Relazione progetto di Ingegneria del software 2019

Edoardo Zorzi, Elia Piccoli, Marian Statache

Luglio 2019

Indice

1 Ingegneria e sviluppo

1.1 Organizzazione 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.

1.1.1 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. In paritcolare

2 Specifica

2.1 Casi d'uso

2.2 Diagrammi di classe

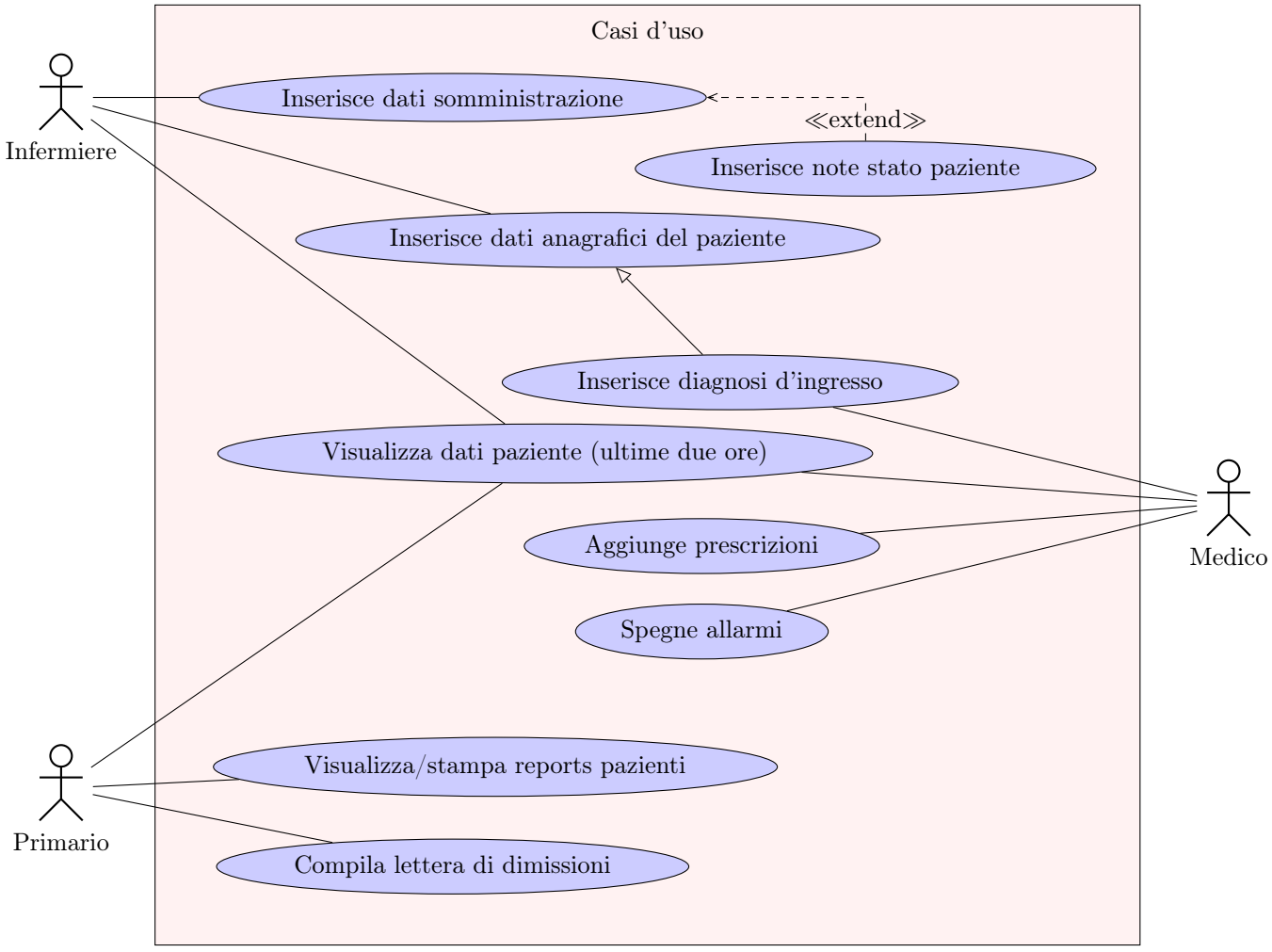

Figura 1 Diagramma UML dei casi d'uso del sistema.

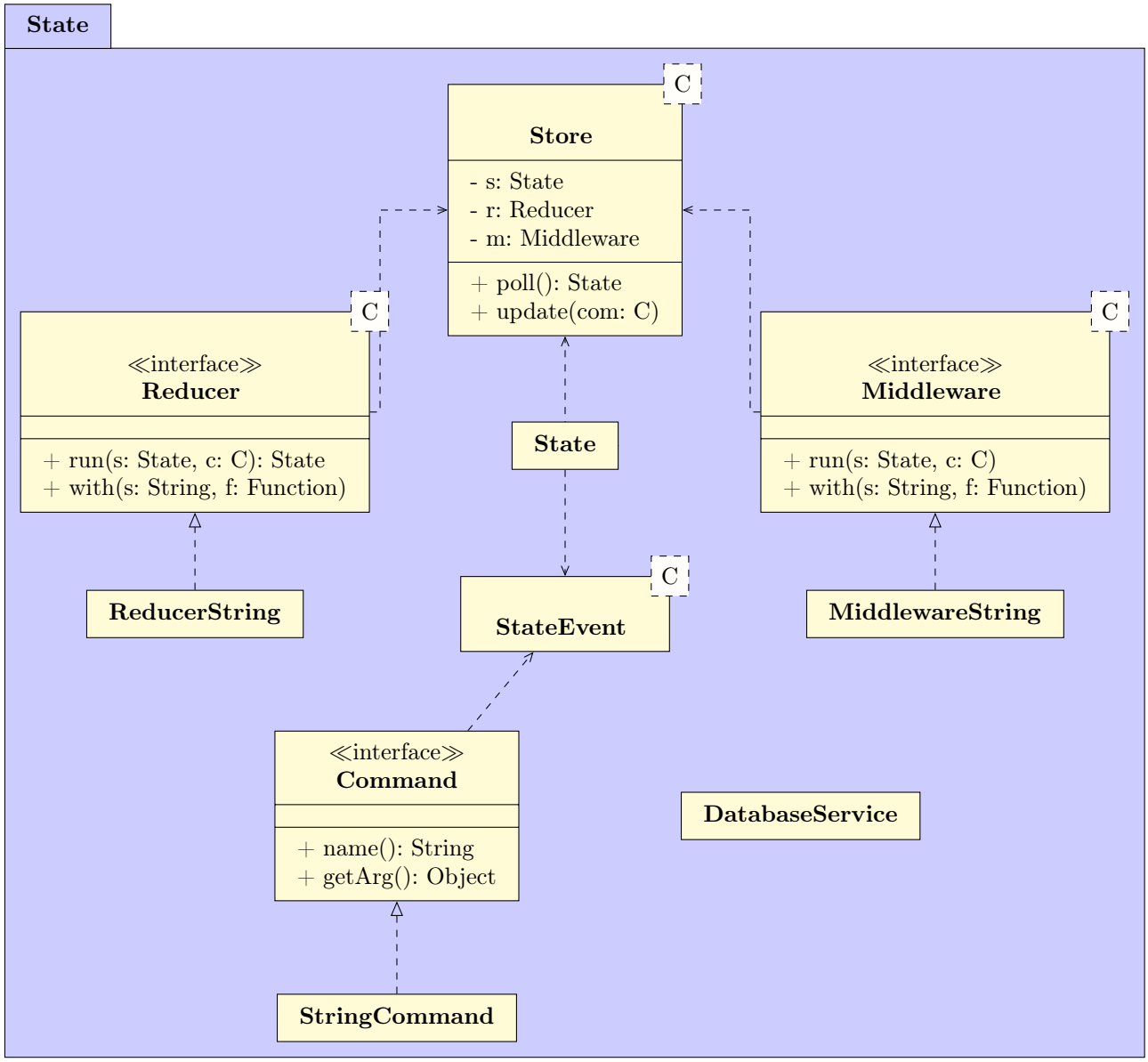

Figura 2 Diagramma di classe del *package* State.

Entità	Campi
Patient	Id, Name, Surname, Fiscal code, Place of birth, Date of birth
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, *Patient*
Monitoring	Id, Date, Diastolic pressure, Systolic pressure, Heart rate, Temperature, *Recovery*

Tabella 1 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).

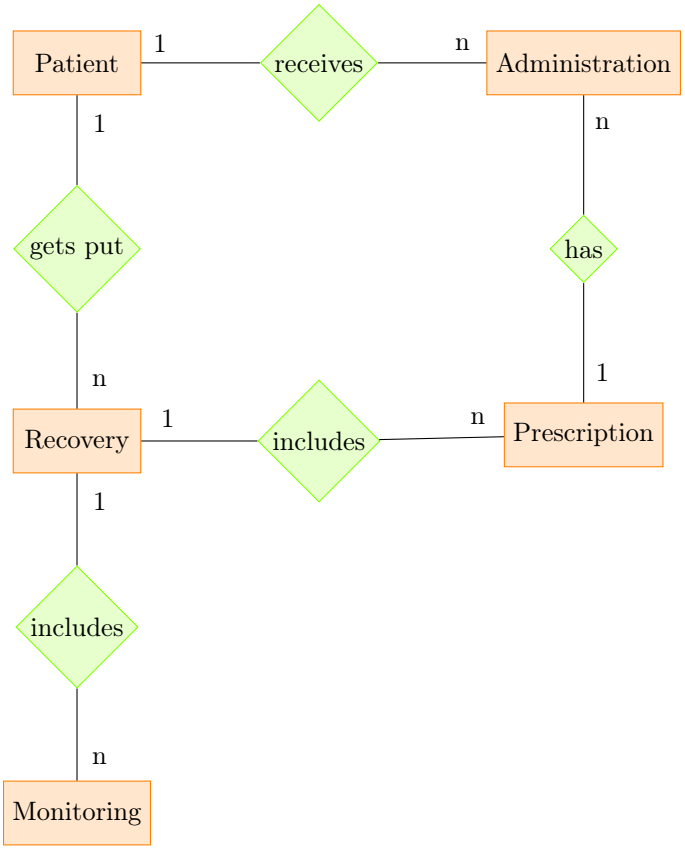

Figura 3 Schema *Entity-Relationship* dei dati del sistema.