CBCJVM: Applications of the Java Virtual Machine with Robotics: Concept and Features

Braden McDorman *(beta/catron)*
Benjamin Woodruff *(pipeep)*
Akshay Joshi *(axyjo)*
Jonathan Frias *(freakinjonathan)*

Norman Advanced Botball `braden@betabot.org`
Fletcher High School `odetopi.e@gmail.com`
American Community School of Abu Dhabi `me@akshayjoshi.com`
Robert E. Lee High `freakinjonathan@gmail.com`

CBCJVM: Applications of the Java Virtual Machine with Robotics: Concept and Features

1 Introduction

The Botball tournament requires programmers to rapidly develop and prototype programs. However a disconnect is visible between the language and the tools provided and the goals of the game. Although very fast, C is generally considered to be a low-level language, making it what many of us believe to be the wrong tool for the job. The result is buggy programs, cluttered code, and a lack of flexibility in user libraries. The CBCJVM is the product of the realization that although slower, interpreted Object-Oriented programming languages are better for the type of prototyping used in the Botball competition. As KIPR has stated many times, it is highly unlikely that the CBC will ever be officially changed from C [5]. CBCJVM is not just a port of JVM to the CBC, but it encompasses a group of libraries and tools to compete with (and often times surpass) those offered by KISS-C. In this paper we will look at the internals of CBCJVM, as well as what it can provide for your team.

2 Concept and Background

2.1 Languages

The CBCJVM originally went under the name "CBCJava", but as the project grew, it was soon realized that using the JVM only for Java was a waste. The Java Virtual Machine has become a de facto standard of virtual machines, and almost every interpreted language imaginable has some port to the JVM, and these languages typically have support for interfacing with Java libraries. The JVM is fast, taking advantage of Just-In-Time compilation and various other optimizations [4], making its performance somewhat comparable to C++

and C (but performance is still in no way better than a native language). CBCJVM has been used with JavaScript (via Mozilla Rhino) and Scala (which compiles directly to Java byte-code), but it can be used with many other languages such as Ruby, Python, Lua, and even LOLCODE. While it cannot always match the performance of native interpreters for these languages, such as WebKit (JavaScript), CPython, or the standard Lua interpreter, it offers a fast way for these teams to get up and running with these high level languages. What's more, is that CBCJVM already includes a set of powerful libraries that can be used in conjunction with these languages to help teams that would otherwise be on their own with a language.

Due to the limited scope of this Document, and the large differences in how various JVM languages manage their bindings to Java code, we will not discuss the specifics on running these languages. Please see the Ending Notes for information on finding help for doing this.

2.2 Libraries

Rather than being a straight port of the JVM to the CBC, CBCJVM includes libraries developed on top of the JVM. These libraries allow access to the KISS-C libraries via the Java Native Interface (JNI), allowing you to control motors, read sensors, and so on. Although bugs occasionally show up in this low-level communication layer, we believe that our implementation is complete and stable enough to be used on a day-to-day basis. These low level functions are available in the `cbccore.low` package.

On top of this, we have classes that make the low level functions more object-oriented, and use camelCase for function names, rather than KISS-C's underscore_naming_system. The majority of CBCJVM's classes use these camelCased classes, and so should you. That said, All simulation happens in this `cbccore.low` package, and thus there is not loss in functionality by using the `cbccore.low` package. There exist two obvious reasons for the use of the `cbccore.low` package in your own code:

- Ease of porting from KISS-C code.

  Although the best option would be to port your code to CBCJVM utilizing the new APIs, in an object-oriented fashion, the cbccore.low package can be used to do a quick ad-hoc port of code, due to it's similarities to the KISS-C's API.

- The use of some feature where no high-level, camelCased class exists to handle it yet.

  We try our best, but as a (currently) small development team, there is the possibility that we have overlooked some functions in our porting efforts. However, the best option would be to fork[3] our github repository [1] *, add that functionality yourself, and submit a pull request[6], or report it in our github issue tracker* [2] *and wait for a fix.*

  More information regarding these libraries can be found in the *Features and Internals* section.

[1] `http://github.com/catron/CBCJVM`
[2] `http://github.com/catron/CBCJVM/issues`

3 Features and Internals

3.1 Running the JVM

CBCJVM uses a lightweight JVM called JamVM[7]. JamVM is designed for embedded platforms, such as the Linux running Chumby used in the CBC. JamVM gives us excellent compatibility with Sun's Java standard library because it uses GNU Classpath[1] for it's standard library. This allows us to run generated byte code on the computer and CBC with no modification. Our simulator exploits this cross compatibility by replacing the `cbccore.low.*` libraries at runtime.

3.2 Simulator

As mentioned, our simulator replaces classes loaded by `cbccore.Device` at runtime if it detects it is running on a computer. These simulated Device classes then update a Simulator GUI singleton. This is done automatically and is transparent to the user. The simulator is still in an early state, but its architecture allows room for fast and easy expansion.

Figure 1: *The CBCJVM simulator, demonstrating motor simulation. The simulator also has support for framebuffer simulation (not shown).*

3.3 Frame Buffer Access

One of the more unique features of CBCJVM is the ability to draw to the CBC's 320 × 240 screen. This is accomplished by writing RGB565 data to the */dev/fb0* pipe provided by Linux. A custom-made library built specifically for CBCJVM gives you helper classes such as pixmaps (Pixel Buffers) with simple drawing functions and Image data types. The frame buffer is also simulated on the computer, allowing you to test your drawing code

directly in the simulator. You can find more details about CBCJVM's frame buffer library in `cbccore.display` package.

3.4 Movement Library

Never again will you have to write your own movement library CBCJVM includes an extensible Object-Oriented library. Its plug-in based architecture allows easy adoption for any piece of hardware; motor based, or create based; a simulator plug-in is even in development, meaning that you will be able to have a KISS-C style simulation system that can be used with any JVM language. Plug-ins are essentially stubs, telling the DriveTrain class (the center of the movement library) what functions to call for a specific piece of hardware. By making all your movements through this library, you can take advantage of features such as position tracking. The movement library can be found in `cbccore.movement`.

3.4.1 Example Code

```
import cbccore.movement.DriveTrain;
import cbccore.movement.plugins.create.CreateMovementPlugin;
/* There is also a MotorMovementPlugin located at:
 * cbccore.movement.plugins.motor.MotorMovementPlugin
 * for traditional Lego-chassis robots. You can read more
 * about this in the later mentioned JavaDoc files.
 */

public class MovementLibraryExample {
    public static void main(String[] args) {
        //create a new DriveTrain instance:
        DriveTrain dt = new DriveTrain(
            new CreateMovementPlugin(
                    //Wrap a plugin in the DriveTrain class
                1.0 //Efficiency, a 'fudge' factor
                , true //"Full" mode, as defined by KISS-C
            )
        );
        /* You are free to extend the DriveTrain class as you wish,
         * using any "plug-in" you wish with the extended version.
         */

        //move one meter forward, half speed:
        dt.moveCm(100.0, 0.5*dt.getMaxCmps());

        //rotate 90 degrees, counter-clock-wise, half speed:
        dt.rotateDegrees(90.0, 0.5*dt.getMaxDegreesPerSec());
```

```
        System.out.println(dt.getPosition().getAngleDegrees());
        //will print out 90.0
    }
}
```

3.5 Configuration System

"Configurators" can get input from a user for several options using buttons, touch sensors, or even analog sensors using an `cbcccore.sensors.analog.AnalogBooleanAdapter`. This is accomplished by passing an array of options along with an array of acceptable `IBooleanSensor`s to use, then mapping each option to a unique sensor. Using Configurators allows a programmer to quickly build menu systems based off of the inputs their robot can handle.

3.6 CBCDownloader

CBCDownloader is a Java based downloader that functions over USB and network connections. Beyond downloading files, CBCDownloader can also send commands to the CBC over USB and network connections. This allowed us to implement a simple console similar to a ssh console.

3.6.1 Eclipse Integration

Ease of use being of the utmost importance to our team, CBCJVM's Eclipse[2] Plug-in allows you to download to your CBC via a network or via a flash drive. Eclipse's built in documentation, code completion, and automatic build system allow for you to develop code even faster. Eclipse also provides thousands of plug-ins for things such as version control with git and svn. When downloading over the network there is no need to recompile your program, just simply press run again.

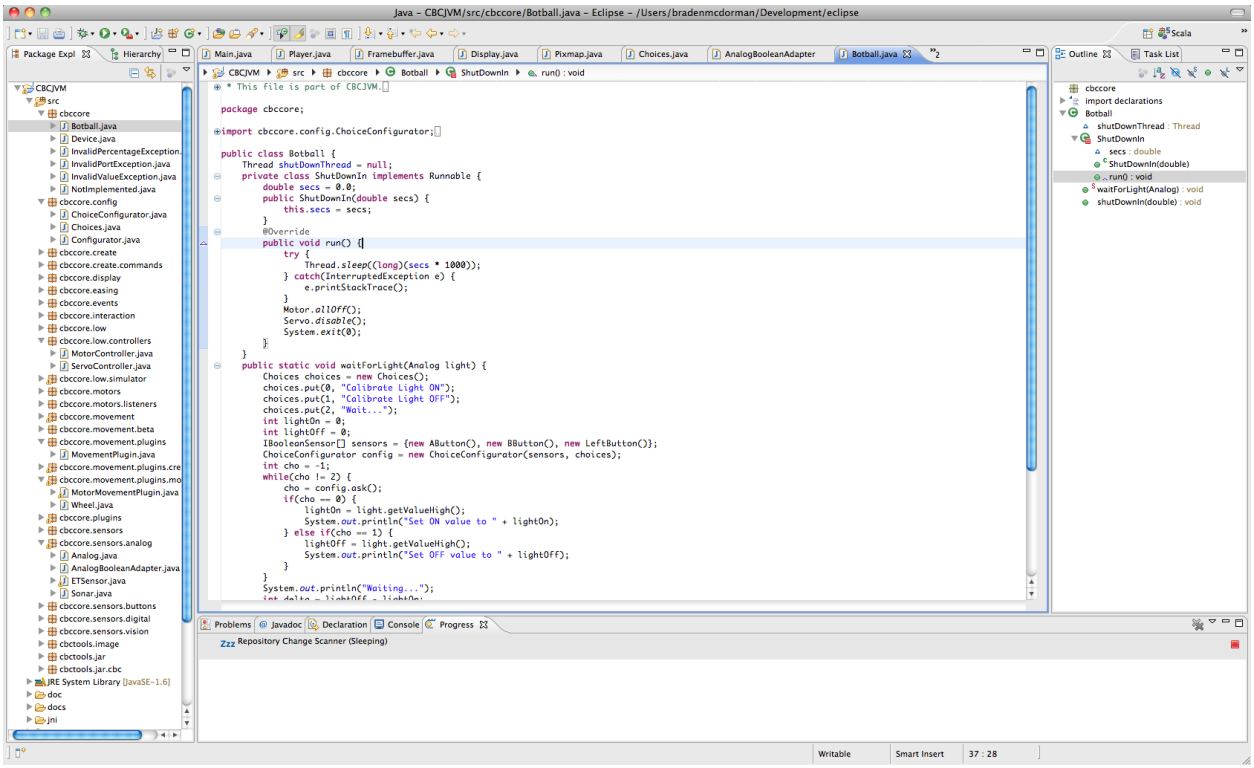

Figure 2: *CBCJVM being edited in the Eclipse IDE*

3.6.2 Network Downloads

Network downloads are accomplished over an ssh connection using scp to transfer files. If you plan to use the network downloader, it is highly recommended you install the Norman/Nease mods [3] on your CBC for wireless downloads, a new version of which will be released at GCER. More information on finding the CBC's IP address and networking with the CBC can be found in the 2009 Nease-Norman CBC hacking paper[4].

3.6.3 USB Downloads

USB Downloading was probably the single most desired feature in CBCJava, and we are pleased to be able to offer it this year in CBCJVM. Due to limitations on the current cbcui serial server, we had to rewrite it to be more accepting of non-C tasks. Our downloader requires a custom cbcui binary you can download and install from Braden McDorman's cbc fork[5]. There are serveral benefits of our server, including simple console support (like ssh), cbcui state manipulation, and downloading of files to any writeable location on the CBC's

[3] `http://community.botball.org/forum/technical/programming/normannease-cbc-mod-installer-v12-supports-`
[4] `http://bit.ly/nn-cbc09`
[5] `http://www.github.com/catron/cbc`

file system. Our serial server can be turned on and off from the Settings page on your CBC, to allow support for KISS-C downloads.

3.7 License

CBCJVM is licensed under the GPLv3 [8]. This license requires that you distribute the source code of any modifications of CBCJVM along with any binary version of it that you distribute. We, the authors of CBCJVM, believe this constitutes a fair and just agreement. You are also permitted to use CBCJVM under any future version of the GPL that you wish.

4 Conclusion

This concludes part one of the CBCJVM paper. We hope to see you the next paper, titled "CBCJVM: Applications of the Java Virtual Machine with Robotics: Setup, Help, and Tomorrow."

References

[1] GNU Classpath — GNU Project — Free Software Foundation (FSF). `http://www.gnu.org/software/classpath`, 07 2009.

[2] Eclipse.org home. `http://www.eclipse.org`, 2010.

[3] Help.GitHub — Forking a project. `http://help.github.com/forking`, 2010.

[4] Performance Features and Tools — Oracle. `http://java.sun.com/developer/onlineTraining/Programming/JDCBook/perf2.html`, 2010.

[5] Logan Cox. Can there be a KISS-C++ rather than KISS-C? | Botball Community. `http://community.botball.org/forum/miscellaneous/suggestions-and-bugs/can-there-be-kiss-c-rather-kiss-c#comment-422`, 2010.

[6] Tom Preston-Werner et al. Help.GitHub — Forking a project. `http://github.com/guides/pull-requests`, 2010.

[7] Robert Lougher. JamVM — A compact Java Virtual Machine. `http://jamvm.sourceforge.net`, 01 2010.

[8] Richard Stallman et al. The GNU General Public License — GNU Project — Free Software Foundation (FSF). `http://www.gnu.org/licenses/gpl-3.0.html`, 06 2007.