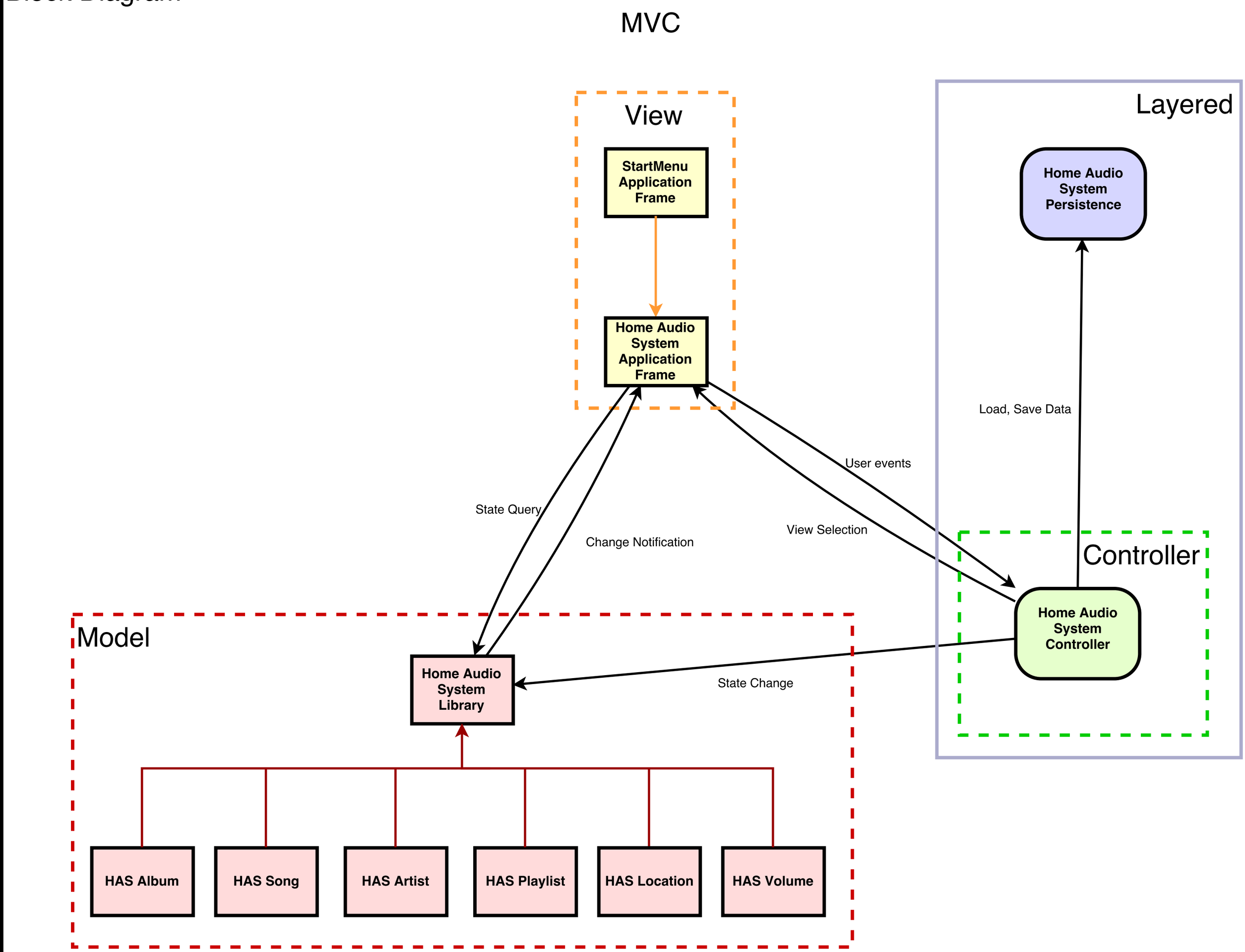
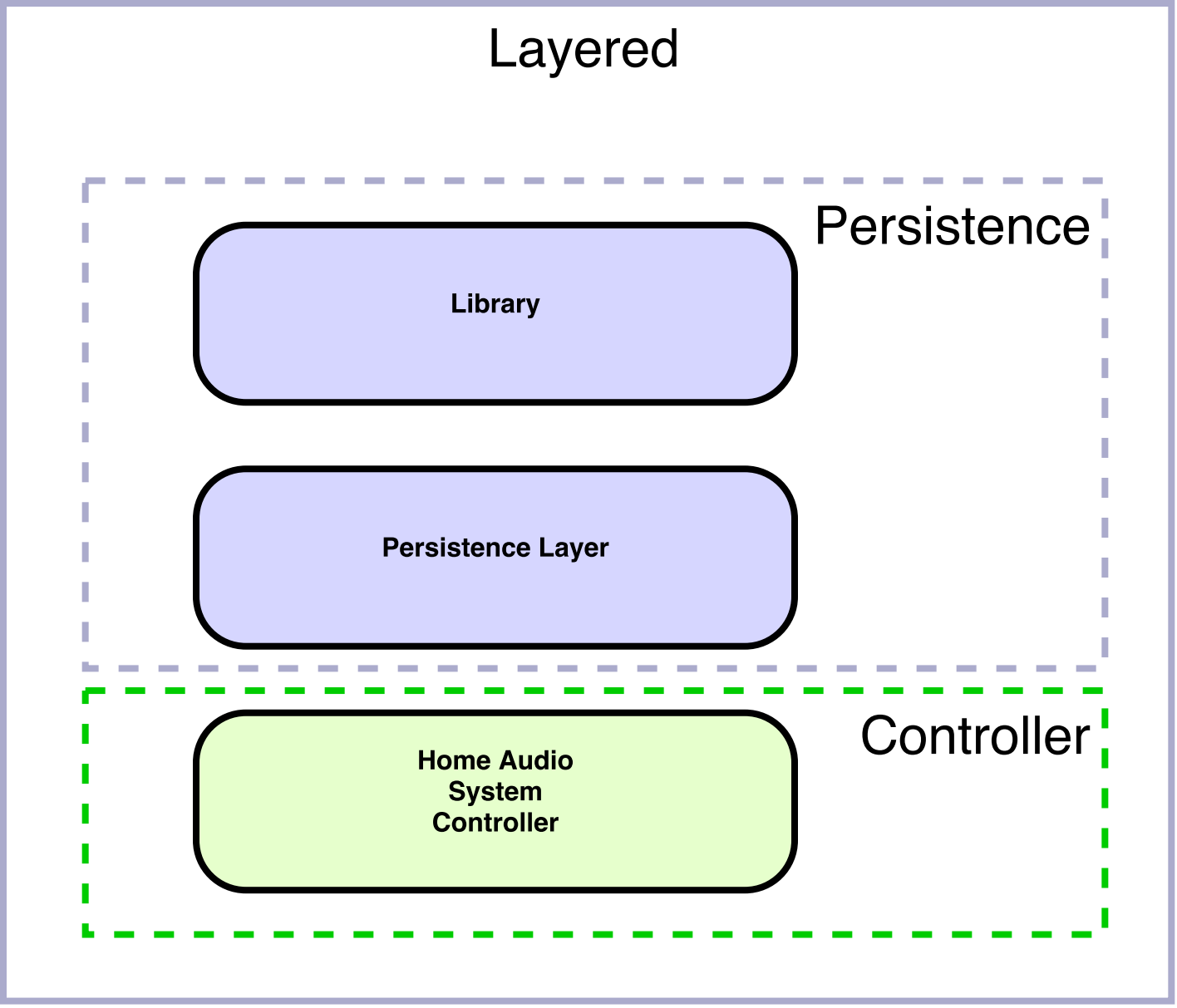
Block Diagram




Layered
Persistence
Library
Persistence Layer
Controller
Home Audio
System
Controller

Block Diagram Description

Our Architecture and Why We Chose It

In our architecture, the Home Audio System application presents a Model View Controller (MVC) architectural pattern. The MVC concept is beneficial as it facilitates the separation and management of the application into three different modules: the Model, the View and the Controller. The View component of the MVC represents the presentation layer, while the Model manages the system data and the Controller is in charge of user interaction. Thus, it is easier to configure one module (i.e. the presentation) without having to modify the other ones (i.e. the Model/Controller). The maintenance of the code is also simpler and it is more transferable. As for the transferability: since the architectural design allows us to change precise modules of the application, it is easier to transfer the application to different platforms (Desktop, Web, Mobile), as the only major difference between them is the presentation layer. Therefore, it is for the reasons listed above that we have decided to implement the MVC architecture for all three platforms. Now, inside the MVC, there is an additional architectural pattern to connect the Controller to the Persistence: it is the Layered architecture. Although the system is designed with the MVC pattern, we believe the communication with the Persistence of the system is done in a Layered fashion. This design gives the possibility to transform the Controller into a "manager" of the Persistence at the core level, and makes the Persistence system even more maintainable as for when a change need to be made, only the layers directly accessing the modified one need to be updated. Regular changes can then be applied to the databases without harm.

Now, all three platforms implements the MVC design, however the Web Application behaves slightly differently. For the Web App, the MVC system is surrounded inside a client-server design. In other words, the application behaves the same, but clients access the information in a different way. In a client-server architecture, the server retains all the knowledge and the client only accesses the views given by the server. Therefore, a single client can access the same view from different browsers (safari, chrome, etc.) or different workplaces, since its is managed by the server.

Why Didn't We Choose the Other Styles?

Alternative #1: Layered Architecture (for the whole system)

While the layered architecture provides the ability to replace entire layers (and thus increasing dependability), this type of architecture requires that the interface be maintained. As well, achieving a clean separation between layers of our applications would be extremely difficult not to mention the fact that in many cases the high-level layer may have to communicate directly with a lower-level layer. It is important to note however that a layered architecture is for the

specific relation between the manager of our system (the HAS controller) and the persistence of the system.

Alternative #2: Repository Architecture

While the repository architecture allows each component to be completely independent, we decided not to implement this type of architecture because the repository is a single point of failure (i.e. any small error in the repository affects the entire system. Therefore, distributing the repository across the five team members would not be ideal.

Alternative #3: Pipe-and-Filter Architecture

Although the pipe-and-filter architecture is easy to understand and supports transformation reuse, this architecture is more suited towards business workflow processes where each decision requires an analysis to determine the next step to be taken. This would be difficult to implement with our applications.

StartMenu Application Frame

This represents the start of the programs. After the user starts the StartMenu, it launches the Home Audio System application. Since the StartMenu is only the initial view, it doesn't contain or share any data. However it is the necessary pipeline to access the MainMenu: the Home Audio System Application.

Home Audio System Application Frame

This is considered the MainMenu of the program. It displays the music, location and volume management as well as the streaming of the music. While it displays all the available actions, it does not perform any modification on them. Once an event occurs, this subsystem relays the information to the controller, which deal with the information accordingly. For example, if the user wanted to access to a different view of the system, it is the controller that will update the Home Audio System Frame to its new view. Also, the Home Audio Application Frame subsystem as a deep relation with the HAS model subsystem, as they continuously share information to keep track of the updates and make changes to the View if necessary. In fact, all the data displayed in this Frame is to be founded in the HAS Library Model.

Home Audio System Controller

The Home Audio System Controller connects all of the models together, and displays it through the view. If any change is made in the view, the controller subsystem is in charge of update the models accordingly. Hence, the controller subsystem shares data with both the Home Audio System Library Model and the Home Audio System Application Frame. In a conventional MVC architectural pattern it is natural to have more than one controller to handle the different

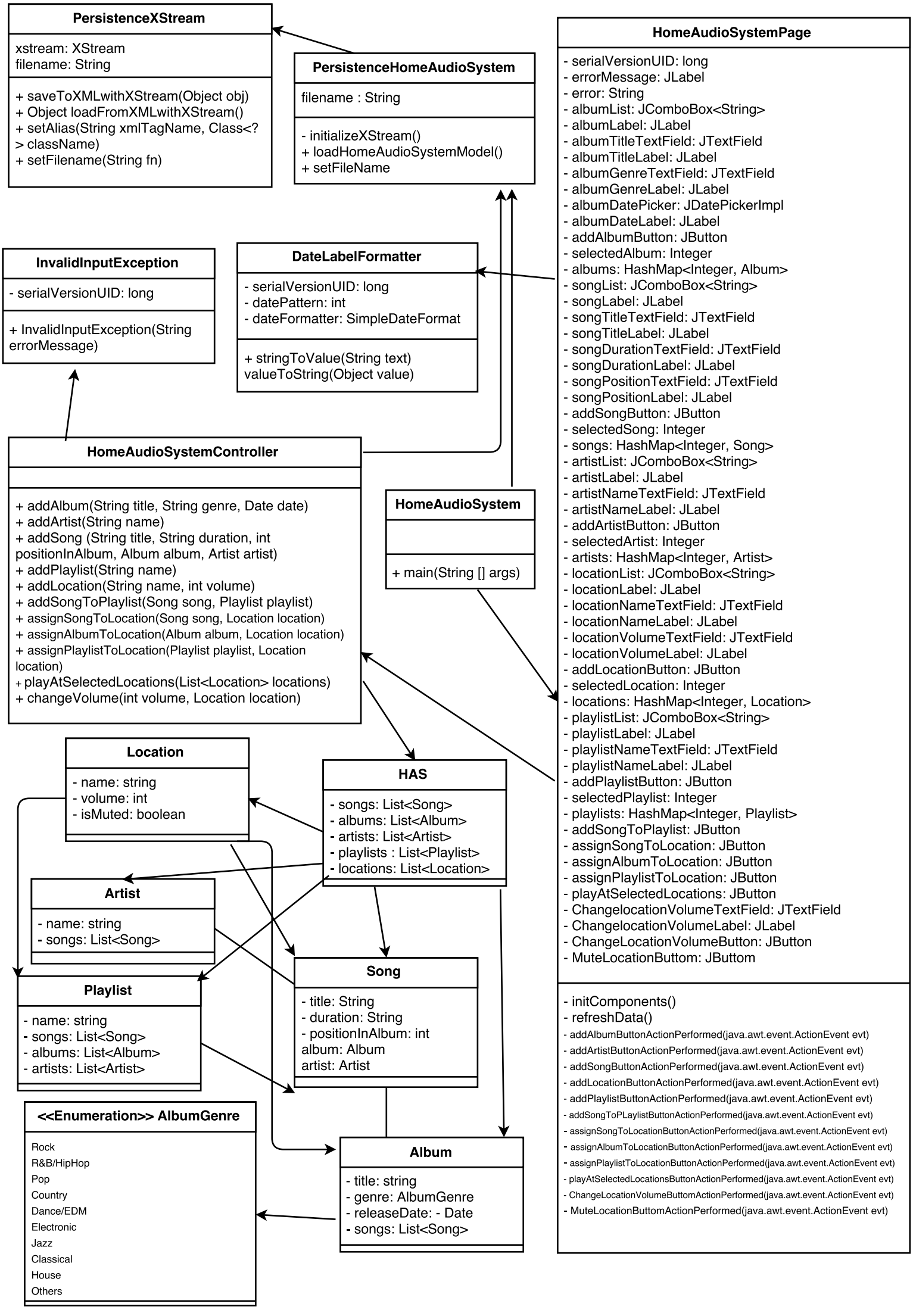
models (Play music, Change location, Manage the songs), however having one controller allows for high-cohesion and low coupling, as it is not necessary for all of the models to be coupled to one-another. Also, the HAS Controller is the subsystem which manipulate the persistence of the system. It saves data to the persistence and also loads data from it to display it on the View.

Home Audio System Persistence

This subsystem provides access to the persistence layer. The persistence layer is the database of the system where the information is organized to be easily accessed and managed in the current session and for the upcoming ones. It is important to note that the architectural style chosen for the Persistence is the Layered design. In the HAS system, the Persistence is accessible by the HAS Controller to save new data or load an existing one to display on the HAS View. The Controller has access to the Persistence Layer, which is the XStreamPersistence in our application. Then the XStreamPersistence has access to the XStream Library to manage the storage of the data.

Home Audio System Library

The Home Audio System Library represents the models of the system. It is the core of the application. Every objects in the application has its own structure and, both the structure and the relation between the objects are described in the HAS Library. Hence, it describes the behavior of the application as it contains the logic and the rules of the entire application. The Model which is occasionally changed by the controller is then sent to the HAS Application Frame to update the View. In other words, the Model actively shares data with the View and receives data from the Controller. However, it doesn't send any data to the Controller as the user doesn't have direct access to the HAS Library Model.


PersistenceXStream
xstream: XStream
filename: String
+ saveToXMLwithXStream(Object obj)
+ Object loadFromXMLwithXStream()
+ setAlias(String xmlTagName, Class<?
> className)
+ setFilename(String fn)
PersistenceHomeAudioSystem
filename : String
- initializeXStream()
+ loadHomeAudioSystemModel()
+ setFileName
HomeAudioSystemPage
- serialVersionUID: long
- errorMessage: JLabel
- error: String
- albumList: JComboBox<String>
- albumLabel: JLabel
- albumTitleTextField: JTextField
- albumTitleLabel: JLabel
- albumGenreTextField: JTextField
- albumGenreLabel: JLabel
- albumDatePicker: JDatePickerImpl
- albumDateLabel: JLabel
- addAlbumButton: JButton
- selectedAlbum: Integer
- albums: HashMap<Integer, Album>
- songList: JComboBox<String>
- songLabel: JLabel
- songTitleTextField: JTextField
- songTitleLabel: JLabel
- songDurationTextField: JTextField
- songDurationLabel: JLabel
- songPositionTextField: JTextField
- songPositionLabel: JLabel
- addSongButton: JButton
- selectedSong: Integer
- songs: HashMap<Integer, Song>
- artistList: JComboBox<String>
- artistLabel: JLabel
- artistNameTextField: JTextField
- artistNameLabel: JLabel
- addArtistButton: JButton
- selectedArtist: Integer
- artists: HashMap<Integer, Artist>
- locationList: JComboBox<String>
- locationLabel: JLabel
- locationNameTextField: JTextField
- locationNameLabel: JLabel
- locationVolumeTextField: JTextField
- locationVolumeLabel: JLabel
- addLocationButton: JButton
- selectedLocation: Integer
- locations: HashMap<Integer, Location>
- playlistList: JComboBox<String>
- playlistLabel: JLabel
- playlistNameTextField: JTextField
- playlistNameLabel: JLabel
- addPlaylistButton: JButton
- selectedPlaylist: Integer
- playlists: HashMap<Integer, Playlist>
- addSongToPlaylist: JButton
- assignSongToLocation: JButton
- assignAlbumToLocation: JButton
- assignPlaylistToLocation: JButton
- playAtSelectedLocations: JButton
- ChangelocationVolumeTextField: JTextField
- ChangelocationVolumeLabel: JLabel
- ChangeLocationVolumeButton: JButton
- MuteLocationButtom: JButtom
- initComponents()
- refreshData()
- addAlbumButtonActionPerformed(java.awt.event.ActionEvent evt)
- addArtistButtonActionPerformed(java.awt.event.ActionEvent evt)
- addSongButtonActionPerformed(java.awt.event.ActionEvent evt)
- addLocationButtonActionPerformed(java.awt.event.ActionEvent evt)
- addPlaylistButtonActionPerformed(java.awt.event.ActionEvent evt)
- addSongToPLaylistButtonActionPerformed(java.awt.event.ActionEvent evt)
- assignSongToLocationButtonActionPerformed(java.awt.event.ActionEvent evt)
- assignAlbumToLocationButtonActionPerformed(java.awt.event.ActionEvent evt)
- assignPlaylistToLocationButtonActionPerformed(java.awt.event.ActionEvent evt)
- playAtSelectedLocationsButtonActionPerformed(java.awt.event.ActionEvent evt)
- ChangeLocationVolumeButtomActionPerformed(java.awt.event.ActionEvent evt)
- MuteLocationButtomActionPerformed(java.awt.event.ActionEvent evt)
InvalidInputException
- serialVersionUID: long
+ InvalidInputException(String
errorMessage)
DateLabelFormatter
- serialVersionUID: long
- datePattern: int
- dateFormatter: SimpleDateFormat
+ stringToValue(String text)
valueToString(Object value)
HomeAudioSystemController
+ addAlbum(String title, String genre, Date date)
+ addArtist(String name)
+ addSong (String title, String duration, int
positionInAlbum, Album album, Artist artist)
+ addPlaylist(String name)
+ addLocation(String name, int volume)
+ addSongToPlaylist(Song song, Playlist playlist)
+ assignSongToLocation(Song song, Location location)
+ assignAlbumToLocation(Album album, Location location)
+ assignPlaylistToLocation(Playlist playlist, Location
location)
+ playAtSelectedLocations(List<Location> locations)
+ changeVolume(int volume, Location location)
HomeAudioSystem
+ main(String [] args)
Location
- name: string
- volume: int
- isMuted: boolean
HAS
- songs: List<Song>
- albums: List<Album>
- artists: List<Artist>
- playlists : List<Playlist>
- locations: List<Location>
Artist
- name: string
- songs: List<Song>
Song
- title: String
- duration: String
- positionInAlbum: int
album: Album
artist: Artist
Playlist
- name: string
- songs: List<Song>
- albums: List<Album>
- artists: List<Artist>
<<Enumeration>> AlbumGenre
Rock
R&B/HipHop
Pop
Country
Dance/EDM
Electronic
Jazz
Classical
House
Others
Album
- title: string
- genre: AlbumGenre
- releaseDate: - Date
- songs: List<Song>

Class Diagram Description

Home Audio System (HAS) is composed of several classes as shown in the class diagram.

HomeAudioSystem

The class contains the main method which loads the model and starts the user interface of the Home Audio System.

Method Index

- Main(String [] args)

HomeAudioSystemPage

HomAudioSystemPage class is responsible for creating the user interface of Home Audio System. The UI created by this class will be used to interact with the user and constantly update when user uses controller to manipulate the model.

Method Index

- addAlbumButtonActionPerformed(java.awt.event.ActionEvent evt)

  Add an album when the button in the interface is pressed
- addSongButtonActionPerformed(java.awt.event.ActionEvent evt)

  Add a song when the button in the interface is pressed
- addArtistButtonActionPerformed(java.awt.event.ActionEvent evt)

  Add an artist when the button in the interface is pressed
- addLocationButtonActionPerformed(java.awt.event.ActionEvent evt)

  Add a location when the button in the interface is pressed
- addPlaylistButtonActionPerformed(java.awt.event.ActionEvent evt)

  Add playlist when the button in the interface is pressed
- addSongToPlaylistButtonActionPerformed(java.awt.event.ActionEvent evt)

  Add a song to the playlist when the button in the interface is pressed
- assignSongToLocationButtonActionPerformed(java.awt.event.ActionEvent evt)

  Assign a song to the location when the button in the interface is pressed.
- assignAlbumToLocationButtonActionPerformed(java.awt.event.ActionEvent evt)

  Assign an album to the location when the button in the interface is pressed.
- assignPlaylistToLocationButtonActionPerformed(java.awt.event.ActionEvent evt)

Assign a playlist to the location when the button in the interface is pressed.

- playAtSelectedLocationsButtonActionPerformed(java.awt.event.ActionEvent evt)

  Play the songs at the multiple locations when the button in the interface is pressed. The user can also choose to play at one location.

- ChangeLocationVolumeButtonActionPerformed(java.awt.event.ActionEvent evt)

  Adjust the volume at certain location when the button in the interface is pressed.

- MuteLocationVolumeButtonActionPerformed(java.awt.event.ActionEvent evt)

  Change the volume to zero at certain location when the button in the interface is pressed.

HomeAudioSystemController

The controller of the HAS has several responsibilities. The methods in this class are responsible for adding album, artist, song, playlist, and location and updates the library of HAS. It will display error message when the input is empty. Also it allows to assign song, album and playlist to a location where user desires, and play at the selected location with adjustable volume.

Method Index

- addAlbum(String title, String genre, Date date)

  Add album entered by user to the library

- addArtist(String name)

  Add artist name entered by user to the library

- addSong(String title, String duration, int positionInAlbum, Album album, Artist artist)

  Add song entered by user to the library

- addPlaylist(String name)

  Add playlist entered by user to the library

- addLocation(String name, int volume)

  Add location entered by user to the library

- addSongToPlaylist(Song song, Playlist playlist)

  Add a song to playlist

- assignSongToLocation(Song song, Location location)

  Assign a song to a specific location

- assignAlbumToLocation(Album album, Location location)

  Assign an album to a specific location

- assignPlaylistToLocation(Playlist playlist, Location location)

Assign a playlist to a specific location

- playAtSelectedLocations(List<Location> locations)

  Play song, album, or playlist that were assigned to the selected locations

- changeVolume(int volume, Location location)

  Change volume at selected location

InvalidInputException

InvalidInputException class allows controller to display error message at certain conditions specified by the controller.

Method Index

- InvalidInputException(String errorMessage)

  Constructs a new exception with the specified error message.

PersistenceHomeAudioSystem

PersistenceHomeAudioSystem class manages the HAS library. It efficiently stores and retrieves data from the library.

Method Index

- initializeXStream()

  Set class with xml tag name for storing data

- loadHomeAudioSystemModel()

  Load the instance of HAS

- setFileName(String name)

  Set name of the file where data is stored

PersistenceXStream

PersistenceXStream class uses XStream library to write and read from an XML file.

Method Index

- saveToXMLwithXStream(Object obj)

  Save data to xml file

- loadFromXMLwithXStream()

  Load xml file where data is stored

- setAlias(String xmlTagName, Class<?> className)

  Set xml tag name for classes when storing data

- setFilename(String fn)

  Set file name with given string

DateLabelFormatter

DateLabelFormatter class is responsible for the format of the date display

Method Index

- stringToValue(String text)

  Parses text from the beginning of the given string to produce an object

- valueToString(Object value))

  Formats a Date into a date/time string

The following classes are the domain models which identify the principal concerns in the system. They are defined using UML class diagrams that include objects, attributes and association and then automatically generated by Umple.

HAS

The HAS class manages how the information is stored for Location, Artist, Song, and Playlist within the library. The HAS was designed using the singleton pattern, such that library which contains all information about the Locations, Artists, Songs, and Playlists will be restricted to instantiate just once. The singleton pattern is most appropriate for this application as it allows for only one library to be created, insuring a singular database for all information to be stored. Therefore, restricting the potential for loss of information through multiple libraries.

Song

The Song class stores information on title, duration, position in an album, album name, and artist name for each song in the HAS library.

Artist

The Artist class stores the name of the artist as well as a list of every song that the artist is associated with.

Playlist

The Playlist class stores the name of the playlist, and lists containing the songs, artists, and albums on the playlist.

Album

The Album class stores the name of the album, the album release date, album genre, and a list containing all songs on the album.

Task	Start Date	Expected Completion Date	isCompleted
Deliverable 1 – Requirements Document and Prototype			
Functional and non-functional system requirements	02/10/2016	02/12/2016	yes
Domain Model	02/13/2016	02/15/2016	yes
Use Cases	02/16/2016	02/20/2016	yes
Requirements-level sequence diagram for “Add Album” use case	02/16/2016	02/20/2016	yes
Source code of prototype implementation of “Add Album” use case on each supported platform	02/16/2016	02/20/2016	yes
Implementation-level sequence diagram for “Add Album” use case for each supported platform	02/16/2016	02/20/2016	yes
Work plan for remaining iterations	02/20/2016	02/21/2016	yes
Make sure everything is in order for submission of deliverable	02/21/2016	02/22/2016	yes
Deliverable 2 – Design Specification			
Description of architecture of proposed solution including block diagram	02/26/2016	02/29/2016	yes
Description of detailed design of proposed solution including class diagram	02/26/2016	02/29/2016	yes
Implement Requirement "HAS7706"	02/27/2016	03/06/2016	yes
Implement Requirement "HAS7707"	02/27/2016	03/06/2016	yes
Implement Requirement "HAS7708"	02/27/2016	03/06/2016	yes
Update of work plan	03/05/2016	03/06/2016	yes
Make sure everything is in order for submission of deliverable	03/06/2016	03/07/2016	yes
Deliverable 3 – Quality Assurance Plan			
Description of unit testing	03/08/2016	03/17/2016	no
Description of component testing	03/08/2016	03/17/2016	no
Description of system testing	03/08/2016	03/17/2016	no
Description of performance/stress testing	03/08/2016	03/17/2016	no
Implement Requirement "HAS7703"	03/10/2016	03/18/2016	no
Implement Requirement "HAS7704"	03/10/2016	03/18/2016	no
Implement Requirement "HAS7705"	03/10/2016	03/18/2016	no
Implement Requirement "HAS7709"	03/10/2016	03/18/2016	no
Implement Requirement "HAS7710"	03/10/2016	03/18/2016	no
Update of work plan	03/19/2016	03/20/2016	no
Make sure everything is in order for submission of deliverable	03/20/2016	03/21/2016	no
Deliverable 4 – Release Pipeline Plan			
Description of release pipeline	03/22/2016	03/25/2016	no
Implement Requirement "HAS7702"	03/22/2016	03/25/2016	no
Implement Requirement "HAS7711"	03/22/2016	03/25/2016	no
Implement Requirement "HAS7712"	03/22/2016	03/25/2016	no
Implement Requirement "HAS7713"	03/22/2016	03/25/2016	no
Implement Requirement "HAS7714"	03/22/2016	03/25/2016	no
Implement Requirement "HAS7715"	03/22/2016	03/25/2016	no
Update of work plan	03/26/2016	03/27/2016	no
Make sure everything is in order for submission of deliverable	03/27/2016	03/27/2016	no
Deliverable 5 – Presentation			
Prepare Powerpoint presentation	03/28/2016	04/09/2016	no
Prepare script to say	03/28/2016	04/09/2016	no
Fully test application	03/28/2016	04/13/2016	no
Present project	04/14/2016	04/14/2016	no
Deliverable 6 – Final Application			
Source code of full implementation on each supported platform	04/15/2016	04/15/2016	no

Table 1: Functional Requirements of HAS

Requirement ID	Requirement Description
HAS7701	The Home Audio System shall allow the user to control the system from a central location.
HAS7702	The Home Audio System shall maintain a library of songs grouped by album or artist
HAS7703	The Home Audio System shall keep track of a song's title.
HAS7704	The Home Audio System shall keep track of a song's duration.
HAS7705	The Home Audio System shall keep track of a song's position on an album.
HAS7706	The Home Audio System shall keep track of an album's genre.
HAS7707	The Home Audio System shall keep track of an album's release date.
HAS7708	The Home Audio System shall keep track of an artist's name.
HAS7709	The Home Audio System shall implement an enumeration style system for genres.
HAS7710	The Home Audio System shall allow the user to set up one or multiple locations.
HAS7711	The Home Audio System shall allow the user to create one or multiple ordered playlists.
HAS7712	The Home Audio System shall allow the user to stream the same music to several locations.
HAS7713	The Home Audio System shall allow the user to play music by choosing a playlist, album or song as well as a specific location.
HAS7714	The Home Audio System shall allow the user to control the volume of each location.
HAS7715	The Home Audio System shall allow the user to temporarily mute one or many locations.

Table 2: Non-Functional Requirements of HAS

Requirement ID	Requirement Description
HAS8801	The Home Audio System shall be portable on 3 devices: desktop, mobile and web.
HAS8802	The Home Audio System shall be able to refresh the screen in less than 2 seconds.