Model-Based Embedded Software

Kevin Albers
Robert Bui
Jose Oyola Cabello
Naren Vasanad

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2015-127
http://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-127.html

May 15, 2015

Acknowledgement

Professor Edward Lee
Professor Sanjit Seshia
Christopher Brooks
TerraSwarm Project

University of California, Berkeley College of Engineering

MASTER OF ENGINEERING - SPRING 2015

Electrical Engineering & Computer Sciences

Robotics & Embedded Software

Model-Based Embedded Software

Kevin Michael Albers

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

Kevin Michael Albers

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.

Model-Based Embedded Software

Final Capstone Report

Kevin Albers
Robert Bui, José Oyola, 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

The goals of the project were very broad, and we had the freedom to decide which direction to go. In the beginning, the goal of the project was to create a model-based code generator for an embedded platform. We were responsible for defining our own goals and deliverables based on this description. Our advisors recommended that we start with an Arduino code generator that was previously developed at Berkeley using the Ptolemy II[7] research project. My team decided to spend the first part of the first semester defining and refining the project goals and deliverables. During this period, we decided that the rest of the first semester we would focus on developing an application for the model-based embedded code generator by using traditional C/C++ programming. In the second semester, we would implement the actual code generator and use this to develop the application from the first semester. This would allow us to compare the generated code to the hand-written version, which would allow us to find the limitations of model based code generation.

After brainstorming many ideas for the first semester application, my team decided to pursue an idea that involved manipulating the lights on an LED cube with a data glove when the user performed certain gestures. We decided that this would be a good project

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

for our target audience of hobbyist. As a team, we decided on what materials to use, and decided to use the mbed[8] as the target platform. This meant that the mbed would be the target platform for the code generator that was developed during the second semester.

This section of the project was divided into four main tasks. The Wi-Fi and data glove packetization needed to be developed to access the sensor information from the data glove. This information was then fed to a correction and gesture recognition algorithm in order to translate the raw sensor information into cube translations. Then the LED cube needed to be constructed and algorithms developed for manipulation of the virtual cube. These first three tasks were the core for a working application. In order to connect this project to our main goal of model-based code generation, a model needed to be made with synchronous dataflow (SDF) and finite state machine (FSM) models of computation. This model would later be used to demonstrate the application as it would appear in the Ptolemy II model in order to code generate the application for the second semester.

The second semester started out with a more uncertain plan, which was developed as the project progressed. The first step was to adapt the previously done Arduino project to generate code for the mbed. This first step involved several tasks that could be divided among teammates. First, we needed to adapt the Ptolemy II actors currently used in the simple blinking light model to generate code for the mbed. Second, an offline compiler for the mbed needed to be adapted, and the makefile generator needed to be able to use the offline compiler to create a binary file directly from the model. In addition, there were

[8] mbed is an embedded platform with an ARM-based microcontroller, which can be used to develop our application. <https://mbed.org/>

several bugs in the C code generator in the Ptolemy II project Java files that prevented it from working with C++, which some of the mbed libraries required.

After the Arduino project was adapted, more work needed to be done in order to be able to code-generate our application from the previous semester. The next step explored the possibility of hierarchical and FSM code generation. In addition, the rest of the required actors needed to be adapted from existing ones or created from scratch.

Overall, every team member contributed in some way to every subtask throughout both semesters. We would work as a group to brainstorm ideas and start the main tasks, then each of us would take responsibility to complete the task further on our own. Since we all have similar backgrounds, we determined how to divide the tasks by working on whatever interested us the most personally. Part of this was determined by whoever made the most progress on a task while we worked on it as a group, since that person would most likely successfully complete the task in the smallest amount of time. I developed the software for the data glove packetization and Wi-Fi during the first semester. During the second semester, I fixed errors with the current code generator which prevented it from working correctly with C++ files. These tasks were my main contributions, and I will address them in the most detail. My team will cover the details of the other tasks depending on their main contributions. In addition to covering these topics, I will cover our future work to create and adapt more actors that will be the building block to model and generate code for our application from the first semester.

In addition to the main technical tasks, our team decided to document our work, and create a guide that will help future developers create a code generator for their platform

using Ptolemy II. We decided to do this because Ptolemy II is a huge research project that took our team several months to understand it enough to contribute to the project. With proper documentation about the methods that we used to develop for Ptolemy II, a future group will be able to get started right away without running into the difficulties we had. In order to achieve this, our team created an online page on the Ptolemy II wiki[9] to write down everything that we did individually. Therefore, every team member contributed to this part of the project.

B. Literature Review

As mentioned before, our project is an adaptation of the model-based code generator for the Arduino Yun[10]. This project managed to produce a working prototype code generator that could generate code to blink an LED on the Arduino. Our project expands on this in two ways. First, we developed the same application on a different target platform. Our project works on the mbed (Freescale FRDM-KL25Z) board, while the previous work's target was an Arduino Yun. Second, we expanded the functionality of the code generator to work with more complex models and be more versatile for a wider variety of applications.

Our project was developed with the Ptolemy II research project at UC Berkeley. A book covers the information necessary to both use and develop for Ptolemy II (2014). This resource covers the different Models of Computation (MoC), which determine the semantics of different types of models that affect how the code is generated. It also provides an overview of the Ptolemy II code structure, which allows us to learn how to

[9] Our wiki for the mbed project. <http://chess.eecs.berkeley.edu/ptexternal/wiki/Main/Mbed>
[10] Wiki for the Ardunio Yun project. <http://chess.eecs.berkeley.edu/ptexternal/wiki/Main/ArduinoYun>

develop new actors to work with our code generator. Another resource that gives a detailed introductory overview of Ptolemy II, is a presentation by Brooks and Lee (Brooks et. al., 2010).

Ptolemy II utilizes a large amount of resources in its implementation. The following literature consists of topics important to our project and relevant to the implementation already done in Ptolemy II. In his work, Edward Lee defines the semantics of the FSM director in Ptolemy II (2009). Our work relies on the use of this type of model to create parts of our applications. Another paper by Johan Eker et. al., argues the Hierarchical approach to creating models (2003). Hierarchy allows smaller models to be encapsulated in the a larger model. This method helps make models more readable and reusable, since these hierarchical blocks can be reused. This is the same approach we used to build our models in Ptolemy II.

In addition, outside of the Ptolemy II project, there are other resources related to Model-Based design. For instance, "Elements of Model-Based Design" by Jeff Jensen outlines the steps to be taken for Model-Based design environment (2010). There are 10 steps total, and our project falls into the "Synthesize Software" stage, which is where code generation happens (Jensen, 2010). This work shows how a model-based code generator can fit into the overall design process and process more efficient. Lublinerman et al's paper uses an approach to reduce code size and optimize modularity of code segments (2009). While code size will be an issue with our project, the actual code generator and any optimization associated with it is already implemented in Ptolemy II. Our work will focus on compiler optimization instead of the generated source code itself.

Other related work includes literature by Krizan, who published a paper about C Code generation from Matlab Simulink models. The goal of the paper is similar to ours, in that it looks to model-based code generation to reduce errors from the designer. However, the focus on the application is on controls for the avionics industry, which is closer to the focus of Simulink (Krizan, 2014).

In the field of Internet of Things (IoT), In-Su Yoon offers a solution to offer low power TCP/IP to embedded devices (2009). Our work requires that this protocol is used, but in our application does not require low power. While Yoon's method would be a more formal approach and interesting to implement in Ptolemy II, this will fall out of the scope of our project.

Ignatyev gives a method for checking constraints in C++, and Ramalho offers a way to check C++ by using models (2014; 2013). This work focuses on checking the semantics of C++ to verify certain constraints. These are possible methods that can be used in our project if we decide to implement verification of the C source code.

C. Methods and Materials : Fall Semester Application Project

In order to build our LED cube application for the fall semester, various materials needed to be selected. The most important piece was the selection of the embedded platform. We decided that we should choose from the most popular hobbyist platforms, since this selection will support our target market (see our business strategy paper). This narrowed down the selection to Arduino, mbed, or Raspberry Pi. The Arduino has the advantage of the previous work done, although instead of the Arduino Yun, we considered a lower power and lower cost version of the Arduino, the Arduino Uno. The mbed was an

interesting new platform that focused on Internet of Things (IoT) applications, which was a part of the market segment we wanted to target. The last main option was the Raspberry Pi, which is a miniature computer that works well as an embedded platform. Unfortunately, the Raspberry Pi does not have the high timing precision required to run the Neopixel LEDs [11], the type of LEDs we used in our application. In the end, we chose the mbed, because it was an interesting new platform, with the same focus around IoT that we wanted our project to have. Specifically, we used the KL25Z, which is meant to be a lower end and low power board, ideal for IoT applications. Robert Bui covers the differences between the Arduino Uno and mbed in more detail.

The LED cube was physically constructed by our team using Neopixel LEDs. We decided to layer the outside of three faces of a cube, since that is the most a user could use at any time. Foam board was used to construct the surfaces of the cube to attach the Neopixel LEDs. For network communication,we could have used either Wi-Fi or Bluetooth. Since the CC3000 wifi module was available to use, we decided to use this for our wireless connectivity to connect the mbed to a network over Wi-Fi since the mbed does not have an internal Wi-Fi module.

The last major part of our application required us to decide what data glove to use. A data glove will generally have accelerometers and a gyroscopes to be able to detect the rotation of the glove. They also have bend variable resistors in the fingers to be able to detect how bent the wearer's fingers are. In order to transfer this information, data gloves will need some sort of interface to transfer information, such as a Wi-Fi module or USB

[11] See <http://www.adafruit.com/product/1138>

port. We had two choices for a data glove, either build one or buy a premade one. The benefits of building one would give us much better control over the protocols and timing of the communications and hardware of the data glove itself. However, it would take a lot more time and effort to build one from scratch, which would take time and resources away from the rest of the project. In the end, we decided to purchase a data glove, the VirtualRealities DG5[12], which supported Wi-Fi over TCP/IP. The data glove has an accelerometer, gyroscope, and bend sensors. It simplifies the rotation process by sending rotation information directly, instead of raw sensor data that would need to be processed to obtain the rotation. In order to support Wi-Fi connection, we set up our own Wi-Fi router to keep a reliable connection.

The main development work for this application was C++ programming on the mbed online compiler. The mbed was the central core to our project, since it interfaced to the LED Cube using GPIO (General Purpose IO) peripherals. José Oyola will talk about this in detail in his paper. In addition, the mbed interfaced to the data glove over wifi using the CC3000. Both of these interfaces required hardware to be wired in correctly, then interfaced with the software on the mbed.

As mentioned above, the data glove is used to determine the state of the user's hand based on its internal sensors. Accelerometer and gyroscope data are used to output rotational quaternion data internally, which is sent to a client over Wi-Fi, along with finger bendness data as a percentage. This data is used to determine gestures and cube translations, which will be talked about in more detail in José Oyola's and Naren Vasanad's

[12] For details see: <http://www.vrealities.com/products/data-gloves/dg5-glove-3-0>

paper. The data glove offered a simple packet structure for data sent over the TCP Wi-Fi network. The structure is as follows:

[Header (‘$’)] [Command] [Package Length] [Package Data] [Checksum] [End Character (‘#’)]

A simple start command begins streaming requested data from the the data glove over a Wi-Fi network. The data packet provides information not only about quaternion and finger sensor data, but also a clock counter that timestamps when the data was logged. This information would be useful if we needed to process the raw data to determine the position, rotation, and overall state of the glove. However since no one in our team had knowledge about these subjects we did not explore this method.

D. Results and Discussion: Fall Semester Application - WiFi

The Wi-Fi connection to the glove turned out to be the most troublesome part of the system in terms of latency. The data glove required the use of a TCP connection in order to receive data from the glove. This can cause problems with real time systems in embedded systems. The system on the mbed device is triggered by receiving packets from the data glove, which means the timing of the entire system is reliant on receiving data over Wi-Fi. When packets are missed, TCP will request for them to be redelivered, and wait for the new packets to arrive even if new packets have arrived, so that they are not received out of order. This causes delays up to 1 to 2 seconds which will cause the application to freeze occasionally throughout the process.

In order to see the consistency of data delivery from the data glove, an LED on the mbed was set to blink for every 50 packets received. Since the PC client sends 50 packets

per second, this will blink once a second. When observed during runtime, the LED blinked inconsistently. For short periods it would blink at a constant 1 Hz, then it would slow down or stop blinking for a moment, then recover by blinking faster over the next period, which reflects the problems that would be expected by using TCP. Preferably, UDP should be used for this type of application, since it will simply ignore dropped packets. The data glove provides time data which can then be used to properly time the packets and adjust the algorithm for any dropped packets detected by this method. However, the limitations of the data glove that we used prevented us from using this type of connection.

E. Methods and Materials: Spring Semester Code Generation

The code generator segment of our project was built using the Ptolemy II research project at UC Berkeley. Our first task during this time was to learn how to use and develop this research project. We met with the manager of the project, Christopher Brooks, to help us get started and learn how code structure works. Our job was extend this research project for the the current GUI and code generator to work with the mbed platform. We used the Arduino project, which was also built into Ptolemy II, to copy over the same file structure that we could use to adapt for the mbed. During this learning phase we also learned about the different Models of Computation (MoC), which we can use to build our models. Robert Bui will cover this portion in more detail. This stage allowed us to break the project up into smaller pieces and get a better idea about what our project would be capable of in the end.

The first stage was to adapt the Arduino project to work with the mbed. In this case, for any mbed model with a display actor, the display actor was adapted to turn on or off an

LED if a true or false value was sent to it, but only when the code generator target is the mbed. This was done by editing a template .c file which would substitute in code for the fire command. Robert Bui covers the template .c and actor structure in more detail in his paper. After these changes were made, the C code that was generated was uploaded to the mbed online compilers, and the errors that were produced were fixed manually. Once these errors were logged, we could then go back to the code generator in Ptolemy II, and find how to fix the bugs by making changes there.

The second part of this first stage was to create a makefile that would allow Ptolemy II to build a .bin file that can be directly downloaded to the mbed, instead of requiring the user to upload the c source code to the online compiler. The mbed online compiler allowed us to download a project to work with a makefile and the offline gnu compiler. By combining the information from this makefile, with the template makefile from the Arduino project, it was possible to create a template makefile for the mbed project. Naren Vasanad covers this in more detail in his paper.

The next steps required us to expand upon what was done with the Arduino project by using similar methods. With the Arduino project, only the SDF director was used to produce code and the models did not include hierarchy. Both FSM directors and hierarchy must work correctly for our application to be modeled in a reasonable fashion. The method used to develop this part of the project was incremental. We first made a model of the simple blink light application but with a FSM inside an SDF model. This allows us to find the bugs and figure out what breaks down when FSM and Hierarchy are used. When this works, then a slightly more complicated model can be built, which can be parts of our main

application. This incremental method reduced the overwhelming amount of bugs that would happen by building our application into a model all at once. Instead, the bugs were isolated and solved in smaller steps.

In order to develop the rest of the actors that were required for the main application, an incremental approach was used again. Instead of just adapting the "display" actor that was required for the blinky LED, more actors such as a GPIO actor, Wi-Fi actor, or even a Neopixel LED actor were adapted or created. The same approach was used as before, and small models were built with only one new actor added with each iteration, usually by first using the embedded code actor to test the actor first. When the actor was adequately tested, we moved on to the next one until enough actors were created to be able to complete the overall model of our application.

The final step was then to create the model in Ptolemy II for our application from the first semester. When the previous tasks were finished, all the building blocks were available to create the model. Each of the three main parts were tested individually to assure that all the actors were working correctly. We started by creating a small model to test the Neopixel LEDs, and worked backwards from there. Next the LED cube was implemented, then the gesture recognition and training using fake input data. Then finally the CC3000 was added to complete the model of the application.

F. Results and Discussion: Spring Semester Code Generation

We ran into problems with the C code generator in Ptolemy when the generated code needed to be converted to C++. In general, C++ was designed to support everything written in C, but there are a few exceptions to this case, which we ran into when dealing

with the Ptolemy Code generator. Naren Vasanad talks about this from the perspective of the makefile in his paper.

C allows a program to define a variable twice with the same name, and the linker will take care of matching the variables accordingly depending on their names. This means that in C, it is okay to declare a variable in the header file, and if multiple files access this header file there will be no error for defining the variable more than once. In C++, this is not the case. The linker will produce an error telling the user that the variable was declared from multiple object files. The proper way to define a variable that might be accessed from several files, is to declare the variable as "extern" in the header file, and then put the actual declaration of the variable in the source file. The "extern" keyword tells the compiler to look for the variable at link time, but does not actually declare an instance of it.

In most cases, Ptolemy II used this method to declare these types of variables. In a few instances, this was not done correctly. The bug was corrected in the Java source code of Ptolemy II. In another instance, variables were being defined multiple times because of an error with hierarchy. The higher level director would also define the set of variables used in the hierarchical model, while the director in the hierarchical model generated the same variables. In C, this would not cause an error since the linker would take care of the problem by making them the same variable. In C++ this was problematic and the model would not compile. This was fixed by only allowing the director of the model to produce the variables for themselves, and not the models in their hierarchy.

A second difference between C and C++ is that C allows a parameter to be passed as "void *", which is a pointer to an unknown type. C++ will not accept this, and needs the type

to be declared in order to compile correctly. Since this was a strategy used in a large amount of Ptolemy II, this error is not as easily solved without making major changes to the code generator. The solution to this problem is to be careful which files are converted to have a .cpp file extension. Only the files that have code exclusive to mbed need to be converted. This is because interface to mbed libraries and drivers are done by using classes, which can only be done in C++. The source files that require "void *" to be used only happen in the director files, and not the source files produced for mbed. When only the appropriate files are renamed, the files compile correctly without errors. In order to rename files, a script written in BASH in the makefile is used. Naren Vasanad covers this in more detail in his paper.

V. Concluding Reflections:

The goal of our project was to create a model-based embedded software environment using Ptolemy II, and develop a sample application with the created environment. Our team successfully adapted the Ptolemy II code generator to generate code for two mbed boards, the KL25Z and the K64F. In addition, a small library of actors were created to work with embedded boards and the Ptolemy environment itself. Finally, the main task of the project was completed when a model for the final application was built by using standard and custom actors created in the previous steps. While some of created custom actors were completed, some of them are missing the Ptolemy implementation, so the model can generate code, but cannot be ran in Vergil.

The capstone project produced some value insights into project management. Our project was not well defined and the tasks difficulty and length were hard to know at the

time of the creation of the project plan. Due to this, our team decided to use a rolling wave planning style. This allowed us to create a rough project plan at key points throughout the semester, then iterate on them every week when our team would meet. Project management also brought challenges with allocating tasks that aligned with everyone's interest in the group. Even though we all had similar backgrounds in Electrical Engineering, each member had a different task preference. This made the task division best done at the weekly meetings, when the tasks were better defined and each member knew what aligned with their interests. In addition, we had challenges with our risk aversion plan when the code generator ran into memory problems. While this put the project behind schedule, we were able to continue with other tasks, while a few members continued working on the problems. In the end, this led to a few fewer features that were desired in our project, but the project was still a success.

Our project required us to learn about the Ptolemy II code generator, with sparse documentation. Part of our project included improving the documentation on our wiki[13]. Future work should start with this documentation to learn how the develop under Ptolemy II. From here, there are two recommended options to continue work. They could continue the work we have done, and create more actors and adaptors for the mbed platform. This would expand the current libraries and make Ptolemy II much more useful for code generating on the mbed. Another option is to develop a code generator for another platform, to create a larger breadth of devices covered.

[13] See 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/>

Christopher Brooks, Edward A. Lee. "Ptolemy II - Heterogeneous Concurrent Modeling and Design in Java". Talk or presentation, 11, February, 2010; Poster presentedat the 2010 <http://www.eecs.berkeley.edu/BEARS>.

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/>

Claudius Ptolemaeus, Editor, "System Design, Modeling, and Simulation using Ptolemy II", Ptolemy.org, 2014. Print and Web. <http://ptolemy.org/systems>

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.

Edward A. Lee, "Finite State Machines and Modal Models in Ptolemy II", 1 Nov. 2009. Web. 16 Mar. 2015, <http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-151.pdf>

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>

Ignatyev, V., "Static Analysis Usage for Customizable Semantic Checks of C and C++ Programming Languages Constraints," *Software Testing, Verification and Validation Workshops (ICSTW), 2014 IEEE Seventh International Conference on* , vol., no., pp.241,242, March 31 2014-April 4 2014

In-Su Yoon; Sang-Hwa Chung; Jeong-Soo Kim, "Implementation of Lightweight TCP/IP for Small, Wireless Embedded Systems," *Advanced Information Networking and Applications, 2009. AINA '09. International Conference on* , vol., no., pp.965,970, 26-29 May 2009

"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>

Jeff C. Jensen, " Elements of Model-Based Design", 19 Feb. 2010. Print.

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.

Johan Eker, Jorn W Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Sonia Sachs, Yuhong Xiong, Stephen Neuendorffer. " Taming heterogeneity - the Ptolemy approach". *Proceedings of the IEEE*, 91(1):127-144, 2003.

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.

Krizan, J.; Ertl, L.; Bradac, M.; Jasansky, M.; Andreev, A., "Automatic code generation from Matlab/Simulink for critical applications," *Electrical and Computer Engineering (CCECE), 2014 IEEE 27th Canadian Conference on* , vol., no., pp.1,6, 4-7 May 2014

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.

"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.

R. Lublinerman, C. Szegedy, S. Tripakis. "Modular Code Generation from Synchronous Block Diagrams." *ACM Symposium on Principles of Programming Languages*. Jan. 18, 2009. Print.

Ramalho, M.; Freitas, M.; Sousa, F.; Marques, H.; Cordeiro, L.; Fischer, B., "SMT-Based Bounded Model Checking of C++ Programs," *Engineering of Computer Based Systems (ECBS), 2013 20th IEEE International Conference and Workshops on the* , vol., no., pp.147,156, 22-24 April 2013

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

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.