Plant Habitat Telemetry / Command Interface and E-MIST

Uriae M. Walker

NASA's Kennedy Space Center

Major: Computer Engineering

Program: Space Grant - OR Summer 2013

Date: 29 July 2013

Plant Habitat Telemetry / Command Interface and E-MIST

Uriae Walker[1]
Portland State University, Portland, Oregon, (97201)

Plant Habitat (PH) is an experiment to be taken to the International Space Station (ISS) in 2016. It is critical that ground support computers have the ability to uplink commands to control PH, and that ISS computers have the ability to downlink PH telemetry data to ground support. This necessitates communication software that can send, receive, and process, PH specific commands and telemetry. The objective of the Plant Habitat Telemetry/ Command Interface is to provide this communication software, and to couple it with an intuitive Graphical User Interface (GUI).

Initial investigation of the project objective led to the decision that code be written in C++ because of its compatibility with existing source code infrastructures and robustness. Further investigation led to a determination that multiple Ethernet packet structures would need to be created to effectively transmit data. Setting a standard for packet structures would allow us to distinguish these packets that would range from command type packets to sub categories of telemetry packets. In order to handle this range of packet types, the conclusion was made to take an object-oriented programming approach which complemented our decision to use the C++ programming language. In addition, extensive utilization of port programming concepts was required to implement the core functionality of the communication software. Also, a concrete understanding of a packet processing software was required in order to put all the components of ISS-to-Ground Support Equipment (GSE) communication together and complete the objective.

A second project discussed in this paper is Exposing Microbes to the Stratosphere (E-MIST). This project exposes microbes into the stratosphere to observe how they are impacted by atmospheric effects. This paper focuses on the electrical and software expectations of the project, specifically drafting the printed circuit board, and programming the on-board sensors.

The Eagle Computer-Aided Drafting (CAD) software was used to draft the E-MIST circuit. This required several component libraries to be created. Coding the sensors and obtaining sensor data involved using the Arduino Uno developmental board and coding language, and properly wiring peripheral sensors to the microcontroller (the central control unit of the experiment).

Nomenclature

APH	=	Advanced Plant Habitat
CAD	=	Computer Aided Drafting
E-MIST	=	Exposing Microbes to the Stratosphere
EXPRESS	=	Expedite the Processing of Experiments to Space Station
GPGGA	=	Global positioning system fixed data
GSE	=	Ground Support Equipment
GUI	=	Graphical User Interface
IC	=	Integrated Circuit
IDE	=	Integrated Development Environment
IP	=	Internet Protocol
ISS	=	International Space Station
LED	=	Light Emitting Diode

[1] KSC NASA Intern, NE-A1, KSC, Portland State University.

PH = Plant Habitat
STELLA = Software Toolkit for Ethernet Lab-Like Architecture
VAB = Vehicle Assemble Building

I. Plant Habitat - Introduction

With NASA expanding it horizons to deep space missions reaching asteroids and Mars, its experiments, not to be left behind, are also breaking new ground. Plant Habitat, a NASA experiment to be sent to the ISS in 2016, takes the ambitious steps of growing and harvesting plants in space. The experiment will be kept in the U.S laboratory of the ISS, and sit as a large quad locker payload on an Expedite the Processing of Experiments to Space Station (EXPRESS) rack. It "'is the first Kennedy-led space station payload of this magnitude'", explains Bryan Onate, APH Project Manager[1]. It is also the first payload of its kind; PH boasts a complete plant growth chamber equip with environmental sensors, pumps, and LEDs. These features give Ground Support Equipment (GSE) complete control of the experiment as well as the ability to monitor experiment variables desired for scientific research. It is an exciting time for KSC as Plant Habitat and its PH team push the limits of space exploration by innovating new solutions for sustainable life in space.

(Photo taken from www.nasa.gov)

II. PH Software

The ability of Ground Support to send control commands to PH and to receive telemetry from PH on the ISS is an integral part of the experiment. Managing these processes is a concern of the software component of PH, and will be the focus of this paper. Commands allow Ground Support to adjust PH moisture levels, and light levels, for example, ultimately insuring that optimal levels for plant growth are sustained inside PH. Science Telemetry allows scientists to analyze the effects zero gravity on plant growth, and the feasibility of plants as a food source during deep space missions and beyond. Relaying these critical commands and telemetry is accomplished by encapsulating this information in Ethernet packets and transmitting them between PH, on the ISS, and the GSE.

III. STELLA

NASA, departing from standard IP/Ethernet standards, appends NASA unique headers to packets, adding a level of complexity to the process of transmitting information from GSE to PH. The Software Toolkit for Ethernet Lab-Like Architecture (STELLA) software developed by Boeing, and newly introduced to the KSC PH software team, simplifies the packet passing process by taking care of these NASA packet headers. This gives the PH team a vested interest in understanding and utilizing the STELLA software.

A. Overview of STELLA

The purpose of STELLA is to make communication between the ground software and the ISS software act as if it were taking place between two lab computers. STELLA does this by handling NASA's unique packets headers which get appended to packets when they travel between the ISS and ground. Figures 1 and 2 below illustrate this concept:

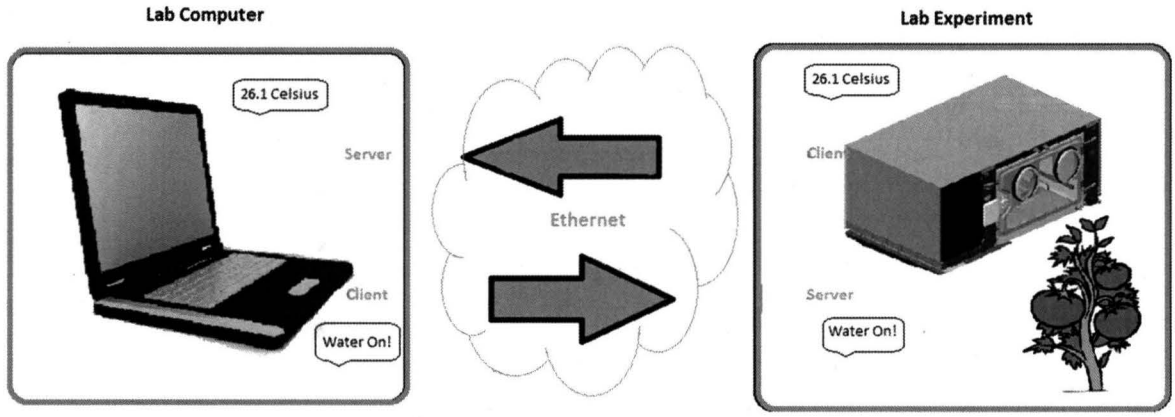

Figure 1. Communication in Lab Environment

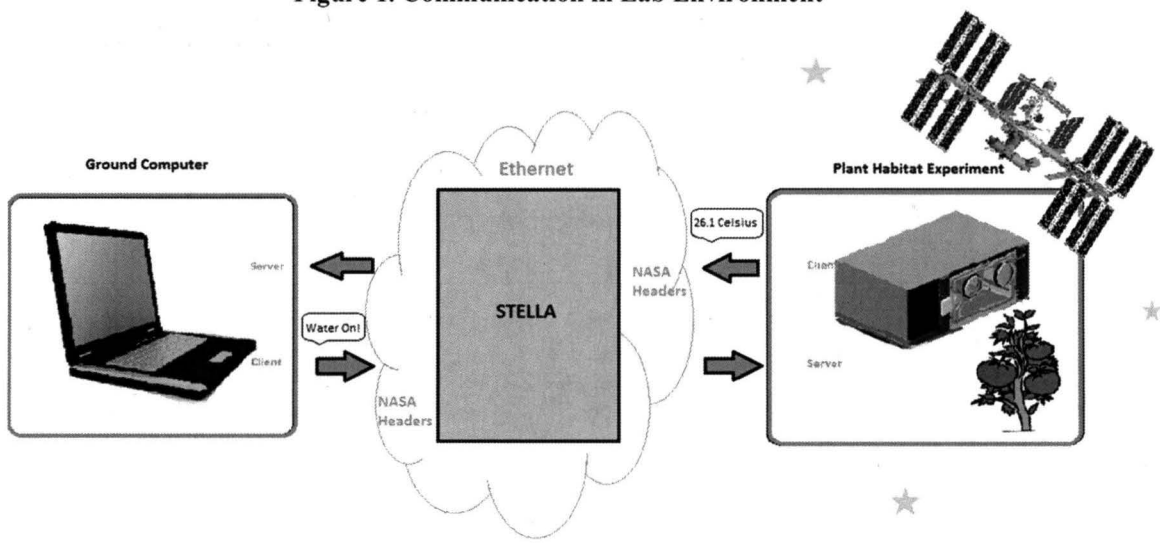

Figure 2. Communication with STELLA

The first figure shows communication taking place in a lab environment. The second shows communication taking place with STELLA. In Fig. 2, STELLA is sitting between ground and the ISS, handling NASA headers, and emulating communication in a lab environment (Fig. 1); experimenters can simply send and receive messages just as they would during regular Ethernet communication. This allows focus to be placed on the experiment rather than on specialized ISS communication protocols and packet recovery – the main point of STELLA.

B. Testing STELLA

Insuring that STELLA operated as advertised required systematic testing. A major aspect of this was verifying that packets sent though STELLA routed to their correct destination without any alternation of packet contents. Verifying this involved the use of a networking tool that could read and write to network connections.

Testing was done by mimicking a client and a server with a networking tool and attempting to send packets (messages) across STELLA. If packets were observed traveling through STELLA and being correctly received from client to server, it could be concluded that STELLA was operating as expected. An illustration of this testing is depicted in the figures below:

Figure 3. Client Sending Message

Figure 6. View of Packet in STELLA Supplementary Software

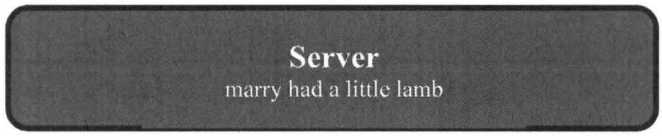

Figure 5. Server Sending Message

Here, Fig. 3 shows a client sending a packet containing the message "marry had a little lamb" to a STELLA port. Fig. 4 illustrates the packet traveling through one of STELLA's supplementary software components. The final figure shows a server listening on a STELLA port and grabbing the message that had passed through STELLA. In summary, what can be observed here is a client successfully passing packets to a server across STELLA. This observation lead to the conclusion that STELLA was operating as advertised.

IV. Communication Software

The message relaying software component of PH encompasses three major components at its core. These software components are STELLA (described above), the communication software, and GUI application (described later). The communication software serves the purpose of grabbing packets which have been transported through STELLA, processing them, and passing them to the GUI application.

A. Language Choice

The communication software needed to be robust and compatible with existing software infrastructures. These factors were the motivation for choosing to program the communication software in C++.

C++'s object-oriented characteristics makes it a robust and powerful language well-suited for the task of handling packet transfers between the GSE and the ISS. A diverse range of packet formats were forecasted to be produced by PH. Utilizing C++'s object-oriented nature would enable the varying packets to be organized into classes according to their types. Additionally, with C++, actions specific to each packet type could be encapsulated into unique functions and placed into classes to further modularize packet-related operations. Having this structure in place sets the stage for a robust communication program capable of processing and passing the multiple packets that would stream between the GSE and the ISS.

C++ is one of several object-oriented programming languages; other object-oriented languages could have also provided the structure described above. However, the code managing the various control devices of the PH experiment was written in C++ (or a C-based language). Therefore, C++'s characteristic of being compatible with existing software infrastructures combined with its characteristic of being object oriented lead to the final decision of programming the communication software in C++.

B. Basic Operation

The primary requirement of the communication software is to send and receive packets to and from the ISS to the GSE. Seeing that these are common networking operations, an experienced port programmer would recognize that, an adaptation of the common client-server model is suited for carrying out this requirement.

The client server model is an application layer communication model. It is used by different applications to talk to each other over the internet. To send a message, the sender requires data to send and addressing information of the application to which to send the data (in some protocols, connection establishment is required as well). Adapting this model was appropriate for the straightforward task of sending commands and receiving telemetry.

A client, specifically, is an application that sends a message to an application usually with the intent to receive data and or a service. A server is the application which listens and "services" these requests made by "clients" by sending a response; a response can be anything from a text string to a video stream. The communication software is an adaptation of the fundamental client-server structure. It combines the ability of the client to send data and the server to receive data to effectively send command packets, and receive telemetry packets.

C. Design Challenges

There were two distinct design approaches discussed during formulation of the communication software. One approach discussed was a single application that combined the communication and GUI software. The second approach discussed was standalone communication software that separated the GUI and communication software. The standalone design was chosen despite it being more complex because of the extra layer of protection this design provided.

Separating the GUI software and communication software into two applications increased the complexity of the communication software considerably. This approached required the communication software to simultaneously handle both data from STELLA and the GUI application requiring implementation of an advanced programming concept called threading. It also required a method for transferring data between the GUI software and the communication software to be developed. Several data transfer strategies were considered; however, a port programming method for data relaying was ultimately chosen. This decision resulted from the fact that a client-server structure was already put in place during design of the interface between STELLA and the communication software. Replicating this structure was more practical than devising and implementing an entirely new system.

Although separating the two applications resulted in an increase in complexity, it also added a layer of protection to the PH experiment– a major advantage. The GUI application may be developed in an open source IDE which would require release of source code. Separating the communication software from the GUI relieves the communication software from the requirement to release code.

D. Final Design

The final design of the communication software was multithreaded client-server program architecture. An illustration of this architecture is shown in Fig. 6 below:

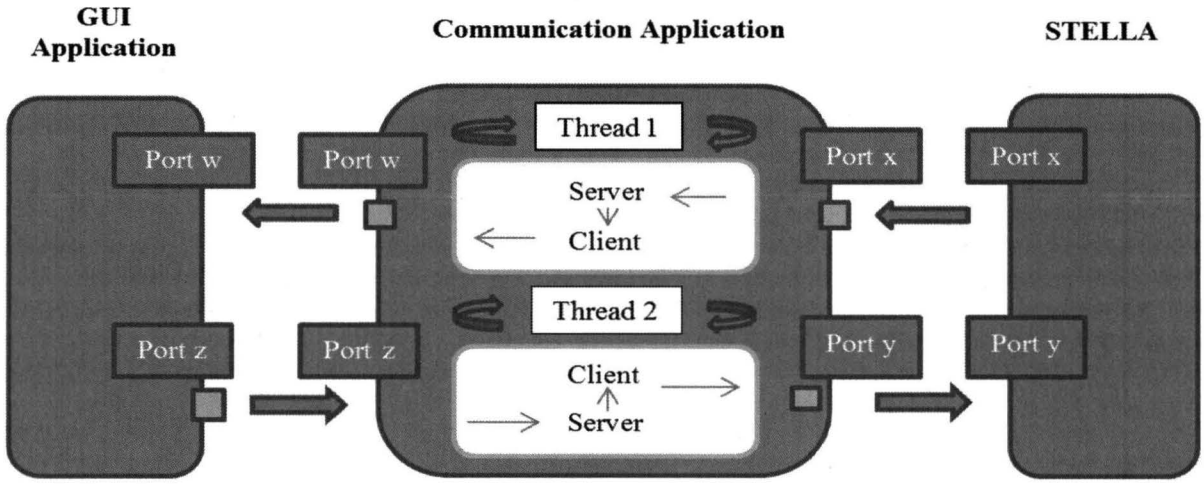

Figure 6. Communication Software Architecture

Shown here is the communication software sitting between STELLA and the GUI application. It acts primarily as a packet passing and processing application. The top data stream carries telemetry packets (packets are depicted by orange filled boxes) from STELLA to the communication application to the GUI application. The bottom data stream carries command packets from the GUI application to the communication application to STELLA. The two data streams and the packets within these streams are each handled by a designated thread (thread 1 and thread 2 in the figure). In each thread, there is a server application listening for incoming packets, and a client structure ready to accept and send the packets received by the server to its neighboring application. It is important to note that a transformation procedure will be applied to the data within the packets before the client retransmits the packet to the next application. This is done, in part, to promote data abstraction. Data abstraction is a technique that involves separating the application interface from the implementation. For example, when driving a car, the user needs to know how to execute tasks such as starting the car, and steering the car. He does not, however, need to know the detailed mechanics of the car in order to drive it. This model of separating the user interface from the internal mechanics is common in programming and is emulated in the relationship between the GUI application and the communication software.

V. GUI Software

The GUI application is the front-end software utility that serves as the interface between users and the PH experiment. Through the GUI, users can control various aspects of the PH system such as water delivery. When a user presses a button on the GUI, a packet containing command information is sent through the communication software and through STELLA to the PH experiment. In this way users can modify PH environment variables. Users can also observe PH environmental conditions through text fields that are filled as the GUI application receives science telemetry packets from PH.

A. GUI Expansion

The GUI application prototype, developed in an open source IDE, stands as a conceptual user interface template with inactive buttons and textboxes. The application lacked an infrastructure to communicate with the PH experiment. In addition, the buttons and text boxes lacked the capability to send meaningful commands and display downlinked data, respectively. A communication infrastructure needed to be put in place, and the buttons and text boxes need to be programmatically connected in order to make the GUI application fully functional.

The figure below is an illustration of the infrastructure implemented in the GUI application to facilitate communication with the PH experiment:

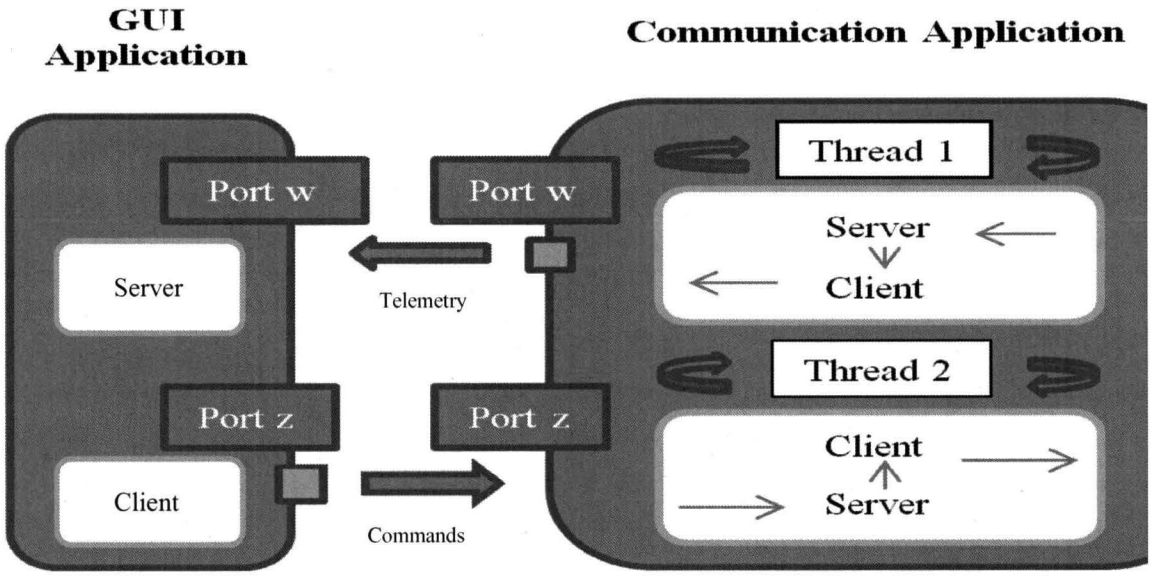

Figure 7. Communication Software and GUI Application Interface

Fig. 7 above illustrates a server and client infrastructure integrated into the GUI application. Adding this infrastructure gave the GUI application the ability to receive downlinked telemetry and uplink command data. The client-server model was chosen because it was implemented as the communication system between the communication software and STELLA, and replicating this infrastructure was easier than designing and implementing an entirely new data transferring strategy.

To activate the GUI buttons and text boxes, an interrupt-handling mechanism was utilized. An interrupt is a signal that notifies a program that an event is waiting to be handled. An event can be anything from the removal of a USB stick to the arrival of an Ethernet packet on an application port. Once a program receives a notification it will switch contexts and execute the code tailored to handle the particular event. The code segment that executes to address the interrupt is called the interrupt handler or callback function.

In code, interrupts can be programmatically connected to events like button pushes. By associating a button push with an interrupt handler that executed code to send a command packet, the task of activating buttons was accomplished. Similarly, by connecting the arrival of telemetry packets to an interrupt that passed execution to a handler that executed code to carry data to a textbox, the task of displaying downlinked telemetry was accomplished. Through implementing a mechanism to send and receive data, and to activate the buttons and text boxes, the ground work for a fully functional GUI application was laid.

VI. Introduction - E-MIST

The E-MIST project aims to access atmospheric effects on microbes that naturally occur in the stratosphere. It will do so by launching a massive balloon carrying a payload containing these microbes into the stratosphere. To observe the effects on the microbes at varying altitudes, the microbes will be placed on rotating skewers which will expose the microbes at a range of atmospheric levels. This process will be controlled by a central microcontroller which will take readings from peripheral sensors such as GPS and altitude sensors, and determine when to expose the microbes.

In order for this system to be realized, the control circuit and logic of the system needed to be developed. This entailed drafting the circuit schematic (an initial drawing complete with circuit connections was already completed) showing the connections of the electrical components, and programming the microcontroller to communicate with the peripheral devices. Drafting of the E-MIST circuit was done in Eagle CAD[2]. Developmental programming of the peripheral sensors was done though utilization of the Arduino[3] development platform.

A. Eagle CAD

Eagle CAD is a circuit-drafting software capable of producing both a schematic and board view of a PCB (Printed Circuit Board). The software is precise enough that files produced from Eagle can be sent to board manufactures to produce complete PCB complete with etches.

E-MIST required a schematic of its microcontroller and connected peripheral devices to be drafted. Below is a screen capture of an example Eagle Schematic View (this is not the actual schematic produced for E-MIST):

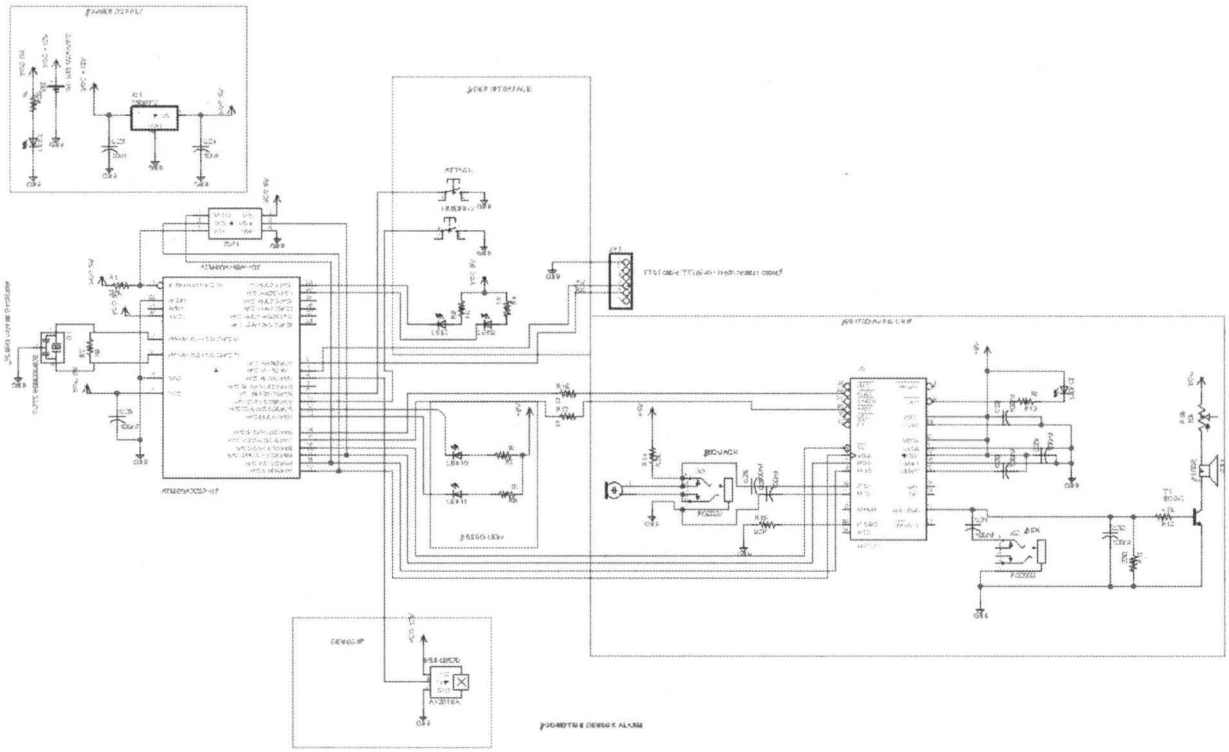

Figure 8. Eagle CAD schematic View

The Eagle CAD schematic view uses actual part libraries, and allows users to draft accurate representations of circuits, down to each individual wire connection. Shown here is a central microcontroller (large IC on left) connected to several peripheral devices. E-MIST had a similar design where a central microcontroller was connected to devices including a temperature sensor, an altimeter, a GPS module, and a light sensor. From the schematic layout an Eagle board layout can be produced. Below is a screen capture of an example board layout (this is not the actual E-MIST board layout):

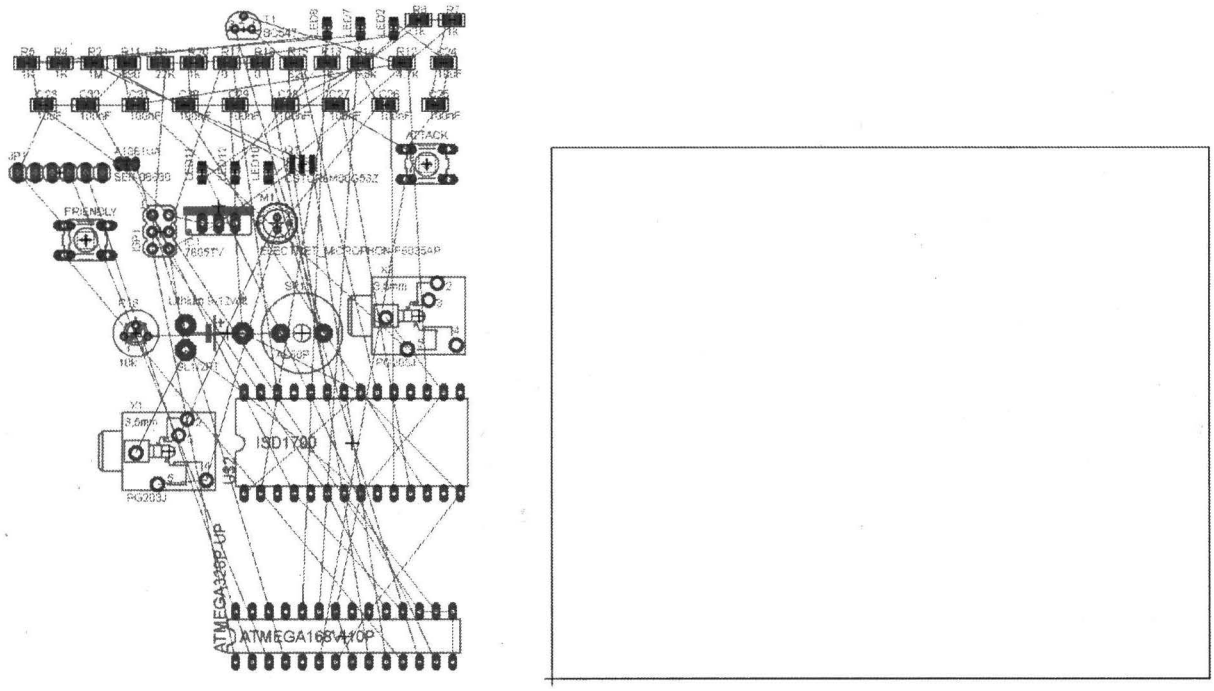

Figure 9. Eagle CAD Board View

The components shown on the left are to-scale footprints of each component drafted in the schematic layer. This view allows you to design the physical layout of the board. Many of the parts shown required custom creation. This process involved obtaining measurements from device datasheets and drafting precise device footprints. A screen capture of this process is shown in Fig. 10 below:

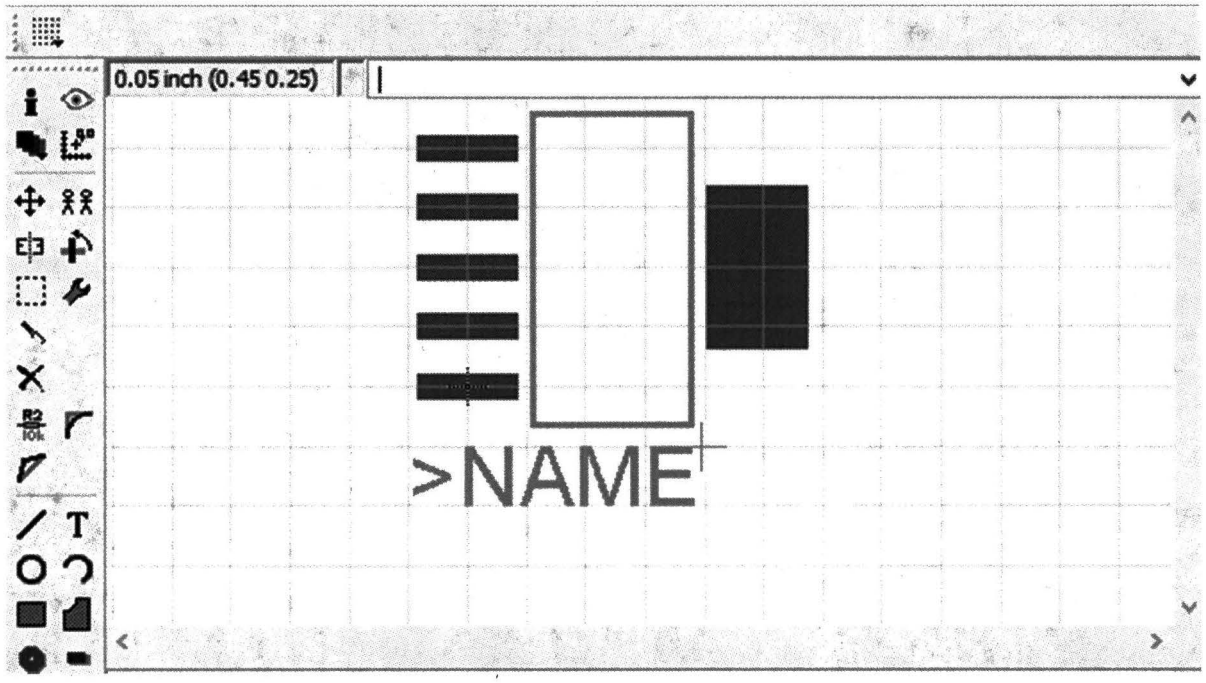

Figure 10. Eagle CAD Custom Component Creation

Shown here is the outline of a voltage regulator chip (middle gray square) and its surface mount pads. The component was sized and spaced according to device manufacturer specifications with aid of the tools shown at the left and top slides of the screen. An accurate footprint is crucial in ensuring proper connection and placement of components on a PCB. If a device foot print is inaccurate, it can lead to shorted connections and other undesirable outcomes.

Although complete drafting of the physical board layout was not required, the groundwork is laid if a complete PCB is needed to be drafted. The schematic of the board, meanwhile, serves as a useful reference for understanding the connection and basic capabilities of the E-MIST circuit.

B. Arduino and GPS module

A GPS sensor was one of the numerous sensors required to be interfaced with the central microcontroller. The objective in programming the GPS module was to obtain time, and altitude data.

Accomplishing this required interfacing the GPS module with the Arduino microcontroller board. It also required writing Arduino code to extract altitude and time information from the data stream sent from the GPS sensor to the microcontroller onboard the Arduino. (Source code that provided a frame work to receive GPS data streams was already developed). Extracting altitude and time information from the GPS data stream required obtaining a basic familiarity with GPGGA formats[4], and parsing the GPGGA string. Once the format of the data stream was understood, programming structures and concepts such as array buffers, baud rates, and loops were used to parse and extract the desired data from the stream.

VII. Conclusion

My experience at NASA has been extremely rewarding. I was able to work on two amazing projects, PH and E-MIST, that gave me the opportunity to significantly strengthen the skills that I had only begun to develop in school. I also had the opportunity to be a part of a great team that made my internship at KSC a memorable experience I will cherish.

The PH software project required implementation of various high-level programming structures including port programming, threading, interrupt handling, and GUI programming. Working with these structures were skills that were at the very outer edges of my abilities. So, I am still ecstatic, and in disbelief over the fact that I was given a chance to improve upon all these skills at once. However, what more is that I was able to do so while working on an exciting real life application whose objective, facilitating research on plant growth in space, is out of this world.

On top of this I was able to explore historically significant buildings while working on projects. Just when I thought programming a microcontroller would expose microbes into the stratosphere was cool enough, it got better! In working on E-MIST / MIST, I was given the chance to participate in a balloon launch from the top of the Vehicle Assembly Building (VAB) roof! It was amazing to be able to see the launch pads in the distance while working on such an interesting project. This was one of the coolest experiences in my life.

In addition to all this, in being a part of the NE-A1 team I gained solid experience working in a team environment. Moments that stood out where the times when I met closely with my teammates and discussed project strategies. These moments were not only enjoyable, thanks to amazing mentors and teammates, but rewarding as well, because it gave me an opportunity to improve my communication and listening skills. All in all, being a KSC intern and part of NE-A1 was an absolutely wonderful experience; I will never forget it!

References

[1]Granath, Bob. "Kennedy Engineers Designing Plant Habitat for Space Station." *nasa.gov*, NASA, 19 Feb. 2013. Web. 25 July 2013. http://www.nasa.gov/mission_pages/station/research/news/plant_habitat.html#.UfEiuI3VA1K

[2]"EAGLE." *www.cadsoft.com*, CadSoft, 2011. Web. 25 July 2013.

[3]"Arduino." *www.arduino.cc*, Arduino, n.d. Web. 25 July 2013.