EfficienSeat

Dennis Donoghue, CSE, Matthew Donnelly, EE, Kristina Georgadarellis EE, and Aarsh Jain, EE

***Abstract*—The EfficienSeat system is proposed as a solution to address problems and inefficiencies presented by overcrowded dining halls. Through real time monitoring of individual seats via sleek, LED based, modular units on every table, users walking through the dining hall can easily ascertain seating occupancy at a glance and claim their seats with ease. Further, these units will ultimately communicate this information to users outside the dining hall via a phone application in the form of a real time map, giving a simple visual way to determine dining hall occupancy and viable seating locations.**

I. Introduction

DINING halls on our campus suffer from problems in seating that limit their overall efficiency. During busy times, incoming patrons require info on available seating. Without this info people travel up and down walkways searching for a potential open seat. Larger parties face more difficulty in finding seats. Patrons are then faced with the equally unfavorable choices of using their limited time walking throughout the entirety of the dining hall an unknown amount of times until they find a seat, or leaving and wasting the meal swipe or money that they committed to enter the hall. This also inhibits staff ability to quickly refill food and dishware throughout the dining hall. These time delays, though potentially minimal in single instances, add up and are detrimental in a food service setting.

We sought to design a system that would decrease these frustrations and time delays and explored several methods of implementation. A big factor was whether to use an active or passive system. An active system completely controls where patrons sit while a passive system reacts to where patrons choose to sit. A completely active system would be akin to a restaurant reservation system. This system would be good for maximum seating efficiency but is not suitable for a dining hall environment. Patrons would be averse to the idea of being told where to sit, especially when the dining hall is relatively empty. The sheer number of seats to manage in a dining hall also makes this approach inefficient.

On the opposite end of the spectrum we contemplated an entirely passive system, one that indicates seat status based on the seats at which people choose to sit. This system senses when someone is sitting at a particular seat and then relays this information to patrons inside and outside the dining hall. This system is favorable because patrons will not have to do anything new, however, it would be expensive and complex to implement. Our solution was a hybrid of the two approaches.

With the hybrid system in mind, we then considered the specific needs of our solution. The solution must improve time and travel efficiency for patrons and staff. To do this, it must accurately identify seating status and indicate this to incoming patrons. It must also be simple to service without interrupting the normal operation of the dining hall e.g. easy to install and requiring infrequent maintenance. The only maintenance that should be required would be battery replacement once a month for individual monitoring units. The system should be IPX4 [1] compliant to comply with food safety standards and to ensure reliable operation.

To satisfy these needs, the system will consist of nodes situated on every table in the dining hall, all reporting to a central hub. The nodes need to be small and self-sufficient in terms of power; this coincides with the low maintenance requirement, where routine "refilling" of power should be monthly. Every node will be connected to a central hub, creating a network that will reliably monitor the seating situation and ultimately report this to the user via a phone application. Table 1 quantifies these requirements.

TABLE I
REQUIREMENTS AND SPECIFICATIONS

Requirement	Specification
Table search functionality	Users can search for available seats by party size through a mobile app
App users will receive timely graphical response	Updated map displayed within 2 seconds
App users can find their tables from the map on the mobile app	Table locations will be accurate to ½ a table length
System can manage entire dining hall	Can support > 100 seats
Table unit is splash-proof and safe for use in a dining hall	Table unit is compliant with IPX4
System can accommodate non-app users	Patrons can claim seats by pressing a button on the Table Unit
Infrequent maintenance	Monthly battery replacement

D. Donoghue from Dover, Ma (e-mail: ddonoghue@umass.edu).
M. Donnelly from Springfield, Ma (e-mail: matthewdonne@umass.edu).
K Georgadarellis from Dartmouth, Ma (e-mail: kgeorgadarel@umass.edu).
A. Jain from New Delhi, India (e-mail: aarshjain@umass.edu).

II. Design

A. Overview

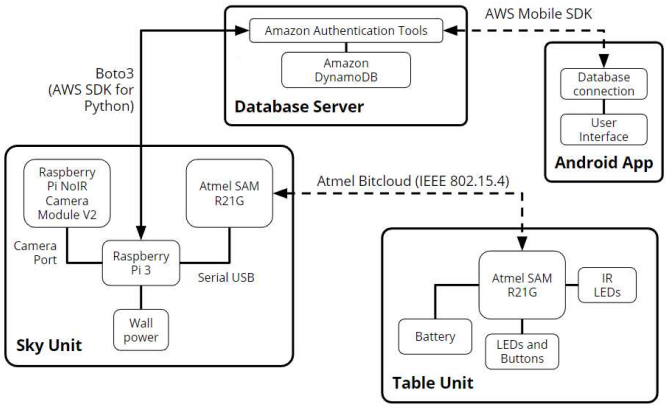

Fig 1. Block diagram of our system. There are four main blocks that make up our system: the Table Unit, Sky Unit, Database Server, and Android App.

To solve this problem, we will design a network within a dining hall that will have real time monitoring of seat occupancy and consistent reporting of this data to users. The data is accessible to users both inside and outside of the dining hall. As shown in the block diagram in Figure 1, this network will consist of four subsystems: 1) the aforementioned nodes - dubbed 'Table Units,' 2) a 'Sky Unit,' 3) a database, and 4) a phone application. The purpose of each subsystem and the interaction between them is outlined in the overview, with more detailed descriptions provided in the following sections.

Patrons of the dining hall, the users of our system, will interact with it from two points: the phone application and the Table Units. Table Units will populate the dining hall as compact units, one per table of four seats. These units provide the first point of user interaction within the dining hall: users claim seats by pressing a button, and the LEDs on the unit indicate seat status, providing a simple method of identifying occupancy at a glance. By having an easy-to-interpret system to claim seats, the issue of patrons marking their spots with valuable items like phones or keys is eliminated and seat status will be clearly marked via the LEDs, eliminating the ambiguity of a stray coat or left behind dishware.

Seat status is reported from each Table Unit to the Sky Unit, the central hub of the network. As the hub, it processes the Table Unit data and functions as the medium of communication between the Table Units and the database/app side of the system. Alongside this, the Sky Unit will also perform table localization; this information is used to accurately report seat status to users via the app.

From the Sky Unit, seat status' will be sent to the database. The database will be constantly updated in real-time by both the Sky Unit and the phone app so that it will contain up-to-date information of all seats within the dining hall.

The phone application provides the other point of user interaction with our system from outside of the dining hall. Namely, users can view a map of the dining hall with table occupancy, search for available seats by party size, and reserve seats, all from the convenience of their phone. The map of the dining hall is generated using information queried from the database for table occupancy and the localization data for the position of each table within the dining hall.

With a basic outline of the system's functionality and interactions established, the following sections delve into each subsystem more, discussing design choices and alternatives, protocols, and component interactions.

B. Table Unit

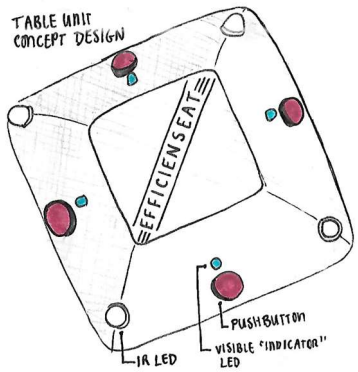

Fig 2. Conceptual drawing of the Table Unit design. The unit will be no more than 10cm by 10cm and features a pushbutton for seat confirmation and reservation, a visible LED for seat status, and an IR LED for table localization.

The Table Unit is a small modular unit for user interaction at the table itself containing a visual LED and button pair for each of the table's four seats (See Figure 2). It also contains four IR LEDs, which will be discussed later regarding table localization. The visual LEDs are used to indicate the status of the seats to the user, namely whether the seat is taken or not. The buttons are used to confirm seat reservations made from the app and to occupy and vacate seats without the app. The final product will be splash-proof (IPX4 compliance mentioned in the introduction) and easy to maintain with low power consumption. The following table shows the seat state

TABLE II
Table Unit Chair States

Chair State	LED Response
Vacant	OFF
Occupied	ON
Reserved	Slow Pulse
About to Expire	Slow Pulse

corresponding to the LED responses:

The following bullets guided us on selecting the processor to be used on the Table Unit:

- The processor needs to support a wireless network with low bandwidth and energy.
- The processor shall be able to handle at least hundred nodes on a network.
- It should be low powered to keep the maintenance of the unit to a minimum.
- Needs to also take input from four buttons and output to eight LEDs.

The SAMR21 microcontroller [2] by Microchip Technology Inc. seemed to be the right choice. The microcontroller has a System on Chip to connect the Table Unit to a network based

on IEEE 802.15.4 communication protocol [3]. This is a low bandwidth network protocol due to reduced power requirements to run the network. The stack provided by Microchip – Bitcloud [4] – is well supported for this microcontroller. Some of its features are:

- Network management of more than sixty-five thousand nodes.
- Queuing of incoming packets.
- Callbacks to confirm data received.

The project started with a few example codes in Atmel Studio Framework to better understand the overall code that would govern operation of the Table Unit. Simple network communication was established between two SAMR21 boards via their built-in RF antennas. The range of communication was established to be beyond 50 feet through testing with a wall barrier, this range is more than enough to communicate to any corner of a dining common from its center. Further testing was done including button debouncing optimization to give a smoother interface for the user. The Table Unit will be tested using a default set of commands including the following responses:

- Button press shall toggle states between vacant and occupied.
- Message received from Sky Unit shall be processed and signified on LEDs.
- Reserved state shall go to occupied when button is pressed.
- Timer state shall give a pulse on the corresponding LED.
- Data shall be sent when a button is pressed at the Table Unit which should be properly received at the coordinator in Sky Unit.

C. *Sky Unit*

The Sky Unit is composed of the Atmel SAMR21G processor and Raspberry Pi 3 board [5]. For table localization, a Raspberry Pi NoIR camera module [6] is used. These three components will all operate from wall power. The Sky Unit functions as a central hub for all the Table Units, and as an intermediate link between the Table Units and the database/app. The SAMR21G board is used to communicate wirelessly with the Table Units via the Atmel Bitcloud stack, as explained in the Table Unit section. The Raspberry Pi was chosen as the central processing system for the Sky Unit due to its power and versatility as a computer coupled with easy-to-use functionality and an abundance of available documentation. For our database needs, we ultimately decided upon Amazon Web Services (AWS) - which is discussed in more detail in Section II.D - and this influenced the functionality of our Pi. Python code using Boto 3, Amazon's own SDK for Python [7], [8], is used on the Pi to communicate data between the Table Units and the database. Given the plentiful documentation for Boto 3 from AWS, as well as the convenient use of Python scripts aboard the Pi, this communication method proved to be the most efficient way of both writing to and reading from the database with the Pi. By utilizing the Atmel Bitcloud stack and the Boto 3 SDK, we have successful communication between the Table Units and the database.

Alternatives for interfacing the Pi with the database were explored as per the database chosen; Boto 3 was our final choice as we settled on Amazon DynamoDB [9] with which Boto 3 is closely coupled, but a previous option was Amazon RDS [10] - the choice to switch databases is discussed in detail in the following section.

TABLE III
PROTOCOL

Address	Command	Data	Pin
2 Bytes	1 Byte	1 Byte	1 Byte

To communicate seating data efficiently throughout the system, we needed to design a protocol for the information being sent. To do this, we identified the pertinent information required for successful system operation, which boiled down to Table ID (with what Table Unit are we communicating?), Command Type (are we changing seat occupancy? telling the Table Unit to sleep?), and Data (seat is now occupied, Table Unit action). Our resultant protocol consists of a five-byte string, as displayed in the table below (Table III).

The first two bytes of the string correspond to the Table ID, which is assigned during initial system calibration. The next three bytes each correspond to the Command Type, the Data, and the Pin (seat), respectively. An example command for a seat reservation sent by the Sky Unit and received by the Table Unit would be as follows: "01 1 2 2," where:

- **01** corresponds to the address of the Table Unit
- **1** corresponds to a "Change Seat State" command,
- **2** corresponds to a "Reserved" seat status for the "Change Seat State" command. The meaning of this byte will change based on the command given.
- **2** indicates to alter Seat Two

Putting it all together, this protocol tells Table Unit #01 to shift Seat Two into the "Reserved" state. Visually, the LED of Seat Two will start to pulse. Figure A in the Appendix displays all of the possible values for each byte and their corresponding meanings.

To test this subsystem in terms of the Sky Unit's ability to process and transmit data to and from the server, we used our protocol to send example data to the database from the Raspberry Pi. We wrote a Python program using Boto 3 to connect to the database and access and update table information, thereby showing reliable and efficient communication between the Sky Unit and the database. This functionality will further be tested by automating the process of sending and receiving data to and from the database while also having the process occur automatically in response to changes elsewhere in the system, such as when a seat's status is changed on the Table Unit itself. The protocol itself was also tested via communication between the Table Unit and the Sky Unit, where data sent using the protocol from the Sky Unit successfully resulted in the desired change on the Table Unit.

Table Localization: Another important aspect of our system is determining the position and orientation of each Table Unit in the dining hall. We need this data for an accurate mapping of

the tables and seats on the mobile app. Without it, our system can tell the app user that a particular table and seat is open but not where the table is or which of the four seats are available. We considered using a static implementation of the Table Unit locations, with each unit being mapped to a template of the room prior to operation, but we wanted our system to have adaptability in case tables were moved, e.g., if patrons pushed two tables together to seat eight people. We chose this because it better reflects a practical implementation. Now that we know what kind of system we want, finding a suitable method is the next step.

We first thought about what our system needed to have reliable, effective, and dynamic table localization. In terms of accuracy, tables do not need to be identified down to the exact inch, but rather within at least half of a table's length. As such, minimal error is tolerated and tables are reported correctly enough that users do not fail to find their table. For dynamic response to changes in the table layout, we decided that periodic updates every hour are sufficient, since major table movement is not frequent. The implementation also must be non-intrusive to the operations of the Table Unit, i.e., not consume too much power or be distracting to patrons. Keeping these specifications in mind, we first looked at creating an indoor positioning system, however, that idea was quickly abandoned. Since the dining hall is a relatively small area, we must detect differences between signals on the order of nanoseconds. We also thought to use ultrasound, as is a common method for indoor localization, but this technology proves to be too complex and robust for our needs.

We consulted Professor Goeckel and Professor Kelly about different ways to implement table localization functionality. Professor Goeckel suggested we try to use the variation in Wi-Fi access point strength to map the room, where individual locations would have unique but relatively consistent Wi-Fi strengths based on distance, but warned that this is not always the case, even over larger distances than would be used in the dining hall. We did some tests in the Worcester dining hall by downloading software that measures the Wi-Fi signal strength at particular locations. Unfortunately, the signal strength was not consistent, and it did not provide enough resolution to be able to distinguish between one location and another location one meter away. Professor Kelly suggested that we place a unique pattern on the ceiling and equip the Table Units with small cameras that can take a snapshot of the ceiling. Depending on what piece of the pattern was shown in the snapshot, we could determine the location of the Table Unit. We considered this option but ultimately did not pursue it due to the extra cost the camera would bring to the Table Units.

The method that we are implementing uses an IR camera on the Sky Unit and four IR LEDs on board the Table Unit. The IR camera will not pick up the visual spectrum, allowing the IR LEDs to be easily seen when turned on. The position of each Table Unit will be found using the following steps: First, before the system is in service, the Sky Unit will be calibrated to the room. This is done by placing the Sky Unit where it will be mounted, in a corner of the room, and placing an IR LED at every other corner. The dimensions of the room will be known. The IR camera will take a picture of the room and then map the IR LEDs to the corners of the room. This will create a rectangular space. This space will then be divided into a grid so that the table locations can be pinpointed within it. To find the location of each Table Unit, the Sky Unit will send a command to a particular Table Unit to turn on all four of its IR LEDs. The IR camera will take a picture of the dining hall and compare this with the reference grid, finding the position of the table. The Table Unit ID and position will then be sent to the database and finally the mobile app for rendering.

Additionally, the Table Unit orientation is needed to determine which button and LED corresponds to which seat, since the Table Unit is not fixed to the table (for cleaning purposes). The Sky Unit will determine orientation by sending a command to a particular Table Unit to turn on one IR LED corresponding to one of the seats. The IR camera will then take a picture of the dining hall. From the light intensity, the Sky Unit will determine the orientation of the unit by using a reference guide to match light intensity to orientation.

D. Database/Server

Our system will feature a set of databases, each of which will store table data corresponding to a specific dining hall. These databases will be populated with information obtained from the Sky Unit regarding each Table Unit, as explained in the previous sections, corresponding to table layout and orientation, and seating occupancy status. Our app will fetch data from the database corresponding to the user-selected dining hall and use it to populate the app's dining hall map and respond to user searches. To this end, we decided to use Amazon's DynamoDB. DynamoDB is fast, flexible, and highly scalable noSQL database hosted by Amazon Web Services. DynamoDB is fully integrated with the full suite of AWS services, including identity management and security. The techniques needed to produce this part of our project were acquired both in basic coding classes and through work experience. A test of this part of our system would likely start with the basic CRUD functionality and then test the integration of the database with our greater system. If we can perform CRUD operations properly and correctly connect the parts of the system together, then the test will have passed.

Before using DynamoDB, Amazon RDS was also explored as a potential option, however it proved difficult to interact with in the way we needed. MySQL is available for simple Pi-database interaction using RDS, but at the cost of security; in order to successfully interact with the database, a lot of the security authentications need to be removed. With DynamoDB, initially connecting to the database involved a bit more work but the overall process of interacting is simpler with respect to the Pi, and more secure in terms of the connection. This led us to choose DynamoDB for our project.

E. *Android App*

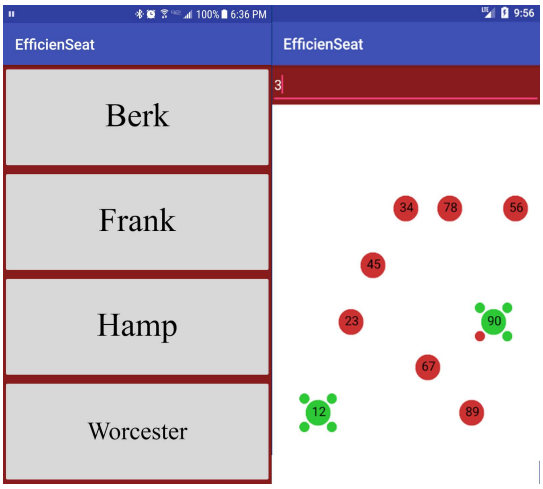

Fig 3. Screenshots of our Android app. The screen on the left allows the user to select the dining hall and the screen on the right allows the user to view and search the dining hall map.

Our app is one of the outward-facing parts of our system. The app will allow the users to browse a seating map of the dining halls and search the available tables by party size, as shown in Figure 3. Android Studio IDE and Java were utilized to write and build the app. A testing suite for this app would test the user's ability to select a dining hall, the app's ability to update the table map based on database changes, the app's ability to communicate with the Pi, and the app's ability to search the table data by party size. Proper outcomes from these tests would confirm proper app functionality.

III. Project Management

TABLE IV
MDR Deliverables

Deliverable	Percent Complete
Demonstrate basic Table Unit functionality: • Change of state via button • Communication to Sky Unit	90
Demonstrate functional app: • UI – rendered map and ability to search for seats • Communication to AWS	100
Demonstrate Sky Unit Communication to AWS	100

As per the table above, our MDR goals focused on implementing necessary functionality of our system and integration of its components. Part of this was displayed by a functional Table Unit with 4 LEDs to indicate the states of the 4 seats monitored by that unit. The states could be changed physically on the unit via a button press and wirelessly at a distance via the Sky Unit, which shows integration of Sky and Table Units and a step towards transmitting data from one end of the system (phone app) to the other (Table Unit). Integration of the Sky Unit and the server/app is currently in place with the app providing a rudimentary user experience of viewing a rendered, dynamic map of the dining hall tables, entering search queries, and viewing the response. The app also uses data from and responds to changes made in the AWS database, which can be successfully interacted with via the Raspberry Pi on the Sky Unit. These functionalities satisfy nearly all the deliverables listed.

The MDR deliverable that was only partially delivered upon was the communication between Sky Unit and Table Unit. In effect, the Table Unit and Sky Unit are successfully communicating via the two SAMR21G processors aboard either unit, but the processor aboard the Sky Unit is not yet successfully connected to the Raspberry Pi. Currently, the SAMR21G processor on the Sky Unit can successfully transmit the necessary data when received serially from a computer, but there is an issue with connecting serially to the Pi that will be solved. As the Pi functions as a central hub of sorts, having a successful connection between the Pi and the processor aboard the Sky Unit results in full connectivity of the system from Table Unit to the phone app.

Alongside this issue, table localization is the next major step to tackle in terms of its successful use, implementation, and integration with the rest of our system. We will first test out the IR camera and IR LED interaction, making sure that the camera can detect the IR LEDs from an appropriate distance (the farthest distance across the dining hall). After that is established, we will work on creating a reference grid for a room, starting out small, and then expanding for the size of the dining hall. After that, we will then test the finding of the table positions. As for orientation, we will test that light intensity is a reliable way to tell orientation and create a reference guide based on these tests. Then we will implement this in a small setting, and then expand to the dining hall setting.

There is also still overall optimization and automation to be done to further enhance the basic functionality that we have into what is necessary for the system to work as a whole. More specifically, full Table Unit functionality needs to be implemented in terms of relevant table localization aspects and complete code for all possible table states and the behavior of the LEDs to indicate said states. The code for the Pi aboard the Sky Unit also needs to be completed such that the processing, receiving, and sending of data is automated. A line of communication needs to be opened between the app clients and the Pi, so that the Pi is the only device making changes in the database, to prevent reservation conflicts. A more advanced searching functionality must be implemented for the App, as well as a handful of UI updates.

Since our project is easily broken up into components, we split our team to work on each component relatively individually. Discussion as a group, however, occurs regarding each design decision and any changes or issues that come up. Though everyone has an assigned focus on different aspects of the project, there is great team involvement to help flesh out the most efficient approach and to ensure that everyone is consistently on the same page. We hold weekly meetings with our advisor, Professor Wolf, where we discuss the project's current state, alternatives to current designs and potential issues that might arise, and next steps. As a team, we also meet one to two times per week to discuss the project amongst ourselves, to work on components together, and to ensure proper integration of our separate components. The team maintains email communication with Professor Wolf and constant communication within the team via GroupMe. Project

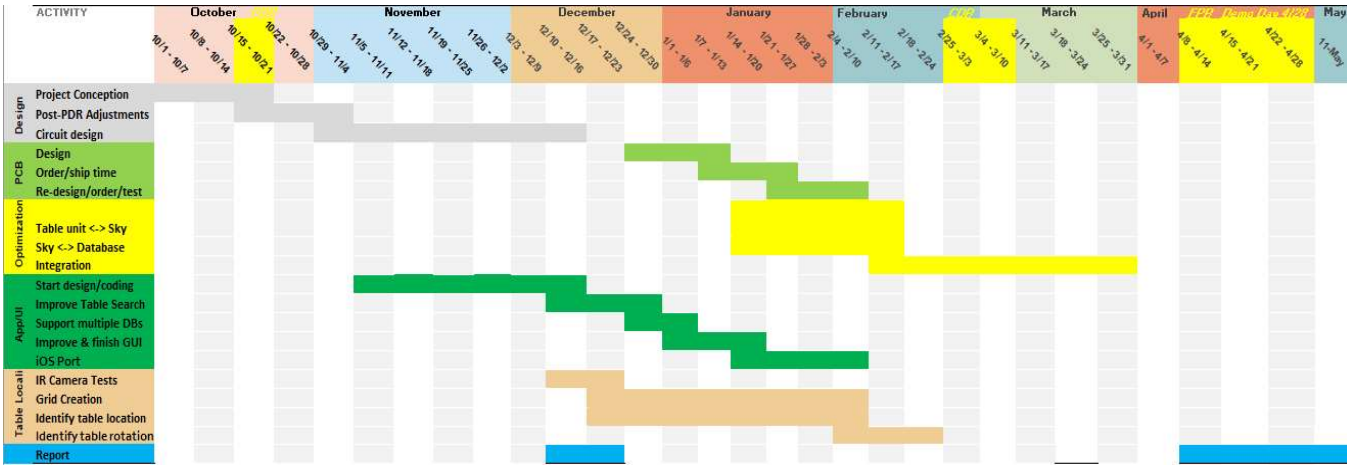

Fig. 4 Gantt chart outlining the timeline of our project management, starting at the beginning of October and ending in May.

documentation and organization is maintained on Google Drive.

Dennis, our computer systems engineer, handles phone app development and integration with the AWS server. Relevant information in terms of functionality and communication protocol are related to the rest of the team when necessary so that the rest of the system may effectively communicate with the database and app when needed. The Table Unit is being primarily handled by Aarsh, who has great familiarity with Atmel processors, code, and functionality. Matt is mainly working with the Raspberry Pi aboard the Sky Unit in terms of ensuring communication with the database and the Atmel processor aboard the Sky Unit, and processing of data that is transmitted throughout the system. Kristina is working on table localization and how this interfaces with the Sky Unit. As previously mentioned, each component has substantial input from the team as a whole to ensure that the best method to solving the problem and implementing the solution has been found and agreed upon.

Figure 4 is our Gantt Chart detailing the start of our project up to its end.

IV. Conclusion

By meeting our MDR deliverables, we have established almost complete end-to-end integration and communication. The fundamental skeleton upon which our project resides has been built and communication protocols have been determined. Piecewise, our system is integrated. We still have yet to link the Raspberry Pi to the SAMR21G aboard the Sky Unit and implement the table localization functionality, the former being trivial and the latter is in the developmental stage. Our project has taken a few twists and turns to arrive at this point. We scaled back our original idea and also sought simpler solutions to the problems we were trying to tackle. We met frequently and held each other accountable for the assigned tasks. Through this, we were able to create the framework for our project and create a clear idea of what our end goal will be. The next step is to build upon this framework and optimize power consumption for the Table Unit. We are aiming for a low power embedded system and will need to utilize the wake and sleep functionalities to our advantage. We will be drafting up the PCB schematic in the coming months so that we can have a running start to next semester. Since we are unfamiliar with the designing of PCBs, this part will prove to be a challenge for us, but starting early on the design will set us up for success. We will start working on the table localization functionality early on in the semester as well, since it is a key component in integrating the entire system. We have a few challenges ahead of us and a lot more mileage to put in before SDP day, but we are well on our way to getting there.

Appendix

A. Communication Protocol Definitions

Below are the definitions for each byte of our Table unit to Sky Unit communication protocol, as discussed in Section I.C.

ID	Command
0	Invalid
1	Change table state
2	Sleep
3	Wake up
4	Turn IR LED on

ID	Data Entry
0	Vacant
1	Occupied
2	Reserved
`3	About to expire
4	Sleep
5	Turn IR LED on

ID	Pin Name
0	Invalid
1	Seat 1
2	Seat 2
3	Seat 3
4	Seat 4
5	All Seats

Acknowledgment

We would like to thank our advisor Professor Tilman Wolf and evaluators Professors Eric Polizzi and Csaba Moritz for their very useful critiques and insightful advice that helped guide our project. We would also like to thank Professors Patrick Kelly and Dennis Goeckel for taking the time to meet with us and discuss potential options for a part of our project. Dennis Donoghue would like to thank the creators and users of the website StackExchange for boundless advice and help.

REFERENCES

[1] "IP Rating Chart | DSMT.com," DSMT.com, 2018. [Online]. [Accessed 2017].

[2] "ATSAMR21G18A - Wireless - Wireless Modules," Microchip.com, 2018. [Online]. Available: http://www.microchip.com/wwwproducts/en/ATSAMR21G18A. [Accessed 2017].

[3] "IEEE 802.15.4-2015 - IEEE Standard for Low-Rate Wireless Networks," Standards.ieee.org, 2018. [Online]. Available: https://standards.ieee.org/findstds/standard/802.15.4-2015.html.

[4] "BitCloud SDK," Atmel, 2018. [Online]. Available: http://www2.ee.ic.ac.uk/t.clarke/projects/Resources/BitCloudBitCloud SDK for. [Accessed 2017].

[5] "Raspberry Pi 3 Model B," www.rs-components.com, 2018. [Online]. Available: http://docs-europe.electrocomponents.com/webdocs/14ba/0900766b814ba5fd.pdf.

[6] "Pi NoIR Camera V2 - Raspberry Pi," Raspberry Pi, 2018. [Online]. Available: https://www.raspberrypi.org/products/pi-noir-camera-v2/. [Accessed 2018].

[7] "boto/boto3 Repository," GitHub, 2018. [Online]. Available: https://github.com/boto/boto3. [Accessed 2018].

[8] "AWS SDK for Python," Amazon Web Services, Inc., 2018. [Online]. Available: https://aws.amazon.com/sdk-for-python/.

[9] "Amazon DynamoDB Product Details - Amazon Web Services," Amazon Web Services, Inc., 2018. [Online]. Available: https://aws.amazon.com/dynamodb/details/. [Accessed 2017].

[10] "Amazon RDS Product Details - Amazon Web Services (AWS)," Amazon Web Services, Inc., 2018. [Online]. Available: https://aws.amazon.com/rds/details. [Accessed 2017].