**Checkpoint #3**

**SWENG 587**

**Nick Forleo, Steven Lathrop, Manish Shrestha, Brian Stahl**

**Step 1: Review Inputs**

|  |  |  |
| --- | --- | --- |
| **Table 1: Architectural Drivers for Internet-based Collaborative Work Environment** | | |
| **ID** | **Architectural Drivers** | |
| **Design Purpose** | | |
| **DSN-1** | The system design shall support increased efficiency of collaboration between geographically dispersed employees | |
| **DSN-2** | The system design shall allow for the product to differentiate itself in the product in a competitive market | |
| **DSN-3** | The system shall reduce employee downtime and increase productivity | |
| **Primary Functional Requirements** | | |
| **UC-1** | The system shall support 5 different services including voice communication, video conferencing, instant chat, file sharing, and collaborative whiteboarding. | |
| **UC-2** | Real-time features such as voice, video, and whiteboarding shall have a response time of no more than 100ms. | |
| **UC-3** | The system must be profitable | |
| **Quality Attribute Requirements** | | |
| **QA-1** | Usability - A user attempts to access any of the 5 services. They are able to connect and use any of the services at any time. | (H, H) |
| **QA-2** | Security - A user will sign into the product and the system will verify the credentials. Once authenticated, only features that are within the users’ role permission will be available. | (H, H) |
| **QA-3** | Performance - The system detects a change in connection strength or availability while a user is using a service. It will try to reconfigure the settings for an optimal connection within 10 microseconds or it will alert the user of possible network disruption. | (M, M) |
| **QA-4** | Performance - When a users’ data in any of the 5 services is sent to the server, it will be compressed and then decompressed once the client receives it. No packets will be dropped. | (L, M) |
| **Constraints** | | |
| **CON-1** | The system shall be able to support a variety of hardware and operating system inputs and outputs | |
| **CON-2** | Any registered user must be over the age of 13 | |
| **Architectural Concerns** | | |
| **CRN-1** | The 5 system components shall be developed and run independently within the system | |
| **CRN-2** | The system shall be continuously and quickly deployable | |
| **CRN-3** | Metadata for the system shall be stored in a database | |

**Iteration 1**

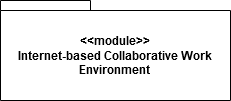
**Step 2:** Establish iteration goal by selecting drivers

We will focus on the following drivers this iteration because they describe the fundamental piece of functionality for the system and because of their similarity. We also want to keep in mind one of the reasons for creating the software when starting the architecture design.

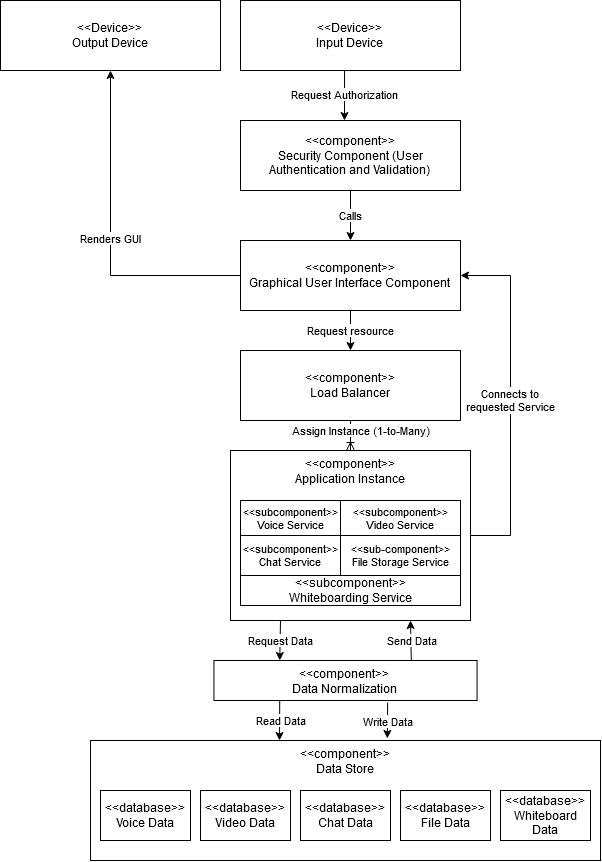
* **UC-1:** The system shall support 5 different services including voice communication, video conferencing, instant chat, file sharing, and collaborative whiteboarding.
* **QA-1:** Usability - A user attempts to access any of the 5 services. They can connect and use any of the services at any time.
* **DSN-2:** The system design shall allow for the product to differentiate itself in the product in a competitive market

**Step 3:** Choose One or More Elements of the System to Refine

Since this is the first iteration of a Greenfield system, we will refine the entire Internet-based Collaborative Work Environment.



*Figure 1:* *Internet-based Collaborative Work Environment we are focusing on in iteration 2*



*Figure 2: Component-and-connecter view of Internet-based Collaborative Work Environment*

**Step 4:** Choose One or More Design Concepts That Satisfy the Selected Drivers

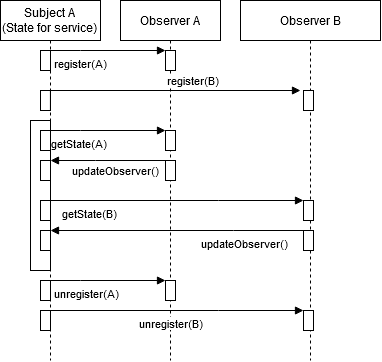
By meeting QA-1, we will have already met UC-1. Therefore, a tactic and a design pattern that meet the needs of QA-1 will also suffice for UC-1. We will use the following tactic under Support System Initiative:

* *Maintain system model:* The system will know what the expected response time for service is, so it can automatically reallocate resources as needed for the user to get response times in the expected range. The load balancer Figure 2 will help to achieve this.

The design pattern that the system will employ is:

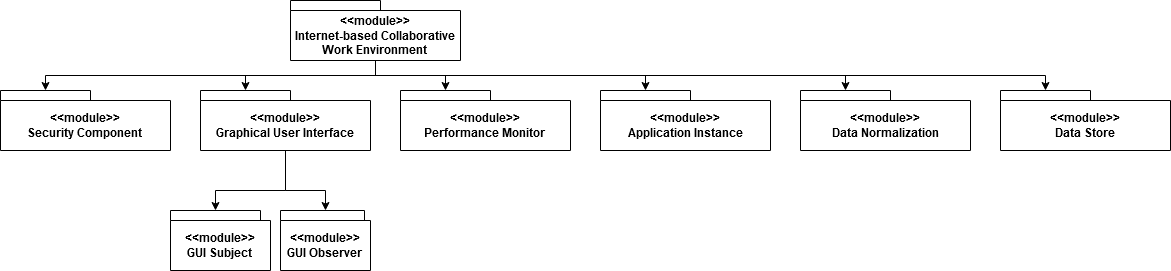
* *Observer:* The state of each service will be a subject and the GUI (Graphic User Interface) will use an observer to render the data the subject. This will be beneficial because you may be accessing more than one service at once. If you are in a video call and want to chat with someone, the interface will look different than it would if you were just to be messaging someone. This way the data will be updated correctly, regardless of what the view looks like. An initial state is loaded when the service is requested by the user

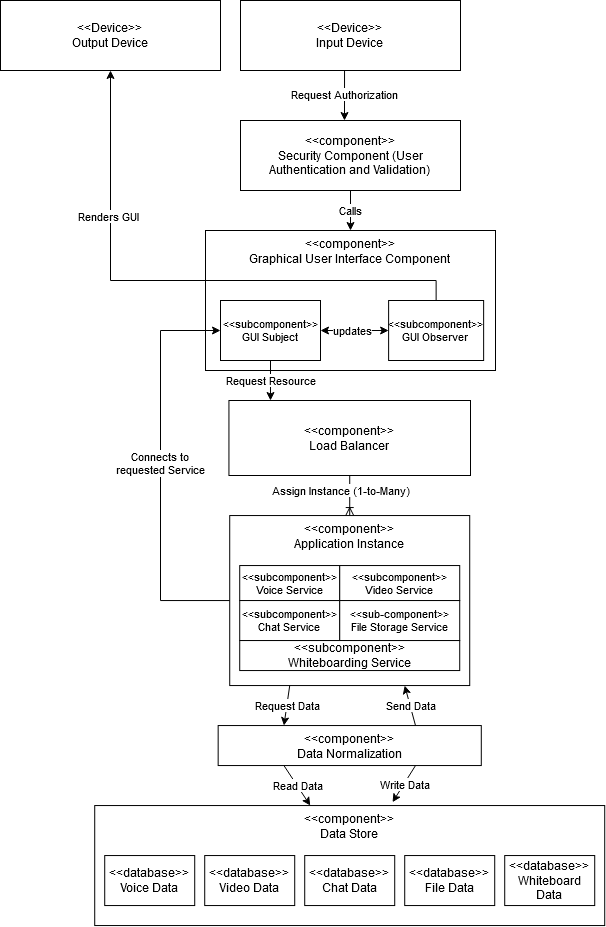
Here is a sequence diagram for the observer pattern:



*Figure 3: Sequence diagram for Observer pattern*

**Step 5:** Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

*Figure 4: Model view with Observer design pattern*



*Figure 5: Component-and-connector view for observer pattern*

**Step 6:** Sketch Views and Record Design Decisions

The model view, component-and-connector view, and sequence diagram were created in the previous steps. Design decisions for tactics and patterns were also previously described.

**Step 7:** Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Purpose

The current architecture of the system fulfills the requirements of drivers UC-1 and QA-1, which means that the iteration goal has been met. With these drivers met, we also differentiate our product in the market so we can say we also fulfilled DSN-2.

**Iteration 2**

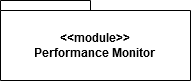
**Step 2:** Establish iteration goal by selecting drivers

We will focus on these drivers because they all relate to performance, which will also lead to increased employee efficiency.

* **UC-2:** Real-time features such as voice, video, and whiteboarding shall have a response time of no more than 100ms.
* **QA-3:** The system detects a change in connection strength or availability while a user is using a service. It will try to reconfigure the settings for an optimal connection within 10 microseconds or it will alert the user of possible network disruption.
* **QA-4:** When a users’ data in any of the 5 services is sent to the server, it will be compressed and then decompressed once the client receives it. No packets will be dropped
* **DSN-1:** The system design shall support increased efficiency of collaboration between geographically dispersed employees
* **DSN-3:** The system shall reduce employee downtime and increase productivity

**Step 3:** Choose One or More Elements of the System to Refine

We are focusing on performance this iteration. The first iteration of the design already included a load balancer and multiple application instances which helps offset any performance issue brought on by using the Observer pattern. However, it will also help address UC-2 by keeping response times to a minimum. We will focus on the Load Balancer component, except it will be renamed to Performance Monitor to reflect its purpose more accurately.



*Figure 6: Performance Monitor (renamed from Load Balancer) that we are focusing on in iteration 2*

**Step 4:** Choose One or More Design Concepts That Satisfy the Selected Drivers

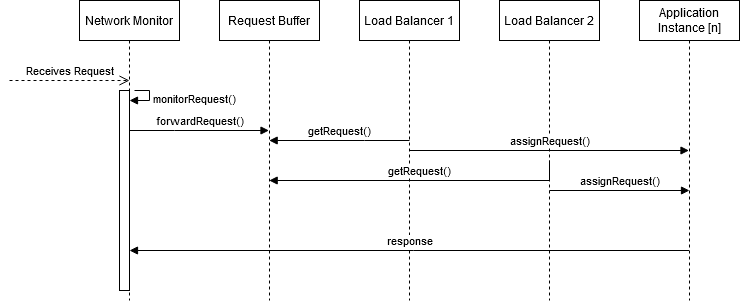
To address our performance concerns, we will consider two design tactics:

* *Increase efficiency of resource usage:* The best way to increase hardware efficiency is to write optimized software. Our algorithms will be a key aspect in providing rapid response times and reallocating resources as efficiently as possible. The system will monitor the metadata of the requests and system performance.
* *Introduce concurrency:* We already have the Load Balancer to distribute requests to different application instances, but what about the requests to the Load Balancer? We can introduce a second Load Balancer to handle these requests.

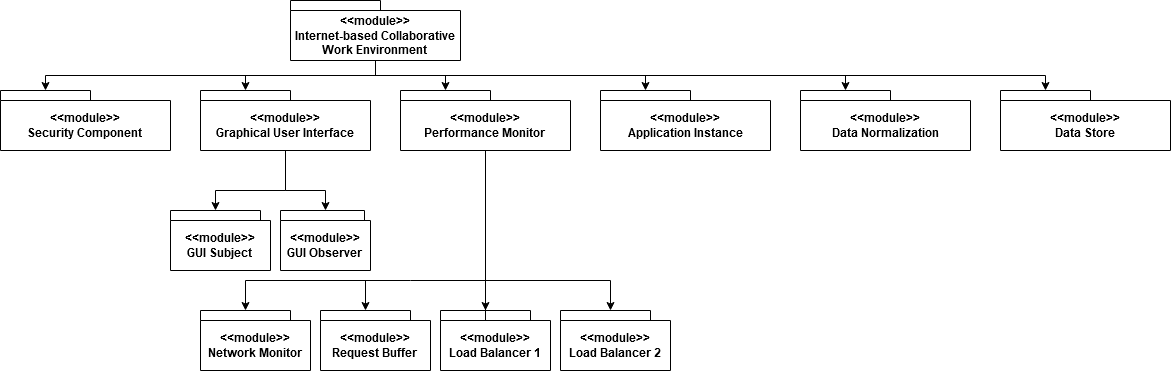
The design pattern that will use to address these tactics is:

* *Load Balancer:* Using a load balancer was part of the initial design of the system, but to improve its reliability and introduce concurrency, we will add another one. Each one will check the load on a given application instance before assigning a request for a service to a given application instance.

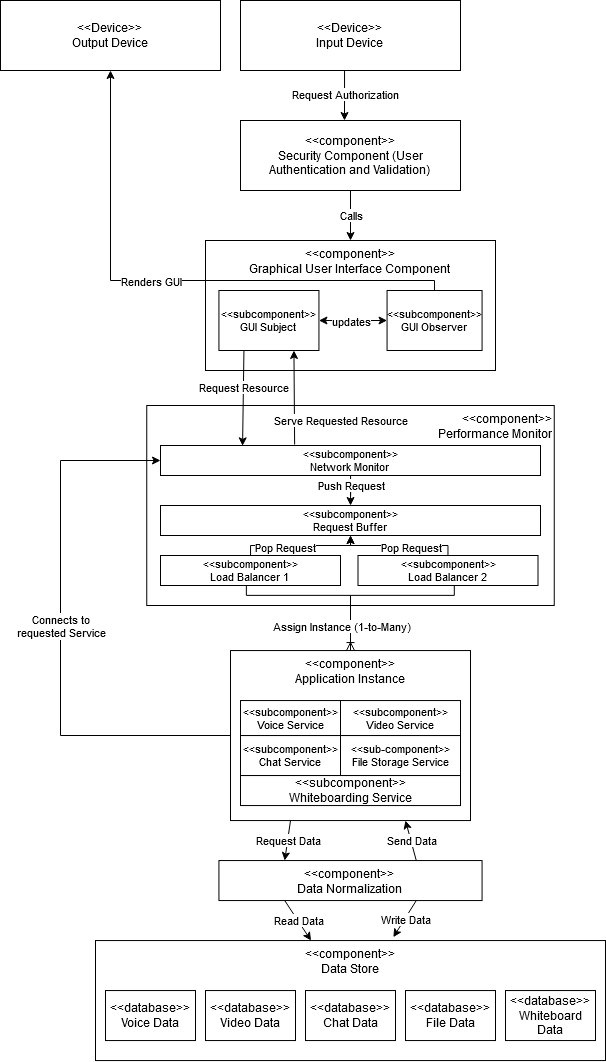
The sequence diagram for the changes we are going to make is shown below:

*Figure 7: Sequence diagram for Network monitor and Improved Load Balancer Architecture*

**Step 5:** Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces



*Figure 8: Model view showing decomposition of the Performance Monitor*



*Figure 9: Component-and-connector view with the expansion of the Performance Monitor.*

**Step 6:** Sketch Views and Record Design Decisions

The model view, component-and-connector view, and sequence diagram were created in the previous steps. Design decisions for tactics and patterns were also previously described.

**Step 7:** Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Purpose

With the application of the Load Balancer pattern directly address UC-2. With the addition of the Network Monitor that will analyze all the requests to and from the client, we can identify and optimally reallocate network and hardware resources to directly address QA-3 and QA-4. By fulfilling these drivers, we also indirectly address DSN-1 and DSN-3. Since we are addressing all the drivers in step 1, we can say that we have met our iteration goal.

**Iteration 3**

**Step 2:** Establish iteration goal by selecting drivers

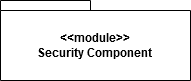
In this iteration, we will focus on the security drivers. While CON-1 is not a direct security, we need to be sure that we can authenticate users coming from any hardware or software profile.

* **QA-2:** A user will sign into the product and the system will verify the credentials. Once authenticated, only features that are within the users’ role permission will be available.
* **CON-1:** The system shall be able to support a variety of hardware and operating system inputs and outputs
* **CON-2:** Any registered user must be over the age of 13

Addressing these drivers will be the goal of this iteration.

**Step 3:** Choose One or More Elements of the System to Refine

We are focusing on Security for this iteration. QA-2 and CON-1 relate to the authentication of users in the system and CON-2 is a legal requirement that our system must fulfill. This can be achieved by passing that requirement onto the customer if they already use an authentication service. Alternatively, we can do this if users sign up directly in our system. The part of the system that will be most affected by this will be our security component.



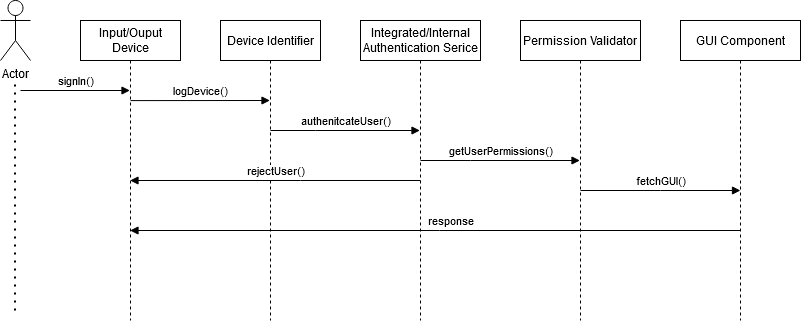
*Figure 10: Security Component we are focusing on for iteration 3*

**Step 4:** Choose One or More Design Concepts That Satisfy the Selected Drivers

To address security concerns, there a few tactics we can utilize:

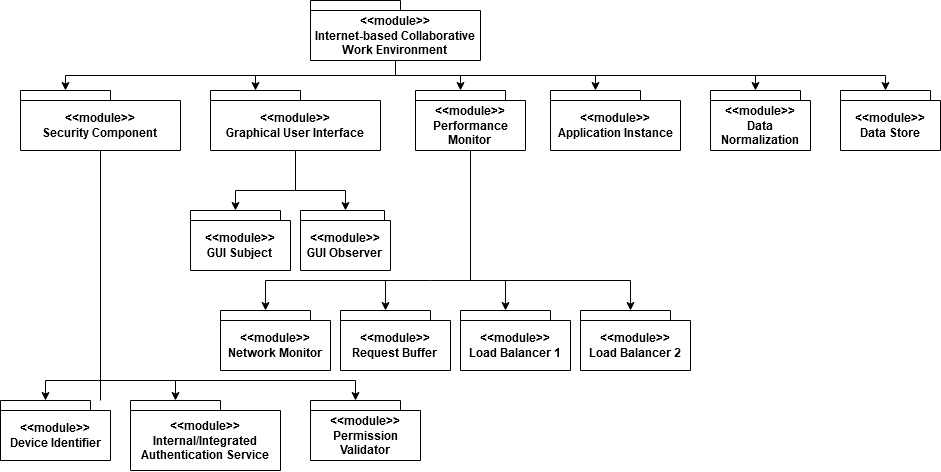
* *Identify actors*. We can track all the users who attempt to authenticate to our system. We can collect data like browser version, IP address, OS (Operating System), hardware profile and more. This achieves two things for us. If we run into an issue with a specific OS or hardware profile, we will have a record of it so we can ensure that we support it. We can also check to make sure a single account is being used correctly. For example, if a single user is signed in twice from the same IP address, then maybe they are on two different devices in the same spot, but if they are in two different countries, then the account may have been compromised.
* *Authenticate actors:* Any user must be authenticated prior to using or being allowed access to our system. This can be done by integrating with a customer’s existing authentication service or with customers who register directly with our system.
* *Authorize actors:* Once a user has been authenticated, it will only have access to features, permissions, data, etc..., that have been assigned to the user directly, group, or role. This will protect users from accidental events that may occur when one user has access to data it should. It will also limit the number of users that can cause potential harm, thus reducing the chances of a comprised user account causing more harm.

Using these tactics will allow our system to address our iteration goal, so we do not need to explicitly utilize a specific design pattern. Below is a sequence diagram to represent these tactics.

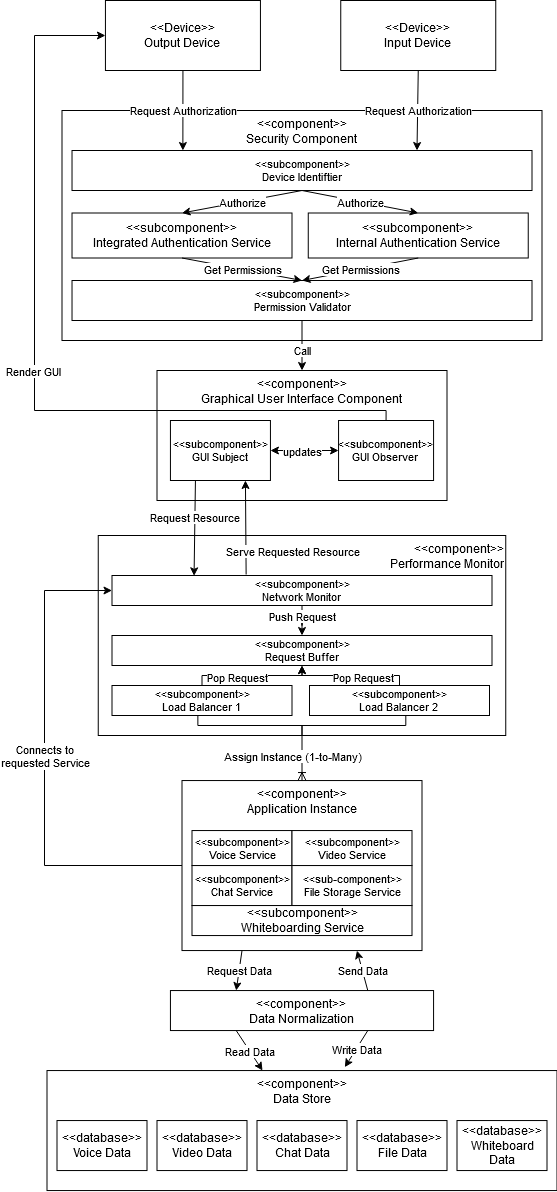


*Figure 11: Sequence diagram showing user authentication addressing security concerns*

**Step 5:** Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces



*Figure 12: Module view of showing the modules of the Security Component*



*Figure 13: Component-and-connector view with the Security component utilizing the security tactics*

**Step 6:** Sketch Views and Record Design Decisions

The model view, component-and-connector view, and sequence diagram were created in the previous steps. Design decisions for tactics and patterns were also previously described.

**Step 7:** Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Purpose

By introducing the tactics used to address our security drivers, we have fulfilled our iteration goal. We meet QA-1 by authenticating and authorizing users. One way we address CON-1 is by logging all metadata when accessing the system by using the *identify* *actors* tactic. Again, this helps identify bad actors and any device we may not fully support. We can track errors back to the hardware and software of the device. Finally, we add CON-2 by not allowing unauthenticated users to use the system. Our internal authentication service shall verify the users age and we will only allow third party authentication services to integrate if they also meet government restrictions on their users.

**Overall Design Trade Offs**

1. The first design decision we made for the application to be browser based. This will ensure that the software is compatible with a wider range of devices than a stand-alone application would. We would have to make sure the application is compatible with all operating systems. We know that web browsers can be used on almost any type of operating system.
2. Using the *Observer* design pattern will separate the functionality of manipulating the state from how it is represented. This means that each observer may have to do a lot of work to reflect a state change, thus negatively impacting the performance of the system. This can be resolved by using a load balancer. A load balancer will distribute users to different instances of the application to reduce resource competition instead of only having 1 instance of each service available. This will improve performance and resiliency.
3. In using the *Load Balancer* pattern, you introduce a single point of failure into the system. To address this problem, we will use two load balancers. This will result in a fail-safe if one of the load balancers stops working and it introduces concurrency.
4. Allowing third party authentication services to integrate with our system will allow customers who use SSO (single sign on service, such as Okta) in their businesses to use our product seamlessly. This means we will have to have different licensing agreements for different ways of authenticating, but that can be solved by a legal or accounting department.

|  |  |
| --- | --- |
| **Table 1: Business Goals for the Building Automation System** | |
| **Business Goals**  **(Mission Objective)** | **Goal Refinement**  **(Engineering Objectives)** |
| Increase efficiency of collaboration between geographically dispersed employees | Ensure data privacy of user when connecting from unsecure networks during usage of any of the 5 services |
| Ensure users have a nominal experience when using any of the 5 services from any location |
| Optimize networking for when accessing the system |
| Support the addition of a user to an existing session of any of the 5 services |
| Support Desktop and mobile operating systems |
| Implement an easy to use and intuitive interface for each service in the system |
| Reduce employee productivity downtime or time spent on menial tasks | Maintain high server uptime to keep productivity high |
| Implement an in proved process for integrating changes to any piece of the system |
| Reduce the time spent troubleshooting disruptions in the system |
| Increase efficiency when releasing changes for the teleconferencing services |
| Increase efficiency when releasing changes for the system |
| Reduce the operating costs of our business systems | Decrease man hours spent making changes to the system |
| Reduce the processing power and kilowatt usage of other services when a using service |