Model-based Embedded Software

Naren Vasanad
Kevin Albers
Robert Bui
Jose Oyola Cabello

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2015-124
http://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-124.html

May 15, 2015

Acknowledgement

Professor Edward Lee, Professor Sanjit Seshia

Model-Based Embedded Software

Final Capstone Report

Naren Vasanad
May 15, 2015

University of California, Berkeley College of Engineering

MASTER OF ENGINEERING - SPRING 2015

Electrical Engineering & Computer Sciences

Robotics & Embedded Software

Model-Based Embedded Software

Naren Shivashankar Vasanad

This **Masters Project Paper** fulfills the Master of Engineering degree requirement.

Approved by:

1. Capstone Project Advisor:

Signature: _____ Date _____

Print Name/Department: Edward A. Lee / EECS

2. Faculty Committee Member #2:

Signature: _____ Date _____

Print Name/Department: Sanjit Seshia / EECS

Abstract

Model-Based Embedded Software

by

Naren Shivashankar Vasanad

Master of Engineering in Electrical Engineering and Computer Sciences

Professor Edward A. Lee and Professor Sanjit Seshia

Embedded software is typically developed using traditional programming languages like C and C++. However, these traditional types of programming languages are not well suited for embedded systems development. The model-based embedded software project extends the code-generating capabilities of Ptolemy II to help users develop software using model-based design techniques for ARM mbed devices. In particular, this project primarily focuses on automatically generating C/C++ code in Ptolemy II for Synchronous Data Flow (SDF) and Finite State Machine (FSM) models. This makes it easier to design and debug, leading to faster and more robust software development.

Table of Contents

I. Problem Statement

The Internet of Things (IoT) encompasses all small scale embedded systems which are interconnected wirelessly through the internet and are continuously transmitting data. Currently, programming embedded systems requires knowledge of intricate details of the platform being used and the software is typically written using traditional programming languages such as C and C++. In addition, embedded software for complex systems becomes very long and difficult to understand as it grows. Our project involves the creation of an environment to make designing applications for IoT easier through the use of model-based embedded software techniques. The product abstracts all the finer details of implementation and exposes the features that the designer is concerned with. Today, designers widely use embedded computing devices such as Arduino[1] and mbed™[2], from ARM®, for prototyping embedded applications, because they are open-source and low power. They are also inexpensive and have a large community of developers. The design environment we are developing will specifically target these types of embedded platforms.

Hardware and software of a cyber-physical system can be complex and difficult to implement. “Cyber-physical systems” refers to embedded computer systems that interact and are affected by physical elements (Mueller et al. 2012:219). A technique for designing a cyber-physical system is model-based design, which applies mathematical modeling for designing and verifying systems (Jensen et al. 2011:1666). Our project focuses on the creation of a model-based design environment for programming embedded platforms. In particular, our project targets applications aligned with the Internet of Things.

[1] “Arduino is an open-source electronics platform based on easy-to-use hardware and software. It's intended for anyone making interactive projects.” <arduino.cc>

[2] mbed is an ARM based microcontroller that can be used to develop applications for the internet of things. <https://mbed.org/>

Over the course of the project, we created a model-based design environment and demonstrated its use with an embedded platform application. In order to test and determine the effectiveness of the application, the project included designing an example system. The application used to demonstrate the model-based design environment's capabilities was an interactive LED cube that could be controlled with hand gestures. The application was initially developed using regular coding techniques by writing C and C++, and later developed using the model-based design environment for comparison. The models for the components of this application were included in the final application.

Code generation is one of the primary aspects of the model-based design approach. As described by Jensen et al. (2011:1666), the model-based design methodology involves the use of a code synthesizer to produce code that executes the desired models of computation. Typically, designers will write C code that can be programmed on an embedded platform to perform some task. However, model-based design techniques allows a developer to build graphical models that represent their application. This project involves the creation of an environment using Ptolemy II [3] to allow designers to represent their application as graphical models. Based on the model created in the design environment, code can be automatically generated for an embedded platform.

Due to the nature of model-based design and specifically code generation, designers can spend less time writing and debugging code. Rather, designers can focus on the design of their application and verify its expected behavior. The use of a model-based design environment allows designers to represent how they expect their application to perform and allow the software environment to produce reliable code. The modularity of graphical models allows designers to easily reuse models in different applications and change aspects of their design, and

[3] "Ptolemy II is an open-source software framework supporting experimentation with actor-oriented design." <http://ptolemy.eecs.berkeley.edu/ptolemyII/>

the graphical interface allows a user to easily view concurrent processes and how distinct units of a program interact with each other.

II. Industry/Market/Trends

A. Introduction

Open source embedded platforms have become popular for rapid prototyping. The market for embedded platforms has been growing as the number of connected devices continues to increase. Our capstone project aims to contribute to the community of embedded developers by solving the challenges of efficient code generation using the approach of model-based design.

The motivation for this project was twofold. First, a model-based design environment specifically for mbed devices does not currently exist. There are a few competitors, as described further in this section, that provide a graphical interface, but they do not offer a design environment focused on model-based design. Secondly, our project targets an emerging market and offers an opportunity for us to differentiate from our competitors. Embedded platforms have become very popular with hobbyists and the maker community, but there are not many tools such as ours that directly contribute to helping design for applications involved with the IoT. The stakeholders for this project include three segments: end users, sponsors, and customers. End users include hobbyists who work on IoT projects. Since these users will be working on fast prototyping of solutions and also have basic knowledge about building products, this would be the ideal market to target. These users could potentially give feedback of our product to improve and focus it towards being viable to a larger audience. Once the software gains traction amongst hobbyists it will be easier to reach a broader market like students, major companies, and universities. Our sponsors include the EECS Department, Embedded Systems Lab, TerraSwarm Research Center, Professor Edward Lee, Professor Sanjit Seshia, and the project team members

(Kevin Albers, Robert Bui, José Oyola, Naren Vasanad). Our customers will be discussed in detail in the Customers sub-section.

In this section, we use Porter's five forces model to analyze the five major forces in our embedded software market in order to create a go-to-market strategy: competitors, customers, suppliers, new entrants, and substitutes (Porter, 2008). In his article "How Competitive Forces Shape Strategy", Michael Porter (1979) discussed how the "strength of these forces determines the ultimate profit potential of an industry". We describe each of the forces and its effect on our strategy in the sections ahead and provide a strong or weak label. A force that is labeled as strong means that it could have a strong effect on our competitive strategy, whereas a weak force is an area that our strategy could take advantage of. Porter's five forces was important to use because it offers a unique analysis to determine the strength of our product's position, potential to make a profit, and create a strategy to move the balance of power to our favor.

B. Market Trends

Our target industry includes anything which encompasses IoT. Gartner (2014) published a study indicating that the IoT is on the peak of the hype cycle. It is expected that IoT will reach the plateau of productivity, the point where the technology is stabilized, in the next five to ten years. Furthermore, Clarice Technologies (2014) talks about how there will be close to 50 billion devices connected to the internet by 2020. Based on these studies, the IoT industry has the potential to grow immensely in the near future.

Most of these IoT devices will be small scale devices which sense the environment and connect over the internet to communicate with other more complex devices. A Markets and Markets (2014) report expects that by 2019, the IoT market will be close to $500 Billion. IoT has the potential to create waves in many industries worldwide, spanning from medical and wearable

devices to transportation and automation, as well as improve social connectivity between people everywhere (Hulkower 2014; Ma et al. 2011).

C. Competitors

There are three main competitors that offer model-based programming with a graphical interface. These include MATLAB's Simulink®[4], National Instrument's LabVIEW[5], and an open source project named PyLab_Works[6].

Mathworks' product, MATLAB, is one of the world's best super calculators that runs on a computer. It uses a scripting language to solve complex computations, often by using calculus. Simulink is an environment within MATLAB that allows programs to be built using graphical blocks. Mathworks has provided an interface, called Simulink Coder, a Simulink extension that allows user to generate and execute code from stateflow models.. This allows people to use Simulink to build model-based programs, then use the interface to and from the Arduino to provide Simulink with the inputs and outputs. However, Simulink must be installed on a computer to run, so the embedded device must be connected to a computer in order to work.

National Instruments improves upon Simulink's flaws with LabVIEW. LabVIEW is similar to Simulink, but it switches the focus from computations with calculus to data analysis and program logic. The best advantage that LabVIEW has over Simulink is the downloadable model. It allows code generated by the model to be downloaded to the embedded platform and run without the help of a computer. While LabVIEW offers substantial advantages for embedded devices compared to Simulink, our solution offers further improvements with the use of model-based approaches.

---

[4] "Simulink® is a block diagram environment for multidomain simulation and Model-Based Design." <http://www.mathworks.com/products/simulink/>

[5] "LabVIEW is a graphical programming platform that helps engineers scale from design to test and from small to large systems." <http://www.ni.com/labview/>

[6] "PyLab_Works is a free and open source replacement for LabView + MatLab, written in pure Python." <https://code.google.com/p/pylab-works/>

In the open source community, PyLab_Works offers an open source solution that attempts to accomplish model-based embedded programming. It offers a block graphical interface similar to LabVIEW, but it does not have much support. Each block must have written code in Python, meaning it is not completely model-based software.

Our solution differs from our competitors since it's open source and open platform, whereas MATLAB and LabVIEW require a license to use them. A MATLAB license for personal use costs $149 for non-students, and the basic LabVIEW license costs $999 (MathWorks n.d.; National Instruments n.d.). This license cost is prohibitively expensive to many potential users of these systems. In contrast, our solution is open source and freely available. In addition, our solution is open platform. MATLAB and LabVIEW are closed to specific platforms that the developers have chosen to support. If a user wishes to use one of these software tools with a different platform that is not supported, then there is little he or she can do. By making our solution available to the open source community, it is able to expand and grow the amount of supported platforms. Overall, the threat of rivals is weak, though with a change in strategy, it is possible that these competitors could enter the hobbyist space.

Open source software has been known to disrupt markets dominated by proprietary software in the past. According to IBISWorld, "open-source software (OSS) has been growing as a share of the global software market" (Kahn 2014:31). OSS (such as the Linux operating system) is a threat to some proprietary software, but will also promote interoperability and new software developments (Kahn 2014:31). Since our software is associated with open source software, we anticipate that we can leverage on the OSS structure and increase traction on our product.

The success of our application can be measured with market adoption. A study has shown that the number of updates to open source software created by members of open source communities has increased exponentially in the recent past (Deshpande et al, 2008:205). This

further supports our claim that acquiring more users would lead to more development of our project. Handling a community is not a straight-forward task. Øyvind et al. says that it may be beneficial to release the product as executables in the beginning to increase usage and decentralize the control of power with specific tasks having ownerships also that as the product grows (Øyvind et al. 2009:71-72).

Another factor that affects market adoption is the availability of modules. Our application will have a library of modules that are specific to IoT. These modules include sensors, actuator and communication. Making these modules specific to IoT will help differentiate ourselves from competitors who may not have such libraries. These standard libraries will help to create trust in the open source community and hence will help in building traction amongst hobbyists (Øyvind et al. 2010:114).

D. Customers

Our project would make it easier to communicate with development platforms and also to integrate sensors and actuators into a system. Since the technology is still nascent, it gives the project the right opportunity to grow with an emerging market and adapt to changes from customer needs.

Our main target customers are hobbyists and do it yourself (DIY) enthusiasts. These customers have a large variety of products to build their projects with, as well as a competitive market with low prices for embedded platforms. In addition, there are various tools that they can use to develop on their chosen platform as described in the competitors subsection. The most important factor is our reliance on market adoption to promote our product. We need to create a community that develops libraries and examples that are easily accessible to new users. However, open source software adds additional barriers for customer adoption. It can be harder for customers to trust open source software as much as the paid closed source alternatives

created by established companies (Bianco et al. 2009). For these reasons, the customer market force is strong.

E. Suppliers

Since our project is built using the Ptolemy II, the affiliated Ptolemy II research group at UC Berkeley is our main supplier. Ptolemy II group relies on donations from research grants and businesses that use the software. Our success will help extend the successful functionality of the Ptolemy II project, making it beneficial for us to succeed. This makes our supplier a collaborator rather than a potential threat to our success.

Furthermore, the fact that this is a research project under one of the most reputed universities in its field helps us differentiate from other competitors. Even if there are competitors in the open source community, the backing of the Ptolemy II project will help gain trust from potential users and hence increase the conversion rate of adoption in our favor.

F. New Entrants

According to Hoover's industry analysis of Computer Aided Design (CAD) software, the DIY movement "has sparked interest in CAD/CAM software among hobbyists and tinkerers" (2015). Our software falls into this category as a form of CAD. This industry opportunity shows that not only will this space be attractive to existing players, who can easily enter the market to compete with their products, but also startups that could use our open source code to build their own similar products to compete with our own. This shows that the threat of new entrants is strong.

G. Substitutes

Hobbyists have the option to continue using tools that they know, which makes programming in languages such as C a substitute to our product. Since it might be too time

consuming to learn a new programming method such as using a graphical design environment, many hobbyists might decide it is not worth their time to switch from their current programming methods. We designed our tool to reduce development time when the user has learned how to use it, but over a short period of time this is less obvious to the user and they may become frustrated and return to a familiar tool. In addition, the current communities, such as the Arduino community, have large libraries of tools and project guides, which pose a strong threat to our product adoption. This makes the threat of substitution a strong threat.

H. Critique and conclusion to Five forces

Given the fact that our project is open source and the current trends in the open source community, we are in an interesting position when it comes to our strategy. After evaluating the five forces, it seems that some of these forces may actually end up working in our favor. First, our main supplier, the Ptolemy II project, is actually more of a collaborator. The project participants frequently and on a daily basis increase the capabilities of Ptolemy II and add to the already large codebase. As will be discussed in the section on Intellectual Property, our success is linked with the Ptolemy II project, which was mentioned in the suppliers sub-section. This further incentivizes the Ptolemy II project stakeholders to continue to pursue the project and ensure its success.

In addition, the customers for our project are hobbyists and the open source community. The open source community is known for expanding projects and making the projects suit their needs (Deshpande et al. 2008:198). Therefore, our open source customers can actually become collaborators and help expand the codebase of Ptolemy, adding support for other platforms, and creating sample applications for others to use and learn from.

The open source nature of the project also has the effect that new entrants can end up helping us succeed. Any new open source alternatives to our Ptolemy project will have to

compete with Ptolemy's 20-year-long history and codebase, which spans over 3 million lines of code. However, open source projects have another option: to join our community and enhance its reach and capabilities. For instance, a new entrant seeking to create an open source model-based environment for the Raspberry Pi can take advantage of Ptolemy's already existing infrastructure and simply add support for their platform instead of building everything from scratch.

Overall, the five forces in our market are moderate, with the strongest force being the customers. This means that without addressing these forces appropriately, the profit in this industry will not be huge, even if successful. The open source business model adds an additional challenge to profitability. We can mitigate the strong forces with the right positioning.

To bring our product to market, our marketing strategy will be focused on the 4 'P's: product, place, price, and promotion. As mentioned in the subsection on Customers, our target customers and users are hobbyists and makers. By making our product initially open-source, it will be very appealing to this customer segment as they are very willing to try new products especially those that are at no cost to them. We plan to market it differently as well since we are targeting the open source community instead of industry professionals like our competitors. From our marketing study conducted early in the project, we learned that many of these types of users learn about the latest technology through websites and complementary technologies to our product such as embedded platforms like Arduino. Therefore, our strategy will be to ensure our product is easily accessible online by hobbyists.

I. Marketing and Productization

Based on the success of providing our product as an open source solution, there are four ways in which we could begin to monetize our project. The first way would be to to offer technical support for those that are interested in advanced applications. Users could pay to receive help from our technical support staff in using and extending our product for their own

needs. This option would be the first one that we would try since it has been successful for other products in the past. In his article, Fitzgerald calls this a value-added service-enabling model which has been very successful for Red Hat, an open source Linux provider (Fitzgerald 2006).

Another alternative would be the use of advertisements. Similar to how desktop and mobile applications are designed, we could incorporate advertisements in our design environment and users would pay a fee in order to use a version without advertisements.

Furthermore, we could offer a professional version of our open source project that would be targeted to advanced users and industry professionals. This version would use a subscription model where customers pay a monthly or annual fee. The professional version would include application specific content and strong technical support and documentation for the most cutting-edge advancements in embedded systems. Fitzgerald also mentions in his article that this would be considered a loss-leader/market-creating model since our first product would be open sourced but a product with more features would be used for monetization (Fitzgerald 2006).

A final option for monetizing our product would be to partner with an embedded platform company and offer our product as part of a bundle. The company would provide the target embedded platform hardware and our software product would complement their device with a custom design environment. An example of this approach would be the mbed collaboration between ARM and several semiconductor companies. In this industry with established competitors, this would be an appealing approach to obtaining market share and brand recognition.

J. Conclusion

Based on our project's unique features and target market, our project has potential to make an impact in the embedded software industry. The IoT era has brought a need for better software design tools and our product helps solves the challenges that designers face. By

targeting hobbyists and the maker community, our product enters a space where it can receive market adoption and not directly compete with well-known embedded software competitors. "Open source style software development has the capacity to compete successfully, and perhaps in many cases displace, traditional commercial development methods" (Mockus et al. 2002). Based on our evaluation of Porter's five forces in this industry, our business strategy should allow our product to make a strong impression in an industry with primarily commercial development methods (Porter 2008).

III. IP Strategy

A. Introduction

Since the Model-Based Embedded Software project is built upon Ptolemy II, it is important to understand the intellectual property surrounding the project before deciding how it should be advanced for commercialization. The concepts and ideas that form the basis of this capstone project are not novel, nor is the particular application that this project seeks to build. In particular, the project is an open source implementation, rather than invention, of the previously existing branch of computer programming known as model-based code generation. Several software solutions already exist that produce code using similar techniques, and they are mentioned later in this section. This makes it highly unlikely that any aspect of the project is patentable. However, this does not mean that the concepts of intellectual property do not apply to this project. This section discusses the intellectual property aspects of the Model-based Embedded Software project and the strategy that can be used to ensure proper use and attribution of our work, as well as the risks associated with infringement of previously existing IP.

B. Open Source Licenses

There are many different open source licenses that are available to protect the work of the open source community. The most widely used open source license, the GNU General Public License (GPL), is an example of what is known as a "copyleft" license, which requires that any work built upon GPL-licensed software must also be distributed under the same license (Lindman et al. 2010:239). This ensures that any GPL-licensed work will forever be freely available for all to use. However, other open source licenses such as the Berkeley Software Distribution (BSD) and MIT open source license are different. These open source licenses, both of which come from academic institutions, allow software covered under the license to be used in any way, including in commercialized software for profit, with no restrictions (Lindman et al. 2010:239). The idea behind this method of licensing is that successful projects coming from these institutions, if available freely for use in successful software, can benefit the institution from where it came by enticing others to provide funding to the institution for further development of the software. An example of successful commercial software built upon BSD-licensed software is Apple's Mac OS X and iOS, both of which are built upon BSD Unix (Engelfriet 2010:49). These open source licenses provide many benefits to those wishing to build upon them, such as software startups, since it does not require the resulting software to have the same license. This means that any other protection can be used for the software, including copyright protection, or even a different open-source license, which would ensure that the software would continue to be available as open source, if that is the goal of the software developer, as is often the case for the open source community (Engelfriet 2010:49).

Since our work is part of a large software collaboration, Ptolemy II, it will be bounded by the same rights of use, the BSD license ("Ptolemy II F.A.Q" 2014). "Ptolemy II is an open-source software framework supporting experimentation with actor-oriented design" and is a part of the

Ptolemy project at UC Berkeley, which is an initiative dedicated to studying models and simulations of embedded systems ("Ptolemy II" n.d.) . The Ptolemy project is well-funded and has many industrial sponsors involved ("Sponsors of the Ptolemy Project" n.d.). The BSD license allows software designed with Ptolemy II to be used for free commercially. Thus, if we decided to extend the software in the future as a separate entity, we would not have any issues commercializing it.

C. Advantages of Open Source Licenses

Furthermore, there are many other advantages for distributing our software through open-source channels. As mentioned in the Industry/Market/Trends section, many large competitors already exist in the embedded software industry. Open-source software offers a way to create market adoption by allowing customers to try a new product for free in order to build a community supporting the software. This is one way that open source software can penetrate a market with large competitors. According to Hoover (2015), "open-source software, which poses a competitive threat to the industry's traditional license-based business model, has grown in popularity in the last decade." There are many examples of immensely popular open source successes in the past, such as Linux and Apache, and PostgreSQL, which have formulated a threat to proprietary software (Kahn 2014:31; Deshpande et al, 2008:197).

Although open source software can pose a threat to proprietary software, its open nature can also be a disadvantage. Since many of the existing large players have a research and development unit, the entrance of a new player could mean that existing players can simply use the new open source software to improve their solution directly (Engelfriet 2010:49). This is not an issue for copyleft licenses, since they require that any software built on it must also use the same license, but this requirement doesn't exist for permissive licenses such as BSD (Engelfriet

2010:49). Because permissive open source licenses allow for this to happen, it is very difficult for open source developers to protect themselves.

Currently, two of the largest competitors in the embedded software industry are Mathworks and National Instruments. Their respective products that are similar to our software tool are Simulink and LabVIEW. Each of these products offers a graphical design environment that can generate code for embedded system. Both of these companies have many patents registered involving the design environment, model types, and methods for code generation. In particular, National Instruments has a patent titled "Statechart development environment with embedded graphical data flow code editor", US patent number 8387002 (Dellas et. al. 2008:1). The patent describes a graphical design environment that uses a model that LabVIEW called statecharts, "a diagram that visually indicates a plurality of states and transitions between the states", to represent an application (Dellas et. al. 2008:35). Furthermore, in the patent, LabVIEW claims the rights to the invention of code generation for statecharts and specifically the transitions linking the states of a model (Dellas et. al. 2008:35). Although this patent seems similar to our product, it is quite different since it involves statechart models which are not used in Ptolemy II. Rather, our software generated code based on the specific model of computation selected instead of solely transitions and states as done in LabVIEW. Based on the limits of the patent to statechart models, the patent should not overlap with our idea.

D. Concluding Remarks on IP

Ptolemy II has existed for almost 20 years as an open source project and many commercial products have been created from Ptolemy such as Agilent's Advanced Development Systems ("Links" 2014). Our capstone project extends the functionality of Ptolemy II by offering code generation for models currently supported in Ptolemy II. Since there are currently no novel

aspects of our projects that could be patented, open source would be the best alternative approach for the current state of our project.

IV. Technical Contributions

Project Overview and Context

The Internet of Things (IoT) encompasses all small scale embedded systems that are interconnected wirelessly through the Internet. Currently, programming embedded systems that cater to IoT requires knowledge of the platform being used. Our project focuses on the creation of a model-based design environment for programming embedded platforms. In particular, our project targets applications aligned with the IoT. This project involves design of an environment to make creating application for a device for IoT easier through the use of model-based embedded software. For our project we chose to work with the Ptolemy II project being undertaken in the EECS department at UC Berkeley. Ptolemy II is a graphical environment that focuses on models of computation in order to depict embedded systems[7]. The focus of our project is to use the code generation aspect of Ptolemy II and focus it for designing IoT based applications for the mbed[8] platform.

The tasks for our project were split according to skill sets and preferences. The project tasks were divided between the two semesters. Such a split allowed me to venture into areas that I had less expertise and also into areas that I had prior knowledge. The first task of the project was to concentrate our project's focus. Narrowing down the project scope was a team effort that involved multiple brainstorming sessions in order to understand our target market, specific details of the underlying structure and tasks to be performed over the course of the

[7] "Ptolemy II is an open-source software framework supporting experimentation with actor-oriented design." <http://ptolemy.eecs.berkeley.edu/ptolemyII/>
[8] mbed is an ARM based microcontroller that can be used to develop applications for the internet of things. <https://mbed.org/>

project. This will be discussed in detail in the section on methods and materials. After the project context was decided, an example application needed to be built to portray model-based design. The example included an LED cube that was controlled by a data glove. The LED cube example included most aspects of IoT applications like processors (mbed), sensors (accelerometers and gyroscopes), actuators (LEDs) and communication modules (Wi-Fi). Communication was using Wi-Fi and specifically using the CC3000 module from Texas Instruments. My initial task was to understand the CC3000 module and provide a base on which the rest of the communication for the project was built upon. Kevin Albers, in his paper, describes the communication between the data glove and the LED cube and the different challenges faced.

Sensor information from the data glove was received by the LED cube and transformed into useful information. Once the filtered information was available it needed to be converted into gestures that could be understood by the LED cube algorithm that will be covered in José Oyola's paper. My task was to create an algorithm that corrects and filters data using a finite state machine (FSM) model followed by a gesture recognition algorithm to interpret the filtered data in order to control the LED cube. In his paper, Robert Bui will discuss about the use of model-based techniques such as FSMs and synchronous dataflow (SDF) models. The gesture recognition needed to understand sensor data and convert them into distinct actions. Gesture recognition is an important aspect for control of devices and is an essential concept for IoT since a lot of devices are being controlled by gestures. This will be discussed in detail in the section on methods and materials.

During the Spring semester, the focus of the project was on Ptolemy II and how our example application from the fall semester could be created using a model-based approach. The code generation code of Ptolemy II needed to be understood initially and changes needed to be made in order to repurpose Ptolemy II for code generation for the mbed. These concepts and

procedures will be discussed by Kevin Albers in his paper. Also, there is a need to understand the creation of models using FSMs and SDFs. These model-based approaches will be explained by José Oyola and Robert Bui in their papers. The mbed platform that was used for development of the LED cube application was an online compiler. However, to make it work on a stand alone software such as Ptolemy II is not straightforward. An offline toolchain is vital for seamless generation of binary files that could be loaded onto the mbed without having to use the online compiler. My task was with regard to understanding and creating the offline tool chain that integrated with Ptolemy II. This task included research about the GNU for ARM tool chain for C and C++. The next step was to understand makefiles and how they could be used in order to automate the process of code generation and binary file creation. Finally, code generation on Ptolemy II was not straight-forward. As we began our implementation we found that even small applications failed to work. The problem was narrowed down to an issue of memory leaks, a situation where variables in the code are allocated memory but never freed and hence consuming the entire memory of the processor resulting in no other operations from executing. The problems and solutions for memory leaks will be discussed in the methods and materials section.

Over the rest of the duration of the project, the goal is to complete the implementation of the LED cube application using Ptolemy II and demonstrate that the application works the same as the handwritten approach in the previous semester. This demonstration will show how a model-based approach to solving IoT-based applications makes design and implementation faster. In finer detail, the tasks to be performed include the creation of actors, modules for specific tasks in Ptolemy II, for specific tasks like gesture recognition and filtering that were developed in the previous semester.

Literature Review

IoT is a large and growing space and there is scope for improvement in applications. Kortuem *et al.* mention how there is a difference between regular "things" and "smart things". They also mention how IoT in the industrial sector is being developed. They focus on three main aspects of embedded systems and explain how they relate to each other: activity-aware objects, policy aware objects and process-aware objects and how each of them influence the other (2010).

As mentioned previously, the first part of the project involved creation of an application that uses the mbed platform. The mbed platform has a large community that generates a vast variety of projects. These projects in turn use various libraries that are open to anyone to use. The open source nature of the mbed community allows exchange of ideas and prevention of duplication of efforts in creating the same libraries again. The online compiler provides an interface using which applications can be developed anywhere in the world. Toulson *et al.* provide a good explanation of how the mbed online compiler can be used to create applications (2012).

Once the connections were made, the next part was to make use of the CC3000 library for mbed that was developed by Martin Kojtal[9]. In order to communicate with the internet, protocols such as HTTP are useful. This library has the bare bones implementation for the CC3000 IC from Texas Instruments. My work in the project builds on this library and repurposes it to create a TCP/IP connection between the data glove and the mbed controlling the LED cube.

With the advent of IoT, there is a need for hand based gestures to control electronics. Stravoskoufos *et al.* mention how fields such as e-health and bioinformatics could potentially gain a lot from gesture based control. They also talk about how connectivity to the internet and sensor data collection are important aspects in IoT. There is also a discussion on how a motion control based system for IoT can be designed (2014).

[9] CC3000 Wi-Fi Library for mbed by Martin Kojtal
<http://developer.mbed.org/users/Kojto/code/cc3000_hostdriver_mbedsocket/>

Code generation in Ptolemy II consists of code generation for finite state machines, synchronous data flow diagrams and other models of computation. It also has a very well structured and documented template file that is used in order to write C code for specific actors. It also has a number of macros that can be used to define different information. It is hence beneficial to our project since the mbed works on C++ and Ptolemy offers code generation for C and C++ (Brooks *et al.* 2008).

There are however different methodologies used for code generation. For example, Doi *et al.* mention how they implement an XML based parsing system for devices that do not have enough computing capabilities. The reason for their use of XML is that it is the standard method of communicating information over the internet and hence most ideal for a field such as IoT (2012). There is a lot of work being done on model-based design practices like in Ptolemy II. Riedel *et al.* talk about how model-based approaches can be used for sensors as small as RFID tags and also mention how web services and gateways on embedded systems can make use of this structure (2010).

One specific model of computation is the synchronous dataflow model that consists of actors that fire tokens when executed. Such a system is useful for applications in IoT since it is dependent on sensor data being available before they are consumed by other processing blocks. Tripakis *et al.* mention how code generation for synchronous data flow models works and also how it is integrated into the Ptolemy II platform. This paper also gives an introduction to how these models of computation work (2008).

Makefiles are useful to create applications that involve a large number of files. Wojtczyk *et al.* explain how CMake can be used to create applications easily. It also talks about the cross platform nature of these kinds of tools making it easy to create portable applications (2008).

Methods and Materials

The initial goal of the project was to define its focus. The target market needed to be really narrow since it is difficult to cater to a large set of users. Through our user studies we came to the conclusion that the best target market would be the hobbyists who work on new and upcoming technology to create interesting applications. Hobbyists are generally more adventurous and are willing to try new techniques and are readily accepting to change. Also, IoT in it's current stage is being developed mainly by hobbyists and there is no complete platform that supports their needs. As discussed in the market strategy section, our competitors aim at institutions and larger companies. The key differentiating factor is hence our target user group.

After the target group was narrowed down, the major task was to create an application that demonstrated IoT. Different aspects of IoT include sensors that collect data, like accelerometers and gyroscopes, actuators that act on data, like LEDs and motors, communication modules, like Wi-Fi and BLE, and most importantly the processor that links all these aspects. Each of these aspects needed to be chosen in a way that the application would be portray IoT.

The first aspect that was studied was the processor that would be used. There were multiple options that included the Arduino, Raspberry Pi and mbed. Robert Bui mentions in his paper about the comparison between these platforms and how they affected mbed as our choice for development.

The sensors that are used for the purpose of gesture recognition for our application are accelerometers and gyroscopes. While discovering different ways these sensors could we used, we came across data gloves that have accelerometers, gyroscopes and flex sensors that indicate how much a finger has been bent. The data glove from VirtualRealities was used for this purpose [10].

[10] The DG5 Data Glove from Virtual Realities uses Wi-Fi for communication and has accelerometers, gyroscopes and bend sensors <http://www.vrealities.com/products/data-gloves/dg5-glove-3-0>

As the name suggests, IoT applications are mainly controlled over the internet. This includes talking over networks in order to read from or write to appliances. Hence, we chose to use Wi-Fi as the communication module and in specific the TCP/IP communication protocol. The module chosen for this purpose was TI's CC3000 module. The module communicates with the mbed over SPI, a protocol that allows serial communication between ICs. Finally, the actuators that were used were LEDs. José Oyola talks in detail in his paper about the choice for LEDs and the different libraries used for this purpose.

The application was used to light up the cube according to gestures performed on the data glove. The data glove would communicate with a server and the server would communicate to the LED cube of the different sensor values. The LED cube would then decipher these messages and hence light the cube in various forms. The algorithm for the LED cube will be discussed in José Oyola's paper.

A modular approach is needed in order to portray how good design concepts can be used in order to develop robust applications. SDFs were used in tandem with FSMs. These design concepts will be discussed in detail in Robert Bui's paper.

During the Fall Semester, my focus was on the communication aspect of the project. The communication protocol used for the project was Wi-Fi and specially TCP/IP. The CC3000 Wi-Fi module from Texas Instruments was used. The data glove used a vendor specific Wi-Fi module and hence was not of concern to us. The first step to implementing CC3000 with mbed was to understand how to connect the module to the mbed. Pin mappings of both the CC3000 and mbed needed to be studied. Data sheets for these devices provided a good start. However, the best help

I got was from an existing project[11]. Since the project had already experimented with pin connections, all that needed to be done was making the right connections.

The first test for the Wi-Fi module was to be able to access the internet and ping a site like www.google.com. This feat was achieved with relative ease and spurred the rest of the project. The part of the project that dealt with Wi-Fi communication between the data glove and the LED cube will be discussed in detail in Kevin Albers' paper.

As mentioned previously, one of my concentrations was on creating the correction and filtering algorithms that help making sense of sensor data. Correction consists of a finite state machine that consists of two states: train and filter. The sensor data is normalized before it can be interpreted by the gesture recognition volume. Since the initial state of the sensors may not be the same on every reset of the data glove, a reference zero is required in order to have consistent models for gesture recognition. After the data is trained, it can be smoothed using filters such as alpha filters or Kalman filters to eliminate jitters in sensor data. The implementation of the train and filter FSM is shown in Figure 1.

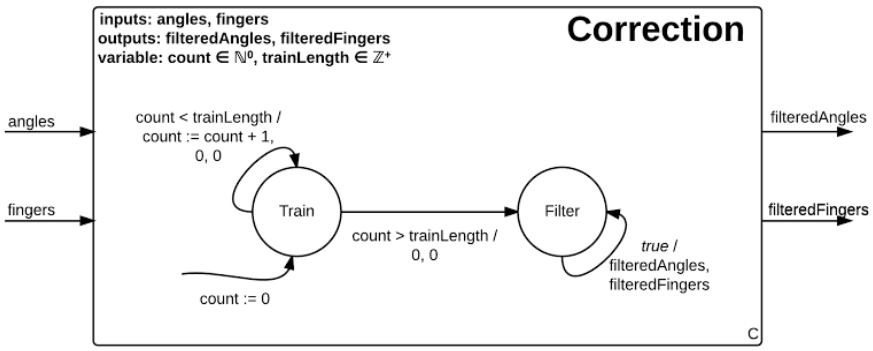

Figure 1: Train and Filter FSM

[11] CC3000 with mbed that includes hardware and software aspects <http://dev.inventit.io/blog/sparkfun/arduino/mbed/embedded_cxx/arm/cortex_m0/frdm_kl25z/2014/08/04/Sparkfun-CC3000-on-mbed.html>

Once the data has been filtered, the data can be characterized into gestures. For every ten packets produced by the correction state machine, the gesture recognition actor fires once. This procedure is used to down sample data obtained from the data glove. This allows for gestures to be interpreted more accurately. The project currently supports the use of ten gestures based on thresholds shown in Table 1. Everyone in the team contributed to the tables and figures shown in this paper. The tilt detection actor, that will be explained by José Oyola, uses angular data to determine if the LED square should move in the x, y or z axis depending on roll, yaw and pitch, respectively. The LED square moves if the roll, pitch, or yaw becomes greater than +/- 10 degrees from the calibrated initial position, for a total of six gestures. Figure 2 shows how data glove movements correspond to movements on the LED cube. The gesture recognition actor reads the bend sensor data to determine if the size or hue of the cube should be changed based on which fingers are bent. The thumb is considered to be bent if the value obtained from the data glove exceeds 200 ADC units. Whereas, the other four fingers are considered to be bent if the value from the data glove exceeds 350 ADC units. Figure 3 shows the four gestures based on the bend sensors. By keeping fingers 2 and 3 unbent and the others bent, the size of the LED square can be increased, while keeping only finger 3 unbent will decrease the size of the cube. The color of the LED square is based on how bent finger 3 was when the finger 4 is unbent and the other fingers are bent. Figure 4 shows how gestures are recognized in the gesture recognition actor.

Gesture Recognition Thresholds			
	Unbent	**Bent**	**Units**
Fingers 0-3	< 350	≥ 350	ADC units
Finger 4 (Thumb)	< 200	≥ 200	ADC units

	Neg. Movement	Pos. Movement	Units
Roll, Pitch, Yaw	< -10	> 10	degrees

Table 1: Gesture Recognition Thresholds

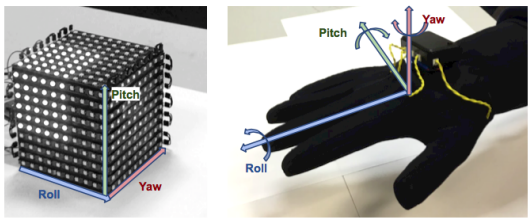

Figure 2: Gestures based on Quaternion Data

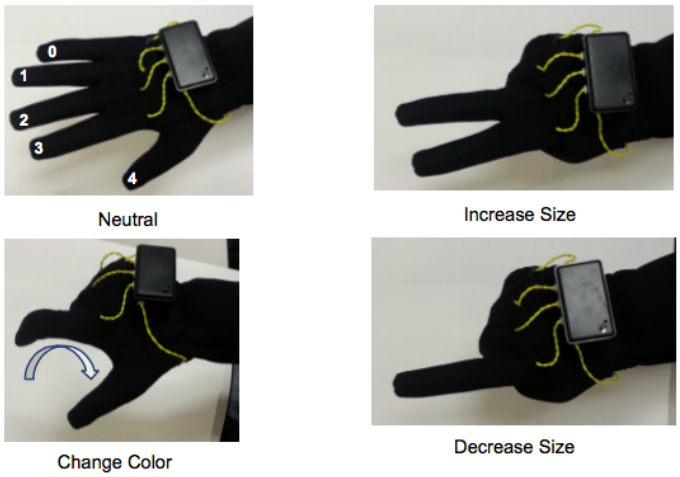

Figure 3: Gestures based on Bend Sensors

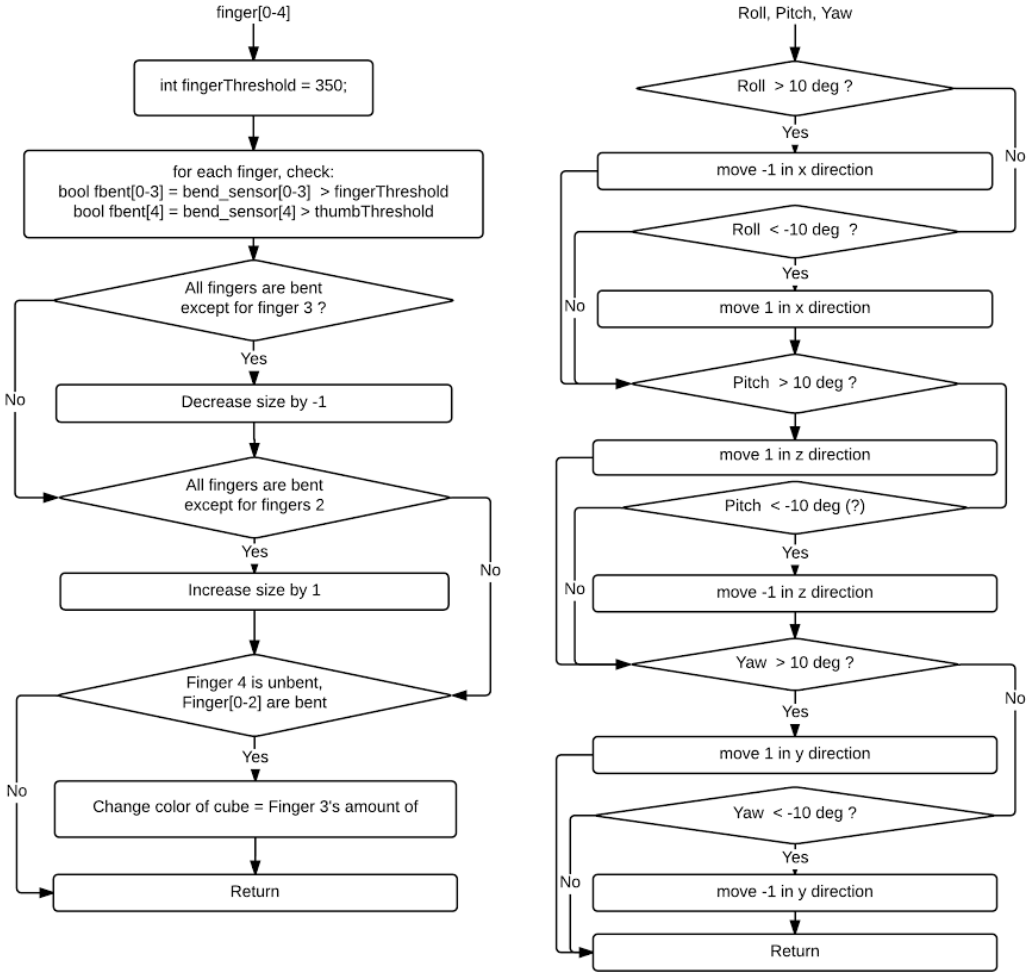

Figure 4: Gesture Recognition Flow Diagram

The second part of the project dealt with the Ptolemy II platform. Ptolemy II is a very vast project and consists of over three and a half million lines of code[12]. Understanding such an elaborate project required a lot of study. In order to get started, we held regular meetings with various contributors to Ptolemy II. These people included Christopher Brooks, who is one of the

[12] The exact number being 3,685,507 as of 11/27/2014 <http://ptolemy.eecs.berkeley.edu/ptolemyII/index.htm>

core developers of Ptolemy II, Fabio Cremona who was working on some aspects of code generation and Ilge Akkayya for Machine Learning. Multiple meetings with Christopher Brooks allowed better understanding of the structure of code and also the build environment from which graphical models were converted into C code. The process of learning involved all members of the project group since the various parts of Ptolemy II were needed by all of us.

The major parts of the project with Ptolemy II includes building of models based on core embedded systems concepts like SDFs and FSMs as will be discussed in Robert Bui's paper. The second part of working with Ptolemy II included features specific to the mbed tool chain. As discussed earlier, mbed has an online compiler that can be used to develop applications. However, since the Ptolemy II project is a stand-alone software, an offline build environment was needed. Such an implementation would allow seamless conversion of graphical models into the final binary file that would be loaded into the mbed in order to run the application.

The offline compiler involved understanding how the arm-none-eabi-gcc toolchain works and how it is used in the context of mbed. My starting reference was the site that gave the source code and binaries to use the GNU toolchain for ARM processors[13]. The toolchain is cross platform and allows applications to be built easily for a particular operating system such as Linux or OS X.

The GNU for ARM toolchain needed to be used to initially build the libraries for the mbed. Most libraries required just a single step build process where a single "make" command needed to be run on the command terminal of the operating system. However, a few libraries such as the CC3000 required modifications to the makefile, which is used by the make command, in order for it to be ready for use by the offline compiler. Paths to the mbed libraries, inclusion of the linker file for the mbed libraries and change of the C++ build command to build using a newer version of

[13] GNU tools for ARM Embedded Processors <https://launchpad.net/gcc-arm-embedded>

C++ and in particular the gnu++11 option needed to be made for the offline compiler to work. These changes have been documented for future users in a wiki page[14].

The next step was to build an existing project offline. When a project is downloaded from the online compiler, it comes with a makefile. However, this makefile doesn't work as is and needs to be modified. The results of these steps is explained in the results section of this paper. Firstly, the object files needed to be removed for library creation. This step is essential since the object files have already been built in the previous step. Secondly, all the object files that will be used by the project needed to be declared. These object files include the various libraries and the mbed specific libraries that are mandatory. Library include paths also need to be mentioned since they indicate where the object files should be accessed from. Finally, the gnu++11 compiler needs to be used due to the reasons mentioned previously.

The results from this process was then integrated into Ptolemy II. Ptolemy II uses a particular build process that includes a makefile according to the target chosen. Hence, for the mbed, the corresponding makefile needed to be altered. Again, this task was not straightforward and it needed the modification of the makefile from the previous step in order for it to work. Some of the challenges faced with this regard will be discussed in the results section of this paper. Many changes needed to be made to the Makefile in order for it to work offline. Firstly, Ptolemy II code generation related source files had to be included into the build system. This includes code for the backbone of different models of computation. Ptolemy II related code is built using using GNU for ARM compilers. I also needed to include information that declared that memory allocation and other memory related libraries are not available and also information that helped the code generator to include code that is specific to the mbed. Finally, specific C files needed to be built using the C++ compiler. Ptolemy II code generates a number of C files that can be

[14] mbed for Ptolemy II wiki page <http://chess.eecs.berkeley.edu/ptexternal/wiki/Main/Mbed>

compiled using the C++ compiler and this is necessary since a number of files include mbed related libraries that are supported only in C++.

As projects grow large they tend to get messy and it is easy to lose track of working versions of software and also fixes that were made. Version control is a great way to avoid these errors. Version Control Systems also promote distributed software development where users from across the world can collaborate on projects and hence speed up development. Bug reports can easily be filed and conflict resolution between different developers becomes much easier (Costa *et al.* 2013:90).

Prahov *et al.* mention how IC projects and software engineering projects have started using version control intensively. The study indicates the different features of version control that are used the most. It also cites a few examples of large scale projects and how version control was useful in their development. It is hence useful for our project to use various version control mechanisms as well (2013).

Multiple environments were used for version control through the course of the project. mbed's online compiler has version control built into it and this was used in order to create the LED cube-data glove application. All the project members worked in parallel and finally merged their code together. Good version control practices like incremental commits and detailed commit descriptions were helpful in order to keep track of progress, contributions and also understand why certain features were implemented.

The Ptolemy II project is version controlled with the help of SVN. SVN provides a concrete solution for version control across large projects and was hence used in Ptolemy II. Each change we made to the Ptolemy II code chain has been tracked by commits on specific files. We also maintained good version control practices and have been able to share each other's

contributions. This has allowed faster implementation time and has avoided dependencies between team members.

As mentioned previously, one of the problems faced was with memory leaks. Embedded system processors have very limited memory and hence have to be managed carefully. It was seen that the code generated from Ptolemy II was not handling memory optimally. Even the simplest of applications resulted in failures. In order to track down and fix these problems we used valgrind[15], a tool that helps find bugs in memory management. Once the source code is built, it can be run through the valgrind tool which in turn provides insightful information about variables that are retained in memory even after they have been used. Using valgrind we were able to track down a number of memory leak issues as will be discussed in Robert Bui's paper.

The final task of the project was to re-create the LED cube application discussed earlier using model-based design approaches using Ptolemy II. The model was chosen to be an SDF model with each specific function being a separate actor. The various actors included the wifi module, filtering and correction, gesture recognition and the LED cube itself. Some of these models were further divided into sub-models hence forming a hierarchical structure. In order to have this system working on the Ptolemy II software, the offline toolchain needed to be completed. One of the libraries, WS2812 for the LEDs, was not working correctly in the offline toolchain and this was fixed by modifying the source code for the library. After the WS2812 library was fixed, the entire offline toolchain was ready, allowing a seamless generation of code for the K64F mbed platform.

Results and Discussion

[15] Valgrind is a memory management bug detection tool <http://valgrind.org/>

Initial successes and failures in the project helped us understand a lot about small scale embedded systems. It was during integration of our various modules that we faced challenges. The CC3000 module that I was working on needed to be integrated with the LED array and algorithm that José Oyola and Robert Bui were working on. However, once the code was integrated the Wi-Fi module stopped working as expected and used to fail occasionally and there was no immediate explanation for this behavior. The usual method of viewing statuses at different parts of the code didn't provide any information. It was then that I started looking through the libraries and tried to gain a deeper understanding of how the libraries were communicating with hardware. It was then that I realized that the Neopixel LEDs and the CC3000 module had an overlap on the external pins. The Neopixel LEDs assumed two pins to be unused but these pins were being used by the CC3000 for SPI communication. The solution was that the mbed had another set of SPI pins through which the CC3000 could be connected. This required physical rewiring of the pins and also the change of pin mapping in the software. This helped us realize that it is important to discover alternatives for different pins on the IC before designing an application.

During the spring semester, the result of the makefile usage was that a user could build the final binary to be loaded into the mbed by clicking a single button on the Ptolemy II model. This ease of use allows users to quickly build applications without worrying about setting up tool chains and other complicated methods for offline compilation. However, the journey was not smooth. With my limited knowledge on makefiles, it was difficult to understand how makefiles were used. Hence, I consulted an expert on the topic, Christopher Brooks, and held a meeting with him. It is then that I understood how to work with makefiles and how they were integrated into Ptolemy II. However, even after this meeting, there were many challenges as I was attempting to accomplish a task that involved expertise from multiple domains, the mbed tool

chain and Ptolemy II. To solve this problem I broke it down into multiple parts as described in the previous section.

The major challenges I faced were during the creation of makefiles for compiling example applications we had built in the previous semester using the online tool chain. I understood what different parts of the makefile meant and started implementing one feature at a time. However, I was unable to move forward for five days while working on a single problem and was not able to solve it. This is when I approached an expert I knew in the field of makefiles, an old colleague of mine from Texas Instruments, Anand Gadiyar. through my discussion with him I was able to solve the problem of relative versus absolute addressing in makefiles. The tool chain required absolute paths instead of relative and also needed slight syntax modifications as the ordering of tasks was wrong in a particular command. Online forums provided a lot of help both through questions that were already asked and through questions that I asked. One example was the GNU forums where I got some help to understand and correct a particular part of the build system for the ARM C/C++ compilers[16]. This reinforces the benefits of an open source community of how support is easily available.

From this point, the task was to integrate the tool chain into the Ptolemy II system and be able to generate a binary file at the click of a button. It also required understanding of the code that was generated from a Ptolemy II model. The challenges I faced here were that all the files generated were C files and not C++ files. The mbed libraries require C++ since they include a lot of C++ specific code and libraries. The solution to the problem was to compile the C files to C++ files since C++ supports most of the features of C. However, this approach led to problems where a few C-specific functionalities were not supported by C++ like automatic type casting from void pointers to other types. Kevin Albers mentions about these errors in detail in his paper. After

[16] Errors building the offline GNU toolchain for ARM
<https://answers.launchpad.net/gcc-arm-embedded/+question/262850>

having another meeting with Christopher Brooks, I was able to understand the different kinds of files and how they could be interchanged from one type to the other. In order to correct for these dependencies I had to add a section to the makefile that converted only the files that included mbed specific libraries into C++. As mentioned previously, I got help from online forums in order to achieve this task[17].

Each of the actors in the LED cube model on Ptolemy II had it's own C code. However, this code was made modular so that any of the blocks could be replaced with another. For example, the CC3000 wifi actor was developed in such a way that it had generic inputs and could be used for any application. Similarly, the NeoPixel LED actor could be replaced by any other actuator like a motor or a screen by a simple replacement of the actor.

As discussed earlier, the final toolchain was ready once the WS2812 library was fixed. The problem was that the NeoPixel LEDs required very precise timing. After a few experiments with an oscilloscope, used to measure timing, the right timing requirements were achieved allowing seamless integration of the offline compiler.

V. Concluding Reflections

Overall, my accomplishments in the project can be summarized into understanding CC3000 communication, correction and gesture recognition algorithms, understanding code generation in Ptolemy II, creation of makefiles that use the GNU for ARM compilers, setting up the offline toolchain and also solving memory leaks in Ptolemy II for code generation. All these tasks were essential to the final result of the project as it helped other members of the team work forward from. A lot of tasks were team based initially and finally the split into teams of two helped faster progress in the project.

[17]How file extentions can be modified using makefiles
<http://stackoverflow.com/questions/28921330/modifying-file-extensions-using-makefiles>

Our original project description was very broad and we decided as a team to approach the project in a two step process. The first was to create an embedded systems application, the data glove coupled with the LED cube, using model-based design concepts. The second was to be able to use a graphical interface, Ptolemy II, to re-create the same application. We were able to accomplish the first half of the project on schedule during the Fall semester. While working with code generation with Ptolemy II we faced a lot of hurdles but we were able to accomplish most of the tasks we planned. There are a few aspects like supporting multiple processors and using other models of computation such as discrete event systems that could have been good additions. However, these tasks were planned to be completed only if the core of the project was completed. These concepts will be a good addition to the entire project if someone plans on building on our work.

Through the course of this project I learnt how important it is to set realistic goals and how to work as a team to accomplish these goals. The initial work with Gantt charts and work breakdown structures helped a lot in breaking down tasks into atomic units. Since the project was a set of peers it was important to understand each one's strengths and weaknesses and leverage them to our advantage. An example of this was while we were working on code generation. Since the other three project members were taking an advanced model-based design course which I was not, I concentrated on building the offline tool chain and learnt aspects of model-based design and Ptolemy II from them so that we could work parallely.

Model-based embedded software practices used in this project will help in building scalable IoT applications. The graphical interface provided will in addition provide a fast prototyping environment. With both these advantages our project stands to gain great traction with the advent of IoT. Our project could open a large range of opportunities for people with

limited programming knowledge to create applications faster and better than what they could find through any other alternatives.

Future Work

A good point for other project teams to continue would be our wiki page. It explains in detail the features we added and the problems we faced in our project. There is also a step by step guide that will help users set up the Ptolemy II project and start developing much faster than we were able to do. Detailed descriptions of different aspects of code generation are also explained in the same page.

The scope of our project still remains vast and future teams can build off our project. An obvious addition to the project would be to support multiple platforms such as the Arduino and Raspberry Pi, not counting the multiple variants of the mbed. Our project focused mainly on synchronous data flow models with finite state machines. However another aspect that was not explored is code generation for discrete event models. This model of computation is wider and many more applications can be created using this model. It would also be beneficial to create a set of libraries for Ptolemy II that focus on IoT. A user of the product would also benefit from example applications for each of the libraries that were created.

References

Audris Mockus, Roy T. Fielding, and James D. Herbsleb. "Two case studies of open source software development: Apache and Mozilla." *ACM Trans. Softw. Eng. Methodol.* 11, 3 (July 2002), 309-346, Web. 16 Feb. 2015. <http://dl.acm.org/citation.cfm?id=567795>

Brooks, Christopher, Edward A Lee, Xieojun Liu, Stephen Neuendorffer, Yang Zhao, Haiyang Zheng, Shuvra S Bhattacharyya, Elaine Cheong, II Davis, Mudit Goel, Bart Kienhuis, Man-Kit Leung, Jie Liu, Lukito Muliadi, John Reekie, Neil Smyth, Jeff Tsay, Brian Vogel, Winthrop Williams, and Yuhong Xiong, "Heterogeneous concurrent modeling and design in java (volume 2: Ptolemy ii software architecture)", Berkeley, CA, USA: California University Berkeley Department of Electrical Engineering and Computer Science, 4/1/2008. Print.

"Buy LabVIEW." - *National Instruments*. National Instruments, n.d. Web. 25 Nov. 2014. <http://www.ni.com/labview/buy/>

Clarice Technologies. "Demystifying the Internet of Things." *Thinking Products: A Weblog by Clarice Technologies*, Clarice Technologies, 6 Mar. 2014. Web. 16 Feb. 2015. <http://blog.claricetechnologies.com/2014/03/demystifying-the-internet-of-things/>

Costa, Catarina, and Leonardo Murta, "Version Control in Distributed Software Development: A Systematic Mapping Study" 2013 IEEE 8th International Conference on Global Software Engineering (ICGSE), pp.90-99, 26-29 Aug. 2013. Print.

Dellas, Christina M., and Hogan, Kevin M. Statechart Development Environment with Embedded Graphical Data Flow Code Editor. National Instruments Corporation, assignee. Patent US 8,387,002 B2. 26 Feb. 2013. Print.

Deshpande, A. and Riehle, D., *IFIP International Federation for Information Processing*, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; Boston: Springer, 2008. pp. 197–209.

Doi, Yusuke, Yumiko Sato, Masahiro Ishiyama, Yoshihiro Ohba, and Keiichi Teramoto, "XML-less EXI with code generation for integration of embedded devices in web based systems," 3rd International Conference on the Internet of Things (IOT), vol., no., pp.76,83, 24-26 Oct. 2012

Engelfriet, A. "Choosing an Open Source License." IEEE Software 27.1 (2010): 48-49. Print.

"Engineering, Scientific & CAD/CAM Software" Hoover's Online. 2015. Web. 16 Feb. 2015.

Fitzgerald, Brian. "The Transformation of Open Source Software." MIS Quarterly. Vol. 30, No. 3 (Sep., 2006) , pp. 587-598. Web. 16 Feb 2015. <http://www.jstor.org/stable/25148740>

"Gartner's 2014 Hype Cycle for Emerging Technologies Maps the Journey to Digital Business", *Gartner,* 11 Aug. 2014, Web. Nov. 2014. <http://www.gartner.com/newsroom/id/2819918>

Hulkower, Billy. "Living Online - US - May 2014." *In Mintel*. n.d. Web. 13 Feb. 2015. <http://academic.mintel.com/display/704619/?highlight>

"Internet of Things Market & M2M Communication", *Markets and Markets*, Nov. 2014, Web. Nov. 2014. <http://www.marketsandmarkets.com/Market-Reports/internet-of-things-market-573.html>

Jensen, J. C., Chang, D. H. and Lee, E.A., 2011, "A model-based design methodology for cyber-physical systems", *Proceedings of the International Wireless Communications and Mobile Computing Conference* . IWCMC 2011 . pp. 1666-1671. Print.

Justyna Zander, Ina Schieferdecker, and Pieter J. Mosterman, 2011 "Model-Based Testing for Embedded Systems", *CRC Press.* Boca Raton: Taylor and Francis Group, 2013. Web. 16 Feb. 2015. <http://dx.doi.org/10.1201/b11321-1>

Kahn, Sarah, IBISWorld Industry Report 51121: Software Publishing in the US. Dec. 2014. Web. 13 Feb. 2015.

Kortuem, Gerd, Fahim Kawsar, Daniel Fitton, and Vasughi Sundramoorthy, "Smart objects as building blocks for the Internet of things," IEEE Internet Computing, vol.14, no.1, pp.44,51, Jan.-Feb. 2010

Lindman, J.; Paajanen, A.; Rossi, M., "Choosing an Open Source Software License in Commercial Context: A Managerial Perspective," *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, 237-44, 1-3 Sept. 2010

"Links." *Ptolemy Project*. UC Berkeley, 26 July. 2014. Web. http://ptolemy.eecs.berkeley.edu/archive/links.htm, accessed February 28, 2015.

Ma, Tao, and Chunhong Zhang. "On the Disruptive Potentials in Internet of Things." *Proceedings 17th IEEE International Conference on Parallel and Distributed Systems: ICPADS 2011: 7-9 December 2011, Tainan, Taiwan*. Los Alamitos, Calif: IEEE Computer Society Conference Publications, 2011. 857-59. Print.

Mueller, W., Becker, M., Elfeky, A., DiPasquale, A., "Virtual prototyping of Cyber-Physical Systems," *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, 219-26, 30 Jan. 2012-2 Feb. 2012. Print.

Øyvind Hauge, Daniela Soares Cruzes, Reidar Conradi, Ketil Sandanger Velle, and Tron André Skarpenes, "Risks and Risk Mitigation in Open Source Software Adoption: Bridging the Gap between Literature and Practice" *Proceedings of 6th International IFIP WG 2.13 Conference on Open Source Systems, Open Source Software: New Horizons, Notre Dame, IN, USA, May 30 - June 2 2010.* Springer. 2010. Web. 16 Feb. 2015. <http://link.springer.com/book/10.1007%2F978-3-642-13244-5>

Øyvind Hauge and Sven Ziemer, “Providing Commercial Open Source Software: Lessons Learned”, *Proceedings of 5th IFIP WG 2.13 International Conference on Open Source Systems, Open Source Ecosystems: Diverse Communities Interacting, Skövde, Sweden, June 3-6, 2009* Springer. 2009. Web. 16 Feb. 2015. <http://www.springer.com/computer/general+issues/book/978-3-642-02031-5>

Porter, Michael. "How Competitive Forces Shape Strategy." *Harvard Business Review*, vol. 57, no. 2, 137-45. Mar. 1979. Print.

Porter, Michael. "The Five Competitive Forces That Shape Strategy." *Harvard Business Review.* Jan. 2008. Print.

Prahov, Radoslav, Holger Schmidt, and Achim Graupner , "Subversion(r): An empirical performance case study from a collaborative perspective on integrated circuits and software development," 4th IEEE International Conference on Software Engineering and Service Science (ICSESS), vol., no., pp.35,42, 23-25 May 2013

"Pricing and Licensing." *MATLAB and Simulink Overview*. MathWorks, n.d. Web. 25 Nov. 2014. <http://www.mathworks.com/pricing-licensing/index.html?intendeduse=home>

"Ptolemy II." *Ptolemy Project*. UC Berkeley, n.d. Web. http://ptolemy.eecs.berkeley.edu/ptolemyII/index.htm, accessed February 16, 2015.

“Ptolemy II Frequently Asked Questions.” *Ptolemy Project*. UC Berkeley, 18 Dec. 2014. Web. http://ptolemy.eecs.berkeley.edu/ptolemyII/ptIIfaq.htm#ptolemy%20II%20copyright, accessed February 16, 2015.

Riedel, Till, Nicolaie Fantana, Adrian Genaid, Dimitar Yordanov, Hedda R. Schmidtke, and Michael Beigl, "Using web service gateways and code generation for sustainable IoT system development," Internet of Things (IOT), 2010 , vol., no., pp.1,8, Nov. 29 2010-Dec. 1 2010

"Sponsors of the Ptolemy II Project." Ptolemy Project. UC Berkeley. Web. http://ptolemy.eecs.berkeley.edu/sponsors.htm, accessed 14 Apr. 2015

Stravoskoufos, Kostas, Stelios Sotiriadis, Alexandros Preventis, and Euripides G.M. Petrakis, "Motion sensor driven gesture recognition for future internet application development," The 5th International Conference on Information, Intelligence, Systems and Applications, IISA 2014, , vol., no., pp.372,377, 7-9 July 2014. Print.

Toulson, Rob, and Tim Wilmshurst, "Fast and Effective Embedded Systems Design: Applying the ARM mbed" Newnes Newton, MA, USA: Elsevier Ltd., 2012, Print.

Tripakis, Stavros, Dai Bui, Marc Geilen, Bert Rodiers, and Edward A. Lee. "Compositionality in synchronous data flow: Modular code generation from hierarchical sdf graphs." ACM Transactions on Embedded Computing Systems (TECS) 12, no. 3 (2013): 83.

Vieri del Bianco, Luigi Lavazza, Sandro Morasca, and Davide Taibi, "Quality of Open Source Software: The QualiPSo Trustworthiness Model", *Springer*, 2009, Web. 16 Feb. 2015.

Wojtczyk, Martin., and Alois Knoll, "A Cross Platform Development Workflow for C/C++ Applications," Software Engineering Advances, 2008. ICSEA '08. The Third International Conference on , vol., no., pp.224,229, 26-31 Oct. 2008. Print.