UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Relazione progetto di Programmazione a Oggetti

Nicolò Bovo - mat. 2042885
Bibliotech

Capitoli presenti nel documento:

1 Introduzione

Bibliotech è un software per gestire una biblioteca multimediale che offre la possibilità di creare e modificare diversi tipi di contenuti come libri, film e album musicali. L'applicazione è stata sviluppata utilizzando il linguaggio di programmazione C++ e ha un'interfaccia grafica basata sul framework Qt che permette agli utenti di interagire facilmente con i vari media e accedere alle funzioni di ricerca e salvataggio dei dati.
Il progetto trae ispirazione dai bisogni tipici di una biblioteca moderna, dove i documenti non sono limitati esclusivamente ai libri cartacei ma includono sorgenti multimediali di vario tipo. Il punto di forza principale di Bibliotech è l'implementazione di un modello basato su oggetti che si ispira a un polimorfismo non semplice, dove ciascuna tipologia di media (libro, film o album musicale) possiede attributi e comportamenti specifici, pur condividendo una struttura comune. Il programma, inoltre, è in grado di effettuare ricerche mirate (ad esempio, per titolo o autore), include moduli per modificare i campi di ciascun tipo di contenuto, e fornisce la possibilità di salvare i dati come file XML, consentendo così di conservarli tra diverse sessioni di utilizzo.

2 Descrizione del modello

Il modello logico di Bibliotech è rappresentato da una gerarchia di ereditarietà incentrata sulla classe astratta `Media`. Quest'ultima definisce le proprietà fondamentali di un contenuto multimediale (titolo, editore, anno di pubblicazione) e dichiara i metodi comuni, come `mathes(...)` per la ricerca testuale e `accept(...)` per il pattern Visitor. Da `Media` derivano le sottoclassi `Book`, `Film` e `MusicAlbum`, ciascuna con attributi e comportamenti specializzati (`Film` introduce regista e durata, `Book` include autore e trama, `MusicAlbum` gestisce informazioni come artista e genere).
La finalità di questa architettura è garantire un'unica interfaccia e un'unica logica di gestione per tutti i tipi di media, pur consentendo di differenziare e personalizzare le funzionalità in base al tipo concreto. Un esempio lampante è la funzione di ricerca: invocando `mathes(QString)`, il filtraggio si adegua automaticamente al tipo effettivo dell'oggetto (un libro cercherà corrispondenze anche nell'autore, un film nel regista, un album nell'artista, ecc.), senza richiedere istruzioni condizionali che discriminino manualmente le diverse sottoclassi.
Un ruolo altrettanto importante spetta alla classe `Biblioteca`, la quale funge da contenitore principale per i media. Internamente, essa mantiene un `std::vector<MediaPtr>` e mette a disposizione metodi quali `addMedia`, `removeMedia` e `search`, incapsulando l'accesso e la modifica della collezione. Inoltre, `Biblioteca` eredita tanto da `Subject` (per notificare all'interfaccia quando vi sono cambiamenti nell'insieme dei media) quanto da `Observer` (per reagire agli aggiornamenti provenienti dai singoli `Media`, anch'essi derivati da `Subject`). Questo schema di notifiche incrociate consente di mantenere sincronizzata la vista con lo stato interno del modello in modo del tutto automatico.

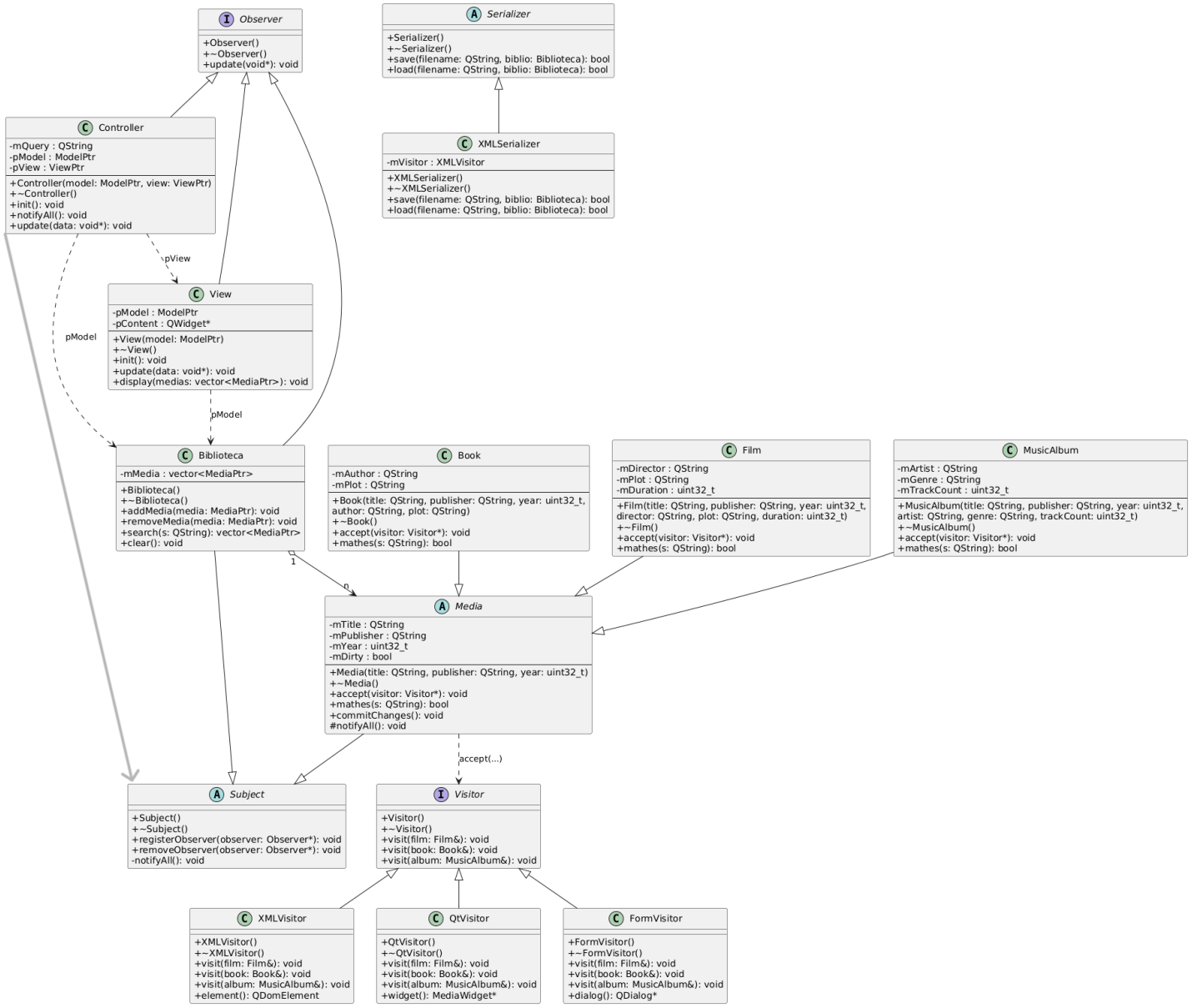

Figura 1: Diagramma UML

Nel diagramma UML (riportato in figura 1) emergono, oltre alla gerarchia di classi `Media -> Book/Film/MusicAlbum`, ulteriori componenti che arricchiscono il modello:

- I **Visitor** (`XMLVisitor`, `QtVisitor`, `FormVisitor`) gestiscono, rispettivamente, la serializzazione su file XML, la costruzione di widget e la creazione di form di modifica.
- L'interfaccia `Serializer` e la sua specializzazione `XMLSerializer` si occupano di "tradurre" l'intera collezione in elementi XML (e viceversa), senza introdurre codice di persistenza nelle classi del modello.

In questo modo, la logica di base come l'attribuzione di campi, la ricerca i, meccanismi di notifica, risultano ben isolati da parti più specifiche, GUI, salvataggio, form di editing.

3 Polimorfismo

L'utilizzo principale del polimorfismo in Bibliotech si concretizza nel design pattern `Visitor` applicato alla gerarchia di classi derivate da `Media`. Questo approccio non si limita a rendere virtuali i distruttori o a introdurre piccole differenze nel comportamento delle sottoclassi, ma offre un vero meccanismo di estensione dinamica delle funzionalità.

- **Visitor per la visualizzazione:** `QtVisitor` e `FormVisitor` consentono di costruire in modo polimorfo i widget grafici e i form di modifica specifici di ciascun tipo di media. Ad esempio, un `Film` ha campi per regista e durata, mentre un `Book` espone autore e trama; tramite il metodo `accept(...)` presente in ogni classe derivata, il codice che interagisce con l'interfaccia non deve sapere a priori il tipo di media su cui opera, poiché il dispatch dinamico seleziona automaticamente la visita corretta.
- **Visitor per la serializzazione:** `XMLVisitor` implementa una logica di conversione distinta per ogni tipo di media, generando elementi XML personalizzati (come `<Book ...>` con l'autore e la trama, oppure `<Film ...>` con il regista e la durata). Anche in questo caso, l'utilizzo di `accept(...)` evita di inserire metodi di salvataggio direttamente nelle classi `Book`, `Film` e `MusicAlbum`, mantenendo il modello più pulito e facilmente estendibile.
- **Polimorfismo nel metodo `mathes(...)`:** Ogni sottoclasse di `Media` ridefinisce `mathes(s: QString)` in base alle proprie peculiarità. Per un film, si potranno cercare corrispondenze nel regista e nel titolo; per un libro, nell'autore e nella trama; per un album, nell'artista e nel genere. A livello di codice, tuttavia, basta invocare `media->mathes(...)` senza preoccuparsi di quale classe concreta sia coinvolta: il polimorfismo si occupa di chiamare la versione corretta del metodo.
- **Estendibilità:** Se in futuro si volesse introdurre una quarta tipologia di media (ad esempio, un "Documentario"), basterebbe definire una nuova classe `Documentario` derivata da `Media`, implementare le visite nei vari visitor (`XMLVisitor`, `QtVisitor` ecc.) e ridefinire i metodi polimorfi necessari (`mathes(...)` e `accept(...)`). L'architettura non richiede stravolgimenti delle classi esistenti, poiché il caricamento, la modifica e la visualizzazione del nuovo tipo possono essere gestiti tramite lo stesso meccanismo di dispatch dinamico.

Questo livello di polimorfismo risulta "non banale" poiché non si limita a piccoli adattamenti, ma incide in modo rilevante sul design complessivo, promuovendo la separazione delle responsabilità (modellazione dei media vs. logiche di interfaccia e persistenza) e favorendo una crescita modulare del progetto.

4 Persistenza dei dati

Bibliotech utilizza il formato XML per la persistenza dei dati, così da poter salvare e caricare la collezione di media tra diverse sessioni. In particolare, la classe `XMLSerializer` si fa carico di leggere e scrivere l'intero contenuto della classe `Biblioteca` su un file `.xml`, seguendo una struttura in cui ciascun `Media` (sia `Book`, sia `Film`, sia `MusicAlbum`) viene convertito in un diverso nodo XML.

- **Struttura XML** Il file salvato è organizzato in un nodo radice `<Biblioteca>`, al cui interno si trovano elementi come `<Book>` o `<Film>` per rappresentare i vari media. Ogni elemento include attributi specifici (titolo, autore, regista, durata, e così via). Poiché ciascun tipo di media deriva da `Media`, la scrittura e la lettura avvengono in modo polimorfo, usando il pattern `Visitor` (`XMLVisitor`) per inserire o estrarre i dati appropriati.

- **File di esempio** Nel progetto è incluso un file `dati_prova.xml` di esempio, per importare dei dati.
- **Gestione degli errori e flessibilità** Se durante il caricamento il file non risulta ben formato, la funzione `load(...)` restituisce `false`

Questo permette al pattern `Visitor` e sull'utilizzo di una classe `XMLSerializer`, rende la persistenza indipendente dal resto del modello.

5 Funzionalità implementate

Bibliotech comprende sia funzionalità "funzionali" (legate alla gestione effettiva della collezione) sia funzionalità grafiche che ne semplificano l'uso all'utente finale.

5.1 Funzionalità funzionali

- **Gestione di più tipologie di media** L'applicazione consente di aggiungere, modificare e rimuovere libri, film e album musicali, ognuno con attributi specifici.
- **Ricerca testuale polimorfa** Tramite il metodo `search(...)`, l'utente può cercare l'autore nei libri, il regista nei film e l'artista/genere negli album musicali.
- **Caricamento e salvataggio in formato XML** La classe `XMLSerializer` permette di importare ed esportare i dati su file `.xml`, sfruttando `XMLVisitor` per serializzare in modo specializzato libri, film e album musicali.
  - *Salvataggio* Il metodo `save(...)` di `XMLSerializer` riceve in input un oggetto `Biblioteca`. Per ogni media, viene invocato `media->accept(&mVisitor)`, in modo che `XMLVisitor` crei un nodo XML dedicato (ad esempio `<Book title="..." author="..." ...>`). Questi nodi vengono inseriti nel documento XML, che infine viene scritto sul file indicato dall'utente.
  - *Caricamento* Il metodo `load(...)` scansiona il file XML alla ricerca dei nodi `<Book>`, `<Film>`, `<MusicAlbum>`. In base al tag, viene creata la relativa classe concreta e vengono inizializzati i campi con i valori letti dal file, aggiungendo infine l'oggetto ricostruito alla collezione di `Biblioteca`.
- **Meccanismo di notifica e aggiornamento** Grazie all'uso del pattern `Observer`, eventuali modifiche apportate all'archivio o ai singoli `Media` vengono propagate in automatico, facilitando il refresh dell'interfaccia.

5.2 Funzionalità grafiche

- **Toolbar in alto** L'interfaccia `QMainWindow` include una barra degli strumenti con pulsanti: "Load", "Save", "Add Book", "Add Film", "Add Music Album", "Search" e "Reset"(fa il reset della ricerca). L'utente ha così a disposizione, in maniera semplice e intuitiva, le funzioni principali del programma.
- **Finestra di dialogo per la ricerca** Cliccando su "Search", compare un `QInputDialog` dove l'utente inserisce la chiave di ricerca.

- **Form di modifica dedicati** Quando si sceglie di editare un media, viene generato un `FormVisitor`, che non è altro che una finestra di dialogo con i campi personalizzati, in base al tipo: `Book`, `Film` o `MusicAlbum`.
- **Widget specifici per ogni media** Un visitor (`QtVisitor`) crea, per ogni oggetto, un widget (`MediaWidget`) con pulsanti "Delete" ed "Edit". Il layout di tali widget varia in base ai campi richiesti (ad esempio, un film mostra regista e durata; un album mostra artista e genere).
- **Dialog di apertura e salvataggio** L'applicazione utilizza le finestre di dialogo fornite da Qt (`QFileDialog`) per selezionare i file `.xml`, evitando percorsi cablati nel codice.

6 Rendicontazione ore

Attività	**Ore Previste**	**Ore Effettive**
Studio e progettazione	8	10
Sviluppo del codice del modello	8	10
Studio del framework Qt	10	10
Sviluppo del codice della GUI	10	12
Test e debug	2	2
Stesura della relazione	2	2
Totale	**40**	**46**

Tabella 1: Tabella di rendicontazione delle ore

Il monte ore è stato leggermente superato in quanto lo studio, la progettazione, lo sviluppo del codice del modello e la GUI ha richiesto più tempo di quanto previsto, in particolare sono stati svolti più test durante la creazione della GUI.