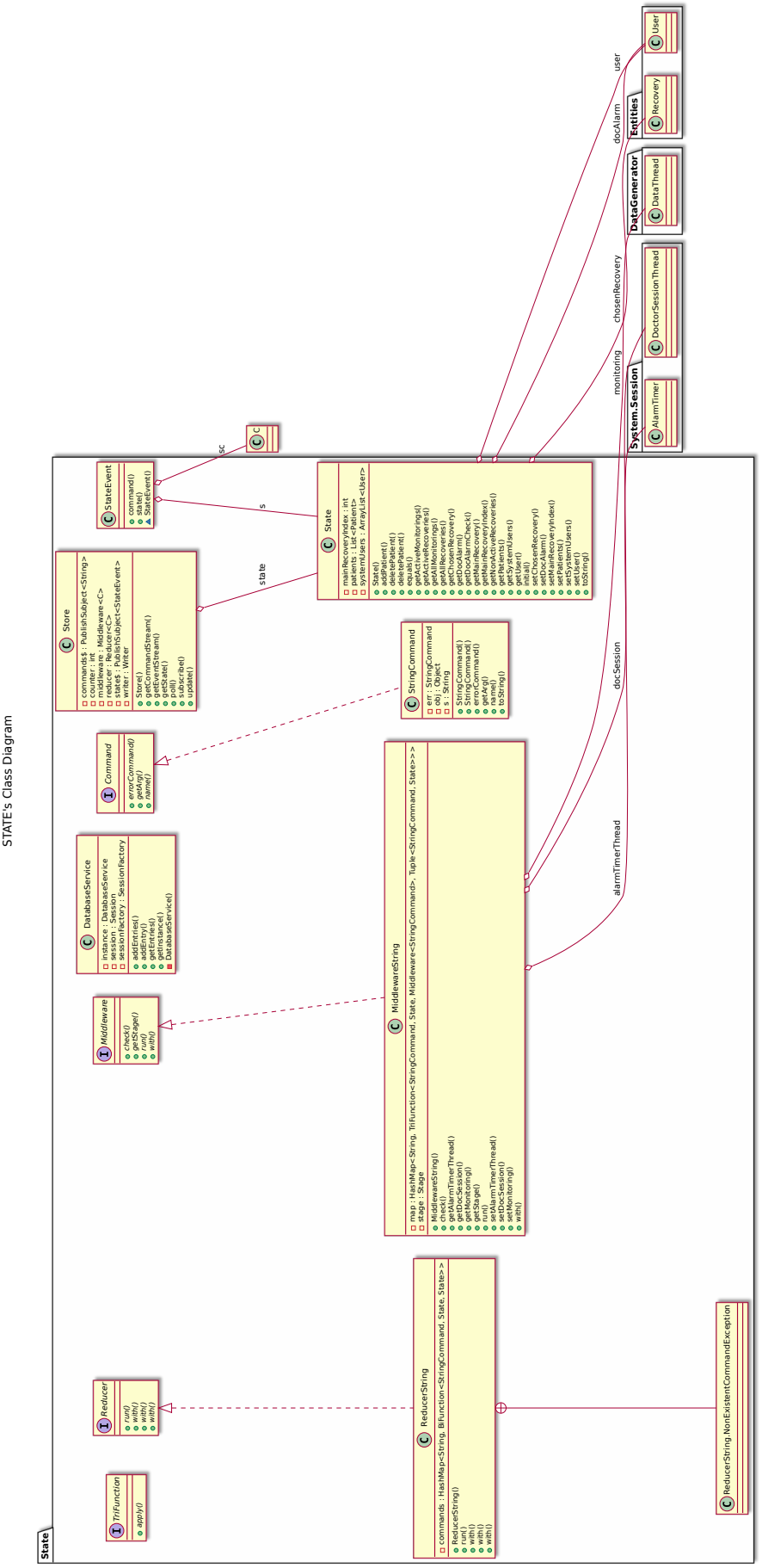

Figura 19 *Class diagram* di State.

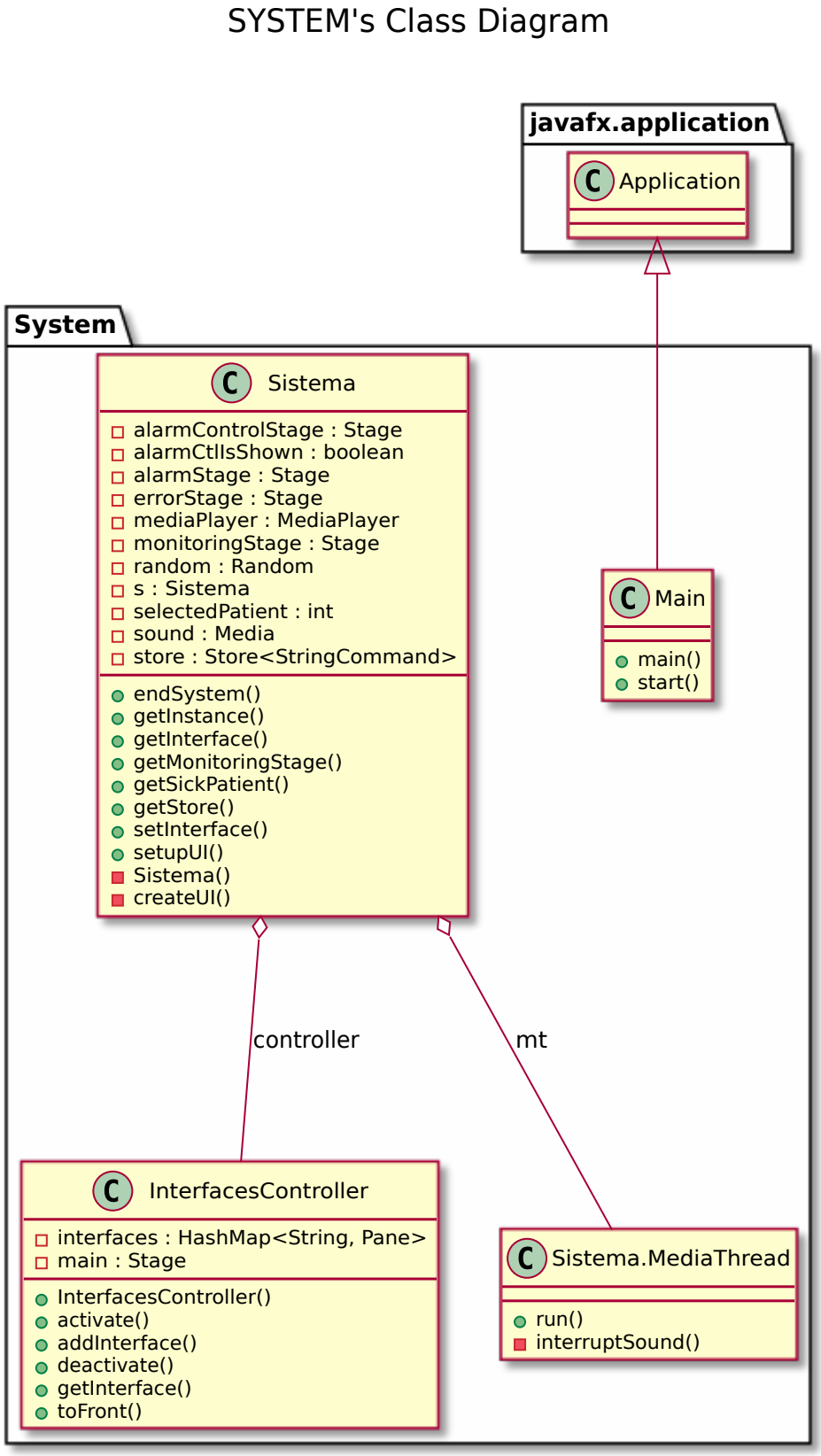

Figura 20 *Class diagram* di System.

Figura 21 *Class diagram* dell'intero programma.