Model-Based Embedded Software

Jose Oyola Cabello
Kevin Albers
Robert Bui
Naren Vasanad

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2015-120
http://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-120.html

May 15, 2015

Acknowledgement

Professor Edward A. Lee
Professor Sanjit Seshia

University of California, Berkeley College of Engineering

MASTER OF ENGINEERING - SPRING 2015

Electrical Engineering & Computer Sciences

Robotics & Embedded Software

Model-Based Embedded Software

José Raúl Oyola Cabello

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

Model-Based Embedded Software

Final Capstone Report

José R. Oyola Cabello, Kevin Albers, Robert Bui, Naren Vasanad

May 15, 2015

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

A. Overview

Throughout the past year, the team worked to complete the goal that was set for the project: to build a graphical interface for automatically generating code for the mbed[7] platform. In order to confirm the effectiveness of the final product, an application was designed such that it could be built manually first and later code-generated, to compare the quality and efficiency of the generated code. As such, the tasks for the year were divided into two main phases: first, building the application with traditional programming, and later automatically generating code for the application. Each of these main phases had many subtasks, some of which were divided amongst the team members, and others that were worked on by more than one team member at a time in cases where the tasks were complex or time-sensitive. The tasks that were done individually were divided based on the strengths of the individual team members, as well as their experience with previous related tasks.

During the first semester, the focus was on designing and building the application. The application that was chosen was an interactive LED cube that could be controlled and manipulated wirelessly with a data glove. There are a few reasons for this choice of application, and they are discussed in the Methods and Materials section of this paper. Each of the components of the application (the LED cube, the data glove, the wireless communication and the microprocessor) had its own set of tasks. In addition, the application needed to be built such that the data flowed from one component to the next following a Synchronous Data Flow model,

[7] ARM's mbed is an embedded platform that can be used for Rapid Prototyping. More information on this platform can be accessed at its homepage: https://mbed.org/

discussed in the Methods and Materials section, in order to ensure its compatibility with code generation in the second semester. My work was mostly centered around the LED cube component of the application. This component is central to the application because it provides the visual feedback to the user who is interacting with it. The initial tasks for the LED cube included designing how the cube would be modeled, as well as designing the algorithms that would produce the desired changes to the cube based on the user's input from the data glove. The next set of tasks dealt with the hardware and software implementations of the cube, testing, and finally integrating the cube into the final application.

During the second semester, the focus was on extending the code generation capability in Ptolemy II. Although the capability already existed, it had a number of issues that needed to be solved. A large portion of the work done in the second semester was dedicated to solving these issues, as well as working with the specific Makefiles for the mbed platform, which are files that help build the software from its source files. Another main portion of the work in the semester was dedicated to building and testing the blocks, or actors, in Ptolemy II that were designed in the first semester in order to generate code for them. We also focused on integrating the actors and code-generating them into the final application. This final code was then tested and compared against the previous semester's code and application to verify functionality, efficiency and responsiveness. Furthermore, a very important aspect of our work throughout the year was documentation. Each step that was taken was documented, including any issues that arose and how they were fixed. This is vital information for future users of the Ptolemy II project and its code generation capability. Part of my work in the second semester was centered around the preliminary building and testing of actors. This was a very important contribution to the project because it led to the finding of many bugs and issues with the code generator as well as the Makefiles. In addition, this preliminary code generation of actors brought to light a significant

issue with code generation, code size, which I will talk about in the Methods and Materials section of this paper. The rest of my work for the semester centered around the building and testing of the final actors. My contributions in the second semester allowed the team to be able to complete the code generation of the application and perform a comparison of this new version and the previous, manually built version.

B. Relevant Work

Our work builds upon many others under the Ptolemy II project. A number of previous projects laid the foundation for the code generation capabilities in Ptolemy II (Pino, Parks, and Lee 1994; Tsay 2000; Zhou, Leung, and Lee 2007). Pino, Parks, and Lee introduced code generation for heterogeneous multiprocessor systems in the original version of Ptolemy, now known as Ptolemy Classic (1994). In Ptolemy II, Tsay's implementation transforms the Java code in each actor into a more simplified, integrated version, then translates it to C using a generic Java-to-C converter (2000). The next implementation builds upon that one by introducing a "helper" concept, a file that serves as a template for the generated code (Zhou, Leung, and Lee 2007). This last implementation is included in Ptolemy II as of version 6.0.1, and the current code generator works in this same way. Our work improves upon this implementation by locating bugs in the code generator and finding solutions to them. The Methods and Materials section below discusses some of these issues and how they were resolved. Our project also leverages the work of B. Lee and E. Lee on integrating the FSM domain with the SDF domain (1998). This allows for the code generation of hierarchical models that include both models of computation. Another related project by S. Lee, Yoo, and Choi deals with the scheduling of such SDF-FSM hierarchical models (2002). Our project uses this feature in one of the main blocks of our application, which is modeled as a Finite State Machine (FSM), while the application as a whole

runs under an SDF model of computation. I discuss this in more detail in the next section of this paper.

A relevant project by Kim et al. deals with performing code generation for different platform targets (2013). Although our work is not directly related to this one, it is somewhat parallel to our work in the sense that our code generation is also platform-dependent. The Ptolemy II code generator used in this project allows a user to set which target to generate code for. The strength of our method when compared to this one is that if a platform-specific target is not implemented in Ptolemy II, then the code generator will use the next higher-up level target instead, whereas in Kim et al. there must be platform-specific code snippets for each target platform (2013). Another relevant project by Manione seeks to develop a model-based framework for Internet of Things (IoT) development (2014). Our work improves upon this idea by providing a framework for IoT and embedded devices that also provides code generation for such platforms. In addition, our project allows for incorporating the tools and features already available in Ptolemy II into the models and applications a user wishes to build.

One of the main issues with the project, which I discuss later in more detail, is the code size of the generated code. A related project discusses the difficult trade-off in code generation between the modularity of the code and the code size (Lublinerman, Szegedy, and Tripakis 2008). Further, a different paper by two of the same authors discusses another tradeoff in code generation: modularity vs. reusability (Lublinerman and Tripakis 2008). They go on to discuss methods that can be used to maintain modularity without sacrificing the code size significantly. However, modifying the code generator in this way falls outside the scope of our Capstone project, and as a result these methods will not be applied to Ptolemy II.

C. Methods and Materials

Fall Semester

As mentioned in the Overview section of this paper, an important aspect of our work in the project was designing an application that would allow us to show off the ability of code generation to develop interesting embedded applications. Many different options existed, but for various reasons, the application that was selected was an LED cube that can be controlled wirelessly using a data glove. First, it is centered around the IoT, which is the main focus of our capstone project. As mentioned in previous parts of this report, the target market for the project is the hobbyist and open source community, which is currently highly engaged in IoT applications. A second reason for the choice of the application is that it could be designed to be modular. This is an important fact, because each of these modules can then be converted to a block, or actor, that can be modeled and code-generated in Ptolemy II. The final reason for this choice of application is that it is an interesting, interactive application with enough complexity to demonstrate the strength and flexibility of code generation. A simpler application could have been selected instead, but the extent of the usefulness of the code generator would not have been shown. The advantage of using a code generator is that it allows a user to design, model and test large and complex applications and automatically develop the code for them. This application shows how easily that can be done with the code generation capability in Ptolemy II.

The application consists of four physical components: a data glove, a WiFi module, an LED cube, and an embedded microcontroller. The first choice we had to make was which board to support. There are many different boards to choose from, with varying amounts of memory, flash and processing power, as well as differing sizes and form factors. Robert Bui discusses the different board options for the project. The mbed FRDM-KL25Z was chosen because its processing power to price ratio is higher than many of the other popular embedded boards and

because it was readily available for us to use. The amount of memory, 16KB, would be enough to run our application when manually programmed in C/C++.

Selecting the data glove was a more difficult decision to make. Many different data gloves exist, but are usually very costly. The application required a data glove with sensors in each finger to detect how bent the fingers are (bend sensors), and an accelerometer and gyroscope sensor. In addition, the data glove should be able to communicate wirelessly with the mbed board, which could be accomplished with Bluetooth or WiFi. Because our application is focused around the IoT, WiFi communication was preferred. The glove would take measurements from all its sensors and send the data over WiFi to the embedded board, which would then interpret the data and use it to update the LED cube based on what gesture the user was performing. The DG5 Data Glove from VirtualRealities was selected. This glove contains electronic sensors for detecting finger bending, hand rotations and accelerations. It is also WiFi-enabled, which fits the selected application perfectly. In his paper, Kevin Albers and Naren Vasanad discuss more details about the data glove and how it was used in the application. The next component is the WiFi module. Because the mbed board does not include a WiFi module, a separate module was required. This module would serve the purpose of receiving data packets from the data glove and passing them to the mbed board. The CC3000, a WiFi Module that can be used with the board, was selected. In their papers, Naren Vasanad and Kevin Albers discuss the CC3000 in detail and how it was integrated into the application.

The fourth and final physical component of the application was the LED cube. The concept was to have a three-dimensional lighted cube that moved around inside a larger LED cube. Its size, color, and position would be changed based on hand gestures from the user, wearing the data glove. Many different LED cubes have been built in the past, mostly from discrete LEDs built up in a complex three-dimensional structure, which requires a large amount of time to build.

Another option was to build only the sides of the cube, and instead model the inner lighted cube by modulating the brightness of the LEDs. The LED cube would be built such that only the sides are visible, with no LEDs on the inside. The inner cube would then be shown at full brightness if it was touching the side, and at a lower brightness as it moved "inside" the cube and away from the sides, as shown in Figure 1. The figure shows the lighted cube attached to one side panel at full brightness, but at lower brightness on the other two panels, since it is "farther away" from them. This version of the cube would be significantly easier to build and would be more aesthetically pleasing. It was decided that only three sides of the cube would be built, such that all three sides could be viewed from the front. To model the inner cube, the only three pieces of information required to fully describe it would be the location of one of its corners as an (x,y,z) coordinate (with the origin located at the bottom left corner of the first panel, as shown in Figure 2), the length of its sides, and its color. This is the only data that would be saved regarding the inner cube, since with this data we could "build up" the inner cube and "draw" it inside the LED cube.

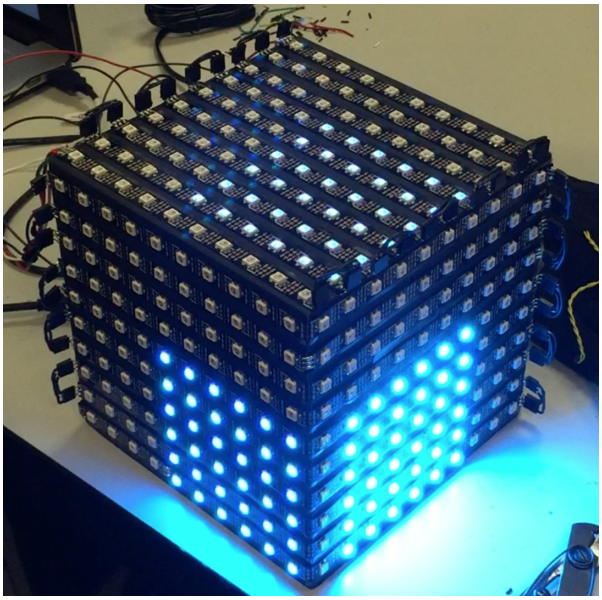

Figure 1: LED Cube showing brightness modulation

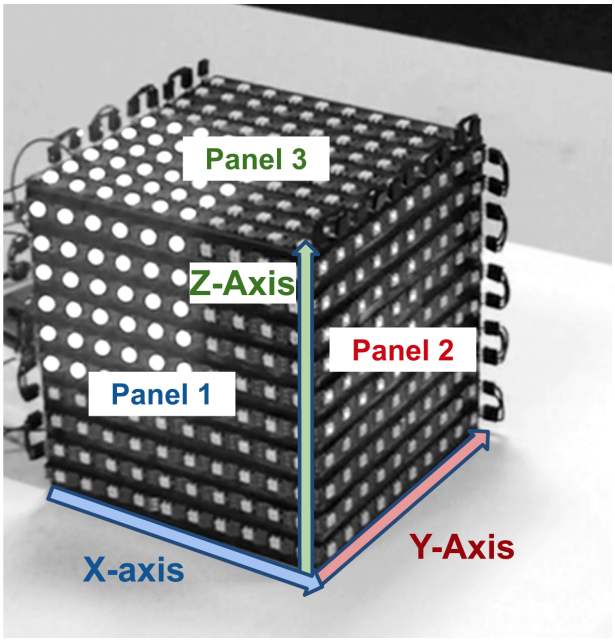

Figure 2[8]: LED Cube showing panel numbers and coordinate axes

The cube was built using NeoPixel LEDs, individually-addressable LEDs that are packaged in continuous strips. These LEDs require only three connections: Vcc, Ground, and a data input, which provides a serial interface through which all LEDs can be individually controlled. The LEDs were cut into 10 strips of 10 LEDs for panel 3 at the top of the cube and 10 strips of 20 LEDs which bent in the middle to form panels 1 and 2 of the cube, as shown in Figure 2. The two sets of strips were daisy-chained using hookup wire, such that only two data pins were required to control all 300 LEDs. This is important because of the limited availability of GPIO pins in the FRDM-KL25Z, taking into account all the components that needed to be connected for this application. The power to the LEDs was provided through an external power adapter to ensure that the maximum power requirements would be met (the case when all 300 LEDs are at full brightness). On the software side, it was of vital importance to design the system to be as

[8] This figure was created by the Model-Based Embedded Software team: Kevin Albers, Robert Bui, José Oyola and Naren Vasanad.

modular as possible, to ensure that it could then be modeled in Ptolemy II. The first step was to find a library for NeoPixel LED strips for mbed. Although a few libraries exist, only one is able to control over 120 LEDs from a single pin: the *Multi_WS2811* library by Richard Thompson[9]. This means that with this library, only two GPIO pins would be required to control all 300 LEDs: one for the 10 strips of 20 LEDs for the side panels and one for the 10 strips of 10 LEDs for the top panel. This library provides a high-level interface with which to interact with the LEDs. In particular, it abstracts away the need to save the current state of the LEDs, and instead allows a user to simply update individual LEDs, as needed. On the other hand, this library is inefficient, saving arrays with static values into memory instead of flash. The LED cube block was built to follow the Synchronous Data Flow (SDF) model of computation, whereby each block "fires", or runs, only when it has all its required inputs, and then produces all its outputs. Robert Bui discusses this model of computation in detail in his paper. Figure 3 shows the overall SDF model for the application. The LED cube block requires three inputs: change in position, the desired color, and the change in size. Structs were used to transfer data from one block to another. In this way, the LED cube block receives input from the Gesture Recognition block and updates the LED cube accordingly.

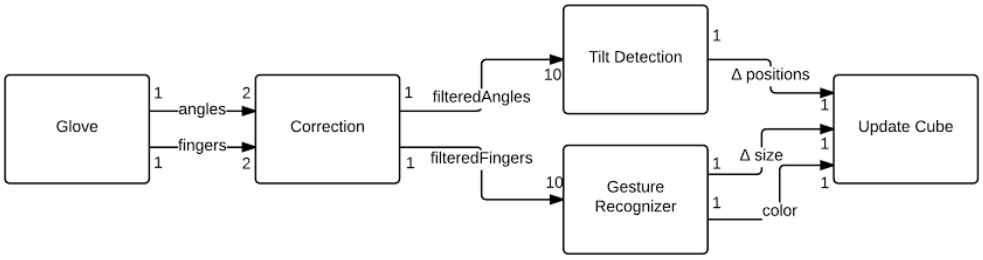

Figure 3[10]: Synchronous Data Flow model of application

[9] The Multi_WS2811 library can be accessed at: https://developer.mbed.org/users/Tomo2k/code/Multi_WS2811/

[10] This figure was created by the Model-Based Embedded Software team: Kevin Albers, Robert Bui, José Oyola and Naren Vasanad.

The LED cube uses algorithms to update the cube, based on the desired changes in position and size. The color of the inner cube can always be changed regardless of its position, because it is based on a desired hue, selected linearly based on a gesture from the glove. However, the LED cube models a space of 10x10x10 LEDs, which constrains the possible the sizes and positions of the inner cube. The algorithm determines whether the size of the cube can be increased or not, depending on its current size, and also updates the corner location in the case that the cube increases its size, causing the corner to change location. The algorithm is as follows:

1. If the desired size is greater than 0 and less than or equal to the maximum size of the cube, continue. Otherwise, exit.
2. If the position of the corner of the cube, added to the new cube size, is within the bounds of the LED cube, then update the size of the cube and return. Otherwise, continue.
3. If the X-position of the corner of the cube, plus the new cube size, is outside of the bounds of the LED cube, then decrease the X-position by the difference.
4. Repeat step 3 for the Y-position.
5. Repeat step 3 for the Z-position.
6. Update the size of the cube and return.

In addition, there is an algorithm to determine which LED corresponds to a coordinate position (x,y), given which panel the LED is on. The algorithm is as follows:

1. If the given panel is panel 1, continue. Otherwise, skip to step 4.
2. If the y-coordinate is even, return `20 * y + x` and exit.
3. Otherwise, return `20 * y + 10 + (9 - x)` and exit.
4. If the given panel is panel 2, continue. Otherwise, skip to step 7.
5. If the y-coordinate is even, return `20 * y + 10 + x` and exit.
6. Otherwise, return `20 * y + (9 - x)` and exit.
7. If the given panel is panel 3, continue. Otherwise exit.
8. If the y-coordinate is even, return `10 * y + x` and exit.
9. Otherwise, return `10 * y + (9 - x)` and exit.

A third algorithm controls the brightness with which to display an LED. Because there are no LEDs inside the LED cube, any LED location that exists inside the cube is modeled by using the LEDs on the sides of the cube, but lowering the brightness of the LEDs following the inverse square law of lighting, simulating the light being further away. The algorithm takes as input the

size of the cube and the position of the corner, and changes the brightness of each LED by multiplying the LED color by a "brightness factor", which is calculated based on panel, as follows:

1. For panel 1, the brightness factor is `1.0 / ((y + 1) * (y + 1))`.
2. For panel 2, the brightness factor is `1.0 / ((9 - x - (size-1) + 1) * (9 - x - (size-1) + 1))`.
3. For panel 3, the brightness factor is `1.0 / ((9 - z - (size-1) + 1) * (9 - z - (size-1) + 1))`.

Spring Semester

In the second semester, the focus shifted to the code generation phase of the project. A previous similar project for Arduino had been developed in Spring 2014, with limited success. However, that work served as an initial starting point for our own. After building an mbed directory in the Ptolemy II structure modeled based on the Arduino directory, we began to manually fix the main differences between the projects. This was the quickest way to get the mbed project going because manually creating the folder structure and the required files would have taken a significantly longer time, and much of the file structure could be reused. In addition to working on understanding Ptolemy II and the process of code generation, a higher priority was given to documentation. The project's wiki page[11] within the larger EECS CHESS (Center for Hybrid and Embedded Software Systems) wiki was used from the beginning of the Spring semester to document the learning process, with particular focus on issues and solutions. The wiki page also serves as a how-to guide for future users of the project.

As mentioned in the overview section, my work in the Spring semester was focused around the building of actors in Ptolemy II for code generation. The Arduino project modified a Display actor in Ptolemy II, usually used for printing data to the console, to turn the Arduino's on-board LED on and off depending on the boolean (true or false) input to the actor. An equivalent modification was made to the Display actor on the mbed project. After generating

[11] The project's wiki page can be accessed at: http://chess.eecs.berkeley.edu/ptexternal/wiki/Main/Mbed

code for the simplest of applications, the main bugs of the code generator and Makefiles began to appear. In his paper, Kevin Albers discusses the issues with the existing code generator and how they were resolved. In addition, Naren Vasanad discusses how the Makefiles for mbed were developed. He also discusses the offline compiler, *gcc4mbed*, which serves as a more streamlined compiler than the mbed online compiler, because it can be integrated into Ptolemy II so that when code is generated, it is also built and compiled.

The EmbeddedCodeActor is an existing actor within Ptolemy II that allows a user to set input and output ports and enter their own C code that reads those inputs and sets those outputs. The actor, when fired, runs the code that the user wrote. It contains in it a series of blocks where the user writes the code. Of note there is the "PreInit" block, where the user can write code that will lie at the top of the resulting C file (such as `#include` and `#define` directives, or global variables), the "Init" block, which is fired at the beginning, and the "Fire" block, which runs each time the actor fires and is where the main C code is placed. This makes the EmbeddedCodeActor an excellent tool for prototyping an idea before making a custom actor from scratch. This actor was used for much of the preliminary code generation tasks because of its flexibility. A first test of the EmbeddedCodeActor was to configure it to flash the LEDs when given a boolean input, mirroring what the Display actors for mbed and Arduino do. This first test resulted in some errors before actually functioning. The main error was that the code generator produces C files, but any files that use mbed-specific code needs to be a C++ file. There was an additional error regarding a multiply-defined struct. The quick solution to the problem was to remove the second definition of the struct, but the underlying problem remained: the code generator multiply defines structs each time there is a composite actor (an actor which encapsulates a model within it, such as the EmbeddedCodeActor) in the model (Ptolemaeus 2014). This problem was later solved after finding it to be a hierarchy problem within the code generator.

Incremental models were built to add functionality. The EmbeddedCodeActor was used to create an initial NeoPixel actor, which takes as input an index of which LED to modify, and three inputs for the color to set (red, green and blue). Code generating this application revealed a very important drawback: code and memory size. In the Relevant Work section, I discuss a relevant paper regarding the tradeoff in code generation between modularity and code size. The overhead associated with code generation, when combined with the size of the NeoPixel library that was used for the mbed FRDM-KL25Z resulted in 88% of memory being used, and the application would not run on this board. A decision was made to instead switch to the FRDM-K64F board, which has 256KB of memory. A different library was required for this board, however, which is more efficient in its use of memory: the *WS2812* library by Brian Daniels. I created a C++ class to encapsulate this library and give it the same interface that was used previously to create the original application on the FRDM-KL25Z. The NeoPixel actor was successful in preliminary tests, and was instrumental in showing that there was another significant problem with the code generator: memory leaks. This problem was noted when applications set to run continuously would only run for a short period of time, one or two minutes at most, and then stop, likely because the memory had been entirely used and there was none left. Memory leaks can be detected by using one of various programs such as *Valgrind*[12] or *Electric Fence*[13] to find where the bug lies. Using *Valgrind*, the leaks were traced and corrected, with the main set of leaks being due to the creation of "tokens", or data that gets transferred between blocks, but never deleted. This main issue was corrected by using creating a macro, `$getAndFree`, for users to use in the C adapter files. This macro gets replaced with a function that returns the token's payload in the payload's type and frees the token. In their papers, Robert Bui and Naren Vasanad discuss the memory leaks in more detail.

[12] The home page for *Valgrind* can be accessed at: http://valgrind.org/
[13] The home page for *Electric Fence* can be accessed at: http://elinux.org/Electric_Fence

After solving the rest of the memory leak issues, EmbeddedCodeActors were built for each of the blocks in Figure 3. Of note is the Correction block, shown in detail in Figure 4, which was originally designed as an FSM model. Ptolemy II and its code generator support including FSM models of computation within an actor that can be included in a larger SDF model. There is a caveat, however, in that not every FSM actor is an SDF actor. In his paper, Robert Bui discusses this in more detail. One problem that was not able to be solved was that FSMs seem to run for only a limited period of time, and then stop. The issue behaves as a memory leak, but Valgrind shows no more errors, making it difficult to trace the source of the problem. Instead, the Correction block was implemented as an SDF actor, with C code that behaves like a state machine.

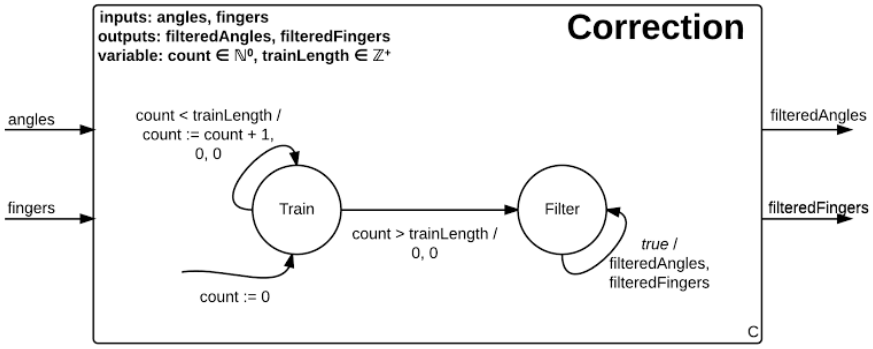

Figure 4[14]: Correction block in detail

The blocks were created starting with the LED cube actor and moving backwards until the CC3000 WiFi actor was built. As each block was created, it was connected to the previously created blocks and sequence actors were used as inputs to the system. This allowed the team to analyze the corresponding outputs, in this case, behaviors in the LED cube. The LED cube itself served as a debugging tool, as in a few instances it was used by the team to investigate where the generated code was failing. After all the blocks were included in the same model simultaneously, the final application was successfully code-generated and placed onto the mbed platform.

[14] This figure was created by the Model-Based Embedded Software team: Kevin Albers, Robert Bui, José Oyola and Naren Vasanad.

Afterwards, the EmbeddedCodeActors were converted into Ptolemy II custom actors and placed in a folder accessible from Vergil. The entire process explained was well-documented in the project's wiki page.

D. Results and Discussion

Fall Semester

By following the design mentioned in the Methods and Materials section, the application for the project was created. It worked as expected, although with a few changes. The WiFi communication between the mbed board and the data glove resulted in issues that were resolved by instead connecting the glove to a computer, which then connected to the mbed. The DG5 Data Glove was limited in its ability to capture gestures from the user's hand, because many different hand and finger positions resulted in very similar measurements that were very difficult to identify. The possibility of using machine learning to more accurately capture gestures was considered, but due to a lack of time and resources, it was decided that simple gestures would be used instead. The Gesture Recognition block was able to capture and correctly identify 10 gestures, six of which correspond to hand rotations (shown on Figure 5) that map to movements of the lighted cube. The other four gestures, shown in Figure 6, are for decreasing size of the cube (pointing with the index finger), increasing the size of the cube (pointing with the index and middle fingers), changing the color of the cube (extending the thumb, and using the index finger, where color changes in a continuous manner with the magnitude of the index finger's bending), and the neutral gesture, which implies no changes. Each of these gestures had a corresponding manipulation of the LED cube, as described in the above section. The LED cube itself was successful in responsiveness and accuracy when updated. A number of difficulties arose when building the cube, mainly due to the solder joints and hookup wire used to connect the LED strips together. Because the cables were desired to be as flush as possible with the cube, the solder

joints were under tension, causing some of the joints to break, requiring further soldering even after the cube had been built. In terms of memory and flash usage, the application used 41.8KB (44%) of flash and 10.9KB (68%) of memory. Before finding the final library that was used for the NeoPixel LED strips, it was thought that the memory would not allow for all 300 LEDs to be used on one mbed board. As mentioned in the previous section, the library does its own memory management, but does so very inefficiently, using a very large portion of memory for a single instance of a NeoPixel object. Once the final library was found, however, only two instances were required, significantly lowering the amount of memory required. Overall, the application was successful and brought significant attention to the project in the Capstone Expo at the end of the Fall semester.

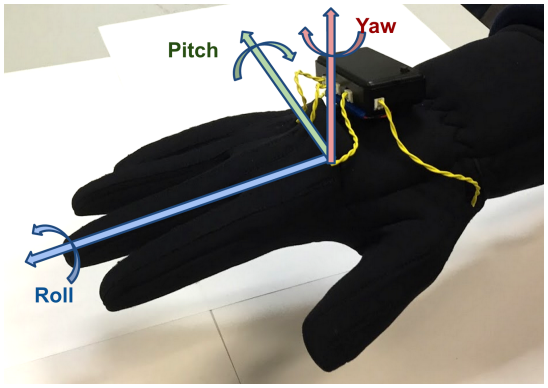

Figure 5[15]: Data glove rotation gestures

[15] This figure was created by the Model-Based Embedded Software team: Kevin Albers, Robert Bui, José Oyola and Naren Vasanad.

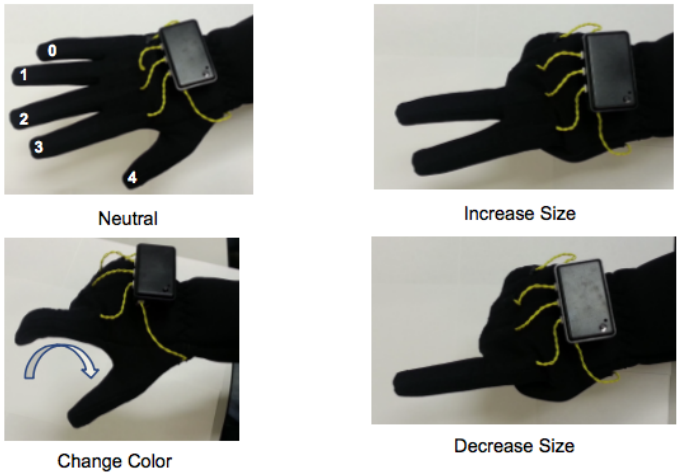

Figure 6[16]: Data glove finger gestures

Spring Semester

After solving the initial issues with the code generator in Ptolemy II, and being able to create Makefiles that correctly built the code for the mbed platform, the code generator is now able to produce code that requires no manual changes in order to make it work with the board. The offline compiler makes code generation with Ptolemy II a very streamlined process, simply requiring the user to run the command which generates the code, and the compiler will automatically build a binary file which can be run on mbed.

The incremental building and testing of the custom actors for the final application was successful. Creating the LED cube actor before all the others allowed the team to simulate the behaviors of the glove and analyze the resulting behavior of the LED cube. The code-generated application actually ran more smoothly than the original, manually coded version from the Fall semester. However, this may not be completely due to code generation itself, but at least partly due to the team's careful inspection of the code while developing the Ptolemy actors, allowing us to find more efficient ways of performing computations and saving memory, as well as a new

[16] This figure was created by the Model-Based Embedded Software team: Kevin Albers, Robert Bui, José Oyola and Naren Vasanad.

WiFi router that suffered significantly less packet loss. Every actor was created as a custom actor and placed in a folder accessible from Vergil, making it easy to build the entire application within minutes. This makes our work on Ptolemy II code generation a successful mechanism for creating embedded software.

V. Concluding Reflections

The Model-Based Embedded Software project was highly successful in meeting the goals that were set at the beginning of the year. The original plan was to create a model-based software development environment for embedded platforms, and the team was able to achieve this goal. The Ptolemy II project already had a functioning code generator capable of generating C code, but there were many bugs that caused the generated code to be unusable in an embedded system due to the limited memory in these devices. As a result, as part of the work required to meet the goal of the project, the team also took on the task of finding solutions to the problems associated with Ptolemy II's code generator, such that the Ptolemy II project has also benefitted greatly from our work. However, not all of the goals were completed as originally planned. One of the goals was to have a large library of components pre-built so that a user could build complex models for our embedded platform and code generate them. However, more time was spent in trying to resolve the existing problems than originally intended, such that less time was available to pre-build additional blocks. On the other hand, all of the actors that were built for the LED-cube application were designed to be general and not application-specific, such that they can be used in a wide range of models for varying purposes.

The capstone project also brought forth many interesting insights with respect to project management and working in teams. Although all four team members have similar backgrounds in Electrical Engineering and Computer Science, it became clear as the year progressed that each member had areas of expertise that could be leveraged to better complete the goals of the project.

By using a rolling-wave project management strategy, the team was able to adjust the project plan as the team members demonstrated technical abilities in completing certain types of tasks, while a fixed project plan would have not allowed these kind of adjustments to take place.

A very important aspect of our work in the capstone project was documentation. The Ptolemy II code generator is lacking in documentation, which translates to a very steep learning curve when someone is first exposed to it. The team made sure to document each step that was taken throughout the year to make it easier for someone to continue or build upon our work. In particular, the team's wiki page[17] was created as a guide, showing step-by-step instructions on code generation in Ptolemy II. There are guides on how to edit makefiles, how to set up the offline compiler for mbed, and how to create custom actors for code generation, to name a few. This will make it significantly easier for someone to continue the work that was done this year, and will allow for addition of some useful features that could be used to enhance the model-based design experience for embedded platforms. In particular, the code generator for Discrete Event models is currently not able to generate code at the same quality as the one for Synchronous Data Flow, which we used in our project. This future work for code-generating Discrete Event models could open the door for very interesting and useful applications that require or can benefit from temporal dynamics.

[17] The project's wiki page can be accessed at: http://chess.eecs.berkeley.edu/ptexternal/wiki/Main/Mbed

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>

"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/>

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.

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.

Kim, BaekGyu, Linh T.X. Phan, Oleg Sokolsky, and Insup Lee. "Platform-Dependent Code Generation for Embedded Real-Time Software." *Proceedings of the 2013 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems CASES '13, Montreal, Canada, 29.09-04.10.2013*. New York (NY): A.C.M., 2013. 1-10. Print.

Lee, Bilung, Edward A. Lee. "Hierarchical Concurrent Finite State Machines in Ptolemy." In *International Conference on Application of Concurrency to System Design, Fukushima, Japan, March 1998, Proceedings*. 34-40. Print.

Lee, Sunghyun, Sungjoo Yoo, and Kiyoung Choi. "Reconfigurable SoC Design with Hierarchical FSM and Synchronous Dataflow Model." *CODES 2002 Proceedings of the Tenth*

International Symposium on Hardware/Software Codesign: May 6-8, 2002, Estes Park, Colorado. New York, NY: Association for Computing Machinery, 2002. 199-204. Print.

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.

Lublinerman, Roberto, Christian Szegedy, and Stavros Tripakis. "Modular Code Generation from Synchronous Block Diagrams: Modularity vs. Code Size." *POPL'09: Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages: Savannah, Georgia, USA, January 21-23, 2009*. New York, NY: Association for Computing Machinery, 2008. N. pag. Print.

Lublinerman, Roberto, Stavros Tripakis, "Modularity vs. Reusability: Code Generation from Synchronous Block Diagrams," *Design, Automation and Test in Europe, 2008. DATE '08* ,1504,1509, 10-14 March 2008

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.

Manione, Roberto, "Short paper: A model based framework for effective Web of Things development," *2014 IEEE World Forum on Internet of Things (WF-IoT)*, 191-192, 6-8 March 2014

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>

Pino, J.L, T.M. Parks, E.A. Lee, "Automatic code generation for heterogeneous multiprocessors," *ICASSP-94., IEEE International Conference on Acoustics, Speech, and Signal Processing, 1994*, 445-448 vol.2, 19-22 Apr 1994

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.

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

Ptolemaeus, Claudius, Editor, *System Design, Modeling, and Simulation Using Ptolemy II*, Ptolemy.org, 2014.

"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.

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

Tsay, Jeff. “A Code Generation Framework for Ptolemy II.” ERL Technical Memorandum UCB/ERL No. M00/25, Dept. EECS, University of California, Berkeley, CA 94720 (May 19, 2000)

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.

Zhou, Gang, Man-Kit Leung, and Edward A. Lee. "A Code Generation Framework for Actor-Oriented Models with Partial Evaluation." *Embedded Software and Systems Third International Conference, ICESS 2007, Daegu, Korea, May 14-16, 2007, Proceedings*. Berlin: Springer, 2007. 193-206. Print.