**Controller**

The controller is a subsystem of the Model-View-Controller (MVC) architecture that responds to user input and performs interactions on the data model objects. The controller contains business-related logic to handle incoming requests. Commonly, the controller is responsible for handling all interactions from the view and updates the database using the model. The controller first receives a request from the user and then interacts with the model to send and receive data. The controller then interacts with the view to render the data. In the context of Jami, the process differs from that of a more common MVC architecture, in which the controller is the interface between the model and the view, and is responsible for all communication and interaction between the two subsystems. The primary role of the controller in Jami is to receive some data and its state, then accordingly route the data and its updated state. The controller handles all incoming requests such as; adding a new contact, messaging processes, and audio/video calls.

In Jami, all inter-process communication is handled by the d-bus. The d-bus acts as the controller and enables communication between the view (client) and the model (daemon).

**Team Issues and Lessons Learned**

For the duration of this first assignment, the team encountered numerous obstacles and challenges that had to be overcome. Most significantly, the initial research proved to be challenging due to the lack of relevant and up-to-date documentation on Jami. Many resources did well in giving a basic overview of the main features offered by Jami, but failed to provide an in-depth description of the software and its structure. As such, to overcome this obstacle, subtasks were given to each member of the team in order to more efficiently conduct research and achieve a complete understanding of all the components of Jami, their functionalities, as well as the architecture as a whole. This obstacle re-enforced the importance of time management for each individual and the team as a whole. The team also had to work together remotely due to the ongoing pandemic. This obstacle brought up another significant challenge in efficient communication. In order to overcome this challenge, the team took two key steps that would prove to be very beneficial. Firstly, the team had to account for time differences in regards to the current locations of each member. Secondly, the team had to consider each members’ point of view. These considerations were crucial as they taught each member of the team to accustom themselves to the situation at hand, and work efficiently in the allotted time frames. The lessons learned from these obstacles helped to build an effective team dynamic.

**Conclusion**

In summary, the derived conceptual architecture of Jami is a high-level structure based on the Model-View-Controller software design pattern, with the utilization of a peer-to-peer architecture for the purpose of communication between users. Additionally, the OpenDHT proxy grants Jami users the ability to exchange messages or conduct audio/video calls without use of a centralized server. Furthermore, this eliminates the need of the company to store personal user data. Jami aims to address an increasing demand for private communication via the internet. Analysis of the conceptual architecture of Jami shows the potential for easy expansion with conservation of its main purpose due to its resiliency and scalability in the technological field.

Through determining the conceptual architecture of Jami, we have built a solid foundation that can be used to derive its concrete architecture. The derivation of a concrete architecture for Jami will provide a deeper look into the system components and how they are implemented.