Introduction

Computers with web technology are popular. Beyond the desktop PC, embedded computers dominate our lives: from the moment our electronic alarm clock wakes us up; browsing the news, email, as we drive to work surrounded by micro-controllers in the engine, the lights, the radio, the heating, ensuring our safety through automatic braking systems and monitoring road conditions; to the workplace, where every modern appliance comes with at least one micro-controller; and when we relax in the evening, watching a film on our digital television, perhaps from a set-top box, or recorded earlier on a digital video recorder. And all the while, we have been carrying micro controllers in our credit cards, watches, mobile phones, and electronic organizers.

Vital to this growth of ubiquitous computing is the embedded processor—a computer system hidden away inside a device that we would not otherwise call a computer, but perhaps mobile phone, washing machine, or camera. Characteristics of their design include compactness, ability to run on a battery for weeks, months or even years, and robustness.

Central to all embedded systems is the software that instructs the processor how to behave. Whereas the modern PC is equipped with many megabytes (or even gigabytes) of memory, embedded systems must fit inside ever-shrinking envelopes, limiting the amount of memory available to the system designer. Together with the increasing push for more features, the need for storage space for programs is at an increasing premium.

Mobile Web technologies are becoming progressively more powerful. On higher end phones such as smart phones, nearly all of the features found on desktop computer browsers are becoming available. A smart phone is a pocket-sized handheld networked device that is a phone, portable media players, a digital camera, a video camera, and a handheld computer. It can browse web sites, send and receive email, download and read certain files and documents, and often, be used for GPS navigation as well. Millions of people simply cannot afford a smart phone. Some of them use a simple cell phone, with very limited capabilities: the ability to make and receive phone calls and text messages. Some people have something that's more than a cell phone but less than a smart phone: they have a feature phone, which has some web browsing capabilities. Consequently, Web applications increasingly are an effective alternative to native applications, following general Internet trends. In this research work, we discuss more about the compiler and interpreter architecture of the feature phone.
Porting browsers to mobile platforms may lead to new vulnerabilities whose solutions require careful balancing between usability and security and might not always be equivalent to those in desktop browsers [1]. Currently, most mobile phones have the possibility to connect to the internet. With upcoming flat-fee data plans, mobile phones are increasingly used to check e-mail, synchronize agenda’s, and browse the web.

Web content are largely authored using markup languages like HTML, XHTML, CSS etc. Various scripting provide more dynamic site behavior: JavaScript, Jscript, and VBScript. Among these JavaScript is an open scripting and a standard by name ECMA exists for JavaScript. JavaScript has been widely accepted and used, as it is not proprietary to any browser manufacturer. Current available open source engines Spider Monkey [2], Rhino [3], V8 [4], JS [5] are not ‘designed’ for mobile devices. Make huge demands in size (both code and runtime: 8 MB), customers are ‘NOT Open’ to using open source stuff. MUST not take more than 512 KB of run time memory and can be integrated into any type of application that needs script execution services. It should be processor intensive operations and must comply with standard ECMAScript–262 3rd edition [6] and would meet the OMA ESMP [7] conformance statement.

ECMA Script is the official name of the language commonly called "JavaScript". ECMA is an international, industry association and dedicated to the standardization of information and communication systems. The language is specified in the ECMA-262 document currently in the 3rd edition from 1999. This edition introduced exceptions into the language. JavaScript is the ECMA Script version of Netscape. To be honest, JavaScript was developed by Netscape and introduced with the Netscape 2.0 browser. Starting with JavaScript 1.4 Netscape also implemented the exceptions. Today Mozilla.org is developing in accordance with ECMA a new version 2.0 of JavaScript that will implement the features of an upcoming 4th edition of the ECMA standard.

Features of Script Engine includes the script type OMA-ESMP V 1.0 *, ECMA Script 262 3rd Edition *JavaScript v 1.5 *. charset support :US ASCII,UTF-8,UCS-2,UTF-16. The core languages are literals, arithmetic, logical, relational, bitwise, conditional branching, unconditional branching, exception handling, function calls, dynamic compilation. The built-in objects are global primitives, global functions such are object, function, array, string, regular expression, number, date, boolean, error, math, argument

In this research work, we introduce an optimized JavaScript engine for feature phones. This introductory chapter discusses about motivation towards and scope of this JavaScript engine’s architecture, research aim and objective, methodology, overview of JavaScript engine, optimization strategies adopted, algorithms used and thesis outline.

1.1 Motivation and Scope:

Looking at the day by day demands of web pages in feature phones, there is lot of necessity to optimize the JavaScript engine in terms of the the code size and the algorithms that are used for interpreting script. In this research work we propose a compiler and interpreter architecture of JavaScript Engine for feature phones.

Feature phones are typically characterized as constrained platforms, due to limitations in computing, storage and interface capabilities. Typically, feature-rich phone manufacturers now use 8 to 16 Mb of low-power SRAM for data backup; 32 to 128 Mb of Pseudo-SRAM (PSRAM) for the working area of the system; and 64 to 128 Mb of NOR Flash for bootable code storage. These devices perform internet browsing, text messaging, games, downloading and playing of music, and digital camera functionality and applications (including the ability to capture, transmit, receive, and display photos). Such applications have caused an increase in the complexity of memory requirements. To implement JavaScript Engine into a feature phone we need to consider execution time, memory and battery constrains of the feature phone and the dynamic behavior of JavaScript. After distributing the memory among all the application, the JavaScript engine should not use more than 512KB of SRAM. To give more priority to user interaction of browser applications, it should be executed within a stipulated time period.

We develop a JavaScript Engine within these limitations. As per ‘Point to analysis’ theory, a function pointer is to be used to access each property of an object. However it consumes lot of static memory of the devices. We define an optimized structure to reduce the static memory consumption.
After compilation, compiler generates AST node as an intermediate representation from the source code. Interpreting any language from the byte code is more preferable than from the self intermediate code (AST). However, byte codes are difficult to modify once they are emitted and it requires two passes to generate from AST node, which consumes lot of time, memory and battery power of feature phone.

We reduce the static memory by optimizing the AST node for different syntaxes of JavaScript. To give the more priority to user interaction, any scripts should be executed within a stipulated time by the browser. We develop an algorithm using non recursive stack which interprets the script within predefined time and define an algorithm to resolve the closure property of JavaScript using runtime stack.

1.2 Research Aim and Objective:

The research work aims at designing a JavaScript compiler and interpreter for small embedded devices like feature phones. The objective of the script interpreter is to design the JavaScript interpreter for embedded devices as per the ECMA (European Computer Manufacturers Association) by reducing memory consumption, reducing CPU cycle consumption, generic in nature, executing within an allocated time period and ease of portability to any devices with constrained memory.

1.3 Research Methodology:

Traditional byte code generation involves two stages of compilation. At first, it generates AST and then byte-code from the AST. Many times, it has been observed that by using the JavaScript libraries of JQuery, the scripts are compiled but not executed. Considering the memory limitation of the mobile devices and the limitation of execution time, it is preferable to generate AST node and execute as and when required rather than converting all AST nodes to respective byte code. However AST node is recursive in nature which can block the high priority user interaction of browser application. In JavaScript, built in objects and their properties occupy a huge static memory during launching of the compiler, and the commonly used syntax identifiers, if-then-else, dot operator and function call takes a lot of memory and time for interpretation. Closure is one of the most powerful features of JavaScript that allows inner functions, i.e. function definition inside the function body of another function.
The research approaches quantitative and qualitative methodology based on a non-recursive AST based stack algorithm for interpreting AST node, solving closure issue using runtime stack, reducing static memory of object behavior by optimizing object behavior and fast interpretation of commonly used syntax of script by optimizing AST node.

1.4 Contribution

Considering feature phone memory and speed, in this research work we introduce an optimized JavaScript Engine. We examine the current state in static code size optimization, and present different data structures for object and AST node optimization. We also define two algorithms to address the AST node interpretation and closure property of the JavaScript.

1.5 Summaries and Thesis Outline:

The remainder of this thesis is organized into the following chapters. The background and overview of JavaScript engine has been discussed in Chapter 2. Chapter 3 looks at the many and varied approaches to reducing code size, optimization of AST node, various compiler optimization, interpreter and stack algorithms those have been proposed in the last forty five odd years, examines their strengths and weaknesses, and considers how they might interact with each other either supportively or antagonistically. "Everything" in JavaScript is an Object: a String, a Number, an Array or a Function. All the objects and their properties are being loaded into the environment before compilation of any script. Depending on the script, these properties are invoked at the runtime through the function pointers. For reducing the high peak memory, we redefine the object structure. The chapter-4 describes more about the differentiation of peak memory before and after optimization of function pointers with a dummy function pointer. We analyze that variables, dot-operators, if-then-else and function calls consumes huge memory during compilation and take considerable time for execution. Chapter-4 explains how to optimize the AST nodes of such operators to improve the compilation and interpretation time. After source compilation, the compiler generates AST node for interpretation. Run-time efficiency is an important secondary requirement in many interpretive languages. In chapter-6 we develop a non-recursive stack algorithm; with all the features of JavaScript which execute the script in predefined time as user interaction of the browser is of more priority than script execution. In JavaScript, a closure is formed by returning an embedded function object that was created within an execution context of a
function call. In chapter-7 we define and algorithm to replace a runtime stack using an execution stack to handle embedded function object, which is a platform independent solution and is capable to run on various mobile devices.