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

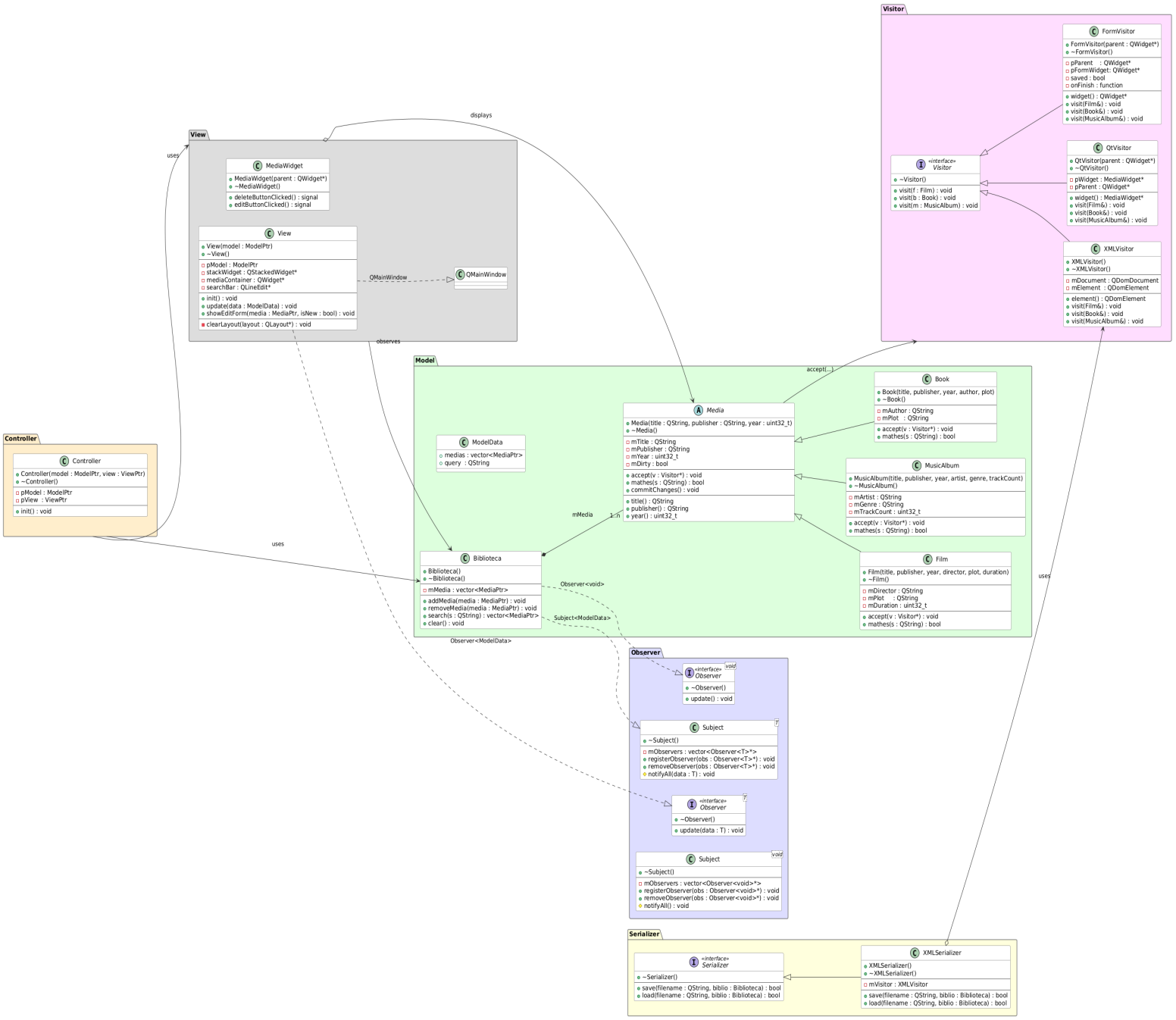

Figura 1: Diagramma UML di *Bibliotech*. 1

Il modello logico di *Bibliotech* si articola in due parti: la **gestione dei tipi di media** e la **funzionalità di ricerca**. La prima comprende sia le classi che descrivono gli oggetti multimediali, come si può vedere dall'UML (Fig. 1), sia alcune classi di supporto che si occupano di convertire tali oggetti in formato `XML` e di gestire la persistenza su file.

2.1 Gerarchia di `Media`

Il modello parte dalla classe astratta `Media`, che rappresenta le informazioni comuni a tutti i contenuti multimediali: *titolo*, *editore* e *anno di pubblicazione*. Questi attributi, insieme a metodi puramente virtuali come `matches(...)` (per la ricerca) e `accept(...)` (per il *Visitor*), costituiscono il nucleo condiviso dalle sottoclassi.
A partire da `Media`, si sviluppa una **gerarchia** che comprende:

- `Book`, dedicata ai libri;
- `Film`, pensata per i film;
- `MusicAlbum`, per gli album musicali.

2.2 `Classe Biblioteca`

Per la gestione vera e propria dei media, *Bibliotech* prevede la classe `Biblioteca`, che fa da *contenitore principale* e, in un certo senso, da "motore di ricerca" semplificato. `Biblioteca` è in grado di:

- **aggiungere** e **rimuovere** oggetti multimediali;
- **cercare** tra i media in base a una stringa, che non è altro che un testo inserito dall'utente;

`Biblioteca` memorizza i propri media in uno `std::vector<MediaPtr>` (dove `MediaPtr` è uno `std::shared_ptr<Media>`), incapsulandone l'accesso.

2.3 Strumenti di persistenza e Visitor

Nel programma si è scelto di non inserire nelle singole sottoclassi `Media` la logica di salvataggio/caricamento, ma di utilizzare il design pattern *Visitor* per arricchire dinamicamente le possibilità di elaborazione degli oggetti `Media`. In particolare:

- `XMLVisitor` fornisce i modi per convertire i campi di un `Film`, `Book` o `MusicAlbum` in nodi *XML*, inserendoli in un `QDomDocument`;
- `XMLSerializer` sfrutta `XMLVisitor` per *salvare* e *caricare* un'intera collezione `Biblioteca` da file `.xml`, creando o interpretando rispettivamente i tag `<Book>`, `<Film>`, `<MusicAlbum>`.

Questa separazione tra modello e logica di persistenza, ci garantisce una maggiore flessibilità in un eventuale cambio di formato, per esempio il passaggio da XML a JSON; o l'introduzione di nuovi campi nelle classi derivate richiedono soltanto di aggiornare i *visitor* e la classe `Serializer`, lasciando invariata la struttura principale di `Media`.

2.4 Pattern Observer e aggiornamento automatico

Un altro aspetto peculiare del modello di *Bibliotech* è il **meccanismo di notifica** basato sul pattern *Observer*. Poiché `Media` eredita da `SubjectNoParameters`, può *notificare* le modifiche interne ai propri osservatori. `Biblioteca`, a sua volta, eredita da `Subject<ModelData>` e `ObserverNoParamets`, reagendo alle variazioni di uno specifico `Media` (ad esempio un cambio del titolo) e propagando in automatico gli aggiornamenti verso la *View*, che si rinfresca di conseguenza. Si specifica che ObserverNoParameters e SubjectNoParameters sono equivalenti a Observer<void> e Subject<void>,

2.5 Estendibilità

Alla luce di quanto detto sopra, l'architettura svilupatta, permette di introdurre una nuova tipologia di media (es. un `Documentario`) o aggiungere campi a quelli esistenti in maniera molto semplice.
In sintesi, il modello di *Bibliotech* enfatizza la **modularità** e la **manutenibilità**, combinando eredità, *Observer* e *Visitor* per coprire le principali esigenze di gestione dei media.

3 Polimorfismo

L'utilizzo principale del polimorfismo in **Bibliotech** si manifesta nel *design pattern Visitor* applicato alla gerarchia di classi derivate da `Media`. Questo approccio offre un vero meccanismo di estensione dinamica delle funzionalità.

3.1 Visitor per la visualizzazione (`QtVisitor` e `FormVisitor`)

- Ogni sottoclasse di `Media` (ad esempio `Book`, `Film` o `MusicAlbum`) implementa il metodo `accept(Visitor*)`. Quando viene invocato, la chiamata è inoltrata in modo polimorfo alla versione corretta di `visit(...)` relativa al tipo effettivo dell'oggetto.

- In questo modo, la GUI può generare *widget* specifici per ciascun tipo di media senza dover utilizzare `if` o `cast`. Tutto avviene in modo automatico, grazie al *dynamic dispatch* gestito da `accept(...)`.

3.2 Visitor per la serializzazione (`XMLVisitor`)

- Il pattern *Visitor* è impiegato anche per la conversione in XML. Ogni classe concreta (`Book`, `Film`, `MusicAlbum`) fornisce la propria implementazione di `visit(...)` in `XMLVisitor`, creando un nodo XML con gli attributi specifici (ad esempio, `<Book author="..." plot="...">`).

- In questo modo, la logica di salvataggio/caricamento non "inquina" le classi del modello (che restano focalizzate sui dati e sul loro comportamento), ma risiede in una classe specializzata (`XMLSerializer` / `XMLVisitor`) che "visita" ogni `Media` e ne genera la struttura XML corrispondente.

3.3 Polimorfismo nel metodo `matches(...)`

- Ogni sottoclasse di `Media` ridefinisce `matches(const QString&)` in modo aderente ai propri campi:

  - `Film` controlla corrispondenze ad esempio su `title`.
  - `Book` ricerca corrispondenze in `title`, `author`.
  - `MusicAlbum` cerca in `title`, `artist`.

- Da un punto di vista esterno, però, basta chiamare `media->matches("testo")` e sarà il polimorfismo a selezionare automaticamente il comportamento corretto in base al tipo concreto.

3.4 Estendibilità

- Questa architettura permette di aggiungere o modificare tipologie di media senza dover stravolgere o duplicare il codice esistente, sfruttando appieno il polimorfismo.

Questo livello di polimorfismo risulta "non banale" poiché non si limita a piccoli adattamenti, ma incide in modo rilevante sul design complessivo, modellazione dei media vs. logiche di interfaccia e persistenza.

4 Persistenza dei dati

Bibliotech usa il formato XML per la persistenza dei dati, salvando l'intera collezione di media (libri, film e album musicali) in un unico file per ogni biblioteca. Questa scelta semplifica il processo di caricamento e garantisce che l'applicazione possa recuperare lo stato esatto della collezione.
La serializzazione è implementata da due componenti principali:

4.1 XMLSerializer

`XMLSerializer` fornisce i metodi `save(...)` e `load(...)` per salvare/leggere l'oggetto `Biblioteca` da un file XML. Durante la fase di salvataggio, `XMLSerializer` genera una struttura `<Biblioteca>` contenente, come sotto-nodi, elementi `<Book>`, `<Film>` o `<MusicAlbum>` che rappresentano i singoli media.
Gli attributi specifici, vengono salvati come attributi XML,permettodno così uan chiara separazione dei campi.

4.2 XMLVisitor

Le fasi sopra esposte avvengono tramite il *pattern Visitor*. Invocando `accept(...)` su ogni elemento, `XMLSerializer` delega a `XMLVisitor` la creazione e l'inserimento degli elementi XML specifici:

- `<Book title="..." author="..." plot="..." ...>`
- `<Film title="..." director="..." duration="..." ...>`
- `<MusicAlbum title="..." artist="..." trackCount="..." ...>`

4.3 Esempio di file XML

Nel repository è presente un file dimostrativo, `dump.xml`, che mostra la struttura XML generata:

```
<Biblioteca>

<!--  Libro di informatica (esempio)  -->
<Book title="Programmare in C++" publisher="Tech Books" year="2022" author="Mario Rossi"

<!--  Film: Il Gladiatore  -->
<Film title="Il Gladiatore" publisher="DreamWorks - Riedizione Speciale" year="2021" dir

<!--  Album di Sfera Ebbasta (Famoso, 2020)  -->
<MusicAlbum title="Famoso" publisher="Island Records" year="2020" artist="Sfera Ebbasta"

</Biblioteca>
```

Questo permette al pattern "Visitor" e sull'utilizzo di una classe XMLSerializer, di rendere la persistenza indipendente dal modello.

5 Funzionalità implementate

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

5.1 Funzionalità funzionali

1. **Gestione di più tipologie di media**

   L'applicazione consente di aggiungere, modificare e rimuovere diverse categorie di oggetti multimediali: *Book*, *Film* e *MusicAlbum*, ognuno con attributi specifici.

2. **Ricerca testuale**

   Tramite il metodo `search(...)`, l'utente può cercare: l'autore nei libri, il regista nei film e l'artista negli album musicali. La ricerca avviene tramite un campo di testo (`QLineEdit`) posizionato nell'interfaccia principale. Man mano che l'utente digita (o clicca il pulsante "Reset" per svuotare il filtro), `Biblioteca` viene interrogata e la lista dei media è aggiornata in tempo reale.

3. **Caricamento e salvataggio**

   La classe `XMLSerializer` si occupa di importare i dati in un file XML, sfruttando `XMLVisitor` per serializzare in modo specializzato libri, film e album musicali.

   - *Salvataggio (`save(...)`)*:
     Viene passato un oggetto `Biblioteca` da salvare. Per ogni media in archivio, si chiama `media->accept(&mVisitor)`, così che `XMLVisitor` crei un nodo XML dedicato (ad es. `<Book title="..." author="...">`).
   - *Caricamento (`load(...)`)*:
     Viene letto il file XML, cercando i nodi `<Book>`, `<Film>` e `<MusicAlbum>`. In base al tag, `XMLSerializer` istanzia la relativa classe concreta, ne assegna i campi con i valori estratti e aggiunge il nuovo oggetto alla collezione gestita da `Biblioteca`.

4. **Meccanismo di notifica e aggiornamento**

   Grazie all'uso del pattern *Observer*, eventuali modifiche alla collezione (o ai singoli `Media`) è automaticamente notificata alla *View*. Ciò semplifica il refresh dell'interfaccia: quando si aggiunge, elimina o modifica un media, la lista visualizzata viene aggiornata senza ulteriori interventi manuali.

5.2 Funzionalità grafiche

1. **Toolbar in alto**
   L'interfaccia (derivata da `QMainWindow`) comprende una barra degli strumenti con pulsanti dedicati: "Load", "Save", "Add Book", "Add Film", "Add Music". L'utente ha così a disposizione, in maniera semplice e intuitiva, le funzioni principali del programma.

2. **Campo di ricerca inline**
   In Bibliotech è presente una barra di ricerca inserita direttamente nella finestra principale. L'utente può digitare liberamente per filtrare in tempo reale i risultati, o cliccare su *Reset* per rimuovere il filtro e mostrare di nuovo tutti i media presenti.

3. **Form di modifica dedicati**
   Quando si sceglie di apportare delle modifiche a un media, viene istanziato un `FormVisitor` che crea un widget di editing specifico per il tipo di oggetto: libro, film o album musicale. Questa finestra raccoglie i dati dall'utente e aggiorna l'oggetto solo una volta salvato. E' in oltre possbile annuallare l'inserimento premendo: "Cancel".

4. **Widget specifici per ogni media**
   `QtVisitor`, si occupa di generare un widget (`MediaWidget`) per ciascun oggetto nella lista principale: qui compaiono i pulsanti *Delete* ed *Edit*, assieme a un layout che visualizza le informazioni. In questo modo, la GUI risulta dinamica.

5. **Dialog di apertura e salvataggio**
   Per aprire e salvare i file `.xml`, l'applicazione utilizza `QFileDialog`, per selezionare i file .xml, evitando così percorsi rigidi nel codice. Inoltre il formato XML è intuitivo da ispezioanre con un qualsiasi editor.

6 Differenze rispetto alla consegna precedente

Rispetto alla consegna precedente, sulla base dei feedback ricevuti, ho risolto il problema della ricerca in tempo reale, inserendo una barra sotto al menù. Ho aggiornato la finestra di modifica, sistemando così la problematica che non permettva di modficare tutti i campi e dall'attuale versione non vengono più creati oggetti vuoti. Sitemato l'Observer come suggerito e sono state apportare modifiche dal punto di vista grafico, introducendo la scroolbar e le icone (file .svg, file style.qss e resources per gestire i percorsi delle icone).

7 Rendicontazione 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.

Attività	**Ore Previste**	**Ore Effettive**
Studio e progettazione	8	10
Sviluppo del codice del modello	8	11
Studio del framework Qt	10	10
Sviluppo del codice della GUI	10	13
Test e debug	2	2
Stesura della relazione	2	2
Totale	**40**	**48**

Tabella 1: Tabella di rendicontazione delle ore