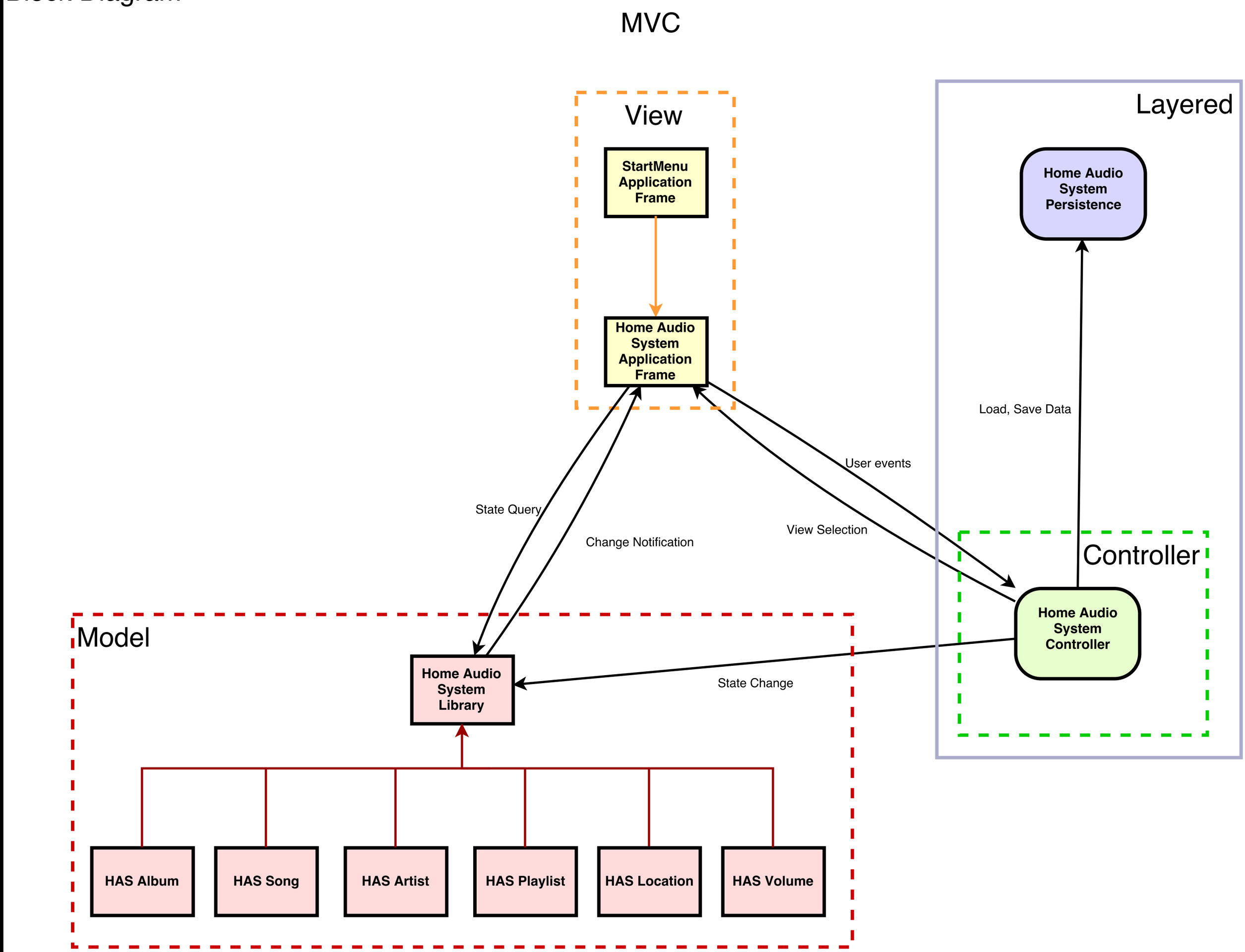
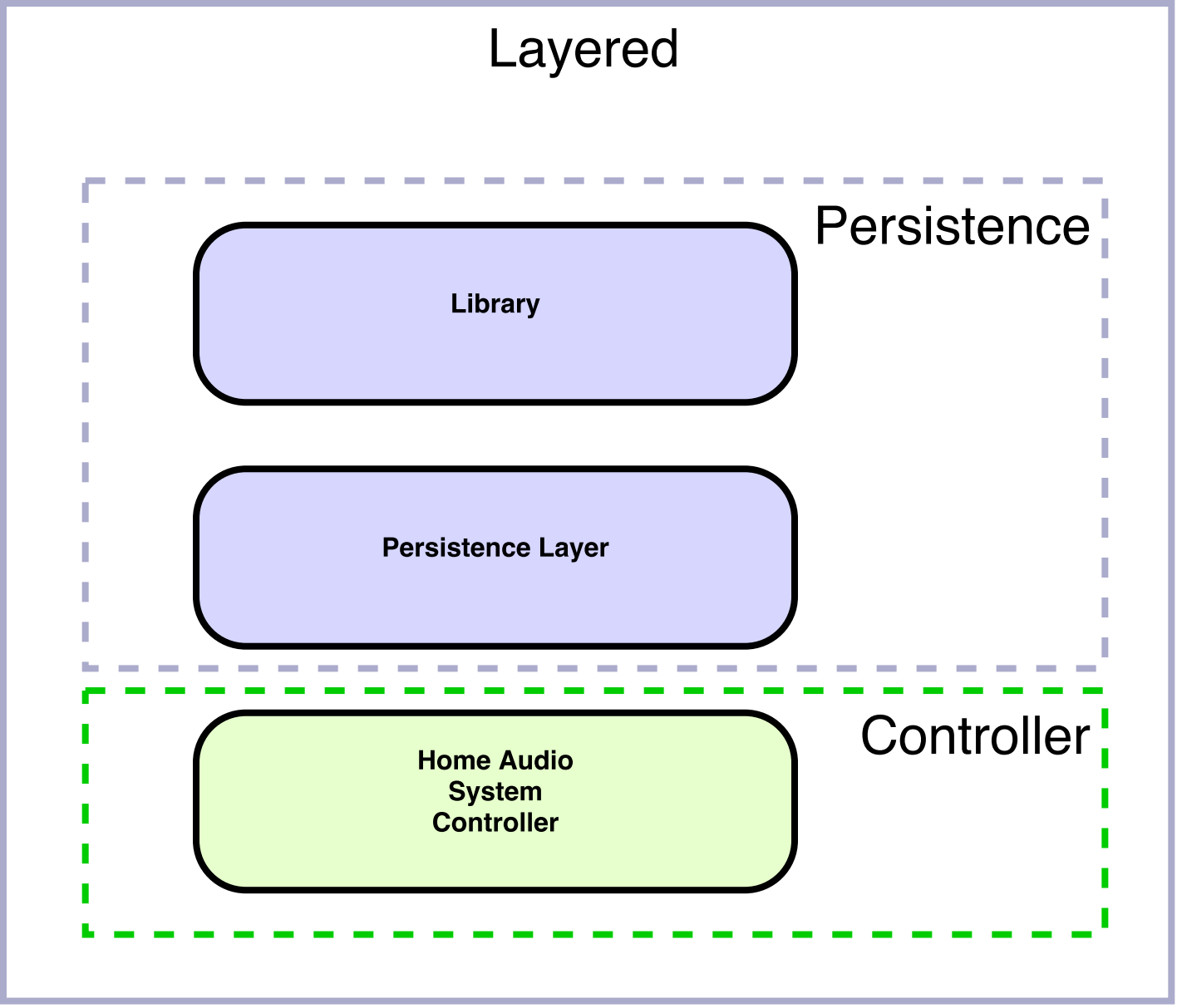
Block Diagram




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.