Component Based MMIX Simulator using Multiple Programming Paradigms - A Proposal

Stephen Edmans

Contents

List of Figures

1 Introduction

As software systems get larger and more complex there is a need to handle this complexity. There is a prevailing design paradigm, which addresses these issues, that is to break these systems up into smaller components. This is a sentiment mentioned by Turner [6] He calls these components "collections of modules", these components will interact with each other to make the complete system.

When you have control over the development of more than one of these components it is a traditional approach to use a single programming paradigm for your components. There is, however, no reason that you cannot use different languages and paradigms for these for each components. The goal of this project is to create a relatively complex system that is made up of multiple components where each component uses the most appropriate programming paradigm for the relevant component.

The system that I plan to create in this project is inspired by Jeliot [7], which is a tool that is used as an aid in the teaching of Java. The Jeliot system allows a user to give it a piece of Java source code and it will show the user what the underlying java virtual machine is doing when it runs the code.

In his seminal work The Art of Computer Programming [2] Professor Donald Knuth designed an artificial machine language that he called MIX. In a later volume of his work Prof. Knuth updated this machine architecture, which he calls MMIX. He later detailed this new version of the architecture fascicle [1] This project will create a system that take MMIX assembly code and shows the user, graphically, what the simulated machine is doing.

2 Existing Implementations

There is already one existing main implementation [3] created by Professor Knuth and his team at Stanford University. This implementation runs as a few command line utilities. These utilities include an assembler which converts a text file containing the relevant assembly language source code into a proprietary binary format. There is a utility which reads files in this proprietary binary format and displays the content back to the console. There is also the main utility which will execute the application stored in the binary files using an MMIX virtual machine. It is possible to get the virtual machine to output some tracing information but it does not have a graphical user interface and it does not allow you to step through the processing of the application.

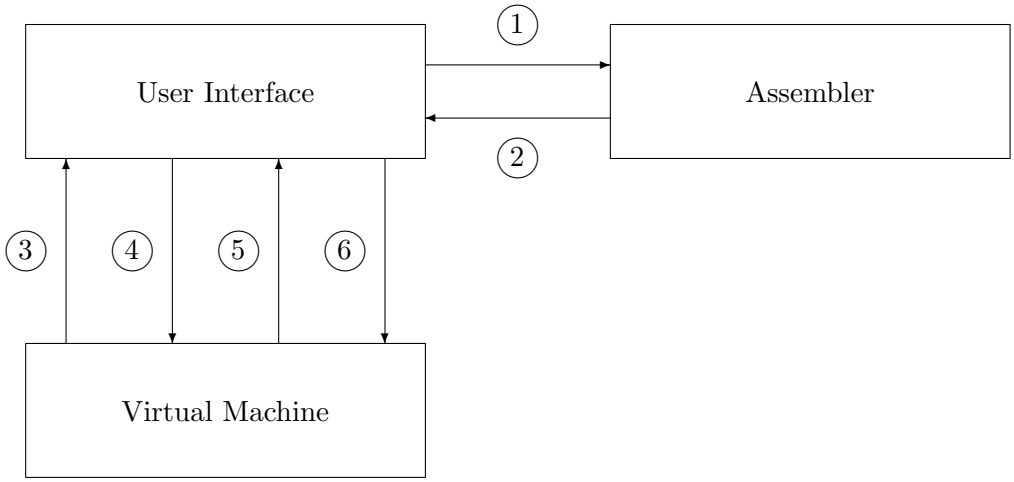
These utilities are all written using CWEB [4] which is to quote the authors:

> a software system that facilitates the creation of readable programs

The underlying programs that are generated by CWEB for this solution are written in C. The applications use some very sophisticated data structures but it is at a very low level. It uses, as is typical for C programs, a lot of shared variables. When you are using the application you need to a large amount of knowledge in the workings of the application to actually understand what is going on.

3 Outline Design

My MMIX simulator application will consists of three separate components. The task of converting the MMIX assembly language source text into the corresponding binary representation will be performed by a component I am calling the Assembler. The task of actually simulating an MMIX computer will be performed by a Virtual Machine component. The final component will be responsible for representing the current state of the MMIX computer to the user, along with orchestrating all of the interactions with the other components. I am calling this component the User Interface. The interactions between the components can be illustrated by figure 1.



1. Source Text
2. Binary Representation
3. Binary Representation
4. Current State
5. Process Next Step
6. Change of State

Figure 1: Component Interactions

One question that needs to be answered is what is the target platform for this application. The application will initially be written for Mac OS X but if there is time available at the end of the development process I would like to make sure this application will run on other platforms.

3.1 Assembler

The purpose of the assembler component is that it will take in MMIX assembly language source code contained in a source file and convert it into a binary representation of the application. The first thing that I need to decide upon, for each component, is what is the most appropriate programming paradigm. The production of an assembler is a fairly well defined process. There should be little, or preferably, no ambiguity in the translations. These requirements make me think that this would be an ideal candidate to be written in a functional programming language, which to quote Turner [6]

> Functional programming is so called because its fundamental operation is the application of functions to arguments. A main program itself is written as a function that receives the program's input as its argument and delivers the program's output as its result.

There are many functional languages available so determining which one to use requires some consideration. One way to classify functional languages is to determine how easily they allow users to create side effects. A procedure performs a "side effect" when a procedure does not just return a value it also amends the underlying state of the system. Functional languages where you have to explicitly declare that code can perform side effects is called "pure". I plan on using a "pure" functional programming language for the assembler.

3.2 Graphical User Interface

The main way that users will interact with the simulator is through a graphical user interface (GUI). The GUI will be responsible for all of the interactions between the other components. The GUI will initially be written to run as a stand alone application. If there is time at the end of the project I will investigate how to push the GUI up onto the internet or even a mobile version.

The only consideration that needs to be taken into account when deciding which programming paradigm and language to use when creating this component is that a GUI, almost by definition, has to handle numerous "side effects". This would make me lean towards an object oriented programming language, but there is a newer family of programming languages that allow the developer to use multiple programming paradigms. I plan on using one of these multi-paradigm programming language for the GUI.

3.2.1 Initial Design

The GUI will be broken up into four separate main sections as illustrated in figure 2. Each of these sections will contain details about one distinct arrear of the MMIX simulator.

3.2.2 Memory

The memory section of the GUI will contain a representation of the simulators current memory. The memory representation will show both an hexadecimal representation for

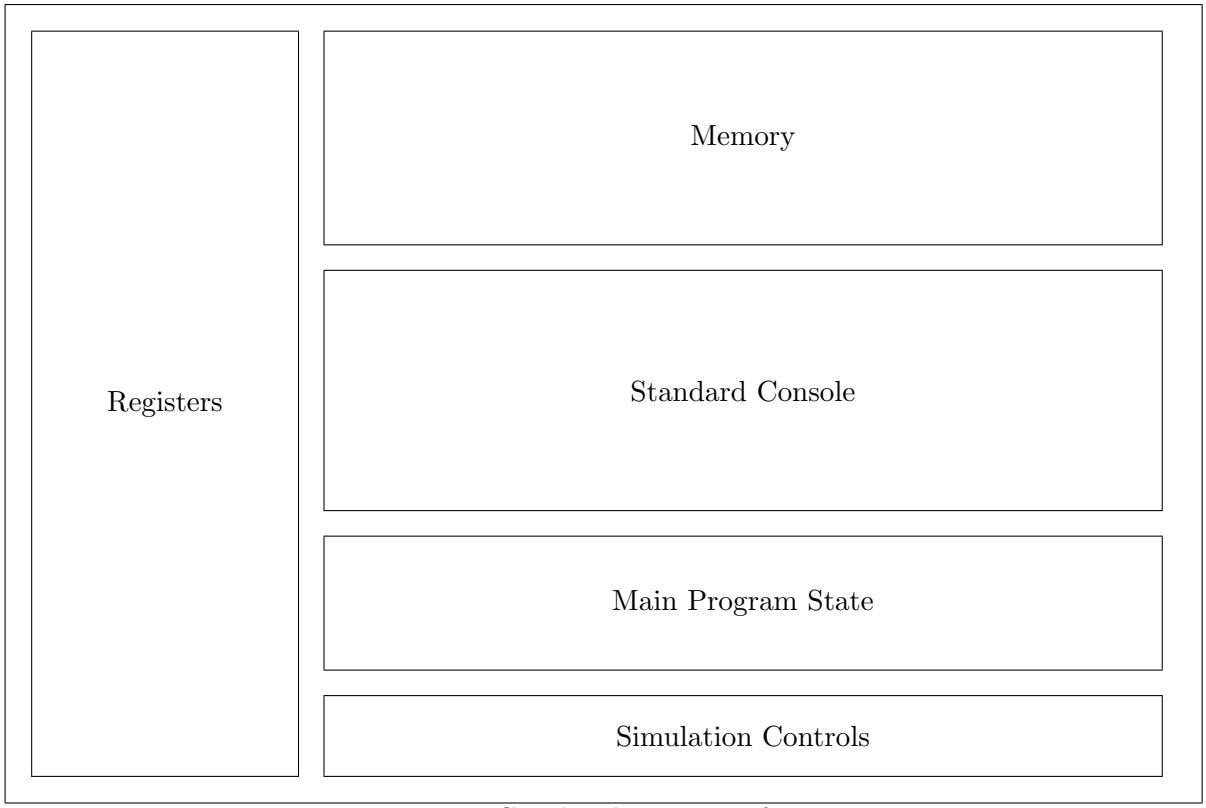


Figure 2: Graphical User Interface

the values in memory but also an ASCII representation of these values. An example of what I am planning to include can be found in figure 3. I am envisaging that when the contents of a specific memory address changes then that location will be highlighted, probably with a change of colour. After a period of time this highlighting will be removed. This period of time will probably be when the user executes the statement in the program.

00000000 :	$8fff$	0100	0000	0701	$f4ff$	0003	0000	0702	.....
00000010 :	0000	0000	$2c20$	$776f$	$726c$	$640a$	00		$...., world..$

Figure 3: Memory Representation

3.2.3 Registers

The MMIX architecture contains two types of registers, general purpose registers and special registers. There are 256 general purpose registers that the developer is free to use as they see fit. There are 32 special registers which either record the internal state of the MMIX machine or how the MMIX machine is configured. The registers section

of the GUI will display a list of all of the available registers in the simulated MMIX machine. It will show not only the names of the registers but there current values. I am envisaging that when the contents of a register changes then, like the memory section, it will be highlighted.

3.2.4 Main Program State

When every statement in the program is executed then the internal state of the MMIX machine will change. I want the user to clearly see what these changes are. One way I am intending of showing these changes is with the highlighting of memory addresses and registers that I have mentioned above. Another way I am intending to show these changes is with this section. This section will list all of the registers that have changed along with possible a sample of the changed memory addresses. I will investigate the appropriateness of this during the development of the application.

3.2.5 Standard Console

There are a number of embedded computers that do not interact directly with a user however it is quite rare for a general purpose computer to have no interactions. The MMIX simulator has got standard input and output channels. These will need to be accessed in the simulator. They will be accessed in the simulator through the standard console section. There will be an area that lists all of the output from the virtual machine along with another area that allows the user to send text to the virtual machine.

3.2.6 Simulation Controls

The users will need some way to interact with the MMIX machine. The simulation controls section will allow the user to control how the GUI communicates with the virtual machine. This will contain all of the interactions detailed in the virtual machine section (section 3.3).

3.3 Virtual Machine

The Virtual Machine component is the heart of the simulator. It is this component that actually simulator the MMIX machine. It will contain a representation of the MMIX machines memory. It will also contain all of the registers that the MMIX machine uses. I am envisaging that the application programming interface for the VM will be very small. An application programming interface is to quote Wikipedia [8]

> An application programming interface (API) is a protocol intended to be used as an interface by software components to communicate with each other.

The way that the virtual machine will change it's state is when it receives a command from the outside world. It will sit there waiting for the next instruction. The virtual machine will respond to these commands by performing the relevant actions and returning details of any changes to the virtual machine. This leads me to want to use

a functional language for this component and it also makes me wants to use a language that has got strong message oriented features.

3.3.1 Application Programming Interface

I am envisaging that the application programming interface for the virtual machine will only contain the following commands:

Reset Simulator This command will close down the existing virtual machine and await for a new program to be specified.

Load Program The GUI will pass the binary representation of a program to the virtual machine which will load this into the memory representation and it will reset the registers.

Reset Program This command will reset all of the registers in the virtual machine.

Process Next Statement This command will execute the current statement and move the state of the virtual machine on to the next statement.

4 Development Plan

The next thing that I need to decide upon is how I plan to actually develop the simulator application. The first detail that I need to decide upon is what methodology should I use for each of the components. As I plan to use different programming paradigms for each of the components it is highly likely that the best development methodology for one of the languages will not be the best methodology for the other languages.

Once I have decided upon the various development methodologies to use I plan on using a risk based approach to determine the order which areas of the application to develop.

Once I have developed the simulator application I need to test that this simulator is working correctly.

4.1 Development Methodology

There are a number of different approaches to developing a software systems. Each language has got a subset of these approaches which are commonly used when developing them. I plan to investigate what the common development approaches are for each of the languages that I use for the application. Once I have investigated the development approaches I will pick one to use developing the application. I am a big fan of test driven development(TDD) and I will, where appropriate, I will chose a methodology that either uses or is derived from it.

4.2 Risk Based Prioritisation

When you are developing an application there will be some parts of the application that you know how to develop and there will be other parts that you need to determine how to create. The parts of the application that you do not already know how to develop are obviously considered to have a higher risk associated with them. When you are developing an application you have to determine how critical each part of the system is. As an example if the purpose of your application is to calculate page ranks then the application must calculate page ranks. If it does not calculate these page rank then the application is a failure. How critical a piece of functionality in increase the risk of that piece of functionality. When you are prioritising what area you are developing next you should take these difficulties into account. You should develop the pieces of functionality in priority order, from highest priority to lowest.

4.3 Testing

The testing of the simulator application I will perform can be split into three phases. The first phase is unit testing where I will be testing that the individual components work as I expect them too. The second phase is integration testing where I will be testing that the interactions between the components work correctly. The third phase is acceptance testing where I will be testing that the simulator application as a whole. To do this I will be taking some of the application that have already been written [5] and run them through the simulator application.

4.4 Development Schedule

I need to review the work that is needed to produce the simulator and confirm the order in which it will be developed. The following list is a break down of the high level tasks required to create the simulator application.

1. Determine the exact mechanism that the various components will use when communicating. I need to be certain that it is physically possible for the languages I have chosen can communicate, if they cannot then I need to choose different languages.

2. Review the development methodologies currently in use for the final language used by the graphical user interface.

3. Create a skeleton application for the graphical user interface.

4. Review the development methodologies currently in use for the final language used by the virtual machine.

5. Create a skeleton application for the virtual machine and make sure that it communicates with the graphical user interface.

6. Determine the data structures required by the virtual machine.

7. Extend the virtual machine so that it can receive a program and execute one or two operations.

8. Extend the graphical user interface so that all of the interactions with the virtual machine are implemented.

9. Review the development methodologies currently in use for the final language used by the assembler.

10. Review the currently existing compiler tools and determine which one to use.

11. Create a initial version of the assembler that understands a small number of operations

12. Extend the graphical user interface so that it will communicate with the assembler.

13. Split the remaining operations into a small number of groups.

14. For each of these groups extend the assembler so that it understands them and also extend the virtual machine so that also understands them.

15. Make sure that all of the special registers have been developed on both the virtual machine and the graphical user interface.

16. When all of these groups and registers have been implemented then the simulator application should be finished we just need to complete all of the testing.

5 Summary

The application that I am proposing to write will simulate the artificial machine developed by Professor Knuth[2]. The application will be split up into three separate components. The first component is an assembler that I plan to develop using a "pure" functional programming language and my initial plans are to use Haskell. The second component is a graphical user interface that I plan to develop using a multi-paradigm programming language and my initial plans are to use Scala. The final component is a virtual machine that will perform the actual simulation that I plan to develop using a message oriented functional programming language. The language that I am planning on using to develop this component is Erlang.

References

[1] Knuth, D.E., 2004, The Art of Computer Programming Fascicle 1 MMIX [e-book] Stanford University: Addison Wesley Available through: Stanford University <http://www-cs-faculty.stanford.edu/~uno/fasc1.ps.gz>[Access 7 April 2013]

[2] Knuth, D.E., 2011, The Art of Computer Programming Volumes 1-4a. 1st ed. Addison Wesley

[3] Knuth, D.E. MMIX Home Page [online] Available at: <http://mmix.cs.hm.edu>[Accessed 1 March 2013]

[4] Knuth, D.E. and Levy S., 1993, The CWEB System of Structured Documentation. Reading, Massachusetts: Addison-Wesley

[5] MMixMaster. 2007. Welcome to the MMIXmasters Home Page [online] Available at: <http://mmixmasters.sourceforge.net/>[Accessed 7 April 2013]

[6] Turner, D., 1990. Research Topics in Functional Programming, Addison-Wesley. Available through <http://www.cs.utexas.edu/~shmat/courses/cs345/whyfp.pdf>[Accessed 2 April 2013]

[7] University of Joensuu, 2007. Jeliot 3. [online] Available at: <http://cs.joensuu.fi/jeliot/>[Accessed 5 April 2013]

[8] Wikipedia. Application Programming Interface [online] Available at: <http://en.wikipedia.org/wiki/Application_programming_interface>[Accessed 7 April 2013]