COLORADOSCHOOLOFMINES
EARTH • ENERGY • ENVIRONMENT

CSCI 370 Final Report

The Epic Wizards

Elijah Cook
Alyssa Hanson
Matt Boughey
Alex Torres

Revised June 16, 2024

Logo adapted from [1]

CSCI 370 Summer 2024

Tree Lindemann-Michael

Table 1: Revision history

Revision	Date	Comments
New	5/16/24	Created sections I. - V.
Rev – 2	5/26/2024	Created section VI.
Rev – 3	6/9/2024	Created sections IX. - XIII.
Rev - 4	6/16/2024	Final revisions

Table of Contents

I. Introduction

Dr. Owen Hildreth and his research group are working on a Nano-Inkjet Printer and would like to expand its capabilities. Currently, the syringe pump that injects materials into the printer operates manually and lacks integration with the printer's interface. Our task is to develop a Swift package that controls the pump and then integrate this package into the desktop app, Fifi, where we will develop an interface for the pump.

Fifi is a desktop macOS application that Dr. Hildreth and students working for his lab have developed to control multiple lab instruments in unison. Before our work, the app controlled the XPSQ8 stage controller, the waveform generator, and a multimeter.

II. Functional Requirements

Using Swift and SwiftUI, the following must be accomplished:

A driver package must be developed to enable command control over the syringe pump via Ethernet. This package must support essential functions such as starting, stopping, and pausing the pump. Additionally, it should allow the configuration of important operational settings, including flow rate, syringe diameter, and units. These functionalities ensure adequate control in operating the syringe pump.

Once developed, this Ethernet driver package must be integrated into the PrinterController package to ensure it works with the existing system architecture. The integration involves ensuring that the driver package communicates effectively with the PrinterController's existing components and follows the system's protocols for operation. This integration is crucial for keeping the system stable and making sure the new features function correctly without causing any errors or issues with other aspects of the existing package.

Afterwards, the enhanced PrinterController package must be incorporated into the existing Fifi app. This process includes creating a graphical user interface (GUI) to facilitate the new syringe pump operations. The GUI must provide clear and intuitive controls for the start, stop, and pause functions, as well as the configuration settings. This requirement ensures that end-users can easily access and manage the new syringe pump capabilities through the Fifi app.

There are also several stretch requirements to consider for further enhancing functionality. One of these requirements is the development of a Fifi operation capable of printing a square. This functionality would likely involve sending commands for precise movement patterns and synchronization along with the syringe pump's flow to create a square. Another stretch requirement is to support non-constant flow rates for the syringe pump, such as implementing

flow rate ramps. This feature would allow the flow rate to vary in relation to the time dispensed, providing more complex and precise application scenarios.

III. Non-Functional Requirements

In addition to the essential functional requirements, several non-functional requirements must be met to ensure the client's satisfaction. Reliability is one of the most critical requirements to this project; the system must consistently perform without failures, which will maintain user confidence in its functionality.

Usability is another important non-functional requirement. The GUI should be intuitive and easy to navigate, allowing users to perform operations and configure settings without needing any training outside of learning how the syringe pumps operate. Compatibility is also crucial, ensuring that the integrated PrinterController and Fifi app work seamlessly across various hardware and MacOS environments.

Finally, maintainability is essential for the system's longevity and efficiency. The system should be easy to update and maintain, with clear documentation provided for developers. Additionally, clear instructions should be available for configuring the Ethernet converter to connect to the syringe pumps.

IV. System Architecture

Technical issues:

We had one major technical issue and that was the interface we used to communicate with the syringe pump. We were provided code that could interface with the pump using an rs-232 serial port, and we were tasked with creating a library that could operate the device over ethernet via an ethernet to rs-232 adapter. We received the adapter and we were able to connect to the adapter itself over ethernet using a TCP connection, however, any communications sent to the adapter were not received by the pump. After many days of troubleshooting, the group resolved to try one last adapter that would invert the transmit and receive wires to the pump. We did not immediately have access to the adapter needed, so we used a breadboard and jumper wires to create our own. This adapter was able to swap the TXD and RXD successfully, and established communications with the pump.

System architecture:

Our basic system architecture is below in figure 1:

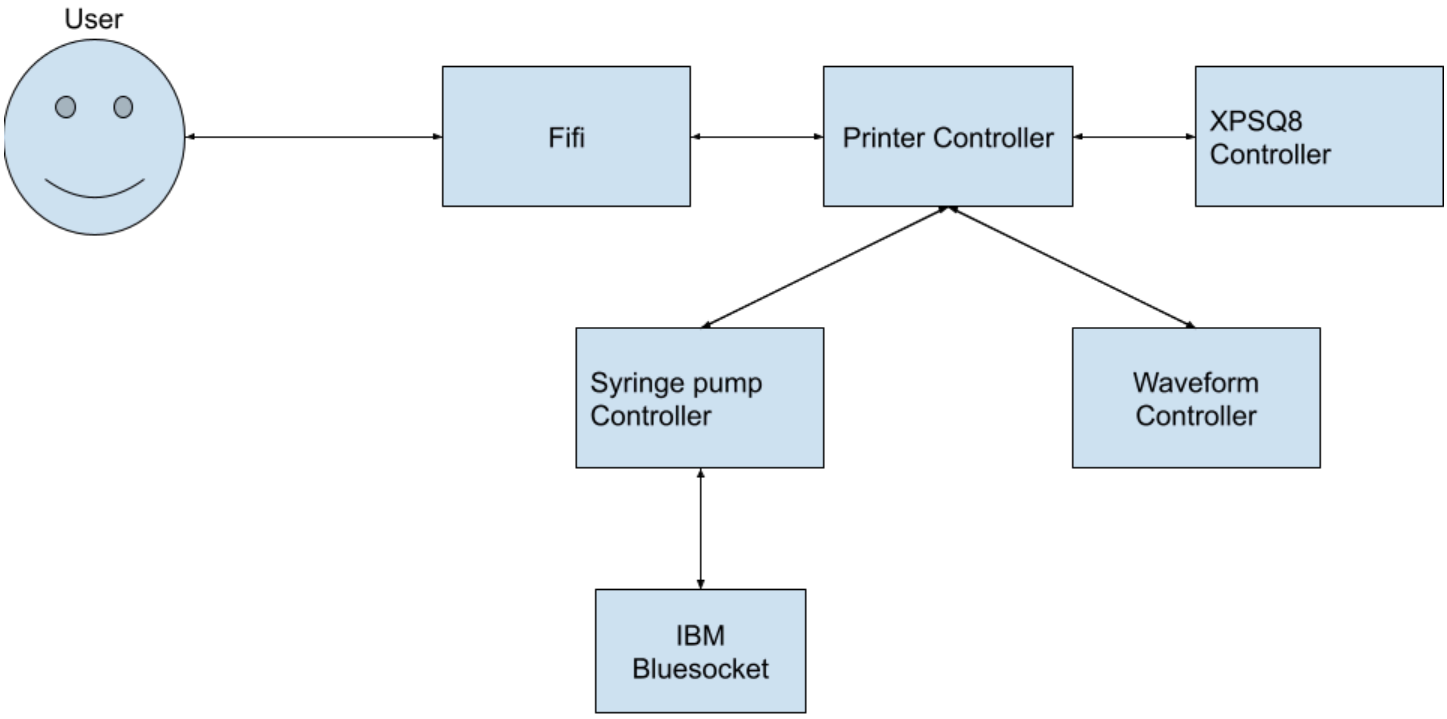

Figure 1: System architecture for the Fifi App

Most of the system architecture was specified to us by our client. The entire project uses Swift. Each part of the architecture is in the form of a Swift package except Fifi. Fifi is a swift app using SwiftUI for its front end. The printer controller is the back end data model which handles the entire printer at a high level. The XPSQ8 controller is used for moving the build plate of the printer and we did not add any functionality to this part of the project. The waveform controller is used for controlling the voltage of the print nozzle and we did not add any functionality to this part of the project. The part of the project we did work on was the Syringe pump controller. We created this controller using the IBM Bluesocket Swift package which handles ethernet connections. We used a TCP connection as mentioned later in "Technical Design".

After establishing the new syringe pump controller we integrated its features into the printer controller and updated both the manual controls and the operation queues. After these back end changes were made, we added the functionality to the UI from within Fifi.

As mentioned before, there are parts of this project which we did not interact with. The system architecture for the entire system is more complicated than shown in our diagram, but for the sake of clarity we abstracted away the waveform and XPSQ8 controllers.

V. Technical Design

RS-232 to Serial Conversion

Our system for connecting syringe pumps involves multiple converters chained together. One of these converters is of more note than the other, that being the RS-232 serial to ethernet converter. Because RS-232 and ethernet are

significantly different protocols that use different wirings, voltages, and baud rates, a special powered converter is needed to convert between the two. The converter our client chose is a small computer capable of a network connection over ethernet which repeats all of the information it receives out to its RS-232 port. We configured what kind of connection the converter uses, opting to use TCP/IP server mode. We chose TCP/IP for its reliability compared to UDP, considering that we want all commands sent to the printer system to be interpreted reliably. The choice of server/client was mostly arbitrary and boiled down to ease of use and configuration. Another important part of the configuration which was discovered after many hours of testing was the need for a null modem adapter. The RS-232 protocol is asymmetric, and devices are designed to be on a specific end of the connection. When two devices that are meant to communicate from the same end are connected, an adapter called a null modem must be used to connect them so that the transmit and receive pins of one device correspond to the opposite pin of the other device. These types of connections are called terminal and non terminal connectors, and have slightly different pin assignments. The pin layouts can be seen in Figure 2. Figure 3 shows a null modem that we constructed as a proof of concept before receiving a commercially manufactured null modem. The makeshift null modem was constructed with a breadboard and jumper cables.

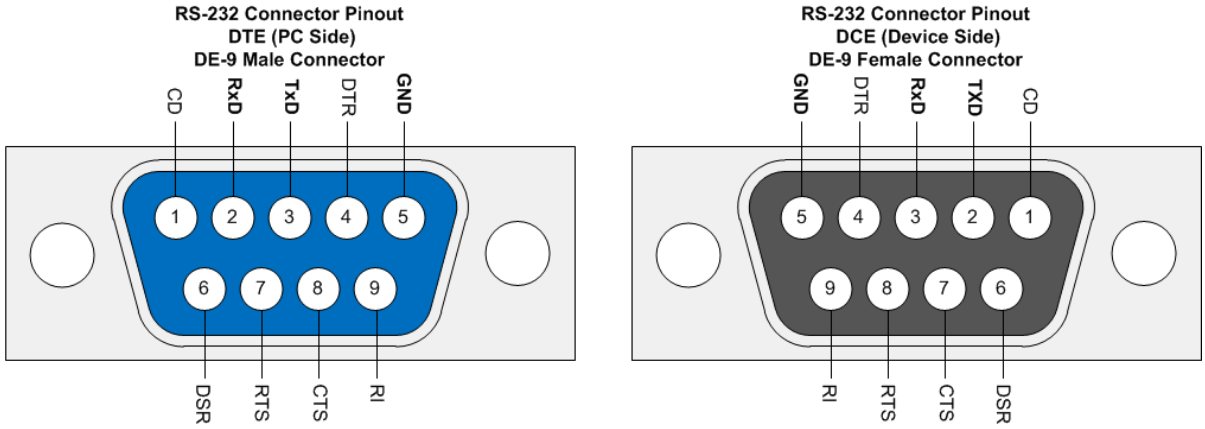

Figure 2: RS-232 PC (Terminal) vs Device (Non-Terminal) Pin Assignments

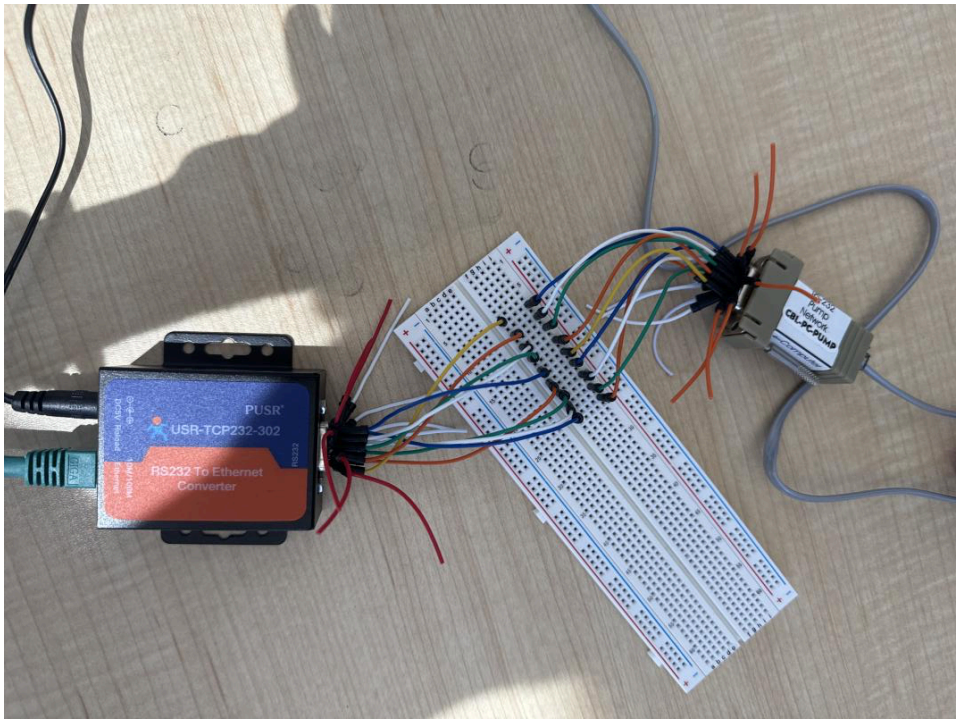

Figure 3: Improvised Null modem connecting the RS-232 to ethernet converter (left) and syringe pump connection (right)

In addition to the physical adapters needed for the conversion, there are also considerations made in our code so that communication with the pump would work properly. One of these considerations is the baud rate of both the syringe pump and the RS-232 to ethernet converter. In order to ensure consistent and reliable communication, these baud rates must be the same. Also, because the RS-232 baud rate is so slow, we must take care of the timing of sending data to and receiving data from the pump. To read data from the pump we must first send a query command, then make the thread that is going to read the data. This involved both using asynchronous commands to do multiple things at once, as well as creating short pauses on the main thread.

Asynchronous programming

Another point of struggle besides establishing a connection to the syringe pump was use of asynchronous operations in the code and the interaction of these with the UI. Because we were working with a GUI and because Fifi needs to control multiple instruments at once, it was vital that we made use of concurrency. The UI needs to be able to update while a command is sent to the syringe pump and a different command is sent to the waveform controller. Swift deals with concurrency by using actors. Actors control their own state and interact with each other via messaging. Only an actor may access its own state. This model helps avoid many pitfalls of concurrency such as race conditions. However, this model led to problems when trying to implement the manual controls for our UI. When trying to implement an on/off toggle we wanted to store information about the syringe pump in the printer controller. This is a logical thing to do since the printer controller already contains the syringe pump controller, this would keep things modular and compartmentalized. However, because the printer controller is an actor, only it is allowed to access its state. When trying to access information about the pump to control the UI (i.e. knowing whether the toggle should start or stop the pump) the code would not compile. The UI is not allowed to access printer controllers private state. The solution to this was to keep the syringe pump state that is needed by the UI in the UI and have the functions and the rest of the back end state for the pump in the printer controller. A diagram of this behavior is shown below in figure 4.

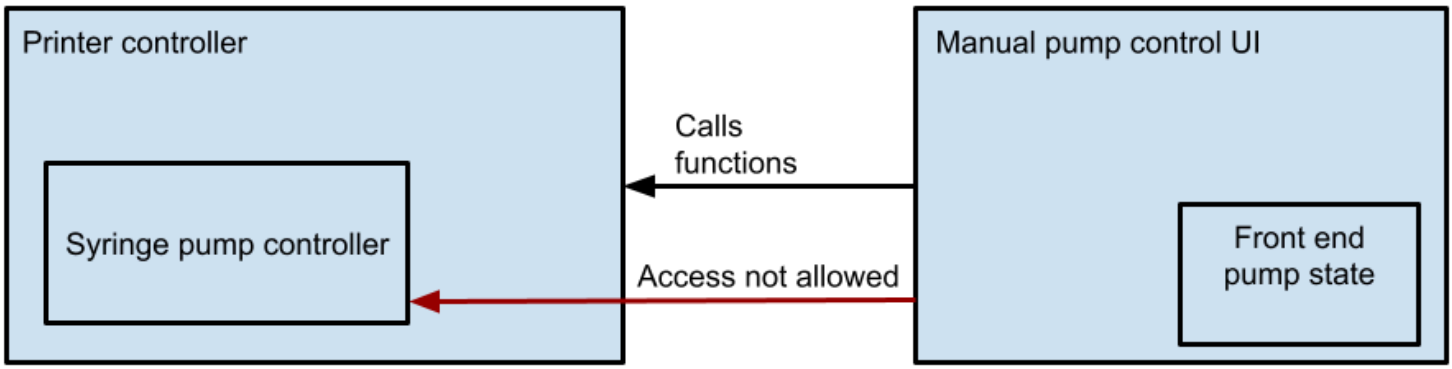

Figure 4: UI and syringe pump control interactions.

This ultimately forced us to take a more functional approach instead of an object oriented approach where the printer controller was exposed only in the form of function calls.

VI. QA

We were explicitly told by our client not to write unit tests. In addition to this, testing of the UI system and of the instrument controls does not lend itself to unit tests. To verify these systems, we had to test them by hand. Due to these factors, our quality and testing relied heavily on functionality, as well as well-defined documentation of the final product. The measure of quality on our project was determined by whether or not the software was functional, meaning that all necessary functions, UI, and integration were accomplished successfully. The integration was our most important

qualitative measure, as the end goal of the project was to create a driver package that could be integrated into the Fifi application.

The first type of testing we did was scenario based testing. This included running through several different scenarios the team had come up with to guarantee our program behaved as expected and had no unexpected crashes. Some of these scenarios included unplugging a pump, stopping a pump mid cycle, power loss recovery, and error handling.

Alongside scenario based testing, we did user acceptance testing. This was an ongoing process for the team that involved consulting the PhD student who used our app the most frequently and getting feedback on the layout of our UI. Through this testing, we were able to make small changes throughout the design process, rather than having to tackle numerous or substantial modifications all at the end.

VII. Results

The overall goal of integrating ethernet based syringe pump control into the Fifi app was successful. We were able to create two different operations for the pumps, a settings operation which configures the pumps when executed, and a toggle operation which starts/stops the pumps when executed. The pumps can be controlled individually with each of these operations.

We also completed the other half of the integration which involved creating a section of the Fifi UI dedicated to the manual control of the syringe pumps which functions correctly. The client expressed approval of the UI design. The UI also displays errors so that the user does not need to check the log.

We were unable to separate the syringe pump controls into a separate swift package, and after many technical difficulties related to version control and building, the client expressed that we should keep the syringe pump controller internal to the Fifi app instead of creating a separate package. The code we wrote is fairly modular, so we are confident that after the technical issues are resolved, separation of the controller into a separate package should be fairly straightforward.

VIII. Future Work

One of our stretch goals that was not completed was to add functionality to vary the rate at which the pump dispensed in relation to the time dispensed. This will likely be an addition to the app in the future, as it would allow easier rate changing on the pumps, which in turns allows the user to do less work.

In addition, the entire app we are working in was coded over several years by one individual, so it lacks documentation and updated coding standards. A future goal is to refactor the app, resulting in clear documentation throughout, more robust and maintainable code, and a bug free app.

Finally, as mentioned above, we were unable to separate the controller into a separate package due to versioning and build issues. We kept our code modular throughout and our client created a new repo with many of the build issues resolved, so we are confident that a future group would be able to separate the syringe pump controller package.

IX. Lessons Learned.

The overall project provided several valuable insights and flexibility in problem-solving was crucial. Effective and frequent communication and thorough documentation were essential for maintaining organization and ensuring future

updates could be made easily. When our initial plan to use Ethernet communication faced issues, quick adaptation and consultation with the client allowed us to not only create a backup plan, but resolve the issue before the backup was needed. These strategies of quick and effective communication were crucial for the team to successfully integrate the pumps with the clients preferred connection methods and are the biggest takeaway the team has from the project. Without them, we would have never worked past our second and third week struggles.

Our advice to teams working on a project like this in the future would be to start working on connecting to the hardware as soon as possible. This was the area that gave us the most trouble and ended up being the largest time sink. In addition, we would recommend that for a project with as little documentation as ours had, begin reading the code and taking notes as early as possible. Deciphering code from another person without having any explanation of what it does takes a lot of time and effort. Having some sort of search function like grep or an IDE feature that can span multiple files and show all callers of a piece of code is immensely helpful for piecing together an existing codebase. Along with this, if your client has some knowledge of the codebase like our client did, talk to them frequently as you make progress, they can help guide you. Additionally, we think it's wise to front load your work, difficult problems are not predictable and it is better to encounter them early and maybe end up completing the project early than to be crunched for time after you underestimate the scope of a problem.

XII. Acknowledgment

We would like to acknowledge our advisor Tree Lindemann-Michael for supporting us during this project. We would also like to acknowledge our client, Dr. Owen Hildreth who helped immensely throughout the project by providing loaner macbooks, helping us learn swift, and being available to meet, discuss and troubleshoot on numerous occasions.

Additionally, we would like to acknowledge Barry Cowen, President of New Era Pump Systems Inc, for his help in establishing communication with the pump, and for the detailed explanation of how to proceed with our work.

XIII. Team Profile

Alex Torres

Junior

Computer Science - Computer Engineering

Hometown: Fort Collins, Colorado

Work Experience: FRC Robotics Team, CAN Bus wire harness assembler

Hobbies: Longboarding, Skiing, Traveling, Backpacking

Eli Cook

Junior

Computer Science - Computer Engineering

Hometown: Erie, Colorado

Hobbies: Jiu-Jitsu, video games, reading

Alyssa Hanson

Junior

Computer Science - Robotics and Intelligent Systems

Hometown: Loveland, Colorado

Work Experience: MIRRORLab Human Robot Interaction Research

Hobbies: Snowboarding, Motorcycle riding, hiking, traveling

Matt Boughey

Junior

Computer Science - Computer Engineering

Hometown: Highlands Ranch, Colorado

Hobbies: Hiking, Shooting, Camping

References

[1] Microsoft Copilot, The Epic Wizards Logo with 4 Wizards. 2024.

[2] “Syringe Pump.” *Syringe Pumps - Peristaltic Pumps - Advanced Precise and Programmable*

[3] Inc., Apple. “Swift Resources - Apple Developer.” *Swift - Resources - Apple Developer*

Appendix A – Key Terms

Term	Definition
Fifi	*The macOS app that controls multiple Devices in Dr. Hildreth’s lab to operate the nano-inkjet printer.*
Swift	*A powerful and intuitive programming language for macOS, iOS, and more developed by Apple.*
TCP/IP:	*Transmission Control Protocol/Internet Protocol. TCP ensures reliable data transfer between devices, while IP handles the addressing and routing*