Design Engineering

Maxim Makhovskyy, Nicola Liguori, Luca Lanzetta, Marco Orefice

December 15, 2024

Contents

1 Introduzione

Il seguente documento illustra l'architettura e il design del progetto di Ingegneria del Software proposto.

1.1 Descrizione generale

Il documento rappresenta un punto di riferimento per lo sviluppo e la comprensione del sistema, descrivendo le componenti chiave, le interazioni e le decisioni progettuali adottate. In particolare, vengono forniti: Diagrammi delle classi, che descrivono la struttura statica del sistema, evidenziando le relazioni tra le entità e la loro organizzazione. Diagrammi di sequenza, che illustrano le interazioni dinamiche tra gli oggetti nelle operazioni più significative, evidenziando i flussi di controllo e di dati. Diagramma dei package, che aiuta a comprendere rapidamente la struttura modulare del sistema. Commenti sui diagrammi con una discussione delle scelte progettuali effettuate, valutate in termini di: coesione e accoppiamento.

La documentazione ha l'obiettivo di rendere chiaro il disegno del sistema garantendo un'implementazione che sia allineata ai requisiti funzionali e non funzionali stabiliti in precedenza.

2 Design architetturale

L'architettura di riferimento del progetto è il Model-View-Controller, un pattern di progettazione utilizzato in molte applicazioni web e desktop. La nostra scelta sull'architettura ha le seguenti motivazioni:

- JavaFX, l'applicazione si basa su JavaFX che è progettata su MVC. Continuare a seguire la stessa filosofia rende più semplice la coesione tra i vari componenti software.
- Separation of concerns, uno dei principi di buona progettazione è la separazione delle preoccupazioni, ossia aspetti diversi del sistema devono essere gestiti da moduli distinti e non sovrapposti; il Model-View-Controller è fondato su questo principio.
- Testing semplificato, l'architettura Model-View-Controller facilita il testing unitario. I modelli possono essere testati indipendentemente dai controller e dalle viste, migliorando la qualità del codice.

3 Diagramma dei package

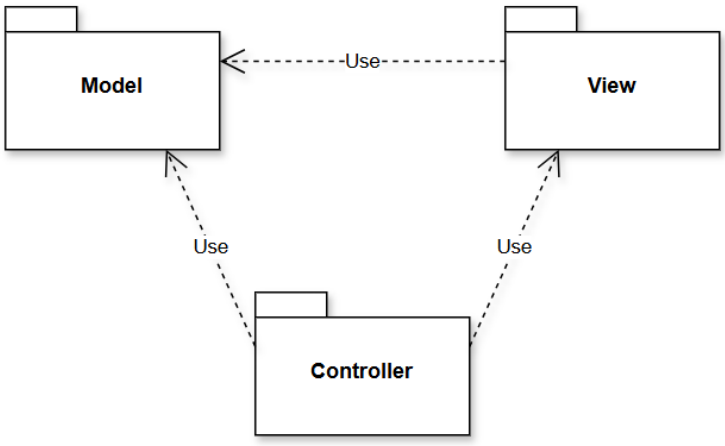

Figure 1: Package diagram

- **Model package**
  - Interfaccia AddressBook
  - Classe AddressBookModel
  - Classe FavouriteAddressBook
  - Interfaccia ExportFileStrategy
  - Interfaccia ImportFileStrategy
  - Classi Import*
    - AsCSV
    - AsJSON
  - Classi Export*
    - AsCSV
    - AsJSON
  - Classe FileManagerModel
  - Classe Component

4 Diagrammi delle classi

4.1 Model diagram

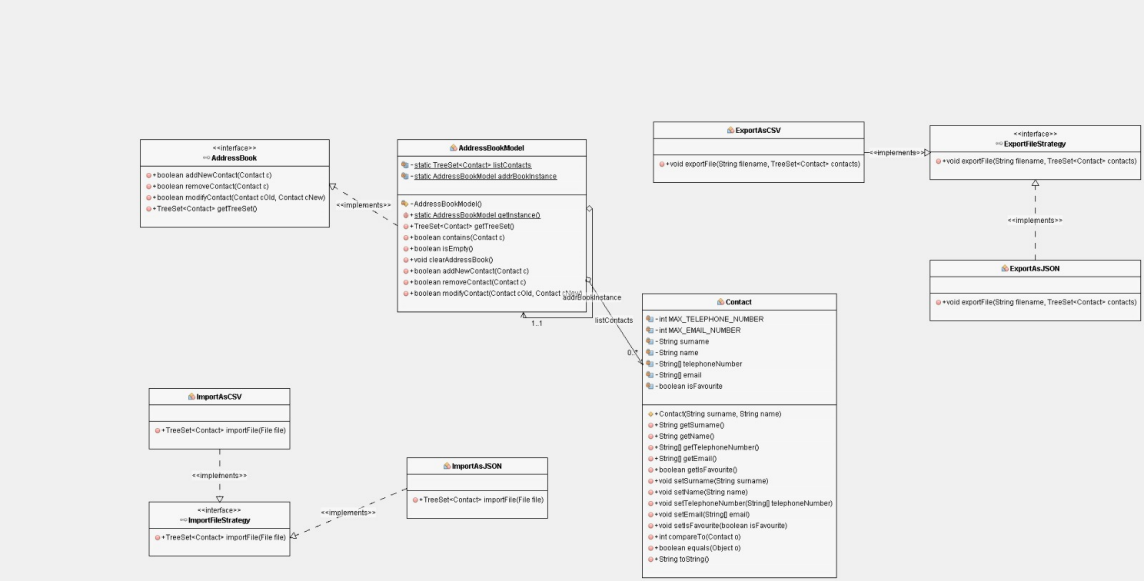

Figure 2: Model diagram

4.2 Commento sui principi di buona progettazione

Il nostro obiettivo primario per la fase di design è stato quello di seguire sin dall'inizio i principi di buona progettazione per lo sviluppo della nostra applicazione. Per semplificare il processo, abbiamo cercato di implementare alcuni design pattern comuni. In questo diagramma si può notare già il primo, ossia il design pattern "Strategy" per la selezione del tipo di file da poter importare ed esportare. Vantaggi del pattern strategy;

- Isolamento delle operazioni di importazione ed esportazione dal resto del sistema
- Open/Close principle, ossia un componente software deve essere sempre disponibile all'estensione, ma chiuso alle modifiche. Il pattern strategy utilizzato rispecchia questo principio.

Per quanto riguarda la classe Contact non c'è nulla da aggiungere, definisce e implementa i metodi necessari per la gestione dei contatti. La classe AddressBookModel è responsabile di tutte le operazioni base per la gestione dei contatti. L'AddressBookModel utilizza una treeSet per poter effettuare le operazioni di aggiunta, rimozione e ricerca di un contatto con una complessità computazionale di tipo logaritmica. La maggior parte dei metodi da implementare riutilizza i metodi già messi a disposizione dalla collezione TreeSet.

4.2.1 Interfaccia AddressBook

L'interfaccia AddressBook definisce il contratto per la gestione dei contatti. Definisce

- addNewContact(): boolean
- removeContact(): boolean
- modifyContact(): boolean

4.2.2 Classe AddressBookModel

Class	AddressBookModel
Descrizione	La classe "AddressBookModel" implementa l'interfaccia "AddressBook", sovrascrivendo i metodi "addNewContact()", "removeContact()", e "modifyContact()". Gestisce i contatti utilizzando una collezione di tipo "TreeSet".
Attributi	- **listContacts**: Collection ⟨ Contact ⟩ Collezione che memorizza tutti i contatti presenti nella rubrica, utilizzando una "TreeSet" che assicura l'ordinamento dei contatti.

Class	AddressBookModel
Metodi	- **AddressBookModel()**: void Questo costruttore inizializza la lista dei contatti come una struttura "TreeSet". - **contains(Contact c)**: boolean Verifica se un contatto esiste nella rubrica. - **addNewContact(Contact c)**: boolean Questo metodo permette di aggiungere un nuovo contatto nella rubrica. Restituisce "true" se il contatto non esiste già, altrimenti restituisce "false". - **removeContact(Contact c)**: boolean Questo metodo rimuove un contatto dalla rubrica. Restituisce "true" se il contatto è stato eliminato con successo. - **modifyContact(Contact cOld, Contact cNew)**: boolean Modifica un contatto esistente nella rubrica. Restituisce "true" se la modifica è andata a buon fine.

4.2.3 Classe FileManagerModel

Class	FileManagerModel
Descrizione	La classe FileManagerModel è la classe principale per la gestione delle operazioni di import ed export dei file.
Attributi	- **exportFileStrategy**: ExportFileStrategy Questo attributo rappresenta la strategia corrente di esportazione del file. - **importFileStrategy**: ImportFileStrategy Questo attributo rappresenta la strategia corrente di importazione del file.

Class	FileManagerModel
Metodi	- **FileManagerModel(ExportFileStrategy strategy)**: void Questo costruttore inizializza la strategia di esportazione con la strategia fornita. - **FileManagerModel(ImportFileStrategy strategy)**: void Questo costruttore inizializza la strategia di importazione con la strategia fornita. - **setExportFileStrategy(ExportFileStrategy exportFileStrategy)**: void Questo metodo imposta la strategia di esportazione del file. - **setImportFileStrategy(ImportFileStrategy importFileStrategy)**: void Questo metodo imposta la strategia di importazione del file.

4.2.4 Interfaccia ExportFileStrategy

Interface	**ExportFileStrategy**
Descrizione	L'interfaccia ExportFileStrategy definisce il metodo di esportazione di un file con contatti.
Metodi	- **exportFile(filename: String, contacts: TreeSet⟨Contact⟩)**: void Questo metodo esporta i contatti in un file specificato dal percorso filename.
Implementazioni	- **ExportCSVStrategy** Esporta i contatti in un file CSV. - **ExportJSONStrategy** Esporta i contatti in un file JSON.

4.2.5 Interfaccia ImportFileStrategy

Interface	**ImportFileStrategy**
Descrizione	L'interfaccia ImportFileStrategy definisce il metodo di importazione di un file contenente contatti.
Metodi	- **importFile(filename: String)**: TreeSet⟨Contact⟩ Questo metodo importa i contatti da un file specificato dal percorso filename e restituisce un TreeSet contenente i contatti importati.
Implementazioni	- **ImportCSVStrategy** Importa i contatti da un file CSV. - **ImportJSONStrategy** Importa i contatti da un file JSON.

4.3 Controller Diagram

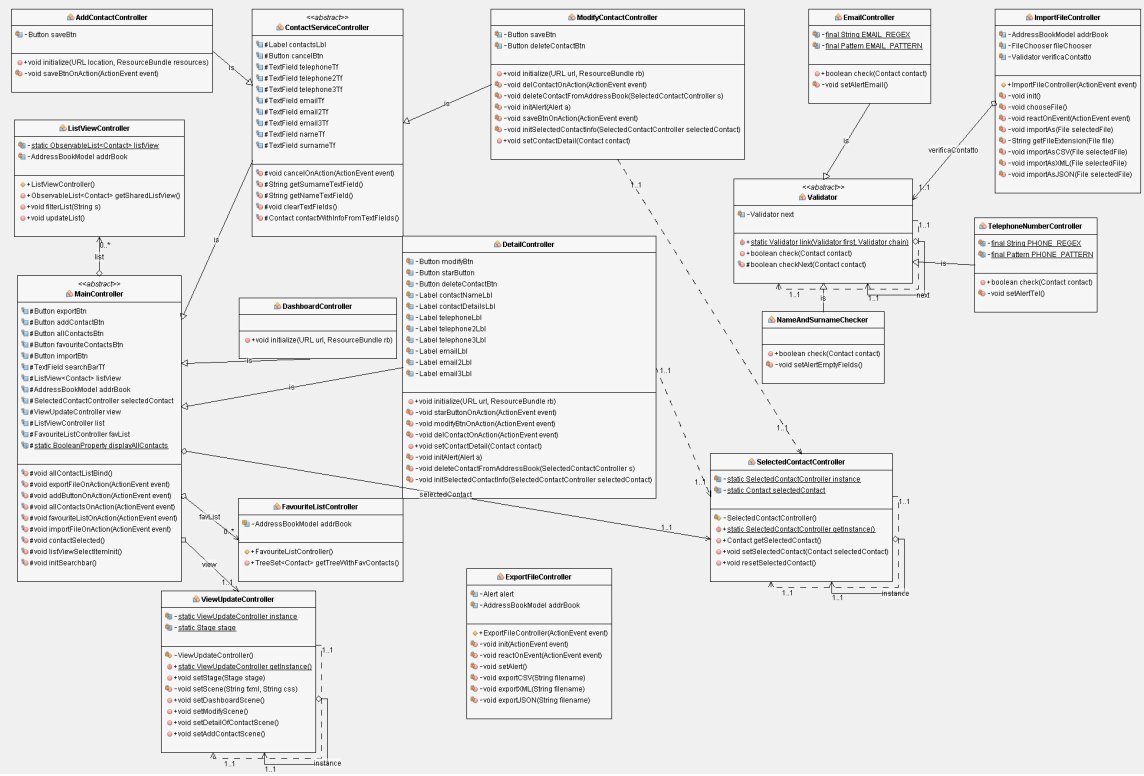

Figure 3: Controller diagram

4.3.1 Panoramica generale e sezioni del diagramma

Il diagramma rappresenta l'architettura che gestisce le funzionalità legate ai contatti (come aggiungere, modificare, eliminare e visualizzare).

4.3.2 Coesione

- Alta Coesione nelle Classi di Controllo

  Le classi AddandModifyController, DashboardController, e DetailController mostrano alta coesione poiché ognuna è responsabile di un compito specifico legato alla gestione dei contatti. Ad esempio, AddandModifyController si occupa esclusivamente dell'aggiunta e modifica dei contatti, mentre DetailController gestisce la visualizzazione dei dettagli.

- Alta Coesione nei Validatori

  Le classi FormatController e MaxTextLengthController hanno alta coesione poiché ciascuna esegue un tipo specifico di validazione. FormatController gestisce la formattazione, mentre MaxTextLengthController verifica la lunghezza dei testi.

4.3.3 Accoppiamento

- Ridotto Accoppiamento tramite Chain of Responsibility

  L'uso del Pattern Chain of Responsibility nelle classi Validator, FormatController e MaxTextLengthController riduce l'accoppiamento tra i componenti del sistema. Questo pattern permette di collegare una serie di oggetti validatori in una catena, dove ciascuno può gestire una richiesta o passarla al successivo nella catena. Le classi di controllo (AddandModifyController, DashboardController, DetailController) non devono conoscere i dettagli di implementazione dei validatori, riducendo così l'accoppiamento.

4.3.4 Pattern Chain of Responsibility

Il Pattern Chain of Responsibility è stato utilizzato per favorire la coesione e ridurre l'accoppiamento.

- Implementazione del Pattern

  La classe astratta Validator definisce un'interfaccia per la validazione dei contatti. Le classi FormatController e MaxTextLengthController estendono Validator e implementano il metodo check per eseguire specifiche validazioni sui contatti.

- Funzionamento del Pattern

  La classe Validator contiene un riferimento a un altro Validator, permettendo di creare una catena di validatori. Quando viene richiesta una validazione, ogni Validator nella catena può processare la richiesta o passarla al successivo. Questo permette di aggiungere nuovi tipi di validazione senza modificare le classi di controllo, migliorando la coesione e riducendo l'accoppiamento.

4.3.5 Classe Validator

Class	Validator
Descrizione	La classe astratta Validator implementa il pattern Chain of Responsibility per la validazione dei contatti da inserire in una rubrica.
Metodi	- **link(first: Validator, chain: Validator...)**: Validator Questo metodo crea la catena di validatori, restituendo il primo validatore della sequenza. - **check(contact: Contact)**: boolean Questo metodo è astratto e deve essere implementato dalle classi derivate per validare un contatto. - **checkNext(contact: Contact)**: boolean Questo metodo permette di invocare il prossimo validatore della catena per continuare la validazione del contatto.
Attributi	- **next** : Validator Riferimento al prossimo validatore nella catena.

4.3.6 Classe MaxTextLengthController

Class	MaxTextLengthController
Descrizione	La classe MaxTextLengthController effettua il controllo del formato testuale dei nomi e dei cognomi dei contatti.
Estensione	Estende la classe **Validator**, implementando la logica di validazione specifica per la lunghezza massima del testo nei nomi e cognomi.
Metodi	- **check(contact: Contact)**: boolean Questo metodo implementa la logica di validazione per il controllo della lunghezza massima del testo dei nomi e dei cognomi del contatto. Se il contatto soddisfa i requisiti, restituisce true, altrimenti false.

4.3.7 Classe FormatController

Class	FormatController
Descrizione	La classe FormatController effettua il controllo del formato dei numeri telefonici e dell'email dei contatti.
Estensione	Estende la classe **Validator**, implementando la logica di validazione specifica per il formato dei numeri telefonici e degli indirizzi email.
Metodi	- **check(contact: Contact)**: boolean Questo metodo implementa la logica di validazione per il controllo del formato dei numeri telefonici e degli indirizzi email del contatto. Se il contatto soddisfa i requisiti di formato, restituisce true, altrimenti false.

5 Diagrammi di sequenza

5.1 Salvataggio contatto

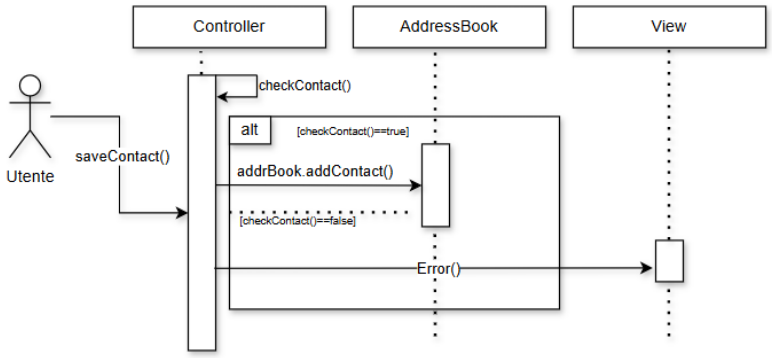

Figure 4: Sequence diagram "saveContact()"

Il diagramma di sequenza mostra l'interazione tra Utente, Controller, AddressBook View durante il processo di salvataggio di un contatto.

- Utente

  L'utente inizia l'interazione inviando il messaggio saveContact() al Controller.

- Controller

  Il Controller riceve il messaggio saveContact() e invia il messaggio checkContact() al Validator per verificare se il contatto rispetta determinati requisiti.

- Controller

  Il Controller utilizza un blocco alternativo (alt) per gestire due possibili scenari:

  1. Se checkContact() restituisce false , il Controller invia il messaggio Error() alla View per notificare un errore.

2. Se checkContact() restituisce true, il Controller invia il messaggio AddressBook.addContact() all'AddressBook per aggiungere il nuovo contatto.

- AddressBook

  L'AddressBook aggiunge il contatto se il controllo è positivo.

- View

  La View riceve il messaggio Error() e gestisce l'errore visualizzando un messaggio appropriato all'utente.

5.2 Rimozione contatto

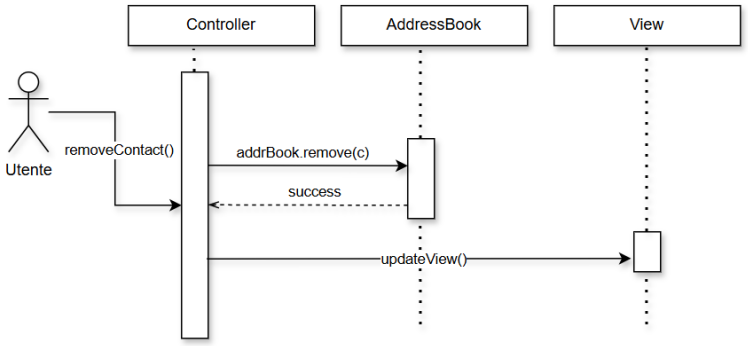

Figure 5: Sequence diagram "removeContact("

Il seguente diagramma di sequenza rappresenta l'interazione tra Utente, Controller, AddressBook View durante il processo di rimozione di un contatto.

- Utente

  L'utente invia una richiesta removeContact() al Controller.

- Controller

  Il Controller riceve la richiesta e invia addrBook.remove(c) all'AddressBook per rimuovere il contatto.

- AddressBook

  L'AddressBook esegue la rimozione e risponde con un messaggio di successo (success) al Controller.

- Controller

  Il Controller invia una richiesta updateView() alla View per aggiornare la visualizzazione.

5.3 Ricerca contatto

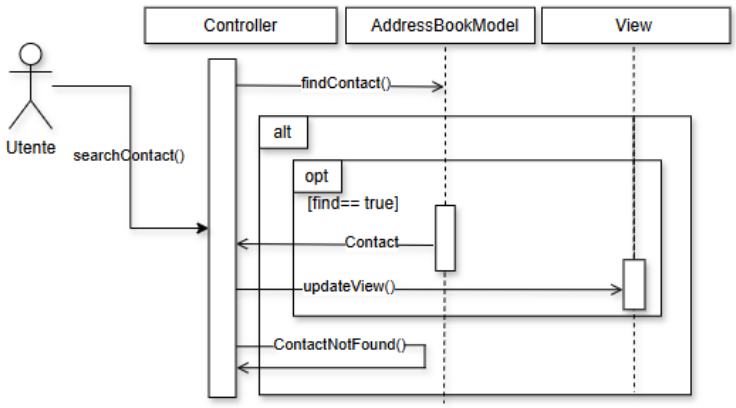

Figure 6: Sequence diagram "searchContact()"

La logica per la ricerca del contatto è ancora in fase di sviluppo, quindi questo diagramma è da considerarsi provvisorio.

Il diagramma di sequenza mostra l'interazione tra Utente, AddressBookController, AddressBookModel e AddressBookView durante la ricerca di un contatto.

- Utente

  L'utente inizia l'interazione inviando la richiesta searchContact() all'AddressBookController.

- Controller

  Il Controller inoltra una richiesta di ricerca tramite findContact() all'AddressBookModel.

- AddressBookModel

  Nel diagramma è presente un blocco alternativo (alt) con due opzioni:

  1. Se il contatto viene trovato ([find==true]) l'AddressBookModel restituisce il Contact all'AddressBookController.
  2. Se il contatto non viene trovato l'AddressBookModel invia ContactNotFound() all'AddressBookController.

- AddressBookView

  L'AdderessBookView viene aggiornata attraverso il metodo updateView() per visualizzare le informazioni del contatto all'Utente.

5.4 Importa contatto

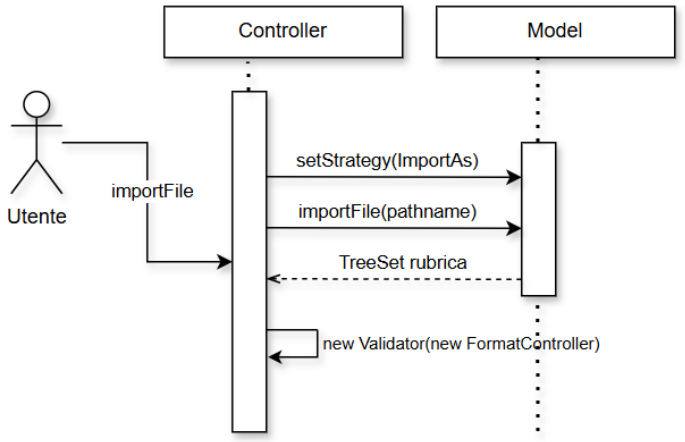

Figure 7: Sequence diagram "importFile()"

Il diagramma di sequenza mostra l'interazione tra Utente, Controller e Model durante il processo di importazione di un file.

- Utente

  L'Utente inizia l'interazione inviando un messaggio importFile al Controller.

- Controller

  Il Controller riceve il messaggio di importFile e invia un messaggio setStrategy(ImportAs) al Model per impostare la strategia di importazione. Dopodiché il Controller invia un messaggio importFile(pathname) al Model in modo da importare il file specificato dal percorso.

- Model

Il Model risponde con una TreeSet rubrica al Controller.

- Controller

  Il Controller crea un oggetto Validator passando come parametro un nuovo oggetto FormatController. Validator controlla ogni contatto della TreeSet. Per ogni contatto che non supera la fase di controllo viene rimosso dalla TreeSet. La TreeSet viene settato nel Model.

5.5 Esporta contatto

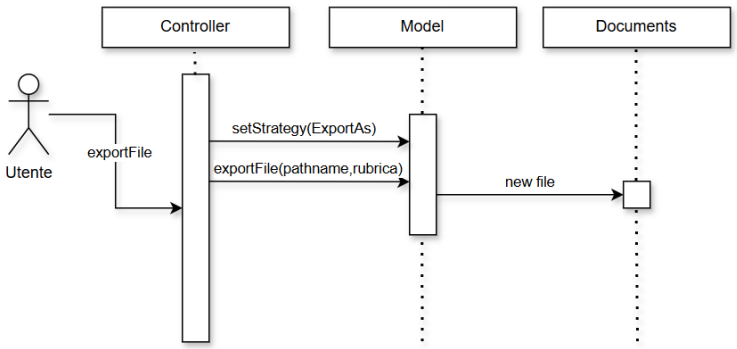

Figure 8: Sequence diagram "exportFile()"

Il diagramma di sequenza mostra l'interazione tra Utente, Controller e Model durante il processo di esportazione di un file.

- Utente

  L'Utente inizia l'interazione inviando un messaggio exortFile al Controller.

- Controller

  Il Controller riceve il messaggio di exportFile e invia un messaggio setStrategy(ExportAs) al Model per impostare la strategia di esportazione. Dopodiché il Controller invia un messaggio exportFile(pathname, rubrica) al Model in modo da importare il file specificato dal percorso con i dati della rubrica.

- Model

  Il Model dopo aver elaborato i messaggi del Controller crea un nuovo file con i dati della rubrica.

- Documents

  Una volta creato il file viene inviato a Documents per la memorizzazione.

6 Matrice di tracciabilità dei requisiti

ID	Design	Implementazione	Testing	Stato
IF-1.1	X	X	X	Design terminato
IF-1.2	X	X	X	Design terminato
IF-1.3	X	X	X	Design terminato
IF-1.4	X	X	X	Design terminato
IF-1.5	X	X	X	Design terminato
IF-1.6	X	X	X	Design terminato
IF-1.7	X	X	X	Design terminato
IF-1.8	X	X	X	Design terminato

ID	Design	Implementazione	Testing	Stato
IF-1.9	X	X	X	Da definire
IF-1.10	X	X	X	Da definire
UI-1.1	X	X	X	Implementazione terminata
UI-1.2	X	X	X	Implementazione terminata
UI-1.3	X	X	X	Design terminato
UI-1.4	X	X	X	Implementazione terminata
UI-1.5	X	X	X	Design terminato
UI-1.6	X	X	X	Design terminato
UI-1.7	X	X	X	Da definire
DF-1.1	X	X	X	Design terminato
DF-1.2	X	X	X	Design terminato
DF-1.3	X	X	X	Design terminato

ID	Design	Implementazione	Testing	Stato
DF-1.4	X	X	X	Design terminato
IF-2.1	X	X	X	Design terminato
IF-2.2	X	X	X	Design terminato
UI-2.1	X	X	X	Design terminato
UI-2.2	X	X	X	Design terminato