|  |
| --- |
| ECSE 321 – Introduction to Software Engineering – Group 1 |
| Use Case Model & Architecture |
| Mission 3: High Level Design |

|  |
| --- |
| Samuel Cormier-Iijima (260174995), Simon Foucher (260223197), Stefanos Koskinas (260211145), Bertin LeBlanc (260191026), Amin Mirzaee (260209556)  4/4/2009 |

# 1. Use case model

The use case model should include:

1. Use case diagrams

2. List of all actors, along with a brief description of the role of each actor and the use-cases they participate in

3. List of all use-cases with a one- or two-sentence description

4. Detailed descriptions of the use cases you feel are the most important ones. Identify at most 15 use cases you feel are the *most important* ones, and include these in the main document.

Additional use cases may be included in an appendix, but will not necessarily be graded. As studied in class and in the tutorials, each use-case description should have

a. A meaningful name

b. Participating actors

c. Entry conditions

d. Flow of events

e. Exit conditions

f. Quality requirements (if any)

g. Exceptions

## 1.1 Use Case diagrams

15 marks

**Stefanos Koskinas**

1.1.1. Diagrams should be consistent with the rest of the use case model (naming, etc.)

1.1.2. Proper use of <<include>>, <<extend>>, and inheritance relationships

## 1.2 List of ACTORS

**Installer**: This actor represents the person who will be installing the software on a PC. The installer has access to the music player’s “setup.exe” or .deb file and a PC running Windows 32 bits or Linux. The installer’s primary objective is to install the software.

**User**: The user represents any preson who will be using the player. When the user uses the player, it is already installed and configured on a computer

**Remote user**: The remote user has the program installed on his machine and wishes to share a playlist over a network with another remote user.

**IT guy**: The IT guy knows where the configuration options are on the music player and configures it to the user’s preference. He does not wish to listen to music.

**Hacker**: The hacker has the music player installed on his machine and wishes to steal playlists from other Remote Users without their consent, or regardless of the configuration of their players.

**Lawyer**: The lawyer wants to see if the player conforms to copyrights regulations and if not, sue the software developers for everything they are worth.

## 1.3 List of use cases

5 marks

## 1.4 Detailed description of 15 use cases

25 marks

**Bertin Leblanc**

1.4.1. Most important use cases are identified

1.4.2. Flow of events is clear and concise with an appropriate level of detail

1.4.3. Entry and exit conditions are clearly defined

# 2. architecture

## 2.1 system architecture and sub-systems

10 marks

**Stefanos Koskinas**

Explain the general structure of your system and the architecture(s) you decided to use. What is the rationale for your choices? What are the main subsystems and what service do they provide? Add a diagram that describes the main subsystems and the relationship between them.

2.1.1. Clearly explains the service provided by each subsystem

2.1.2. Clarifies relationship between subsystems

## 2.2 rationals for the chosen architecture

5 marks

**Samuel Cormier-Iijima**

2.2.1. Lists any tradeoffs taken into consideration

2.2.2. Lists advantages and disadvantages of the architecture chosen

# 3. static model

In this part you should describe the preliminary (system-level) static structure of the system. Focus on finding the main classes in the system, their attributes, and their associations with other classes. **Note:** There is no need to identify and specify the methods of the classes at this stage. Please perform domain analysis and identify classes that will be used in the system. Use the requirements and use-case documents as an aid. When discussing GUI classes, keep the level of details manageable. (There is no need to describe scroll bars or label text locations.) For this sub-task you should submit the following:

1. List of main classes. For each class you should include:

a. A meaningful name

b. Role – describe what it does

c. Important attributes and methods

d. Backward traceability – reference to the use case or requirements that were used to come up with the class

2. Class diagrams. Describe the classes used in the system and the relationship between them. Remember, at this point, this is not a detailed design. No need for design patterns or other forms of reuse at this point.

Prepare tidy and clear diagrams. Messy or unreadable diagrams will be penalized. Pay attention to associations between classes, and multiplicity. Try not to have too many classes; the focus at this point is on major classes so you do not need to include smaller classes (e.g., Date). Follow the instructions that were taught in class and in the tutorials.

## 3.1 list of main classes

10 marks

**Amin Mirzaee**

3.1.1. Only important, meaningful classes are included

3.1.2. Each class is properly named

3.1.3. The role of each class is given

3.1.4. Traceability – references are given to use cases that involve the class

## 3.2 classe diagrams

10 marks

**Amin Mirzaee**

3.2.1. Diagrams are clear, correct, and consistent with the rest of the document

3.2.2. Aggregation, composition, and generalization relationships are used properly.

# 4. requirement tracability matrix

For every requirement listed in your requirement specification document, show how it will be implemented in use cases and in classes. That is, for every requirement (or group of requirements, if convenient), you must ensure that it is implemented by some class, and that it appears in a use-case (for functional requirements). Include a table where each column is a requirement, each row is a use case or class. For each column, indicate with marks in the appropriate cells which use-case(s) address that requirement, which classes are involved in its implementation. This will later be a useful tool for testing use cases, and verifying that all of the requirements have been properly implemented by your system.

## 4.1 tracability matrix

15 marks

**Simon Foucher (Link Use Cases to requirements)**

**Samuel Cormier-Iijima (Link use cases to classes)**

4.1.1. Complete – includes all use cases from part A, all classes from part C, and all requirements as listed in the most recent version of the requirements spec document.

4.1.2. Comprehensive – every requirement and class is illustrated or used in at least one use-case.