Ingegneria del Software
Software per il monitoraggio di un reparto di terapia intensiva

Matteo Meneghetti VR401958
Stefano Cattonar VR402549

Luglio 2019

Indice

Introduzione

L'esame del corso di Ingegneria del Software richiedeva lo sviluppo di un software applicativo da parte di gruppi di studenti, con la possibilità di scegliere tra tre possibili progetti.
Noi abbiamo deciso di sviluppare il progetto riguardante il monitoraggio di un reparto di terapia intensiva di un ospedale.

Capitolo 1

Requisiti

1.1 Casi d'uso

1.1.1 Attori e casi d'uso

Come esplicitato dai requisiti gli attori del sistema sono i seguenti:

- **Gli Utenti non autenticati**
  Che possono:
  - Visualizzare lo stato dei pazienti
  - Autenticarsi
- **Il personale medico:**
  Che può:
  - Scrivere prescrizioni
  - Fermare allarmi
  - Inserire la diagnosi di ingresso di un paziente
- **Il personale infermieristico**
  Che può:
  - Inserire i nuovi ricoverati
  - Somministrare farmaci
- **Il Primario:**
  Che può:
  - Fare tutto quello che può fare un medico
  - Visualizzare il report settimanale
  - Dimettere un paziente

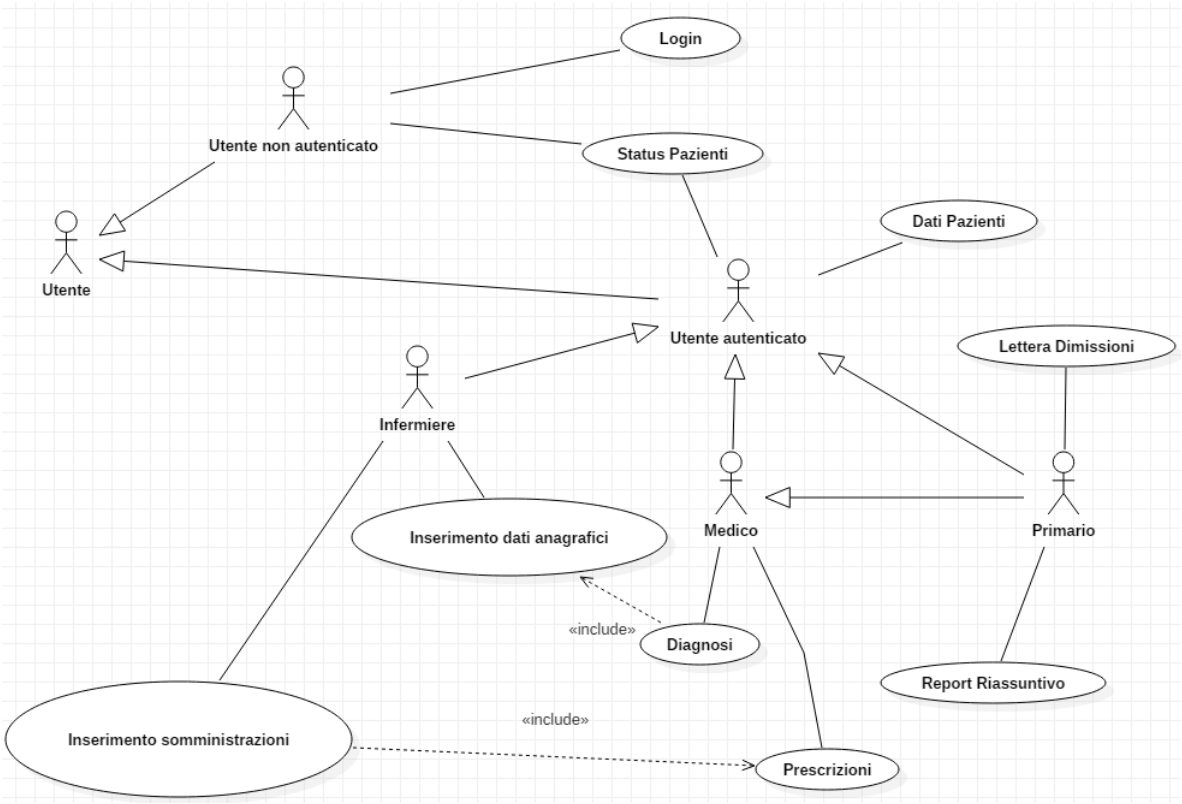


Figura 1.1: Diagramma dei casi d'uso

1.1.2 Use case speciale: Visualizzare lo status dei pazienti

Questo caso d'uso è particolare poiché è accessibile a chiunque e non ci sono azioni da eseguire.

Attori	Tutti
Precondizione	Nessuna
Sequenza	Nessuna
Post-condizione	Nessuna

1.1.3 Casi d'uso degli utenti non autenticati

Login

Il login permette ad un utente non autenticato di autenticarsi:

Attori	Utenti non autenticati
Precondizione	Nessuna
Sequenza	1)L'utente preme sul pulsante "Login" 2)L'utente inserisce username e password 3)L'utente preme sul tasto "Conferma"
Post-condizione	Utente autenticato

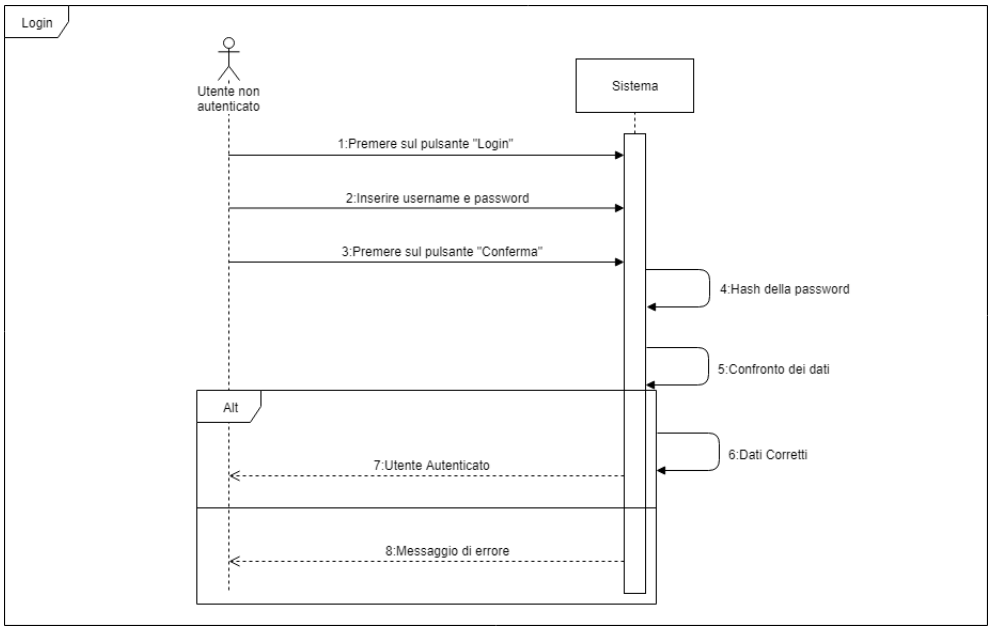


Figura 1.2: Diagramma dei casi d'uso

1.1.4 Casi d'uso del personale infermieristico

Inserimento dati anagrafici

Il personale infermieristico deve poter compilare i dati anagrafici dei pazienti al momento del ricovero:

Attori	Personale infermieristico
Precondizione	L'utente deve essere autenticato
Sequenza	1)L'utente preme sul pulsante "Nuovo Paziente" 2)L'utente inserisce i dati anagrafici del paziente 3)L'utente preme sul tasto "Conferma"
Post-condizione	Aggiunta del paziente e inizio monitoraggio

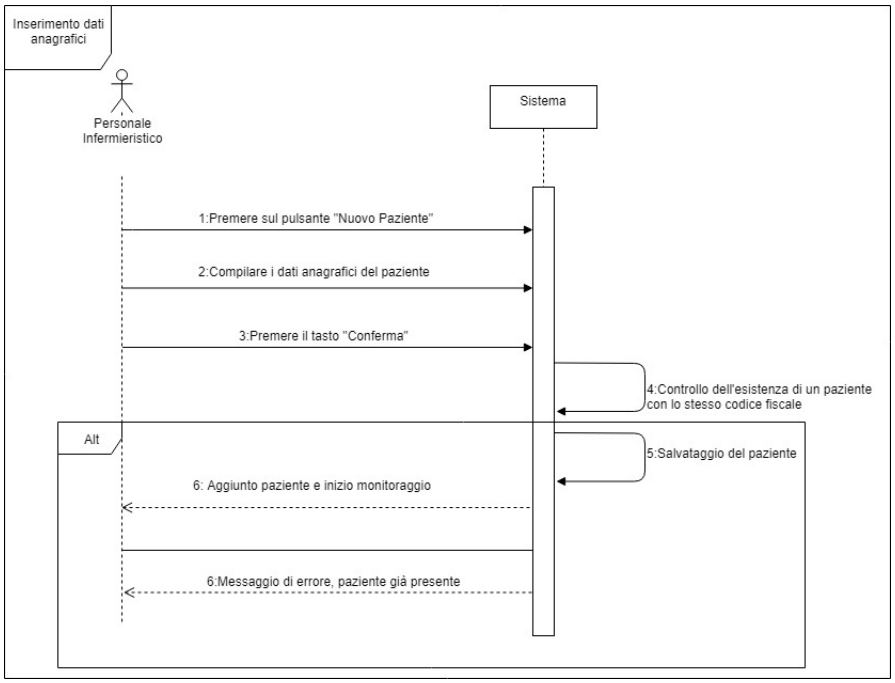


Figura 1.3: Sequence diagram Anagrafica

Somministrazioni

Il personale infermieristico deve registrare ogni somministrazione di qualunque sostanza prescritta ai pazienti:

Attori	Personale infermieristico
Precondizione	L'utente deve essere autenticato
Sequenza	1)L'utente preme sul pulsante "Aggiungi somministrazione" 2)L'utente seleziona il paziente, il farmaco e il numero di dosi 3)L'utente preme sul tasto "Conferma"
Post-condizione	Aggiunta della somministrazione

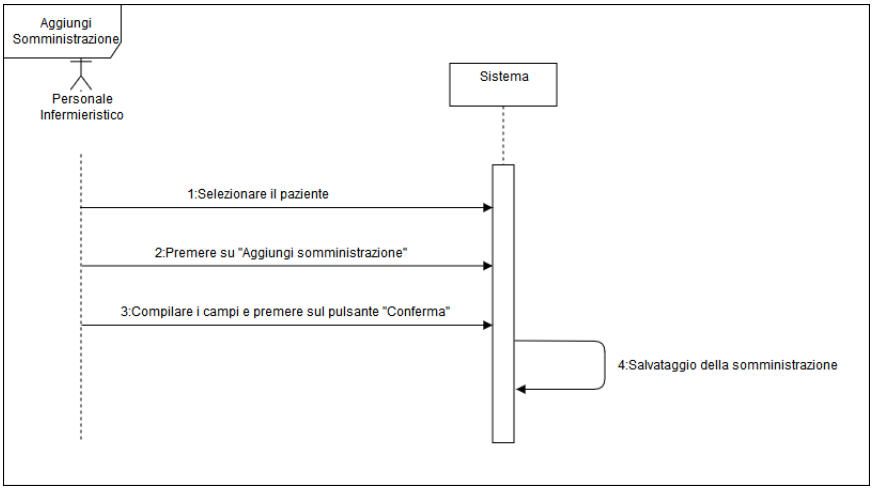


Figura 1.4: Sequence diagram Somministrazioni

1.1.5 Casi d'uso personale medico

Diagnosi di ingresso

Il personale medico quando un paziente viene registrato deve inserire all'interno della cartella clinica la diagnosi:

Attori	Personale medico
Precondizione	L'utente deve essere autenticato e il paziente deve essere registrato
Sequenza	1)L'utente seleziona il paziente 2)L'utente preme sul pulsante "Diagnosi" 3)L'utente compila la diagnosi e preme "Conferma"
Post-condizione	Aggiunta della diagnosi

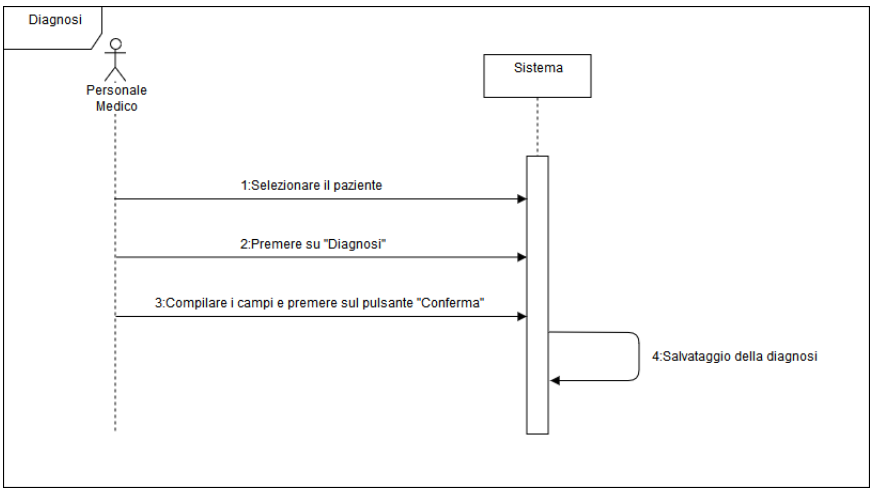


Figura 1.5: Sequence diagram Diagnosi

Nuova Prescrizione

Il personale medico deve poter inserire nuove prescrizioni di sostanze per i pazienti:

Attori	Personale medico
Precondizione	L'utente deve essere autenticato e il paziente deve essere registrato
Sequenza	1) L'utente seleziona il paziente 2)L'utente preme sul pulsante "Aggiungi prescrizione" 3)L'utente compila e preme sul tasto "Conferma"
Post-condizione	Aggiunta della prescrizione

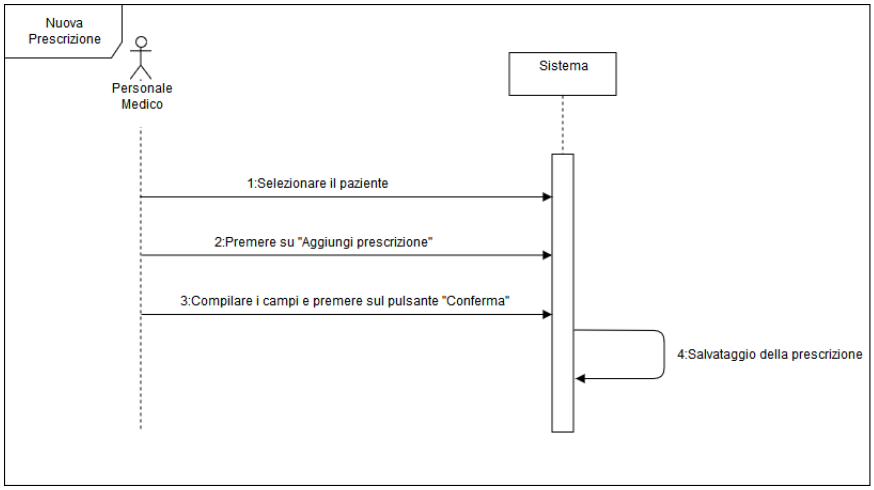


Figura 1.6: Sequence diagram Prescrizioni

1.1.6 Casi d'uso del primario

Il primario eredita tutti i casi d'uso del personale medico(di cui fa parte) in può possiede alcuni use case specifici:

Dimissioni paziente

Il primario può dimettere un paziente scrivendo una lettera di dimissione:

Attori	Primario
Precondizione	L'utente deve essere autenticato e il paziente deve essere registrato
Sequenza	1)L'utente preme sul pulsante "Dimissioni" 2)L'utente redige la lettera di dimissioni 3)L'utente preme sul tasto "Conferma"
Post-condizione	Il paziente viene rimosso

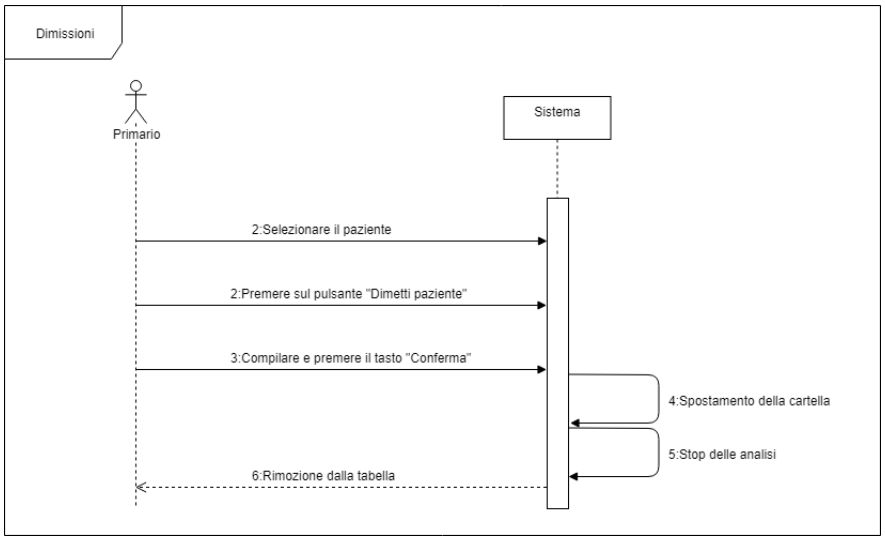
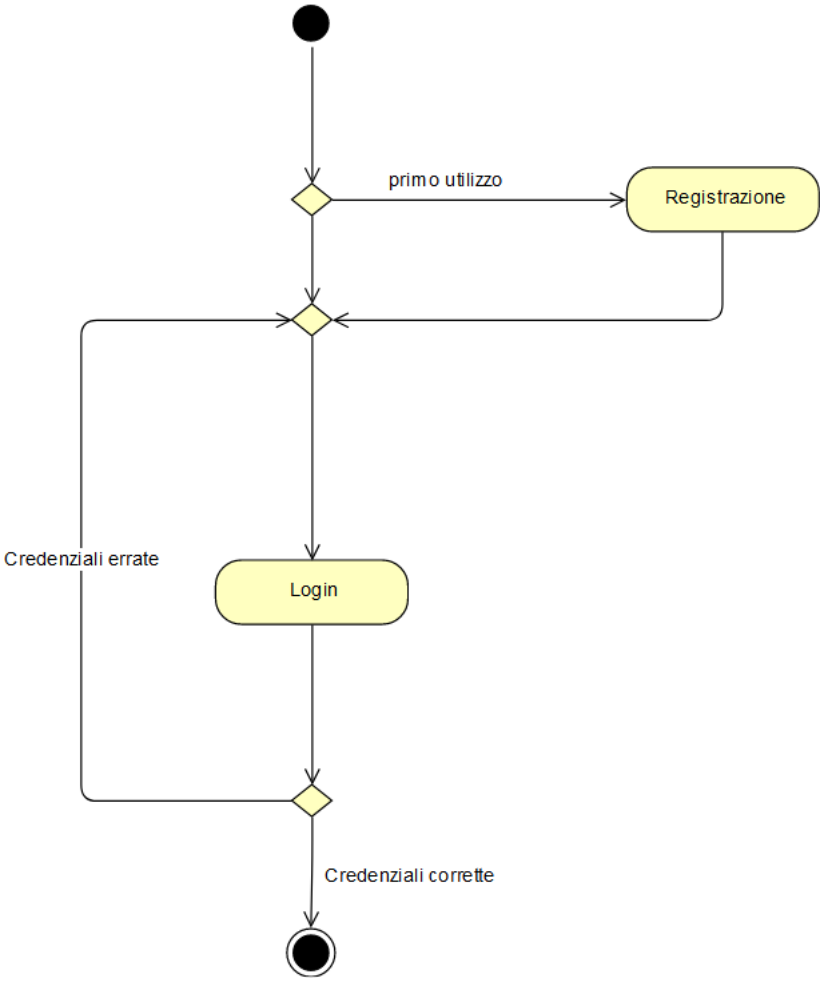


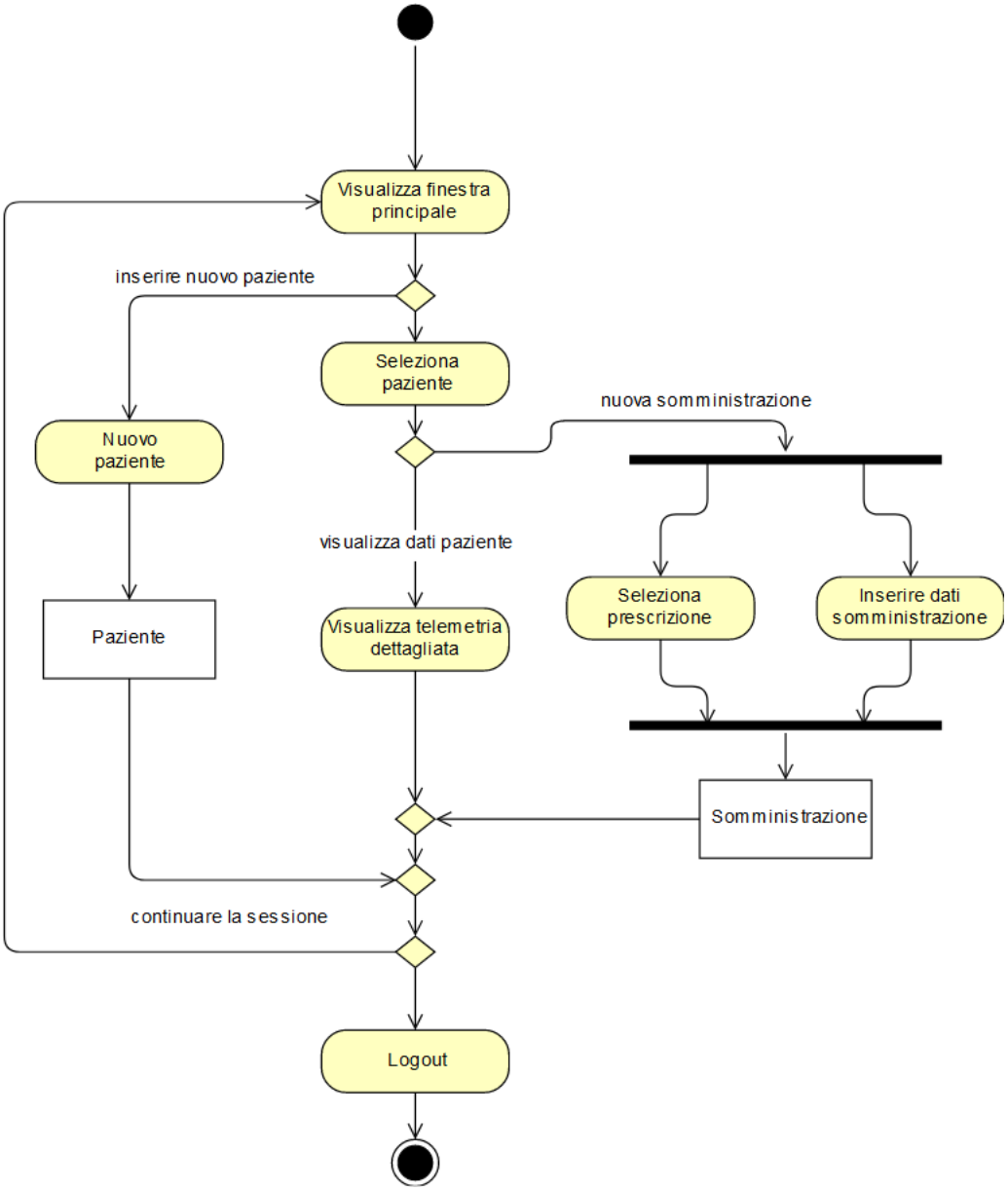
Figura 1.7: Sequence diagram Dimissioni

1.2 Activity diagrams

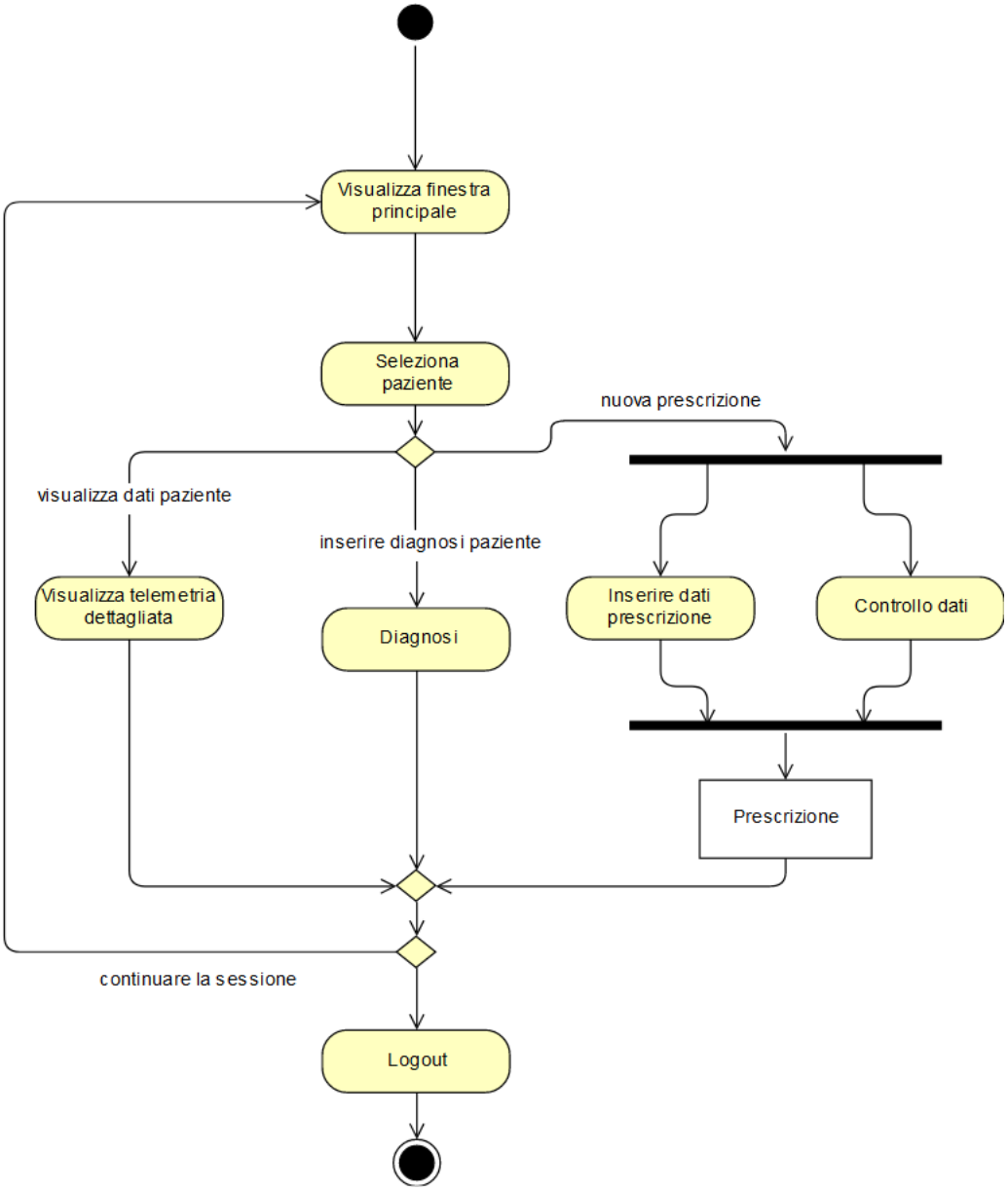
Login



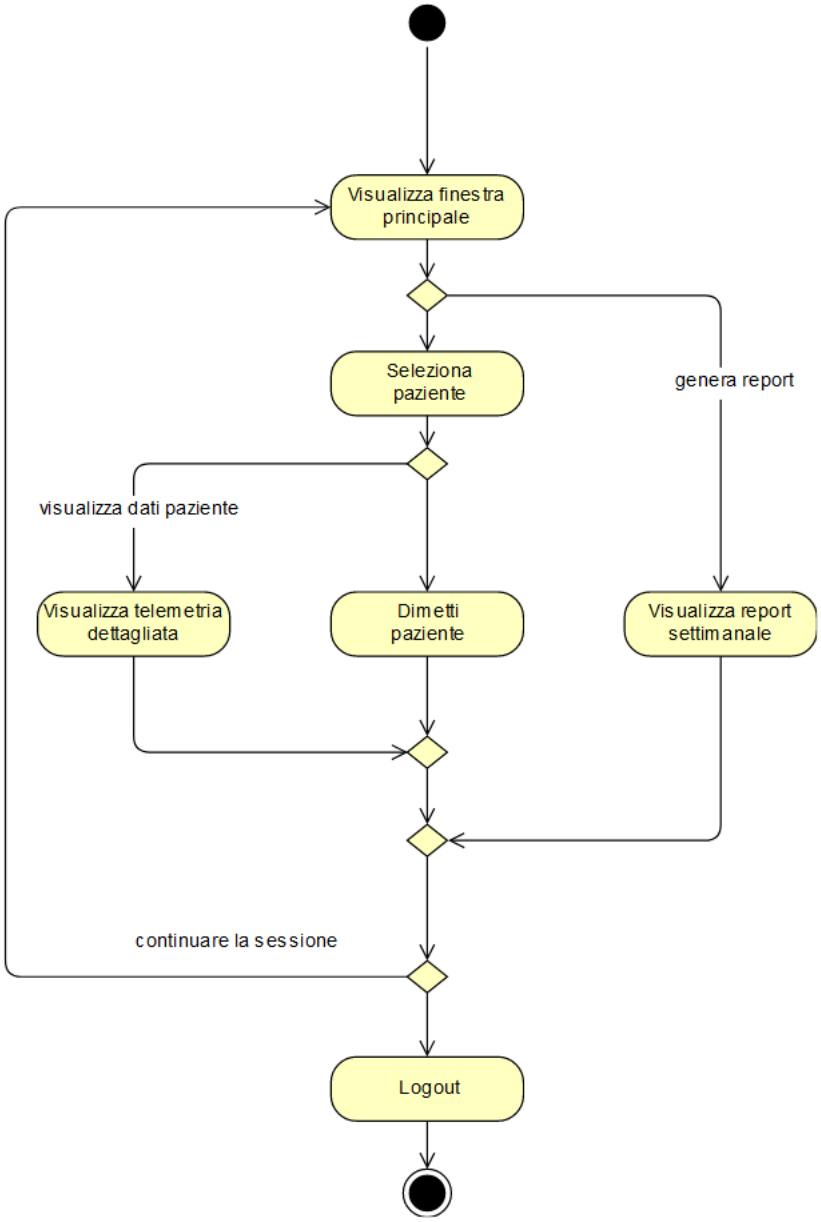
Infermiere


Visualizza finestra principale
inserire nuovo paziente
Seleziona paziente
nuova somministrazione
Nuovo paziente
visualizza dati paziente
Seleziona prescrizione
Inserire dati somministrazione
Paziente
Visualizza telemetria dettagliata
Somministrazione
continuare la sessione
Logout

Medico


Visualizza finestra principale
Seleziona paziente
nuova prescrizione
visualizza dati paziente
inserire diagnosi paziente
Visualizza telemetria dettagliata
Diagnosi
Inserire dati prescrizione
Controllo dati
Prescrizione
continuare la sessione
Logout

Primario


Visualizza finestra principale
Seleziona paziente
genera report
visualizza dati paziente
Visualizza telemetria dettagliata
Dimetti paziente
Visualizza report settimanale
continuare la sessione
Logout

Capitolo 2

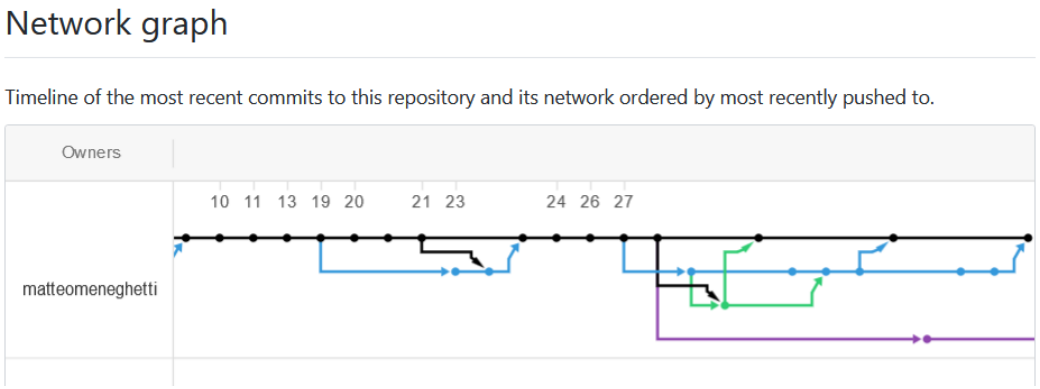
Sviluppo

2.1 Tecnologie utilizzate

Agli albori del processo di sviluppo abbiamo deciso di usare il linguaggio di programmazione **Java**. Questo, infatti, implementa nativamente tutti gli strumenti che abbiamo utilizzato, quali input/output da file, threading e sincronizzazione. Abbiamo infine scelto di utilizzare il framework **Swing** per realizzare l'interfaccia grafica, in quanto di facile utilizzo e di eccellente portabilità.

Abbiamo adoperato metodi di sviluppo di tipo *Agile*; il processo di sviluppo è stato diviso in finestre di tempo limitate chiamate iterazioni, con l'obiettivo di implementare ad ogni iterazione nuove funzionalità pronte immediatamente al testing e seguente uso.

La natura del lavoro di gruppo ci ha spinto ad adottare **Git** come software di controllo versione. Ne abbiamo beneficiato potendo delegare a un tool esterno la gestione dei files, comparare la stesura del codice in momenti separati e poter, in caso di errori, ritornare a versioni del software precedenti. I commit sono stati periodicamente pushati su un repository remoto hostato su **Github**.

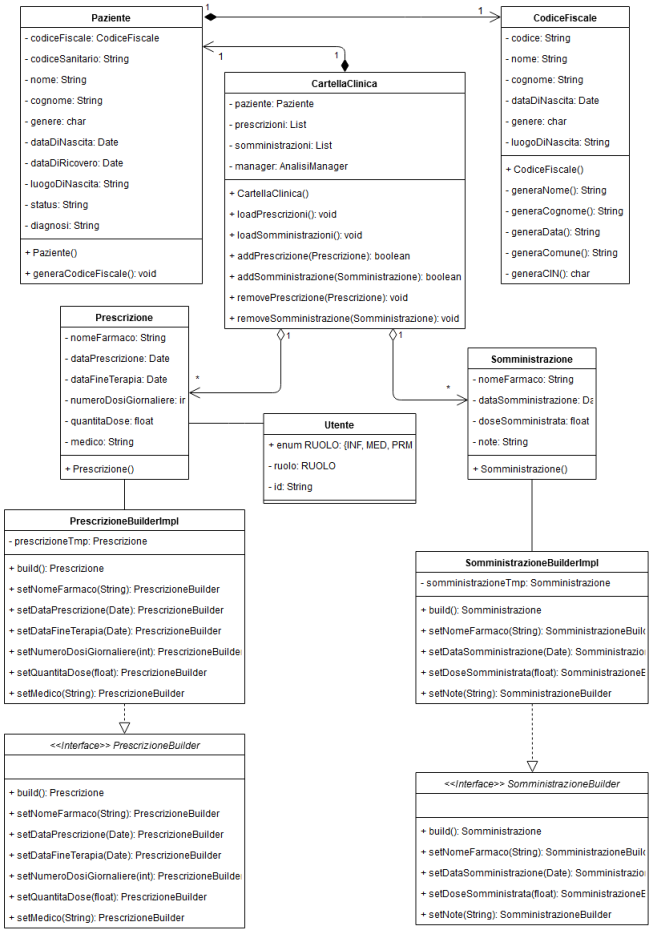


2.2 Design Patterns utilizzati

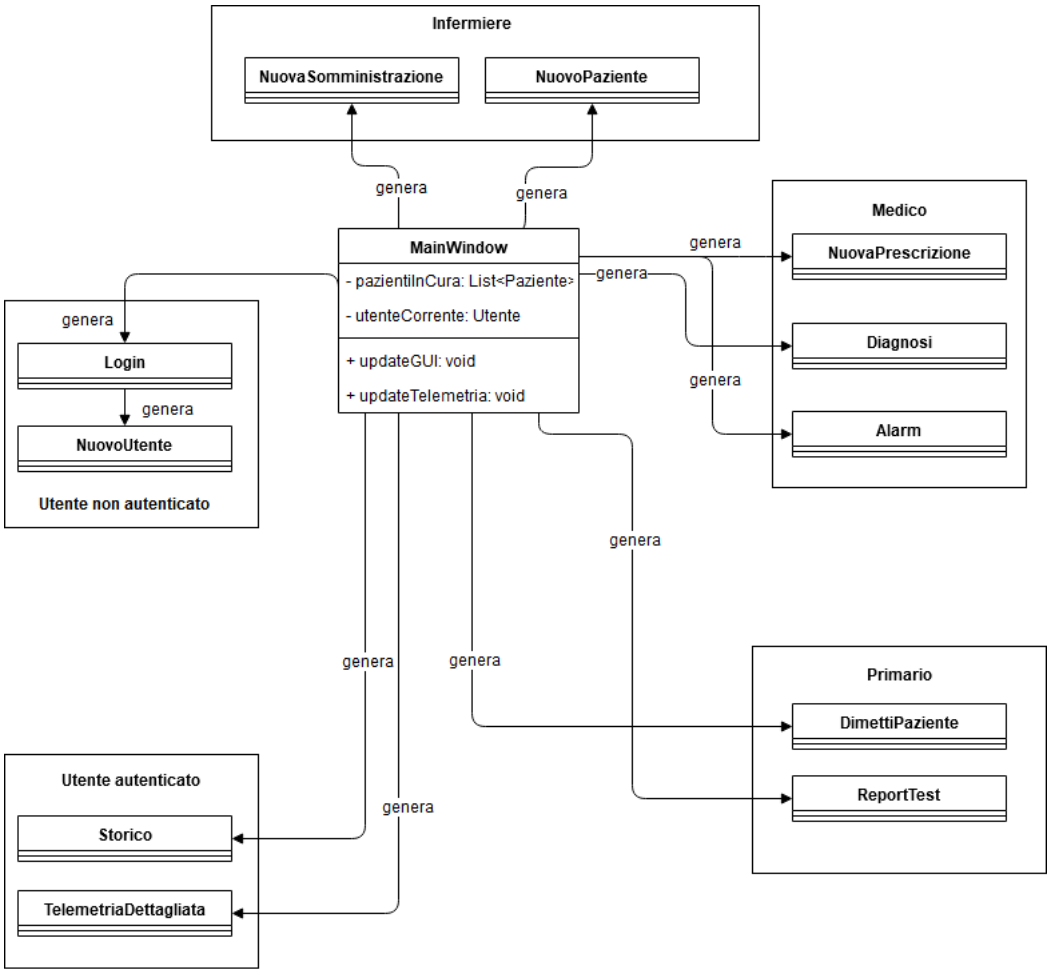
Pattern architetturale

Il pattern architetturale scelto è il **MVC** (Model-View-Controller), supportato nativamente dal framework Swing. Il progetto è infatti separato in tre componenti principali:

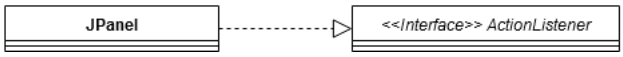
- Model: corrisponde ai package data e analisi, i quali contengono le classi e le strutture dati adoperate dal sistema software.
- View: si trova interamente nel package gui, si occupa di gestire l'interazione visuale con l'utente.
- Controller: anche quest'ultima parte è localizzata nel package gui. Infatti, ogni classe ivi presente corrisponde a una finestra di dialogo, ognuna con compiti diversi da assolvere, quindi necessita di un listener di eventi ad hoc e con accesso personalizzato alle componenti.

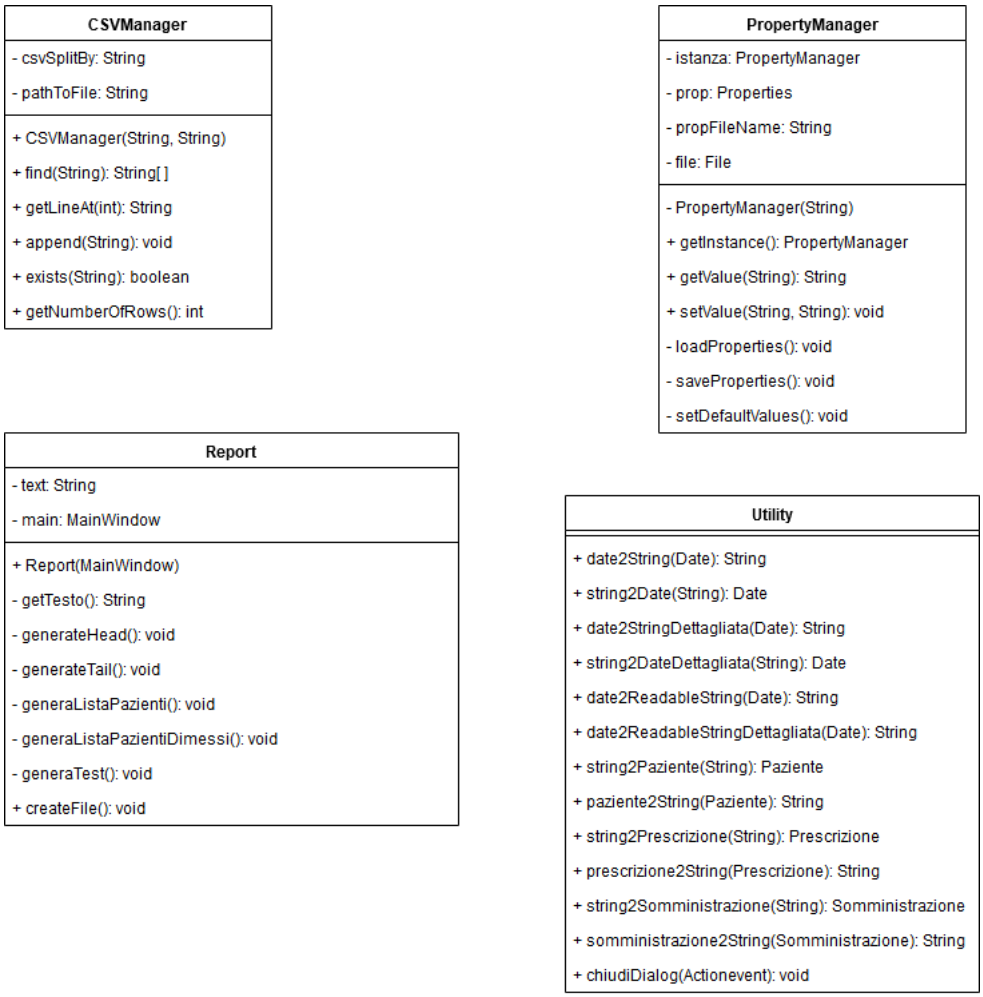


Da notare come la classe CartellaClinica sia l'hub centrale per accedere a tutti i dati rilevanti del paziente: l'anagrafica del paziente stesso, la lista di somministrazioni e prescrizioni.



Ogni JPanel implementa ActionListener




CSVManager
- csvSplitBy: String
- pathToFile: String
+ CSVManager(String, String)
+ find(String): String[ ]
+ getLineAt(int): String
+ append(String): void
+ exists(String): boolean
+ getNumberOfRows(): int
PropertyManager
- istanza: PropertyManager
- prop: Properties
- propFileName: String
- file: File
- PropertyManager(String)
+ getInstance(): PropertyManager
+ getValue(String): String
+ setValue(String, String): void
- loadProperties(): void
- saveProperties(): void
- setDefaultValues(): void
Report
- text: String
- main: MainWindow
+ Report(MainWindow)
- getTesto(): String
- generateHead(): void
- generateTail(): void
- generaListaPazienti(): void
- generaListaPazientiDimessi(): void
- generaTest(): void
+ createFile(): void
Utility
+ date2String(Date): String
+ string2Date(String): Date
+ date2StringDettagliata(Date): String
+ string2DateDettagliata(String): Date
+ date2ReadableString(Date): String
+ date2ReadableStringDettagliata(Date): String
+ string2Paziente(String): Paziente
+ paziente2String(Paziente): String
+ string2Prescrizione(String): Prescrizione
+ prescrizione2String(Prescrizione): String
+ string2Somministrazione(String): Somministrazione
+ somministrazione2String(Somministrazione): String
+ chiudiDialog(Actionevent): void

Pattern progettuali

Nota: si è cercato di non forzare l'utilizzo dei design patterns ma di attuarli con criterio dove ritenevamo necessario, in quanto offrono solidità alla struttura del progetto e ne facilitano la lettura da parte di terzi. I pattern implementati sono i seguenti:

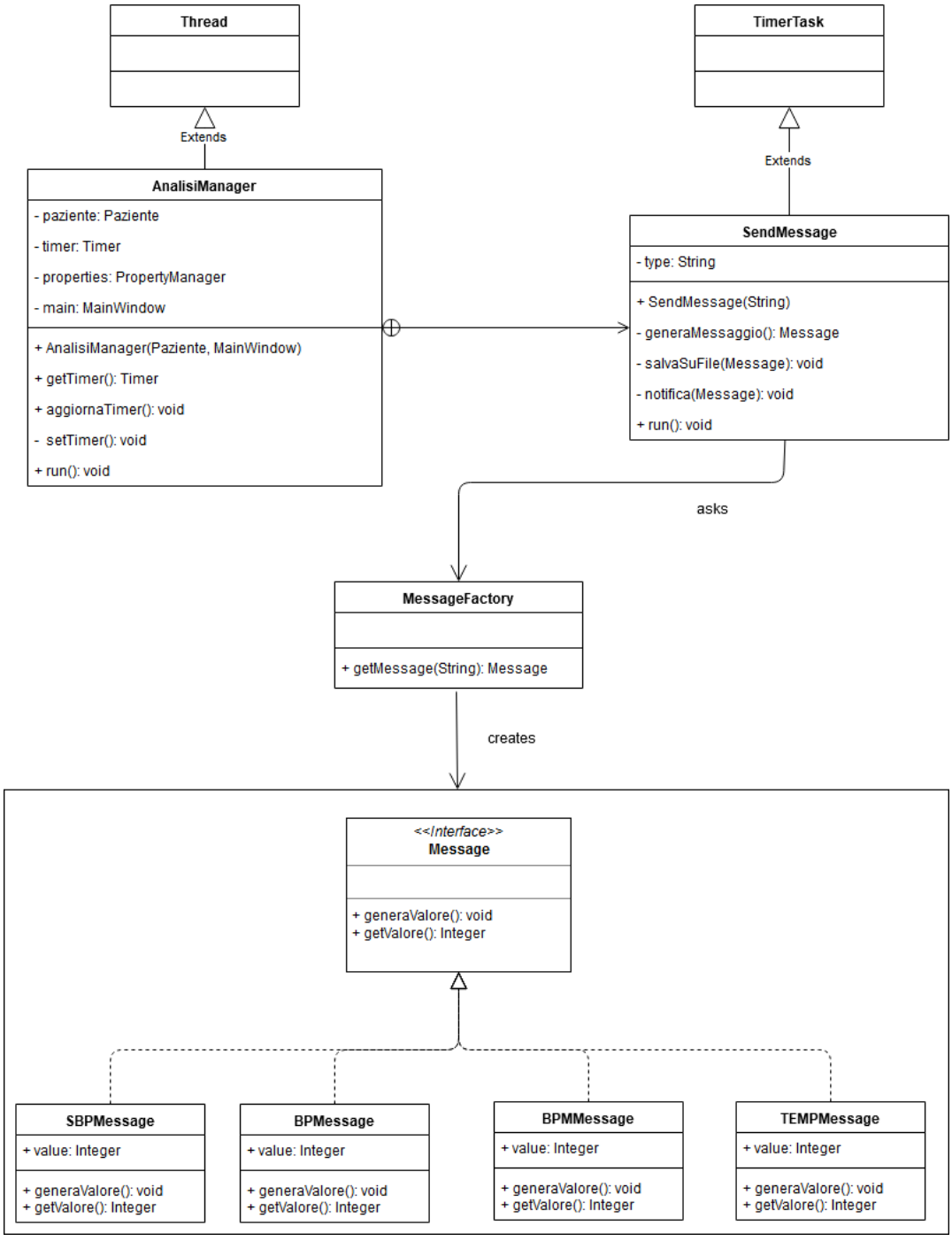
- Per il caricamento e il salvataggio delle proprietà è stata implementata una classe apposita nel package utility chiamata PropertyManager. Questa fornisce un wrapper per gestire la lettura e scrittura sul file .properties apposito. Per garantire che sia aperto solamente uno stream di input/output da file la classe PropertyManager implementa il pattern **singleton**. Il metodo getInstance() che garantisce l'accesso al singolo oggetto è invocato da un discreto numero di thread che scala linearmente con il numero di pazienti monitorati. Per questo nella firma del metodo lo si è definito come *synchronized*.

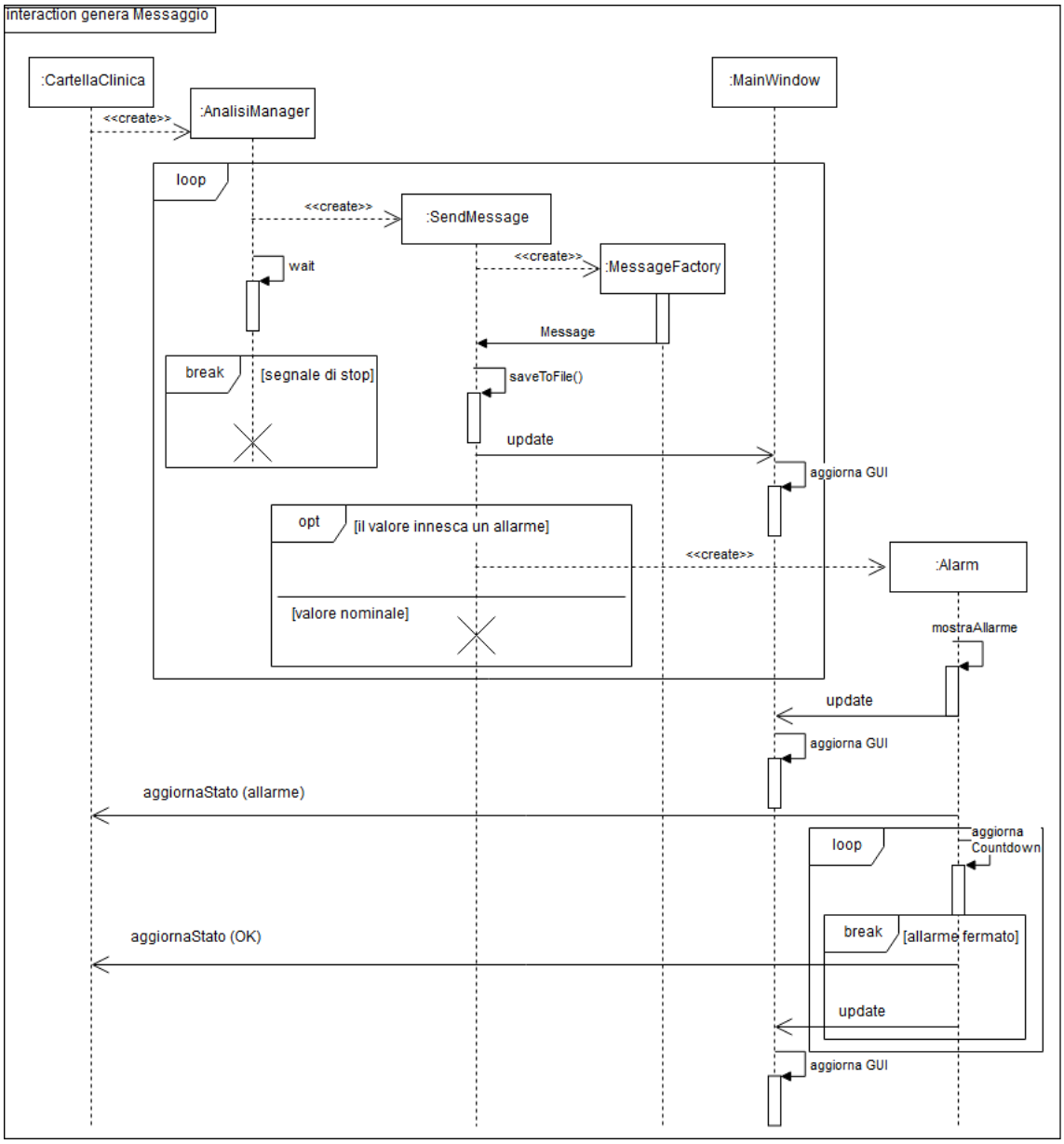
- Le classi Prescrizione e Somministrazione descrivono il comportamento e le strutture dati atte a rappresentare rispettivamente una prescrizione scritta da un medico e una somministrazione eseguita da un infermiere. Il numero di argomenti dei costruttori di queste classi è pertanto elevato e risulta difficoltosa la lettura o rispettare l'ordine dei parametri durante l'invocazione. Per ovviare a queste difficoltà si è deciso di implementare il pattern **builder**, che garantisce un'interfaccia chiara e user-friendly per la creazione di oggetti.

- Come già anticipato, si è ritenuta critica l'astrazione e l'immediatezza per sincronizzare il Model con i dati presenti su disco. Come PropertyManager, anche la classe CSVManager offre diversi metodi per gestire file di estensione .csv, utilizzando metodi nativi delle librerie Reader e Writer di Java ma mascherandole al programmatore e fornendo uno strumento di assai più facile utilizzo. Lo si può considerare come un uso del **facade** pattern.

- Sono stati utilizzati pattern nativi di java, come l'**observer** per la gestione degli eventi generati dall'utente o l'**iterator**, fornito dall'implementazione di una qualunque List.

- Assai più interessante la generazione dei messaggi durante il monitoraggio dei pazienti. Un messaggio è un'istanza di una classe che implementa l'interfaccia Message. Ogni messaggio si differenzia dagli altri per il range dei valori che può assumere. Tuttavia il salvataggio dei messaggi su disco e la visualizzazione degli stessi deve essere medesima, quindi necessitavamo di creare oggetti di tipo diverso a runtime che però si comportassero in modo simile. Il pattern **factory** è perfetto a questo scopo e la sua implementazione è il fulcro del package analisi.


Thread
Extends
TimerTask
Extends
AnalisiManager
- paziente: Paziente
- timer: Timer
- properties: PropertyManager
- main: MainWindow
+ AnalisiManager(Paziente, MainWindow)
+ getTimer(): Timer
+ aggiornaTimer(): void
- setTimer(): void
+ run(): void
SendMessage
- type: String
+ SendMessage(String)
- generaMessaggio(): Message
- salvaSuFile(Message): void
- notifica(Message): void
+ run(): void
asks
MessageFactory
+ getMessage(String): Message
creates
<<Interface>>
Message
+ generaValore(): void
+ getValore(): Integer
SBPMessage
+ value: Integer
+ generaValore(): void
+ getValore(): Integer
BPMessage
+ value: Integer
+ generaValore(): void
+ getValore(): Integer
BPMMessage
+ value: Integer
+ generaValore(): void
+ getValore(): Integer
TEMPMessage
+ value: Integer
+ generaValore(): void
+ getValore(): Integer


interaction genera Messaggio
:CartellaClinica
:AnalisiManager
:MainWindow
<<create>>
loop
<<create>>
:SendMessage
<<create>>
:MessageFactory
wait
Message
break
[segnale di stop]
saveToFile()
update
aggiorna GUI
opt
[il valore innesca un allarme]
<<create>>
:Alarm
[valore nominale]
mostraAllarme
update
aggiorna GUI
aggiornaStato (allarme)
loop
aggiorna Countdown
break
[allarme fermato]
aggiornaStato (OK)
update
aggiorna GUI

Capitolo 3

Test

Test Manuali

Per l'operazione di validazione e bug-fixing del software sono stati effettuati vari test da parte degli sviluppatori tra cui:

- Autenticazione con dati errati o mancanti PASSATO
- Controllo che ogni utente abbia accesso solamente alle funzioni a lui dedicate PASSATO
- Inserimento con dati mancanti di un nuovo paziente PASSATO
- Inserimento di una nuova prescrizione PASSATO
- Inserimento di una nuova somministrazione PASSATO
- Tentato inserimento di oltre 10 pazienti PASSATO
- Possibilità di osservare dati del paziente in modalità live PASSATO
- Impossibilità di dimettere un paziente in stato critico PASSATO
- Generazione del file report.html PASSATO
- Cambiare i valori del file properties a runtime PASSATO

Test utenti esterni

Il software è stato fatto provare anche ad utenti al di fuori del ciclo di sviluppo. Sono stati dati loro indicazioni sulle operazioni che dovevano portare a compimento e nella maggioranza dei casi hanno portato a termine il loro compito. Nei casi rimanenti i problemi erano imputabili all'interfaccia, quindi ne è seguita una fase di analisi con lo scopo di renderla più intuitiva.

Protocollo ARP: perché è stato introdotto?

Il protocollo