# CSC 4001 Assignment 2

# Automatic Melody Generator

Mo Fan (Leader) 115010204

Li Kengjie 115010177

Wang Junce 115010231

Ye Shuqian 115010269

Zhang Ruoqing 115010096

## Requirements

|  |  |  |
| --- | --- | --- |
| Identifier | Priority | Requirement |
| REQ1 | 5 | The generator could generate music automatically. |
| REQ2 | 5 | The user can specify the musical style, tonality and duration. |
| REQ3 | 1 | The program allows trainer to train the generation model, adjust the parameters for generations and adjust the configurations include musical style, tonality and duration. |
| REQ4 | 5 | The program could export the generated music in the MIDI format. |
| REQ5 | 4 | The program could export the generated music as audio file. |
| REQ6 | 2 | The program allows user to create music by using numbered musical notation to generate the melody. |
| REQ7 | 3 | The generator could generate music according to a piece of music. |
| REQ8 | 1 | The generator could visualize the generated music. |
| REQ9 | 3 | The program allows user import MIDI file. |
| REQ10 | 2 | The program allows user to tag the unsatisfactory part in the generated music and regenerate. |

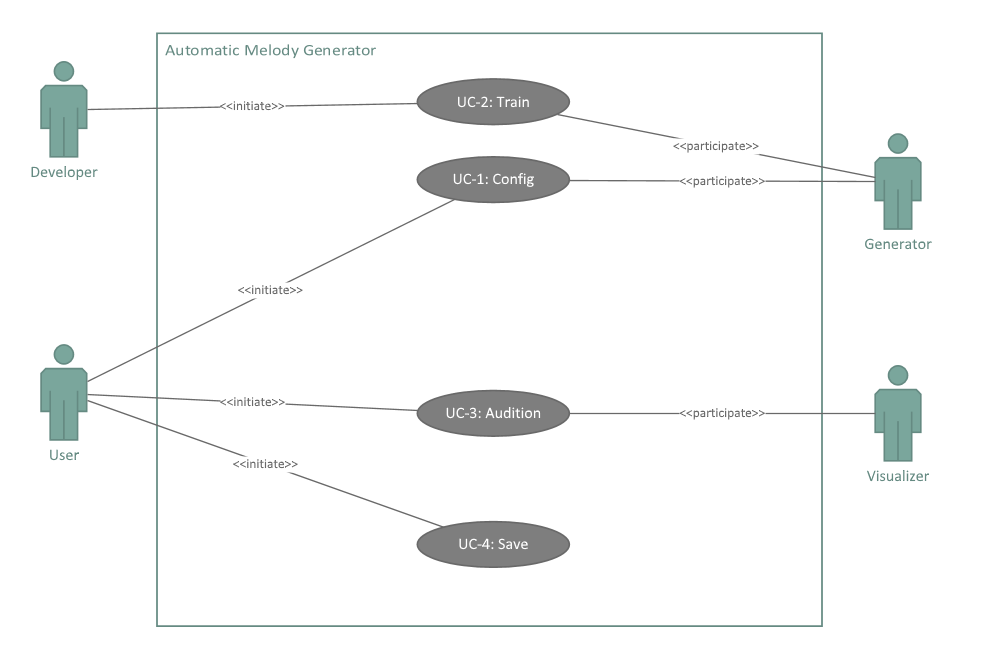
## User Stories

|  |  |  |
| --- | --- | --- |
| Identifier | User Story | Size |
| ST-1 | As a user with no music knowledge, I can generate my music in one-click operation. | 4 points |
| ST-2 | As a user with no music knowledge, I can generate music in my favorite style. | 6 points |
| ST-3 | As a user with basic music knowledge,I can generate music in my specified tornality. | 7 points |
| ST-4 | As a user with rich music knowledge, I can specify the first piece of music, and generate the whole melody with the help of the generator. | 9 points |
| ST-5 | As a user with rich music knowledge, I can audition the generated melody, mark the unsatisfactory part and let the program regenerate. | 10 points |
| ST-6 | As a trainer, I can train models, adjust the parameters and modify the configuration list. | 5 points |

## Use Cases

|  |  |  |
| --- | --- | --- |
| Actor | Actor's Goal | Use Case Name |
| User | To configure the tonality and the style so as to get the generated melody. Or to specify the first piece of music to generate melody. | Config (UC-1) |
| Developer | To train the model based on collected MIDI files, and adjust the parameters which are used to generate the melody and be added into configuration list. | Develop (UC-2) |
| Generator | To generate the melody based on the configurations. | UC-1 |
| User | To audition the generated melody and tag the unsatisfactory parts which need to be regenerated. | Audition (UC-3) |
| Visualizer | To visualize the generated melody, the current playing position and the tagged parts. | UC-3 |
| User | To save the generated melody as MIDI or audio files. | Save (UC-4) |
| Trainer | To train the model based on collected MIDI files. | UC-2 |

## Use Case Diagram



## User Case UC-1 Config

### Detailed Formula

**Related Requirements:** REQ1, RE2, REQ6, REQ7 & REQ9

**Initiating Actor:** User

**Actor's Goal:** To configure the tonality and the style so as to get the generated melody. Or to specify the first piece of music to generate melody.

**Participating Actor:** Visualizer, Generator

**Preconditions:** None.

**Postconditions:** The melody is generated according to decision the user made.

**Flow of Event for Main Success Scenario:**

→ 1. **User** selects the options from the tonality, duration and style lists.

→ 2. **System** (a) saves the choices made by the **User** and (b) sends the data to the **Generator** to generate music.

**Flow of Events for Extensions (Alternate Scenarios):**

← 1. **User** (a) clicks “Import” button to import the first piece of melody from MIDI file, or (b) notes the melody using numbered musical notations.

← 2. **Visualizer** displays the user’s input.

→ 3. **System** (a) saves the file made by the **User** and (b) sends the data to the **Generator** to generate music.

### Responsibilities

|  |  |
| --- | --- |
| Responsibility Description | Concept Name |
| Rs1. Accept choices that what the tonality, duration and the style to be. | Recorder |
| Rs2. Send the choice to Generator. | Sender |
| Rs3. Load the music data from MIDI file. | Loader |
| Rs4. Translate the MIDI data to become the numbered musical notations. | Translator |
| Rs5. Graphical interface allowing the user to edit the numbered musical notations. | Editor |

### Associations

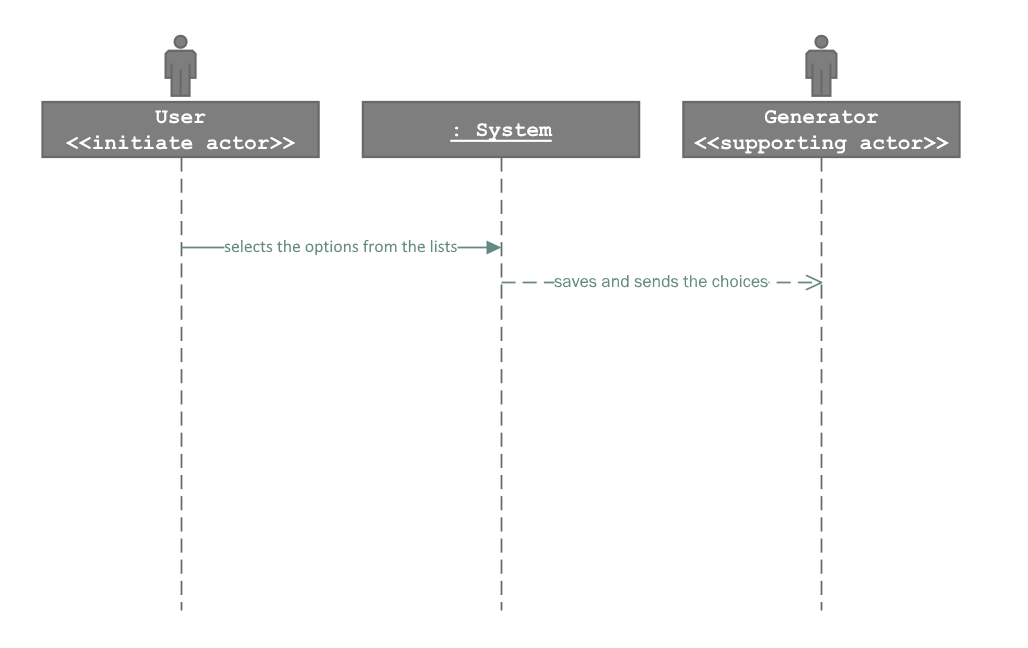
|  |  |  |
| --- | --- | --- |
| Concept Pair | Association description | Association name |
| Recorder↔Sender | Recorder passes the user’s choice to Sender to store the data. | provides data |
| Loader↔Translator | Loader passes the MIDI data to Translator. | provides MIDI data |
| Translator↔Editor | Translator passes the musical notations to Editor. | provides musical notations |
| Editor↔Sender | Editor passes the final edition MIDI data to Sender. | provides data |

### Attributes

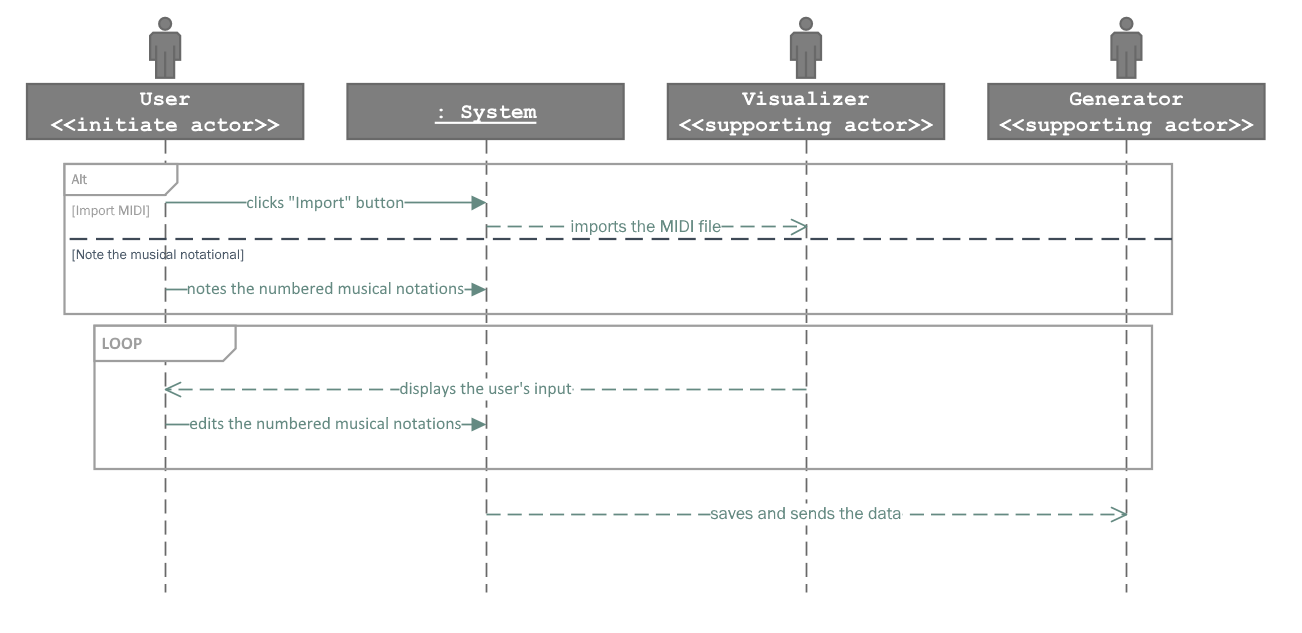
|  |  |  |
| --- | --- | --- |
| Concept | Attributes | Attribute Description |
| Recorder | default configuration | The possible tonality, duration and the musical style settings are defaulted. |
|  | archiver | User's choices are archived. |
| Sender | pass data | The configuration is passed from Recorder to Generator. |
| Loader | MIDI data | MIDI data which is loaded from files. |
| Editor | numbered musical notations | User's noted numbered musical notations. |

### System Sequence Diagram

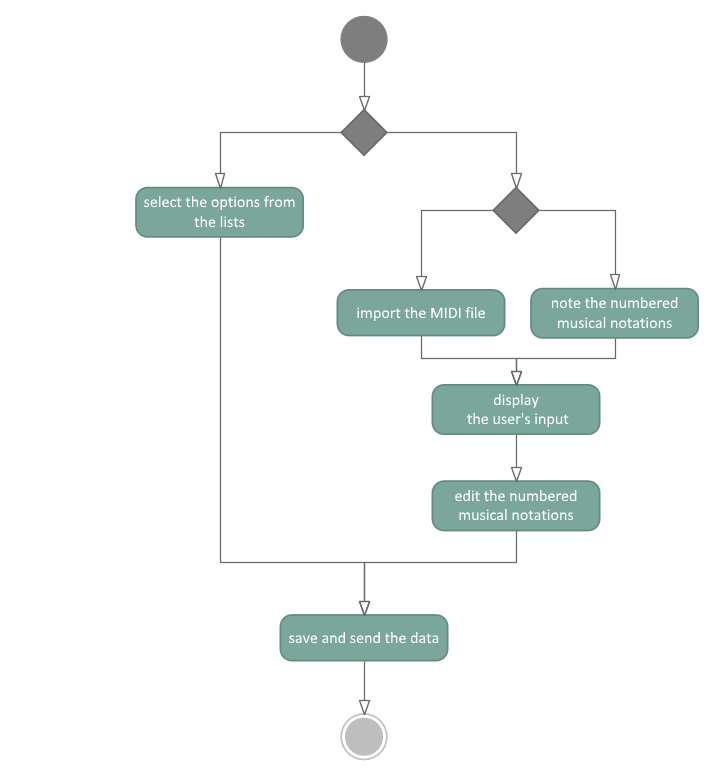
Main Success Scenario:



Alternative Scenario:



### Activity Diagram

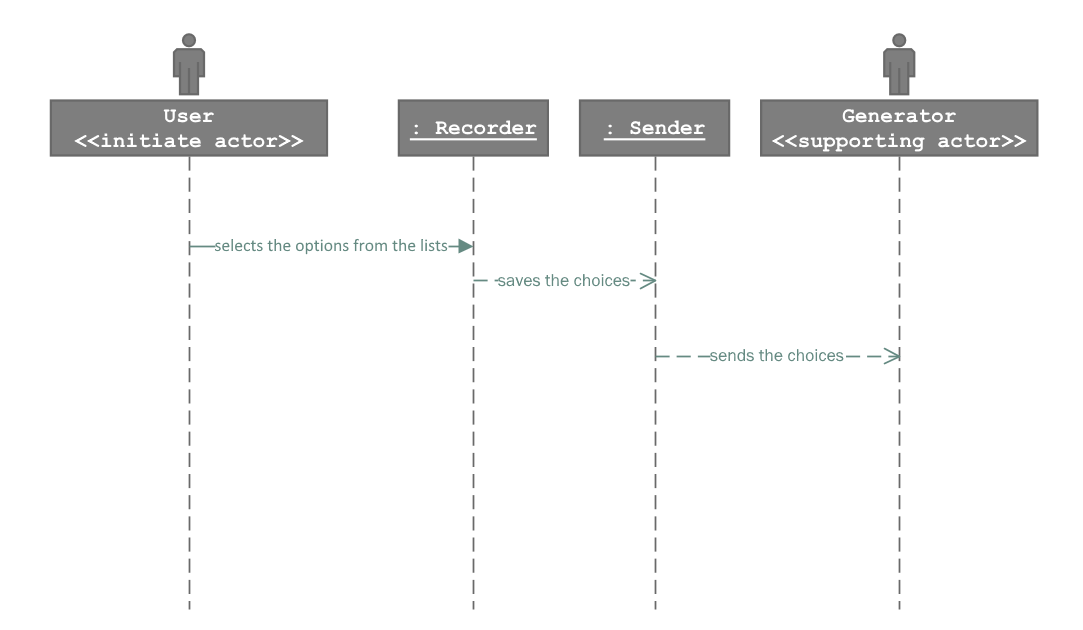


### Domain model

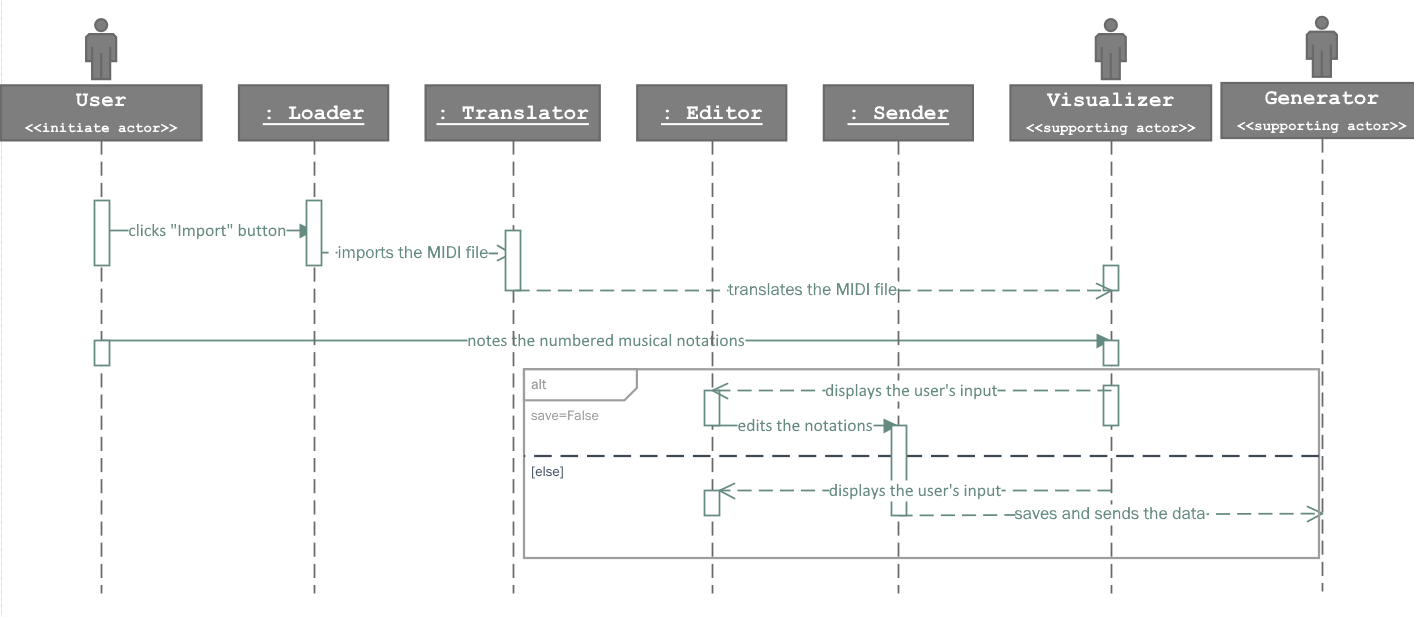
### 

### Design Sequence Diagram

Main Success Scenario:



Alternative Scenario:



## User Case UC-2 Train

### Detailed Formula

**Related Requirements:** REQ3

**Initiating Actor:** Developer

**Actor's Goal:** To train the generation model, adjust the parameters for generation and adjust the configurations including musical style, tonality and duration.

**Participating Actor:** Generator, Trainer

**Preconditions:** None.

**Postconditions:** The configuration list is updated.

**Flow of Event for Main Success Scenario:**

→ 1. **Developer** puts the MIDI files, which are used to train model, into the train set folder.

→ 2. **Developer** (a) inputs the training settings and (b) adjusts the parameters in the configuration source code.

← 3. **Trainer** (a) prints the training result, (b) generates a piece of melody for testing and (c) saves the trained model.

→ 4. **Developer** updates the configuration lists.

**Flow of Events for Extensions (Alternate Scenarios):**

2a. Trainer meets error during training.

← 1. **Trainer** prints the errors.

→ 2. **Developer** modifies the training settings.

2b. Generator cannot generate a test case normally.

← 1. **Generator** signals that it cannot generate the melody.

→ 2. **Developer** (a) modifies the training settings or (b) modifies the parameters in the configuration source code.

### Responsibilities

|  |  |
| --- | --- |
| Responsibility Description | Concept Name |
| Rs1. Receive the training settings and parameters. | Receiver |
| Rs2. Load the training MIDI files. | Loader |
| Rs3. Process the training data and adjust the model. | Processor |
| Rs4. Print the logs including training and generation information. | Printer |
| Rs5. Update the configuration lists. | Updater |

### Associations

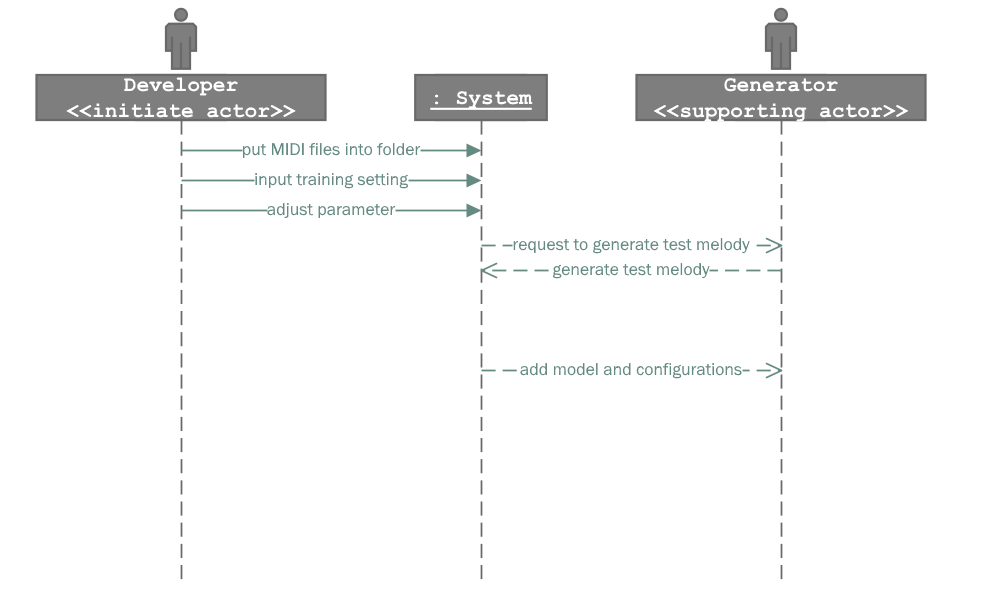
|  |  |  |
| --- | --- | --- |
| Concept Pair | Association description | Association name |
| Loader↔Processor | Loader passes the loaded files to Processor to process the training data. | provides training set |
| Receiver↔Processor | Receiver passes the received parameters and training settings to Processor to control the training process. | provides parameters |
| Processor↔Printer | Processor passes the information for training regeneration to the Printer. | provides training logs |
| Generator↔Printer | Generator passes the information for training regeneration to the Printer. | provides generation logs |
| Receiver↔Updater | Receiver passes the received configurations to Updater to update the configuration lists. | provides configurations |

### Attributes

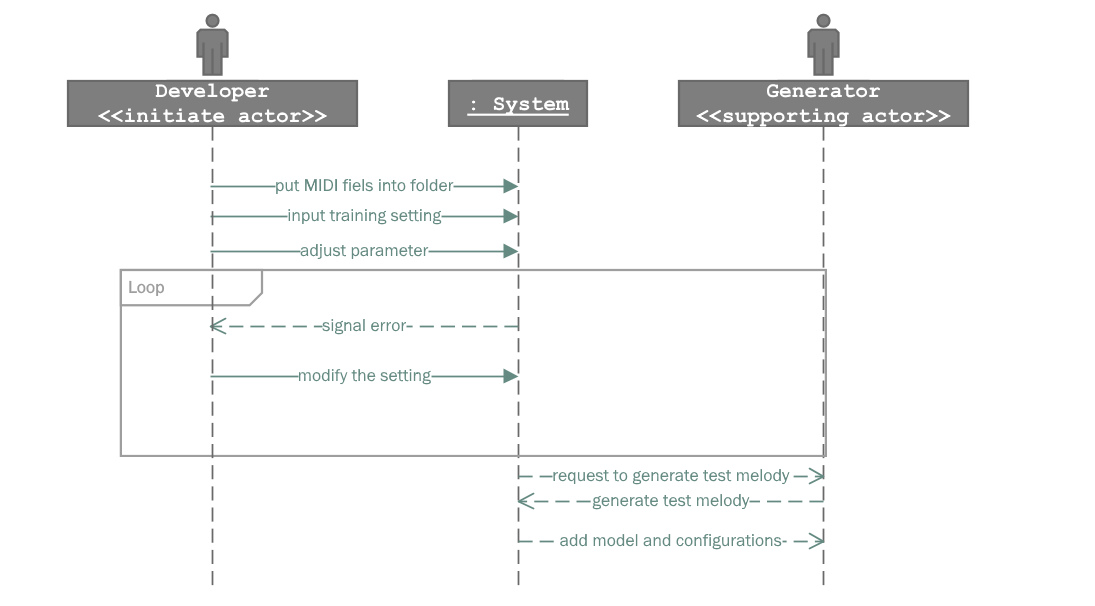
|  |  |  |
| --- | --- | --- |
| Concept | Attributes | Attribute Description |
| Processor | settings | The settings for training. |
|  | training set | The training set used to train the model. |
|  | model | The trained model used to generate melody. |
| Receiver | parameters | The inputted parameters. |
|  | settings | The settings for training. |
| Loader | dataset | The loaded MIDI dataset. |
| Printer | logs | Store the logs received from Processor and Generator. |
| Updater | configurations | Configurations received from Receiver |

### System Sequence Diagram

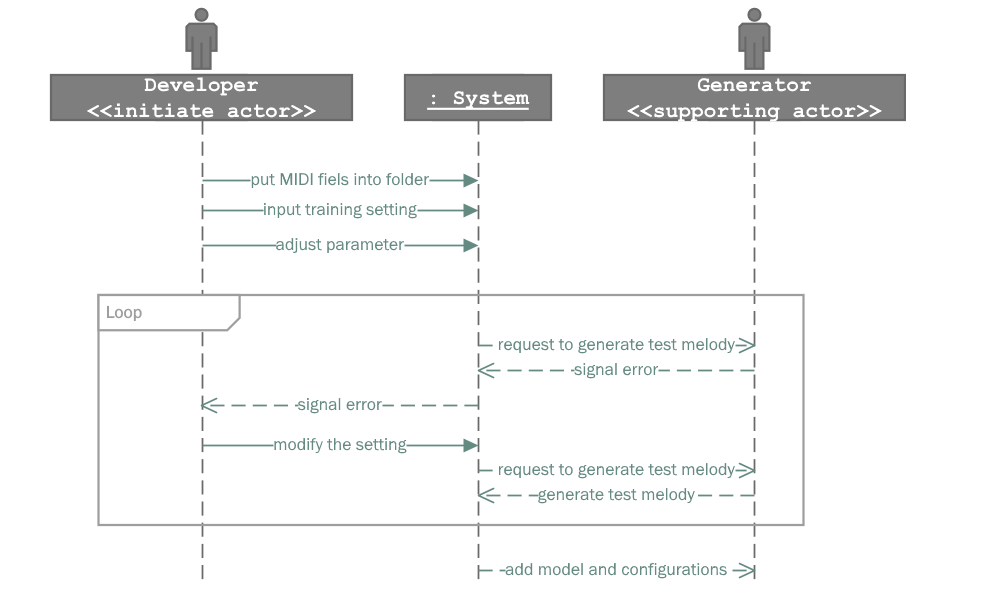
Main success scenario:



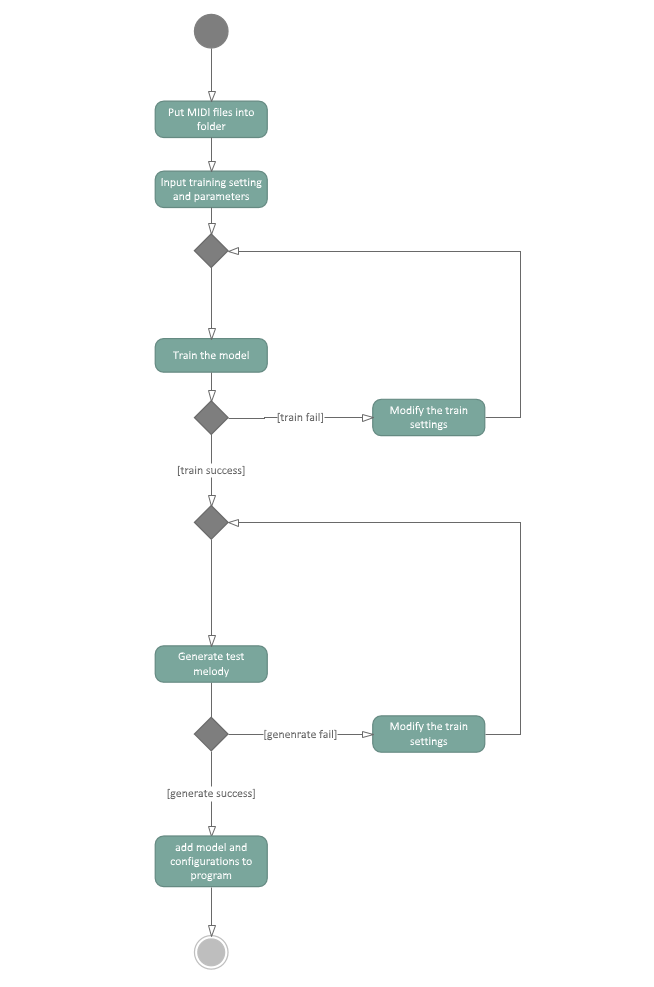
Alternative scenario (trainer meets error during training):



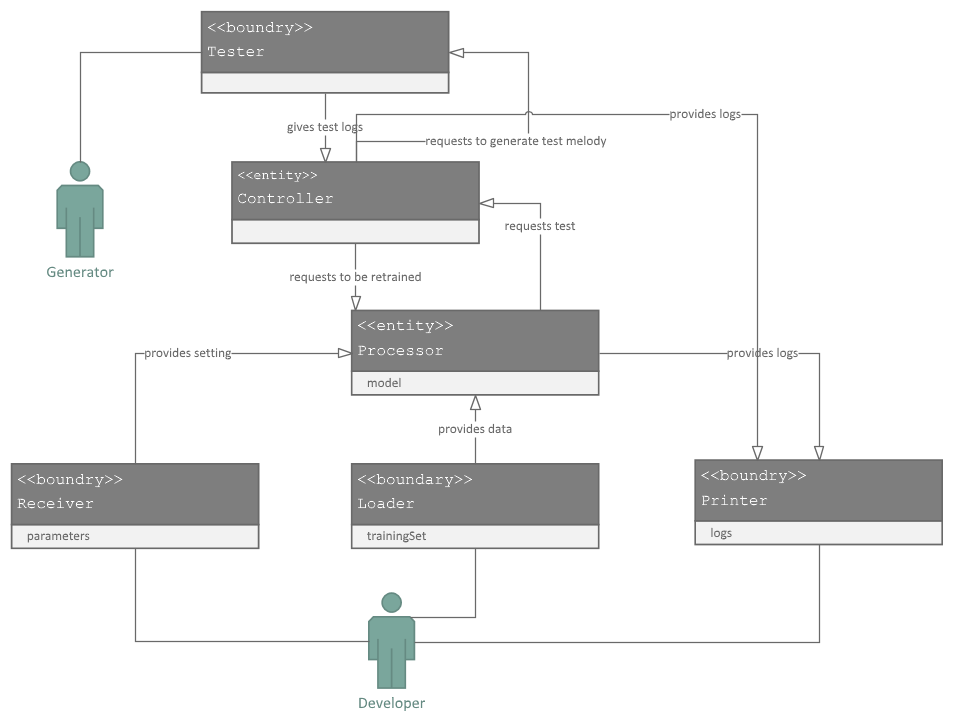
Alternative scenario (Generator cannot generate a test melody normally):



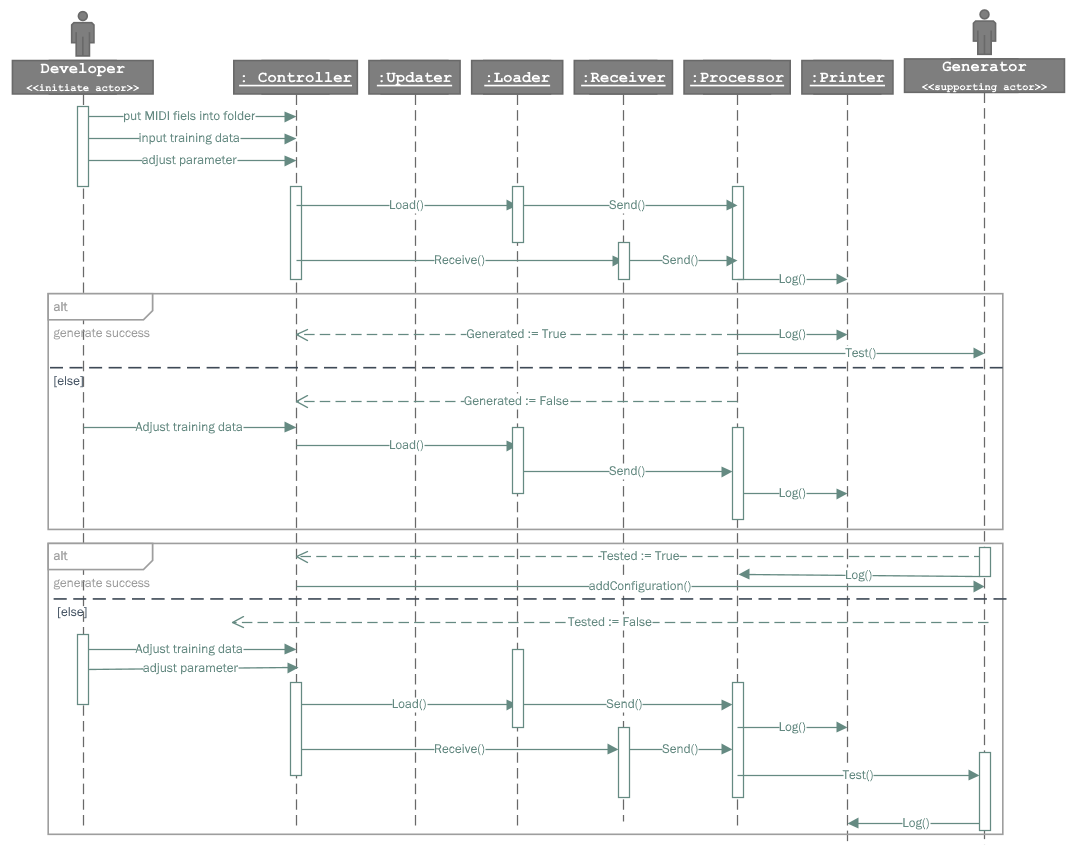
### Activity Diagram



### Domain Model



### Design Sequence Diagram



## User Case UC-3 Audition

### Detailed Formula

**Related Requirements:** REQ8 & REQ10

**Initiating Actor:** User

**Actor's Goal:** To audition the generated melody and tag the unsatisfactory part.

**Participating Actor:** Visualizer, Generator

**Preconditions:** The generator has generated a piece of melody.

**Postconditions:** The program saves the melody.

**Flow of Event for Main Success Scenario:**

→ 1. **User** clicks the button "Play/Pause".

← 2. **Visualizer** (a) plays the melody and (b) visualizes the melody in the graphic interface.

**Flow of Events for Extensions (Alternate Scenarios):**

2a. User finds out some unsatisfactory parts in the melody.

→ 1. **User** tags the unsatisfactory parts.

← 2. **Visualizer** sends the tagged information to the **Generator**.

→ 3. **Generator** regenerates the melody.

← 4. **User** (a) auditions the renegerated melody and (b) visualizes the melody.

### Responsibilities

|  |  |
| --- | --- |
| Responsibility Description | Concept Name |
| Rs1. Receive the generated melody from Generator. | Receiver |
| Rs2. Visualize the generated melody. | Visualizer |
| Rs3. Play the melody. | Player |
| Rs4. Receive the unsatisfactory tags from user. | Tag |
| Rs5. Request the Generator to regenerate the unsatisfactory part. | Regenerate Request |

### Associations

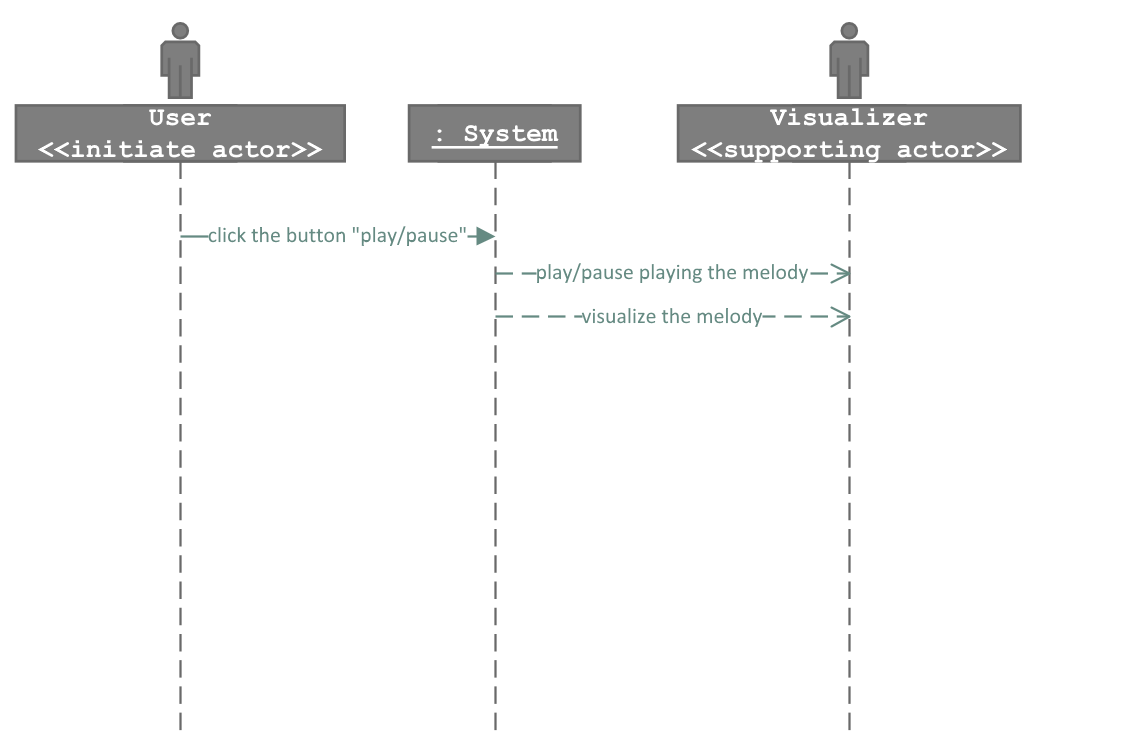
|  |  |  |
| --- | --- | --- |
| Concept Pair | Association description | Association name |
| Receiver↔Visualizer | Receiver passes the received melody to Visualizer to visualize the melody. | provides melody |
| Receiver↔Player | Receiver passes the received melody to Player to play the melody. | provides melody |
| Tag↔Regeneate Request | Tag passes the information for melody regeneration to Generator. | provides tagged data |

### Attributes

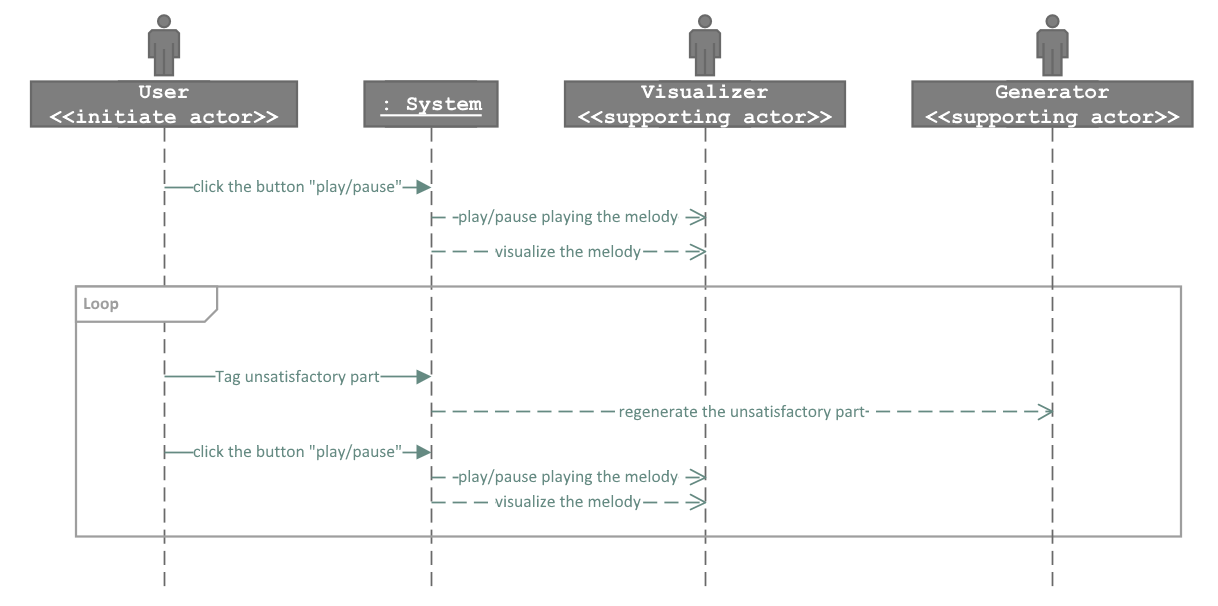
|  |  |  |
| --- | --- | --- |
| Concept | Attributes | Attribute Description |
| Receiver | config | Style, tonality and duration. |
|  | melody | The generated melody. |
| Visualizer | music score | Visualize the melody. |
|  | current moment | Display the current position in the score during playing the music. |
| Tag | tag information | Store the start and end position of each tagged part. |
| Regenerate Request | tag information | Copied from tag; send the information to Generator. |

### System Sequence Diagram

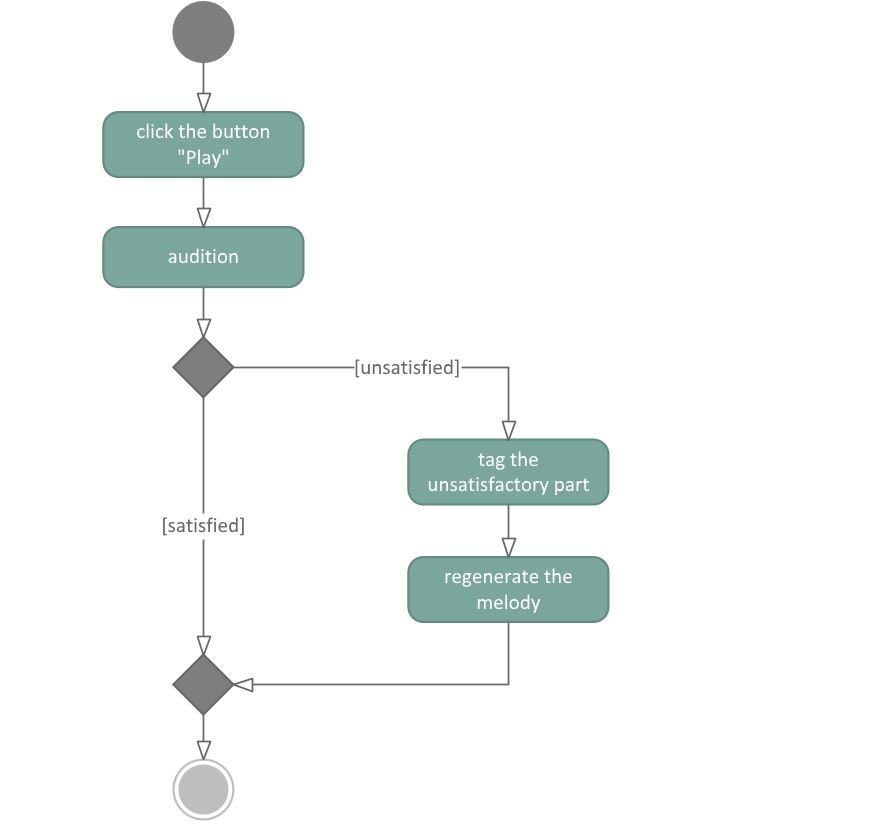
Main success scenario:



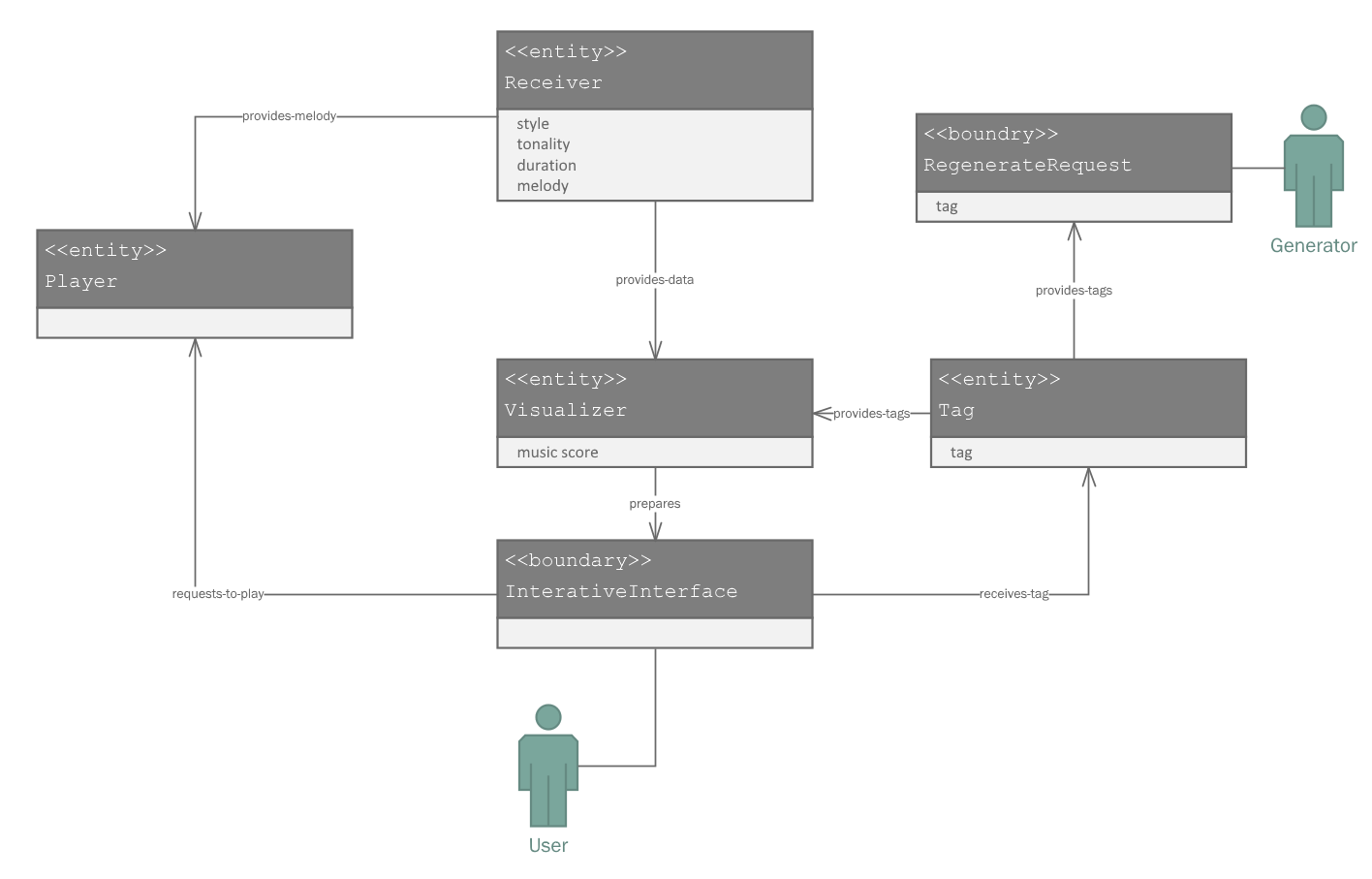
Alternative scenario (find unsatisfactory part):



### Activity Diagram



### Domain Model



### Design Sequence Diagram

