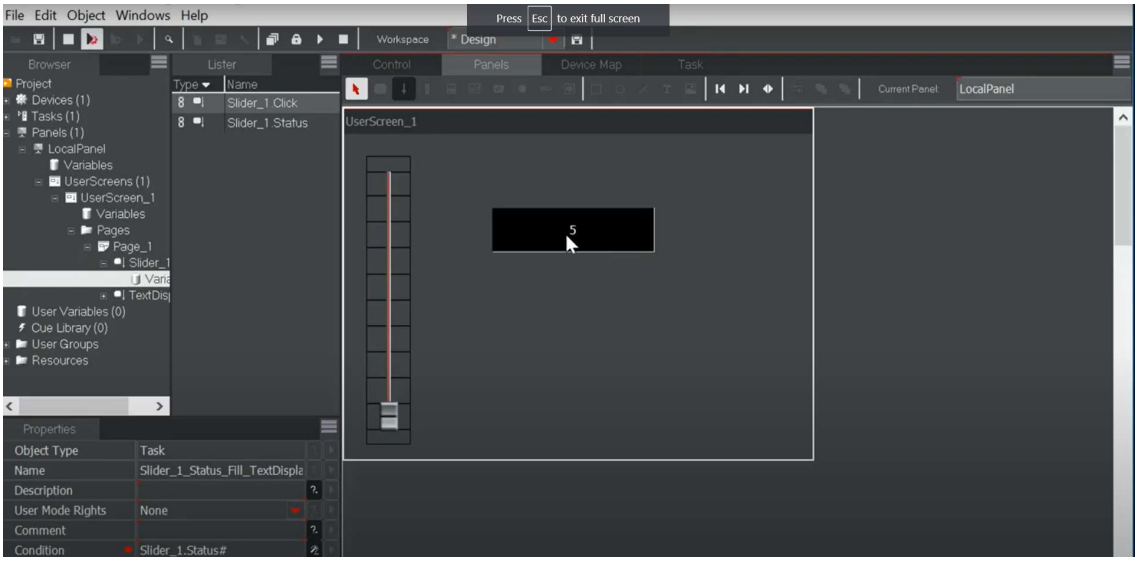
Introduction

I have been working at Medialon/7thsense since mid-March and have been involved in many projects. Medialon provides show control systems to anything from a few lights to a large system. 7thSense brought them over and they provide high quality media servers and a user-interface for editing the media content on a timeline.

I joined the Medialon development team from London. Medialon have a piece of software called Manager which is the interface for creating show control system. As Medialon was previous owned by a Barco the Manager graphical interface had become very out of date compared to the modern applications. As you can see from the picture below it has a very retro feel to it. So, when 7thSense brought Medialon their first plan was to renew the interface.

Manager version 6: (https://medialon.com/products/medialon-manager/)



When I joined one of my tasks was to do research about the best way to do this. Some of the issues in mind were that Manager is an a very reliable product, but it is also very old and large. Therefore, to change the Manager core code would take years and I was told that it was not a possible solution.

Before I could begin to think of the best way to attack this problem, I begun playing around with the application itself. This is because I needed to know what user-ability I needed to keep and what is not necessary to bring into a new graphical user interface.

After building my own mini control system and getting to know what Manager provides, I realised there is this notion of shared variables. In Manager you can create variables of type int, string, float and Enum. You can also create tasks which affect variables. For example, you can have a count task that increments an integer each frame. You can put variables and tasks into a shared group which is then accessible outside of Managers software. This works perfectly for me because I need to create an external add on for Manager. So, if I can somehow link my app to these variables and tasks that would be a valid solution.

After finding this out I spoke to my Manager and was told to look for suitable way to implement this software that could be linked to Managers shared groups.

Managers UI consists of buttons, labels, input fields, sliders, gauges, and the ability to have multiple pages. Manager also can turn its UI into a webpage so this capability would need to be provided as

well. 7thSense and Medialon are a c++ house and they use Juice for their UI which is powerful but for this new app they were looking for possibilities of turning it into a 3D environment in the future for controlling large shows. Therefore, to achieve this we would need to use either OpenGL or DirectX and this is very time consuming for generating even simple UI.

At university I study Games technology, so I come from a game engine background and suggested that we use Unity for providing the graphic system. Unity also provides a desktop and webGL build so we could display the same app on multiple platforms which saves company resource in maintaining this application.

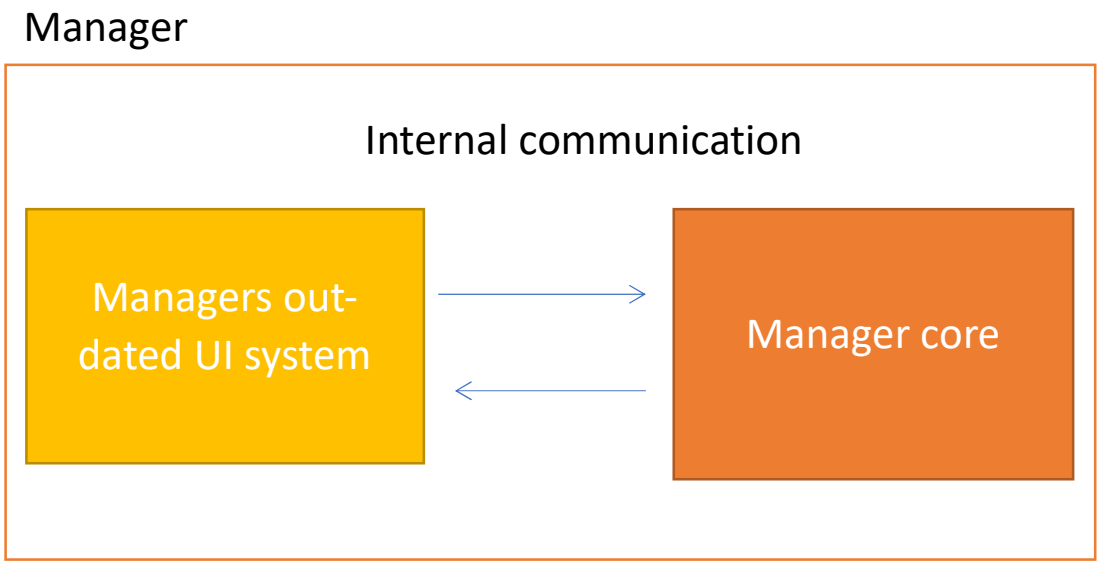
After presenting this idea to 7thSense and showed how easy it was to create UI applications and export it to different platforms they agreed it was the best way forward.

Marquee System Level Architecture Design:

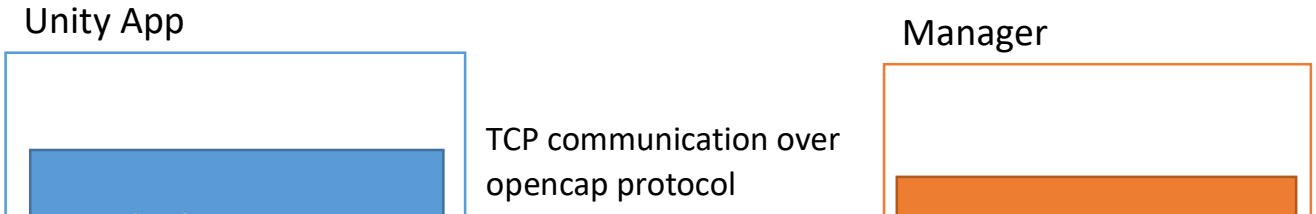
What we knew we needed was a communication between Manager and a Unity app which most likely would be TCP. After studying the Manager manual (190-192) there is a protocol called "opencap" that broadcasts a message over tcp when a shared variables value has been changed. This means I need to catch these tcp packets and produce the correct output. Opencap is also bi-directional. So, I can update these shared vars using the same protocol over tcp. I can also trigger shared tasks over opencap protocol.

An example of this would be having a clock UI object in my unity app and linking this clock to a timer in Manager. This would keep in sync over tcp connections and then if you wanted to reset the clock you could change it in the Unity app and it would automatically reset the managers clock(bi-directional).

Old system level workflow…



New system level workflow…



Summary for separating the front end from the backend:

- To improve the look and feel of any control system made using Manager.
- Managers core is very old and large. So, editing it would be time consuming and tedious therefore separating the front end allows capability to be added to Manager without impacting the core functionality.

Developing the product:

The product being made in Unity will be called Marquee. I am one of four developers programming and taking experience from my university assessments we all had to communicate and keep on the same page. This would normally be easy as we would all be sitting next to each other but because of COVID-19 I was one of the few people in the office, so we all had to stay active on Microsoft teams.

Once of the first things we did was start on Jira and create a Marquee board. This was the first time I used Jira and after little use. It would have helped me in my university project where we were put in teams of students. One of the mechanisms I found to be especially helpful were the sprints and adding points onto tasks to say how long they would take you. At first, I was not accurate at all and really didn't have much clue in time frames but over time I begun to get a grasp of this and judge how long tasks would take me much more precisely.

We also set up SCRUM meetings for twice a week where we could revive and flag and potential problems we encountered. I found this especially helpful with keeping up to date with the areas I was not programming. So, when I came to fix a problem I was always thinking of the big picture and how it would affect persons X code as well.

My product manager told me early on that building a new application is always 2 steps forwards and one step back. This became very true when we started to get problems in our code base or run into limitations we had not thought about.

WebGL Limitations:

Programming TCP communication between Marquee and Manager is just a process and it worked well for desktop applications. However, I really struggled to get marquee to even display on a WebGL build. After doing some more research I found out that WebGL does not support multi-threading or TCP communications for security reasons. After a lot of re-thinking with my colleagues we decided to have two types of communications between Marquee and Manager which were TCP and Web Sockets. This fixed the communication problem. To fix the no-threading we used Asynchronous functions for the socket communication to imitate another thread.

After all this the WebGL would successfully build but it would crash very frequently. The problems with these crashes the fact they are random. Sometimes they could be timing issues or sometimes the program gets in the wrong state early on and the part of the code it crashes on is not the source of the problem. After scanning through the Unity forms, I found that Try/Catches do not work in WebGL. I tested this on and empty Unity project and it crashed. This meant that all four of us had to re-think about areas of code that used a Try/Catch to remove it.

Scene Merge Conflicts:

I used Unity a in my first/second year, but I never used GitHub with it. As there are multiple of us working on it using GitHub was essential but one of the problems, we were encountering in merging a unity scene script. A unity scene is an ASCII file so Git will pick up and change and will allow you to compare it in a text editor but it is not human readable so you cannot manually merge it without a hint of estimating therefore it's not a viable option. After looking through Unity forums it has been an ongoing struggle for Unity to come up with a solution that works for all accounts. In the end we decided to nominate one person to be able to change the scene at a time to prevent merge conflicts with the scene files.

Bouncing Communications:

The idea of bi-directional communication in this case is that a message will be send over the opencap protocol if a change in Marquee has occurred that needs to update Manager or a change in Manager that needs to update Marquee. The issue that we encountered was that once an update was triggered from one program there was no way to tell if it was updated from a message or internally. This causes an infinite loop, and a bouncing affect occurs. An example of this was to slide a slider and it would not be a smooth transition but bounce back a fourth because of the constant opencap messages trying to override your movement of the slider.

The best way to fix this problem would be to include a message parameter in the protocol to say whether the value was updated thought Manager or an update externally (Marquee). We could have then caught this in Marquee and ignored it. However, that would mean editing Managers core which was not viable. So, we created a funnel system in Marquee which meant that if an update was triggered in Marquee and sent to Manager. When Marquee got the reply, it would funnel that data to a different update function from the update function to trigger a TCP/Socket Message. Instead it would hit a dead end.

Conclusion:

For my first every professional development cycle of an application from day one to finish. I think it went well and the product is being used by clients and are impressed with the quality of the UI compared to the original UI/UX system. I have learnt a lot in terms of design an architecture for system and all the features that are involved in the process.

One thing that surprised me was how much using Jira affects a project. The team planning the whole way though was so clear and I felt I knew what I had to achieve for each sprint very clearly. Then if I were having issues, I could raise them in any of the mini scrums we had which prevented me from getting to frustrated with my work. This is a skill I believe to be underrated and I am going to bring it into my final year at university for any group projects.

We did a lot of testing of Marquee before it was released. However, people were finding bugs in the software which is normal for an early application as we did not think of all the use cases from day one. After a month release, we reflected on the bugs were found and decided that in a few areas we did not do the best architecture for every scenario. There is still work needed to be done on this but there are no major crashes in the system.

Overall, I am very pleased and excited that I was involved with this and now have it more clear idea of professional development.

References:

You can download Marquee yourself at https://medialon.com/ under the Marquee section.

Manager Manual: 

M515-2-Medialon-Control-System-Manual.p

Jira: (www.jira.com)

GitHub: (www.github.com)

WebGL Limitations: (https://docs.unity3d.com/Manual/webgl-performance.html)

WebGL Try/Catch: (https://forum.unity.com/threads/webgl-exceptionsupport-performance.307713/)

Unity Scene Merge: (https://forum.unity.com/threads/scene-merging.453901/)

7thsense (https://7thsensedesign.com/)