Electronics and Computer Science
Faculty of Physical and Applied Sciences
University of Southampton

Adam Noakes
Tuesday 8th December 2015

An Open Platform for the Simulation of Java Card Applets

Project supervisor: Ed Zaluska

A project progress report submitted for the award of

BSc Computer Science

Abstract

Smartcards provide a practical element for security-by-isolation design, because small sections of critical code execute inside a protected tamper-resistant hardware/software environment and therefore are shielded from many forms of attack. Software development for smartcards is normally a specialized and costly activity, partly because of the severe resource constraints imposed by the card's internal execution environment, but also due to the somewhat restrictive closed-source and closed-information models imposed by the chip and operating system manufacturers.

The aim of this project is to produce an open platform for simulation of Java Card applications using a reduced-functionality Java Card virtual machine based on the MSc project by Robin Williams. This report details the progress made in refactoring the C# and JavaScript based simulator to run in a Node.JS environment.

Contents

1 Project Goals

Software development for smartcards is a specialized and costly activity partly because of restrictive, close-source models imposed by the chip and operating system manufacturers. Currently, there are only a handful of tools available to test smart card applications, with the official Java Card Platform Simulator (cref) being restricted to Microsoft Windows-based operating systems [1]. The main goal of this project is to produce an open platform that allows the users to simulate the execution of commands sent to smart cards, using the Java Card platform, in order to aid the understanding of Java Card technologies.

The end goal of this project is to provide a tool that will allow easier access to smart card development, especially in educational environments, where the current tools are sparse and difficult for novice programmers to use. Therefore, when designing the simulator, care will be taken to ensure that the application is both easy to use and easy to maintain.

This project is based on the MSc Dissertation by Robin Williams [2] and aims to refactor the C# and JavaScript based Java Card simulator, produced as part of the dissertation, to run in a server based in environment. The server environment will be written in JavaScript and will utilise the Node.js server side framework, a framework that has gained considerable traction since it's release in 2009. This project also aims to extend the simulator's capability in order to support the Java Card packages that are required for applets to perform cryptographic functions (javax.crypto and javax.security), thus enabling the simulator to be used to test applets that have been designed for security purposes.

2 Background and report of literature search

2.1 Smart Cards

Smarts cards are a device, usually made from plastic, which contain an Integrated Circuit (IC) and are consequently referred to as Integrated Circuit Cards (ICCs) by ISO. Smart cards can be produced in two forms, contact and contactless. Contact-based cards typically use 6-8 contacts to connect with a reader [3], providing the ability to transfer APDU commands and power the Integrated Circuit. Contactless cards differ in that they communicate with and are powered by the reader, using an inductive loop, as apposed to a physical contact connection.

Smart cards can be used for many different applications; however, they are usually used to provide a form of security for the user or for the organisation that issues the smart cards [4]. An example of smart cards being used for security purposes would be use of smart cards for Credit/ Debit cards, commonly found in Europe. The smart cards typically rely on a PIN to be inputted from the user when connected to the reader, in order to complete payment transactions. Another example would be the use of a smart card (Universal Integrated Circuit Card) used to provide security for the users on UMTS and GSM networks, typically in a mobile phone.

2.2 ISO/IEC 7816-4 APDU Communication

Connection between the ICC and the reader is arranged in the form of an APDU (Application Protocol Data Unit). The APDU is defined by ISO 7816-4 [5] for contact cards and ISO 14443-4[6] (based on ISO 7816-4) for contactless cards. An APDU can be of two forms; a command APDU (Figure 1), sent by the reader to the ICC and a response APDU (Figure 2), sent by the IC [5] – with the two APDU commands forming the Command-response pair (C-RP).

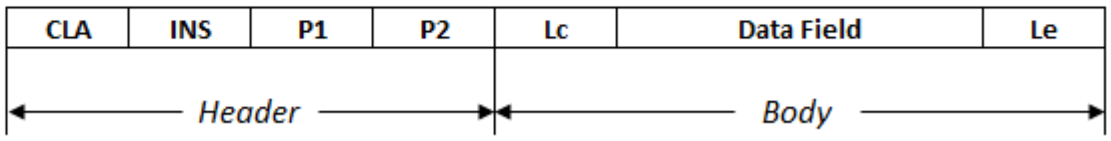


Figure 1: Command APDU

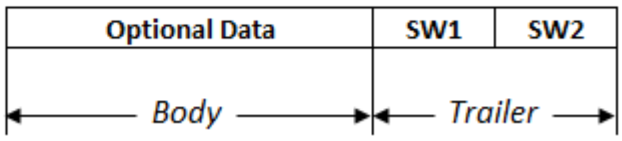


Figure 2: Response APDU

When an APDU command is received by a smart card, the IC will attempt to execute the command in its current state [5]. If command execution is successful; the smart card will send the response APDU 0x9000, proceeded by an output data from the execution (e.g. the sum of a calculation). If the execution is unsuccessful; the IC will send back an error APDU. An example of a possible error APDU returned by a smart card would be the APDU 0x6A82, which is sent for a File Not Found error, where SW1 = 6A and SW2 = 82.

2.3 Java Card

Java Card technology is developed by Sun Microsystems (now a subsidy of Oracle) and enables applications written in a subset of the Java programming language [7], to be run in a secure environment on devices such as smart cards. Java Card technology has been designed with the aim of being highly portable and secure [8], leading Java Card to become the most commonly used platform for developing smart card applications [9].

Java Card applications, referred to as applets, are written using a subset of Java programming language in combination with the Java Card SDK and must extend the javacard.framework.Applet class. When applets have been written in the Java programming language they are converted to a Converted Applet (CAP) file. The CAP file can then be installed onto a smart card by sending the data through a series of APDU commands (an APDU script) [10]. When the CAP file has been downloaded onto the smart card, an APDU command can then be sent to the smart card to create an instance of the applet previously downloaded. Once an instance of the applet has been successfully created, the install process is complete and smart card readers can select the applet and send commands for execution by the smart card.

2.4 Integrated Circuit Chip

The integrated circuit contained in smart cards varies depending on the manufacture and the purpose of the card. The IC in a smart card differs from other cards, such as chip cards, in that they are made up of both a microcontroller (or equivalent micro processor) and some form of embedded memory [11], allowing for more advanced applications. Specifications of the IC vary depending on the intended purpose and the Java Card specification [7] includes a minimum requirement of the system of the hardware (Table 1) that cards must meet in order to be deemed compatible with the platform.

Hardware	Requirement
ROM	16 KB
EEPROM	8 KB
RAM	256 bytes
CPU	Required

Table 1: Java Card Minimum Hardware Requirements

2.5 Node.js

Node.js is a framework for developing fast and scalable server side web application, written in the JavaScript programming language [12]. In computing, the speed of I/O operations can be vastly slower than the speed of processing the data. In such case, when a thread is executing a function and it is required to perform an I/O operation, the thread cannot continue until the I/O operation is complete, resulting in wasted processor cycles. Modern operating systems combat this by switching threads, returning to the blocked thread when the I/O operation has complete (context-switching). However, Node.js runs on a single thread [13], making context-switching impossible.

Node.js uses an event-driven, non-blocking I/O model [13]. The non-blocking, I/O model that Node.js employs, allows the platform to support tens of thousands of concurrent connections and Node.js has been proven to outperform Apache/ Nginx PHP stacks in terms of computational performance [14]. This is partly due to the fact that Node.js does not need to perform thread context switching allowing it be more memory and processor efficient [14].

In order to take advantage of non-blocking model that Node.js uses, applications must be written in a manner that does not the block the thread, referred to as asynchronous I/O. Asynchronous code is code that has been written in such a way, that, what happens next in a function, can be put on to a queue for executing later, when, what is required for that function to execute (e.g. a file) is ready. The thread that executes each function on the queue is called an event-loop.

Synchronous (i.e. not asynchronous) code would block the event loop, an example would be the following code:

```
myFile = fs.readFileSync(largeFile);
console.log(myFile);
```

This would stop the event loop from being able to deal with other concurrent operations (e.g. another user accessing the server), for however long it takes the server to read “largeFile”.

The asynchronous version of this code would be:

```
fs.readFile(largeFile, function(myFile){
      console.log(myFile);
});
```

In this code, the second parameter of the readFile function, is what should be execute after the I/O operation has completed, referred to as the call-back. The event loop event loop will execute the call-back when the I/O operation has finished, and in the meantime, it will execute other functions in the queue.

3 Report on Technical Progress

The primary goal that I aimed to achieve this semester was to have a working prototype of a Java Card simulation that is written entirely in JavaScript and compatible with the Node.js framework. In order to achieve this goal, multiple steps were taken to produce a working prototype from the original simulator code, this section details the main changes to the application structure and challenges of this process.

3.1 Converting C# and client side JavaScript to Node.JS

The original simulator that this project extends, was produced as a web application using an ASP.NET server and client side JavaScript application. The server, written in C#, was used to serve and parse files to the client throughout the execution of APDU commands. The client side JavaScript, was used to perform the actual simulation of the commands and contained the JCVM (Java Card Virtual Machine) code.

3.1.1 Modelling the Smart Card IC

A major problem that was identified whilst converting to application to Node.JS was that; the structure of the simulator's code demonstrated many examples of bad programming practises. One of the problems identified, which made the simulator difficult to understand, was the fact that the virtual smart card computer was not correctly modelled using object oriented programming techniques. In addition to this, the class structure of the modelled smart card computer relied heavily on using global variables in JavaScript, which were accessed and modified throughout multiple files - many of the global variables were also poorly placed.

Consequently, during the development this project, in order to allow the code of the simulator to be more understandable and maintainable by future developers, the virtual smart card in the software has been modelled (Figure 3) to closely match the physical hardware design of the IC (detailed in 2.4). As part of this process, a large amount of time was also used to refactor the dependent functions and classes in the application, so that they are compatible with the new structure.

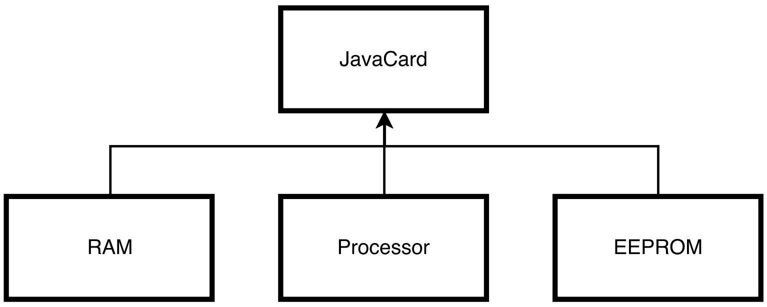


Figure 3: New Software Represnetation of a Virtual Smat Card

3.1.2 Consolidation of Smart Card Information

A design issue of the original simulator, that was identified during the process of modelling the smart card IC, was that a large amount of information was being accessed and written to the files, which were being transferred between the server and the client. This information varied from; virtual components of the smart cards IC, to; lookup tables, which contained information about what was stored on the card, such as the installed applets and their memory location. This design resulted in the structure of the virtual smart card being separated between the server and the client applications.

One example of a component, representing part of the smart card, that was separated from the main JavaScript application was the EEPROM. The EEPROM of the virtual smart cards were stored as a text file on the server, with the client side code using a wrapper EEPROM class, to transfer and parse information during execution of APDU commands. This design made the more simulator difficult to understand and debug, as well as resulting in unnecessary computation and limitations.

In addition to this, storing variables in text files is considered bad practise in software engineering and spreading the structure of the IC across the client and server applications makes the application difficult to understand - providing no real advantages. Therefore, the decision was made to refactor the Java Card simulator, so that all the information relating to the virtual smart card is stored as a JavaScript object, which can then be stored persistently in a database as a JSON.

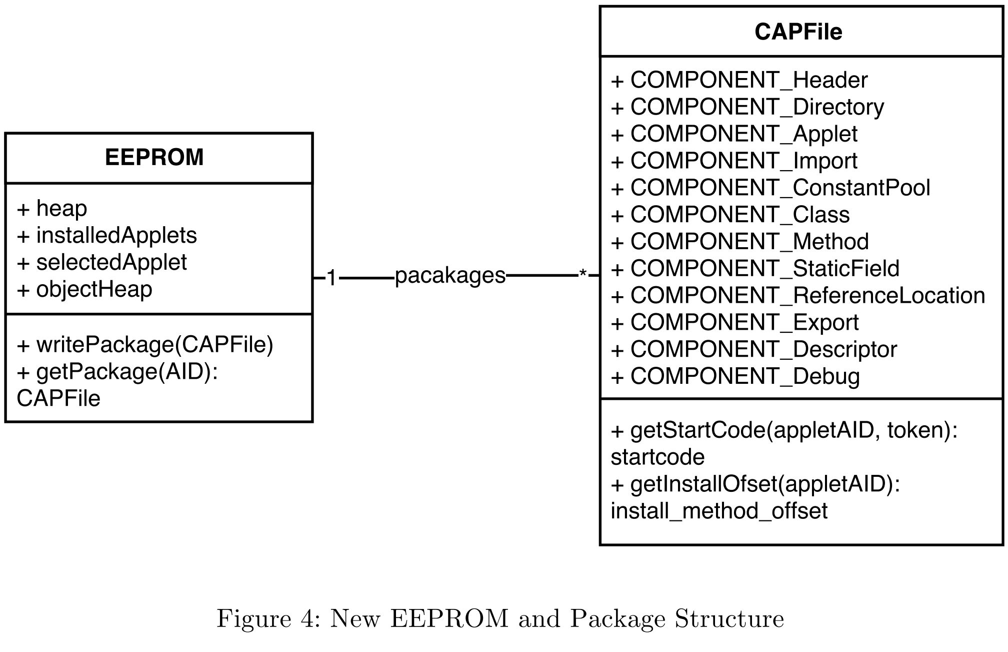


Figure 4: New EEPROM and Package Structure

The new design, and structure of the information has made the application much easier to debug and also allows the Smart Cards to be easily transferred for use in Server Side execution and client side execution, which could be added as an extension to the project in the future.

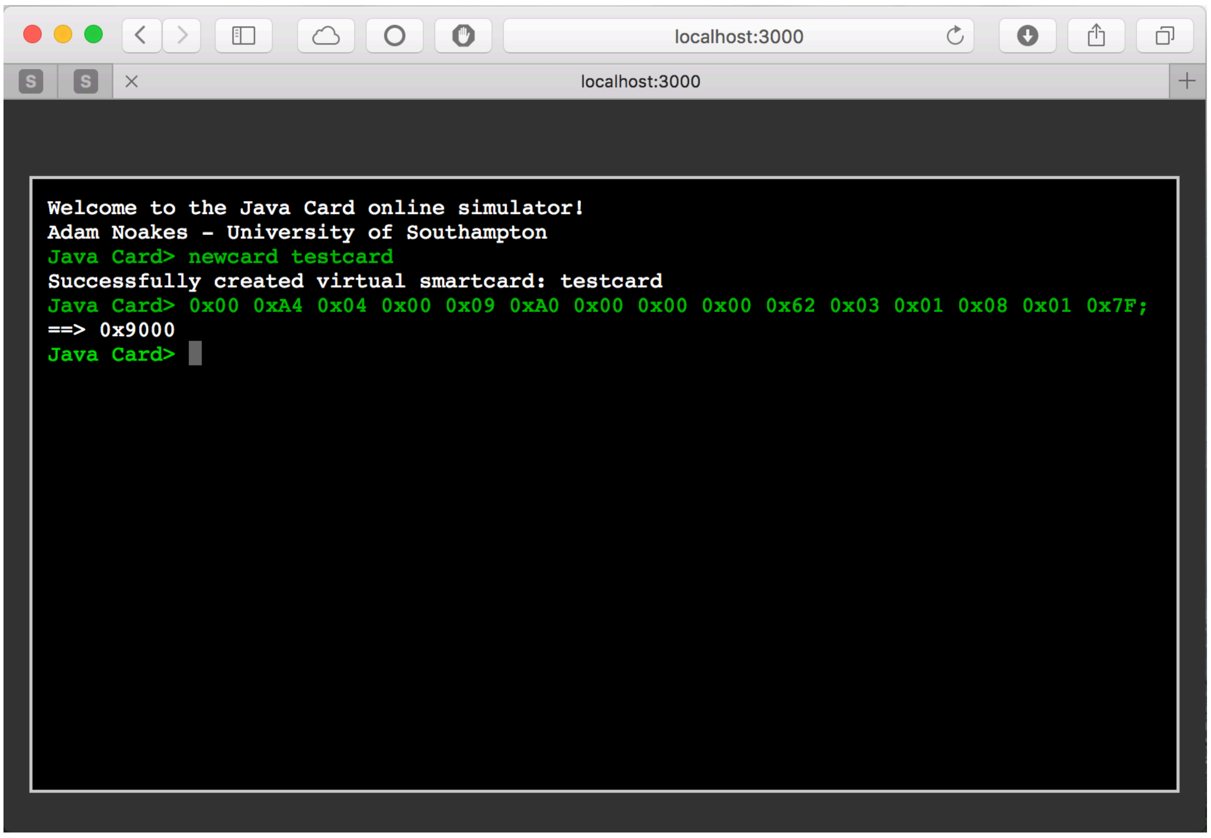
3.2 Implementation of a suitable user interface

In order to provide a suitable environment for using the Java Card simulator in an educational environment, the user interface of the original simulator will be redesigned.

The functional requirements for the new user interface are:

1. Must be suitable for use in an asynchronous application.
2. Must allow the user to send single and multiple APDU commands at once.
3. Must allow the user to view a history of APDU commands.
4. Must allow the user to view the responses of the APDU commands.

During this semester, a prototype interface for sending APDU commands to the virtual Java Cards has been implemented into the application. The interface is based on the open-source project jq-console [15], which provides a web-based console designed for running JavaScript code. The jq-console has been modified for this project so that it can be used to send APDU commands to the Node.JS backend and print the response APDU command after execution (Figure 5).



```
Welcome to the Java Card online simulator!
Adam Noakes - University of Southampton
Java Card> newcard testcard
Successfully created virtual smartcard: testcard
Java Card> 0x00 0xA4 0x04 0x00 0x09 0xA0 0x00 0x00 0x00 0x62 0x03 0x01 0x08 0x01 0x7F;
==> 0x9000
Java Card>
```

Figure 5: New UI Showing a Command and Response APDU

The console was chosen for use in this project as it provides the ability to send single APDU commands as well as multiple lines of APDU commands that have been copied and pasted from an APDU script. In addition to this, as the user interface represents a computer terminal, the user has the ability to scroll through the history of APDU commands and view their responses, something that was not possible in the previous user interface.

In order to maintain the card management functionality of the previous user interface, the virtual smart card management functionality of the original simulator, has been implemented into the console as commands. Commands currently supported by the console for card management are detailed in Table 2.

Command	Function
cards	Prints out a list of virtual smart cards available to the user.
newcard <name>	Creates a new virtual smart card for the user with the <name> specified.
loadcard <name>	Loads a virtual smart card for the user with the specified <name>.

Table 2: Card Management Commands

4 Plan of remaining work

4.1 Cryptography Support

As detailed in the research 2.1, the main application of smart cards involves using the smart card as some form of security. For this reason, the capability to perform cryptographic functions is an essential requirement for smart cards and is commonly implemented into applets, which are designed to provide a form of security for the user (such as when using smart cards as a debit or identity cards - detailed in 2.1).

The javacardx.crypto and javacard.security packages are two packages which must be supported by the simulator, in order to allow applets that are designed for security purposes, to be simulated. Therefore including support for these two packages, and consequently support for applets with security in mind, is seen as a priority for the final application.

4.2 Asynchronous I/O

In computing, the speed of I/O operations can be vastly slower than the speed of processing the data. In such case, when a thread is executing a function and it is required to perform an I/O operation, the thread cannot continue until the I/O operation is complete, resulting in wasted processor cycles. Modern operating systems combat this by switching threads, returning to the blocked thread when the I/O operation has complete.

As detailed in 2.5 Node.js runs on a single thread, meaning that the operating system cannot prioritise another thread for the duration of the I/O operation. If the Node.js runtime thread becomes blocked, the processor will be left idling whilst it waits for the I/O operation to complete.

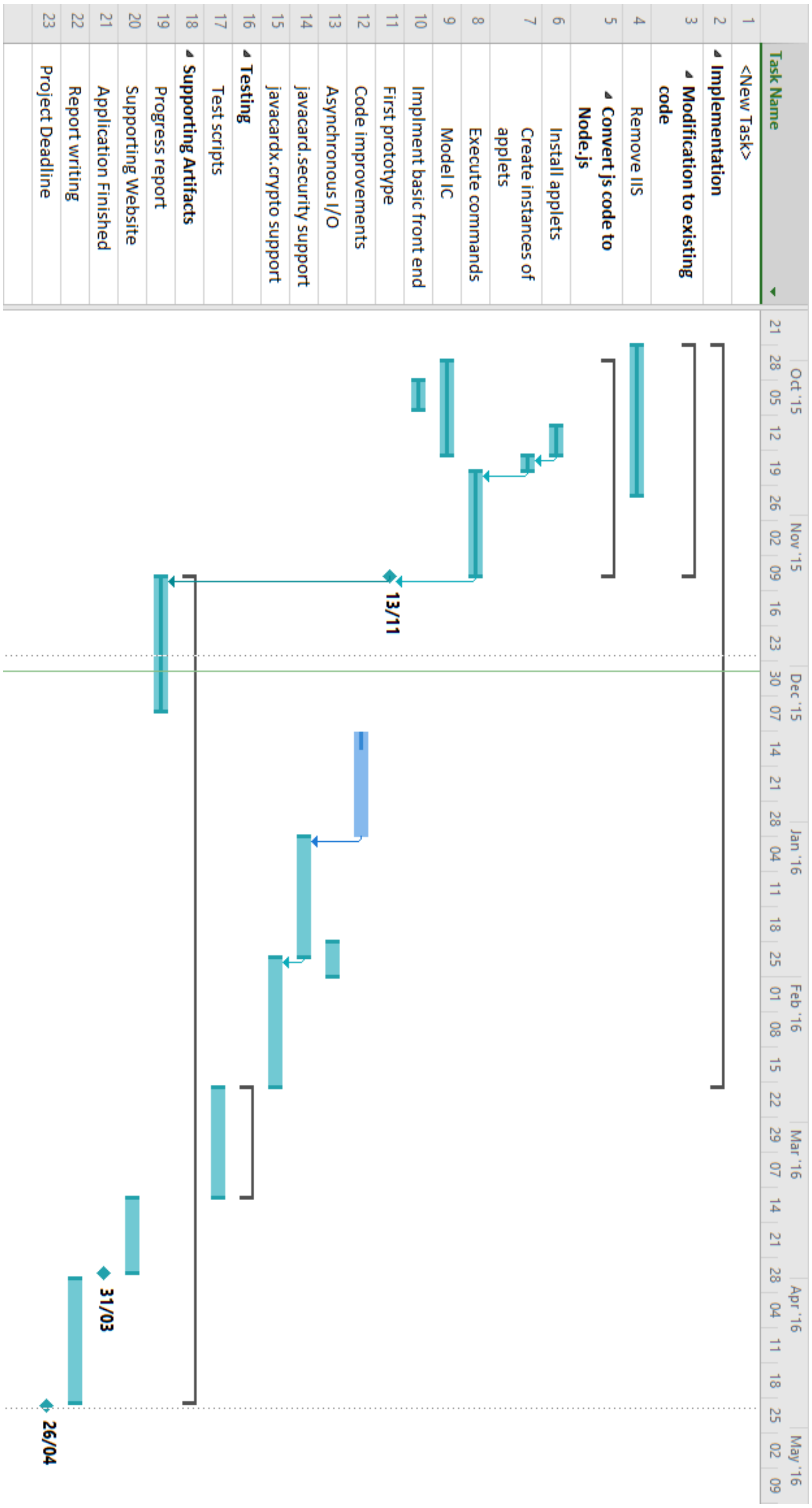
I/O operations that block the runtime thread, would negatively impact the applications ability to scale and therefore this must be taken into consideration whilst developing the simulator for use in a Node.js environment. Currently, the simulator has not been designed with Node.js in mind and therefore functions that invoke I/O operations must be refactored to an asynchronous form.

4.3 Automated Test Scripts

Automated test scripts provide a method of continually testing an application throughout it's development. In this project, test scripts could be used to create virtual smart cards in the simulator and send a range of different APDU commands. The responses from each test script could be automatically compared to the recorded responses of manually performing the test script with a physical smart card.

The goal for this project to be used as an educational tool somewhat mitigates the severity of incorrect simulation of Java Card applets and the execution of APDU commands. However, accuracy of the final simulator is a primary concern and determining factor to the success of this project and the use of test scripts will allow easy testing of the application's integrity. In addition to this, the development of test scripts as part of the project will allow the application to be easily tested by future developers that decide to extend and maintain the project. Therefore, in the second semester a sufficient period of time will be delegated for developing test scripts.

4.4 Project Gantt Chart



	Task Name
1	<New Task>
2	◢ **Implementation**
3	◢ **Modification to existing code**
4	Remove IIS
5	◢ **Convert js code to Node.js**
6	Install applets
7	Create instances of applets
8	Execute commands
9	Model IC
10	Implment basic front end
11	First prototype
12	Code improvements
13	Asynchronous I/O
14	javacard.security support
15	javacardx.crypto support
16	◢ **Testing**
17	Test scripts
18	◢ **Supporting Artifacts**
19	Progress report
20	Supporting Website
21	Application Finished
22	Report writing
23	Project Deadline

References

[1] Oracle Corporation. *Java Platform Micro Edition Software Development Kit.* 2009. [Online] Available: Oracle Corporation, https://docs.oracle.com/javame/dev-tools/jme-sdk-3.0-win/helpset.pdf [Accessed December 1, 2015]

[2] R. Williams, ‘Simulation of the Java Card Platform’, MSc, University of Southampton, 2014.

[3] X. Leng, 'Smart card applications and security', *Information Security Technical Report*, vol. 14, no. 2, pp. 36-45, 2009. [Online] Available: Science Direct, http://www.sciencedirect.com/science/article/pii/S1363412709000223. [Accessed November 30, 2015].

[4] K. Mayes and K. Markantonakis, *Smart cards, tokens, security and applications.* New York: Springer, 2008.

[5] BS ISO/IEC 7816-4:2013 Identification cards. Integrated circuit cards. Organization, security and commands for interchange. BSI Standards Limited, 2013. [Online] Available: BSI Standards, https://bsol.bsigroup.com/Bibliographic/BibliographicInfoData/00000000003031 2035. [Accessed November 30, 2015].

[6] BS ISO/IEC 14443-4:2008 Identification cards. Contactless integrated circuit cards. Proximity cards. Transmission protocol. BSI Standards Limited, 2008. [Online] Available: BSI Standards, https://bsol.bsigroup.com/Bibliographic/BibliographicInfoData/00000000003027 6455. [Accessed November 30, 2015].

[7] Oracle Corporation. *Java Card Classic Platform Specification 3.0.4.* 2011. [Online] Available: Oracle Corporation, http://www.oracle.com/technetwork/java/javacard/specs-jsp-136430.html [Accessed December 1, 2015]

[8] A. Patel, K. Kalajdzic, L. Golafshan and M. Taghavi, 'Design and Implementation of a Zero-Knowledge Authentication Framework for Java Card', *International Journal of Information Security and Privacy*, vol. 5, no. 3, pp. 1-18, 2011. [Online] Available: IGI Global, http://www.igi-global.com/article/international-journal-information-security-privacy/58979 [Accessed November 30, 2015].

[9] D. Vermoen, M. Witteman and G. Gaydadjiev, 'Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems.', in *Reverse Engineering Java Card Applets Using Power Analysis*, The Netherlands, 2007, p.

138. [Online] Available: Springer, http://link.springer.com/book/10.1007%2F978-3-540-72354-7. [Accessed November 30, 2015].

[10] Z. Chen, *Java Card technology for Smart Cards: Architecture and Programmer's Guide*. Boston: Addison-Wesley, 2004.

[11] D. Husemann, 'The smart card: don't leave home without it', *IEEE Concurrency*, vol. 7, no. 2, pp. 24-27, 1999. [Online] Available: IEEE Xplore, http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=766959/. [Accessed November 30, 2015].

[12] P. Teixeira, *Instant Node.js starter*. Birmingham: Packt Pub., 2013.

[13] M. Cantelon, M. Harter, T. Holowaychuk and N. Rajlich, *Node.js in action*. .

[14] I. Chaniotis, K. Kyriakou and N. Tselikas, 'Is Node.js a viable option for building modern web applications? A performance evaluation study', *Computing*, vol. 97, no. 10, pp. 1023-1044, 2014.

[15] M. Shawabkeh and A. Masad, *jq-console*. 2015. GitHub repository, https://github.com/replit/jq-console