

Documentazione Architetturale Sistema SGBU

Commento ai Diagrammi UML

1 Commento al Diagramma delle Classi

Il diagramma delle classi proposto rappresenta il modello statico del sistema SGBU, progettato seguendo un approccio Object-Oriented e adottando il pattern architettonico MVC (Model-View-Controller) a strati.

Per esaminare il Diagramma delle Classi, [clicca qui per andare alla figura](#).

1.1 Struttura Logica

Il sistema è decomposto in quattro livelli logici distinti, ciascuno con una responsabilità chiara e ben definita:

- **Livello di Presentazione (View):**

Rappresentato dalla classe `GUImageView` e `MessaggiInterfaccia`. Questo livello si occupa esclusivamente dell'interazione con l'utente (Bibliotecario) e della visualizzazione dei dati, senza contenere alcuna logica di business.

- **Livello di Controllo (Controller):**

La classe `GUIController` funge da orchestratore centrale. Essa non implementa la logica di dettaglio, ma coordina il flusso delle operazioni, delega le validazioni e smista le richieste ai servizi appropriati.

- **Livello di Servizio (Model/Service):**

Le classi `Catalogo`, `Anagrafica` e `RegistroPrestiti` costituiscono il cuore applicativo. Esse encapsulano la logica di business, gestiscono le collezioni di dati e garantiscono l'integrità delle transazioni (es. verifica disponibilità e limiti).

- **Livello di Dominio (Domain) e Infrastruttura:**

Le entità `Libro`, `Utente`, `Prestito` e `Credenziali` rappresentano i dati puri. Le interfacce `IArchivioDati`, `ILogger` e `IAutenticatore` (con le relative implementazioni concrete) isolano il sistema dai dettagli tecnici di persistenza e sicurezza.

1.2 Analisi di Coesione e Accoppiamento

Le decisioni di decomposizione modulare sono state guidate dalla necessità di massimizzare la coesione interna delle classi e minimizzare l'accoppiamento tra i moduli.

1.2.1 Coesione Funzionale

Tutte le classi identificate presentano un livello di Coesione Funzionale (il livello più alto e desiderabile), in quanto ogni modulo è focalizzato su un singolo compito ben definito.

- La classe `ValidatoreDati` si occupa esclusivamente della verifica sintattica.
- La classe `AuditTrail` si occupa esclusivamente di registrare le operazioni svolte, separando il tracciamento dalla logica specifica dei prestiti.
- Il `RegistroPrestiti` accentra tutte le regole relative alle transazioni.

1.2.2 Basso Accoppiamento e Gestione delle Dipendenze

Abbiamo strutturato le relazioni per evitare un accoppiamento stretto (come il Content o Control Coupling).

- Le dipendenze sono gestite tramite **Accoppiamento per Dati**, passando solo le informazioni strettamente necessarie.
- L'introduzione delle interfacce (`IArchivioDati`, `ILogger`, `IAutenticatore`) ha disaccoppiato il `GUIController` dalle implementazioni concrete.

1.3 Principi di Progettazione Adottati

Il diagramma riflette l'applicazione dei principi di buona progettazione orientata agli oggetti:

- **Separazione delle Preoccupazioni (SoC)**: Il sistema rispetta la separazione tra logica di presentazione, logica di business e gestione dati. La `GUIView` non contiene logica applicativa; il `GUIController` è un puro orchestratore.
- **Single Responsibility Principle (SRP)**: Ogni classe ha una sola responsabilità. La divisione tra `FileArchivio` (persistenza) e `Catalogo` (logica di dominio) ne è un esempio.
- **Dependency Inversion Principle (DIP)**: I moduli di alto livello dipendono dalle astrazioni. Il `GUIController` dipende da `IArchivioDati`, non dai dettagli di basso livello come `FileArchivio`, rendendo il sistema pronto per future strategie di memorizzazione (es. Database).
- **Open-Closed Principle (OCP)**: Grazie alle interfacce, il sistema è "aperto all'estensione ma chiuso alla modifica". È possibile introdurre nuovi meccanismi di log o autenticazione senza alterare le classi esistenti.

2 Commento al Diagramma di Sequenza: Registrazione Prestito

Il diagramma dettaglia il flusso di interazione per assegnare un libro a un utente, rispettando il pattern MVC.

Per esaminare il Diagramma di Sequenza per la Registrazione Prestito, [clicca qui per andare alla figura](#).

2.1 Analisi del Flusso e Scelte Progettuali

- **Gestione dell'Input e Validazione (Controller):** L'attore *Bibliotecario* interagisce con la *GUICollectionView*, che delega al *GUIController*. Quest'ultimo esegue una validazione formale tramite *ValidatoreDati* prima di coinvolgere la logica di business.
- **Logica di Business e Gestione dei Vincoli (Service Layer):** La classe *RegistroPrestiti* gestisce l'elaborazione. Un *Frame di Interazione alt* modella i vincoli:
 - **Limite Prestiti [FC-2]:** Verifica se l'utente ha superato il limite di prestiti.
 - **Disponibilità [FC-1]:** Verifica la presenza di copie fisiche.
- Se una guardia fallisce, il flusso termina con un errore.
- **Creazione e Aggiornamento di Stato:** Nel ramo di successo, il *RegistroPrestiti* istanzia il nuovo *Prestito* e aggiorna *Libro* e *Utente* invocando i loro metodi, mantenendo l'incapsulamento.
- **Tracciamento (Audit Trail):** Al termine, il controller invoca l'interfaccia *ILogger*, soddisfacendo il requisito [IF-11] senza inquinare la logica del prestito.

3 Commento al Diagramma di Sequenza: Registrazione Restituzione

Il diagramma illustra il processo di chiusura di un prestito e il ripristino della disponibilità del libro.

Per esaminare il Diagramma di Sequenza per la Registrazione Restituzione, [clicca qui per andare alla figura](#).

3.1 Analisi del Flusso e Scelte Progettuali

- **Orchestrazione e Delega:** Il `GUIController` delega immediatamente l'intera logica transazionale al `RegistroPrestiti`, centralizzando la responsabilità del ciclo di vita del prestito.
- **Gestione delle Eccezioni di Business:** Un *Frame di Interazione opt* modella il requisito [UC-8] sui ritardi. Se la data di restituzione effettiva è successiva a quella prevista, viene eseguita la logica di gestione del ritardo.
- **Aggiornamento Consistente dello Stato:** Il `RegistroPrestiti` coordina una sequenza atomica di aggiornamenti:
 1. Incremento delle copie disponibili del `Libro`.
 2. Rimozione del prestito dalla lista dell'`Utente`.
 3. Chiusura definitiva dell'oggetto `Prestito`.
- **Tracciamento:** Viene invocata l'interfaccia `ILogger` per storicizzare la modifica allo stato persistente.

4 Commento al Diagramma dei Package

Il Diagramma dei Package illustra l'organizzazione logica del codice sorgente, strutturato per gestire la complessità e garantire la *Separazione delle Preoccupazioni*.

Per esaminare il Diagramma dei Package, [clicca qui per andare alla figura](#).

4.1 Organizzazione Architetturale

Il sistema segue un'architettura a strati che riflette il pattern MVC:

- **it.unisa.sgbu.domain (Dominio):** Il nucleo stabile del sistema. Contiene le entità (Libro, Utente, Prestito) e utility di validazione. È indipendente da tutti gli altri package.
- **it.unisa.sgbu.service (Logica di Business):** Contiene i gestori (RegistroPrestiti, Catalogo, Anagrafica). Dipende dal dominio ma non conosce l'interfaccia utente.
- **it.unisa.sgbu.io (Infrastruttura e Servizi):** Raggruppa componenti tecnici per persistenza, logging e sicurezza (IArchivioDati, FileArchivio, ecc.). L'isolamento permette di modificare le tecnologie senza impatto sul sistema.
- **it.unisa.sgbu.gui (Presentazione e Controllo):** Contiene GUILiveView e GUIController. Agisce da livello superiore orchestrando le interazioni.
- **it.unisa.sgbu.app (Avvio):** Contiene solo il Main per il bootstrap.