HTML5 as HMI in a Command & Control System

Henrik Rydstedt
Abstract

HTML5 as HMI in a Command & Control System

Henrik Rydstedt

This thesis focuses on if it is feasible to create a realtime Human Machine Interface, HMI, solution to a Command & Control System in HTML5. The conclusion is based on some HTML5 features that is of interest for a system of this kind, a comparison between some of the most relevant web server solutions and a prototype. It shows that it is possible to create a HMI in HTML5 for a Command & Control System but with some delimitations.

A proposal would be to create a HMI in HTML5 as a supplement to the current instead of replacing it. The JavaScript API used to present the map in the prototype is releasing an upcoming version with support for WebGL, allowing smoother and graphical richer maps in the future.
Contents

List of Figures

List of Tables

1 Introduction 1
  1.1 Background ................................................................. 2
  1.2 Delimitations .............................................................. 2
  1.3 Question of Formulation .................................................. 2
  1.4 Thesis Disposition ......................................................... 2

2 Previous Experiences 4

3 System Overview 8
  3.1 HTML5 ................................................................. 8
      3.1.1 Compatibility ...................................................... 9
      3.1.2 Utility ............................................................. 10
      3.1.3 Interoperability .................................................. 10
      3.1.4 Universal Access .................................................. 11
      3.1.5 WebSocket ......................................................... 11
      3.1.6 WebGL .............................................................. 12
      3.1.7 Web Workers ...................................................... 13
      3.1.8 Cascading Style Sheets 3 .......................................... 13
      3.1.9 Web Storage ....................................................... 14
  3.2 Backend ................................................................. 15
      3.2.1 Node ............................................................... 15
      3.2.2 JSON ............................................................... 16
      3.2.3 MapServer ......................................................... 18
      3.2.4 Open Layers ....................................................... 20

4 Implementation Choice 21
  4.1 Web Server requirements .................................................. 22
  4.2 Input/Output, Multiple Cores and Single Core .......................... 25
  4.3 JSON and XML ........................................................... 28
  4.4 Alternative geospatial servers .......................................... 29
  4.5 Frameworks .............................................................. 29
## Contents

5 Prototype 30  
5.1 TDLH 30  
5.2 Structure 30  

6 Conclusions 32  

7 Future Work 34  

Bibliography 35
List of Figures

2.1 Runescape 3: Java & HTML5 Client ................................. 6
3.1 HTML5 Specification Diagram ................................. 9
3.2 WebSocket & Polling Latency Comparison ....................... 12
3.3 JSON Object Parsing ............................................. 17
3.4 JSON Array Parsing ............................................. 17
3.5 MapServer Application Architecture ............................... 19
4.1 Web Servers HTTP Throughput Benchmark with 1 Dyno ............ 23
4.2 Web Servers HTTP Throughput Benchmark with 4 Dyno ............ 24
4.3 Web Servers HTTP Throughput Benchmark with 7 Dyno ............ 24
5.1 Structure of the prototype ........................................ 30

List of Tables

4.1 CPU cycles on I/O ............................................. 27
Chapter 1

Introduction

This Master’s thesis will analyze whether HTML5 can be used as Human Machine Interface, HMI, for a Command & Control system at SAAB Group, called 9AIR Compact C2. Compact C2 is a compact command and control system, and ”a swift, straightforward way to integrate a Link 16 Tactical Data Link into the customers operational capability.”[1] The conclusion of this thesis will affect the decision as to whether a further development will occur.

Web sites have previously been a static way to spread information but has evolved tremendously over the last decade due to improvements of the web platform and the ability to provide with dynamic content, see more in Chapter 2. Today we have the opportunity to develop a web site with all the features that the new www-standard HTML5 can provide, allowing an even richer user experience and also to deploy web-sites as stand alone applications. It will most definitely become even easier and more widespread because the development does not belong to any specific company, but instead to a collaboration between several companies. The development of HTML5 has been done with many of the largest companies, which was a necessity to avoid a standard that the market does not want and in that case would cause the whole concept of HTML5 to fail, Chapter 2.

This thesis does not only consists of theoretical content but also some practical implementation. Chapter 3 covers features in HTML5 and backend that might be suitable for a Command & Control System. The prototype consists of a web server that distributes a menu and a map with continuous update of system tracks to the client, a device with a web browser with WebSocket support. System tracks in Compact C2 are units such as military aircraft, ships and ground forces. These tracks are normally provided by a module in Compact C2 called Track Distribution, TRDI. However, for this prototype the tracks are provided by a module called Tactical Data Link Handler, TDLH.
Chapter 1. Introduction

1.1 Background

Compact C2 is a project with an interest in new technology and improvements. The current Human Machine Interface, HMI, is made in Java and the people in Compact C2 are discussing whether to switch to another more future proof front end solution. In the future a communication standard called Data Distribution Service, DDS, is planned to be use to increase the modularity of the system. DDS is an open middleware communication standard for real-time and embedded systems and enables applications written in different languages to interoperate. HTML5 might be considered as a good choice for the new HMI and hence the starting point of this thesis.

1.2 Delimitations

HTML5 is relatively new and so are the API’s and the frameworks built around it. There have been some difficulties finding applicable academic studies regarding HTML5 and therefore this article is based partially upon information that has been extracted from different blogs and other sources that might seem less accurate. However, the information extracted consisted mostly of practical advice, or concrete examples from larger companies that have some experience with HTML5.

1.3 Question of Formulation

To guide my work in this thesis I will try to answer the following research question.

- Is it feasible to create a realtime HMI solution to a Command & Control System in HTML5?

1.4 Thesis Disposition

*Chapter 2: Related Work* - Covers related work regarding companies switching features or whole products to HTML5.

*Chapter 3: Method & Tools* - The most interesting features from HTML5, a web server and geospatial server that could benefit a Command & Control System.

*Chapter 4: Implementation* - Explains the decision process for the web server solution.
Chapter 1. Introduction

Chapter 5: Prototype - How the prototype works.

Chapter 6: Future Work - What should be considered if decisions were made to continue the work further.

Chapter 7: Conclusions - Answers the question of formulation and sums up the thesis.
Chapter 2

Previous Experiences

This chapter focuses on the issues and experiences of HTML5 in complex applications and is divided into two parts. In order to understand why HTML5 exists today, it is important to know its history over the last two decades. With HTML5 comes new possibilities in web development and the second part presents experiences and examples from different web projects.

HTML was first published as a draft in 1993 and evolved considerably during the 90’s. One year it was updated with three larger versions called 2.0, 3.2 and 4.0. In 1999 version 4.01, supervised by the World Wide Web Consortium, W3C, was released and remained for several years. During this era HTML became almost deprecated when the focus changed to other web standards named eXtensible Markup Language, XML, and eXtensible HyperText Markup Language, XHTML. [2, p.1]

A group called Web Hypertext Application Working Group, WHATWG, was founded in 2004 with the focus on raising the level of the web platform and the HTML5 specification was created. HTML should become a platform with features to better suit web applications as the web sites demanded more dynamic information. [2, p.1]

The W3C resumed involvement with HTML in 2006 and in 2008 a new working draft for HTML5 was released. HTML5 is under constant development with the focus on real and practical improvements. Providing with more platform and device independency and also advanced features for web development, for example embedded graphics, client-side data storage and parallel code execution, see more in Chapter 3.1. Browser vendors are eager to update and experiment to the new revisions of the HTML5 specifications due to the improvements that follows, leading to immediate feedback and strengthens the specifications. [2, p.1] The undergoing development is supervised by WHATWG, W3C and Internet Engineering Task Force, IETF. [2, p.3]
Chapter 2. Previous Experiences

In 2011 LinkedIn released a new mobile app in HTML5 and native Android and iOS. The mobile app was 2-10 times faster on the client side compared to its previous native mobile implementations. Initially the server was written in Ruby on Rails and was replaced by Node.js, Node, causing a performance gain with only using “a fraction of the resources”. LinkedIn investigated what web server platform to use, such as Ruby, Java and Scala, and decided to use Node.js, Node. The company educated the the Ruby on Rails developers and also hired some V8 engineers.\(^1\)

Developing the applications took a short time due to the way the developing team reused HTML5 code from the web application in the native implementations for Android and iOS. LinkedIn used Underscore.js, a utility JavaScript library\(^2\), and Backbone.js, a JavaScript library with models and views\(^3\), during the development of the mobile application.

LinkedIn reversed their decision in 2013 and decided to discard HTML5 and use the native applications for Android and iOS. The main reason for this was “we’re seeing that more and more people are spending more time in the app, and the app is running out of memory. It’s not performance issues, like speed or rendering, but it’s still a big problem”. The second reason was to increase the smoothness of animations which they felt had a better performance in the native applications.\(^4\)

According to Kiran Prasad, LinkedIn’s senior director for mobile engineering, there are a few critically important features that are missing in HTML5. Tooling support is one of them, to have a working debugger and performance tools to diagnose memory leakage. Another important feature is to have runtime diagnostics information when distributed to a large number of users.\(^4\)

Not until those two features have been corrected will HTML5 be chosen over a native mobile application implementation, according to Prasad. “It’s not that HTML5 isn’t ready; it’s that the ecosystem doesn’t support it.” Either the tools need to mature or the developers need to build the tools by themselves in order for HTML5 to become more widely used.\(^4\)

Ebay runs for the most part on Java, but when they wanted to bring in a new project called ”Talk” they discovered that the basic Java did not seem to fit the project requirements. There were two main requirements, the first was that the application should be as real time as possible, the server should manage many concurrent live connections. The second, was to manage to display content on the page from various services made

---

\(^1\)http://venturebeat.com/2011/08/16/linkedin-node/

\(^2\)http://documentcloud.github.io/underscore/

\(^3\)http://backbonejs.org/

\(^4\)http://venturebeat.com/2013/04/17/linkedin-mobile-web-breakup/
by eBay that required for example I/O-bound operations. Node seemed to be a perfect fit for the requirements and sparsely consumed the server resources compared to the initial implementation with the basic Java infrastructure.\(^5\)

Mozilla has been working on a new mobile platform called Firefox OS which has been released in many countries and more to come later of year 2013.[3, p.1] Firefox OS is "specifically designed to run on limited hardware while providing a decent user experience".[3, p.2] It also brings up openness in the aspect of using HTML5 as the development platform for applications. In order for a company to release a mobile app, they will have to release the same application in a variety of languages for different platforms, for example Android using Java and Apple using Objective-C. By using HTML5, many web developers are already familiar with the concepts of creating applications for Firefox OS. Garzia [3] also mentions that HTML5 applications are easy to port to other platforms by using wrappers like Phonegap\(^6\).

![Runescape 3, Java and HTML5 Client](http://www.runescape.com/beta)

Figure 2.1: Runescape 3, Java and HTML5 Client

Runescape 3 is a free to play Massive Multiplayer Online Roleplaying Game with over 175 million registered users.\(^7\) The current game client is written in Java, but a beta release of their HTML5 client was released earlier this year. The HTML5 client is at a beta stage and consist of bugs and graphic glitches that will be solved as the beta process.\(^8\) Figure 2.1 shows the difference between the two clients.

EXastum and OS.js are two desktop environments written in HTML5 and both of which are maintained by a single developer. They are very similar to ordinary desktop environments with applications such as video player, web browser, audio player. OS.js can

\(^5\)http://www.ebaytechblog.com/2013/05/17/how-we-built-ebays-first-node-js-application/

\(^6\)http://phonegap.com/

\(^7\)http://www.runescape.com/beginnersguide.ws

\(^8\)http://services.runescape.com/m=news/runescape-3-beta-programme-now-live
also serve on top of X11 as a complete graphical environment. \(^9\)\(^10\)

To summarize the experiences from the different projects, HTML5 allows for reusable code in native mobile applications for Android and iOS and can also be ported using external software. HTML5 is lacking smoothness in animations for native mobile applications and is hard to debug and find memory leaks due to the absence of a proper debugger and diagnostics tools. HTML5 can still be used but in order to perform in large scale applications the tools need to be improved. HTML5 provides better performance on mobile devices than the previous HTML version and its new features opens up new possibilities for more complex applications such as serving as game clients and desktop environments. The web server named Node performs well in heavy I/O and maintaining many live connections.

In this chapter we have seen indications of that the investigated approach would be feasible for the intended purpose. However, we need to investigate the most likely candidates in more detail, in order to find out whether there are any possible problems that may arise from the tools. In order to do this I will go through the most important and interesting implementation tools in the next chapter, and in four, we will have a more thorough discussion on benefits and weaknesses of the various platforms.

\(^9\)http://sourceforge.net/projects/exastum/
\(^10\)http://andersevenrud.github.io/OS.js/
Chapter 3

System Overview

This chapter brings up the new techniques and features in HTML5 that could make it possible to create a Command & Control System. In addition, the choice of the web server and protocols are also taken into account.

HTML5 is a specification written in consent with WHATWG, W3C and IETF to take the web platform to the next step. The earlier specification contained more content and covered a larger spectrum of technologies. As seen in Figure 3.1, the HTML5 specification has been divided into smaller specifications, some of which has been deprecated while some are just working drafts. It has been revised to make it more concise and clear, some features such as WebSocket was transferred to its own specification, but are still standardized by W3C and can still be considered part of HTML5.\(^1\)

3.1 HTML5

HTML5 is the new upcoming web coding standard, with an official release date of late 2014. It will furthermore continuously be updated, with plans of releasing a HTML 5.1 version in 2016.\(^1\)

HTML5 is based on several design principles written in the WHATWG specification.\(^2\)

- Compability
- Utility
- Interoperability
- Universal Access

\(^1\)http://dev.w3.org/html5/decision-policy/html5-2014-plan.html
\(^2\)http://www.w3.org/TR/html-design-principles/
3.1.1 Compatibility

One of HTML5’s main principles is the overall aim to keep it compatible with older HTML. With around 20 years of existing HTML on the web, supporting it is therefore a necessity.

HTML5 is also about improvement and a lot of resources have been put into researching web design behavior. An example is an analysis of the naming convention for the ID and Class names for DIV-tags, conducted by Google. One of their discoveries were the common use of the name ”header” for the DIV holding the top content of the website, therefore the `<header>` element amongst other are included in the new standard.[2, p.3] However, some features in HTML5 are brand new, for example `<video>` and `<audio>` tag, web workers, web sockets but the focus lies on evolving the HTML platform.[2, p.6-7]

![HTML5 Specification and related technologies](http://upload.wikimedia.org/wikipedia/commons/f/f7/HTML5-APIs-and-related-technologies-by-Sergey-Mavrody.png)

**Figure 3.1:** HTML5 Specification and related technologies.
3.1.2 Utility

The HTML5 specification is written based upon a priority of constituencies. It is specified to consider users over authors over implementers over specifiers over theoretical purity. But solutions in favor of multiple constituencies are preferred. This results in HTML5 being very practical. Here is an example.

```
id="html5"
id=html5
ID="html5"
```

All three variations are equally valid in HTML5. Some people might think this is too arbitrary, but the end user is not affected at all. As Peter Lubbers and Salim [2] stated, the end user will be affected if any of the former examples generates errors and leaves the end user with a half generated page.

3.1.3 Interoperability

HTML5 is developed with the endeavor to be truly interoperable. In order to achieve this, the specification follows two principles, well-defined behavior and to avoid needless complexity.

Avoid vague or implementation-defined behavior and strive for a clearly defined behavior. Making it up to the developers of the user agents, such as web browsers, to implement a HTML feature with the focus on how it should behave for the web developer. Having all web browsers display html the same way, makes it easier for web developers. In addition, implementations should still be free to make improvements in quality of rendering, user interface and such areas.

Always use simple solutions over complex, whenever possible. Simple features are easier to implement in user agents, leading to interoperability and are easier for web developers to understand but it should not be exploited as an excuse to avoid upholding the other principles.

---


3.1.4 Universal Access

Access to content by everyone regardless of disabilities is important. HTML5 contains features for screen reader support, aimed at users with disabilities. The specification is called Web Accessibility Initiative - Accessible Rich Internet Applications (WAI-ARIA) and can be used in HTML elements. HTML5 should also be platform independent and the available features should work on all different devices if possible. A feature should not be left out just because some platforms do not have the support, or if not all users can make full use of them. For example interactive features should not be omitted because they can not be presented in a printed document, or a blind person not being able to see images and therefore the image feature should not be omitted but instead provide an alternative text. HTML5 has also a support for all world languages by Unicode. Unicode is an international encoding standard where each letter, digit or symbol is assigned a unique numeric value regardless of platform.

3.1.5 WebSocket

HTML5 enables the support for WebSocket. It allows the server to send data to the client, whenever the server has data ready to be sent. The client does not need to continuously ask the server if there is any new data to receive. Not until WebSockets first implementation, web sites could only use Hypertext Transfer Protocol, HTTP, for data communication. HTTP was not designed for the real-time communication modern browsers require.

Before Web Sockets, other techniques were used such as AJAX polling, Flash, multipart, but the most common is called “Long-polling” or ”Reverse AJAX”, used by Comet. The client opens an HTTP connection to the server and keeps it open until receiving a response. The problem is they carry the overhead of HTTP, which decreases performance in low latency applications. This is where WebSocket shines; it uses minimal overhead due to the use of it own external WebSockets protocol.

If the developer know the exact interval of messages, polling is a good solution because the client can be synchronized to only send a request to the server when there should be new data available. This does not apply to real-time data because it is mostly not updated at a consistent interval and will instead cause the client to send unnecessary requests. WebSocket offers something new, it is designed from scratch to only consist of binary and text, be full-duplex and optimized for minimum overhead in both

4http://www.w3.org/TR/html-design-principles/
5http://www.unicode.org/
6http://www.html5rocks.com/en/tutorials/websockets/basics/
bytes and latency. This leading to, the web browser is able to use any network protocol and opens up a vast amount of possibilities.\[4, p.xiii\]

The WebSocket protocol is an independent TCP-based protocol. The establishment of a WebSocket comes with a handshake by sending a HTTP upgrade request to the server. WebSocket may also be used with TLS to encrypt the data.\(^7\)

![Latency comparison between WebSocket and Polling.](http://www.websocket.org/quantum.html)

The major advantage of WebSocket is the reduced latency. After a connection is established, the server can send data to the client continuously. Unlike polling, where the client asks the server if there is any new data to receive and the server replies with data if it is available, see 3.2. The common polling technique sends these requests at intervals, regardless if new data is available. This is leading to increased latency, CPU workload and bandwidth consumption. Instead, WebSocket makes a single request and the server does not need to wait for the client to ask for new data.\[2, p.7\]

### 3.1.6 WebGL

Over the years several browser manufacturers including Mozilla, Opera and Google worked and experimented on their own with 3D APIs for JavaScript. \[2, p.314\] WebGL is a web standard for low-level 3D graphics API based on OpenGL ES 2.0 and can be

\(^7\)http://tools.ietf.org/html/rfc6455
used with the HTML5 Canvas element. The Khronos Group, responsible for OpenGL, and browser vendors Apple, Google, Mozilla and Opera are all members of the WebGL Working Group. It is royalty free, support cross platform and stays very close to the OpenGL ES 2.0 specification but with some modification to work better with JavaScript.

3.1.7 Web Workers

JavaScript is single-threaded which results in complex computations will block the UI thread and hinder the user from interacting with the UI components such as buttons, text fields or CSS effects. To avoid such a thing, HTML5 comes with Web Workers. Web Workers typically runs on separate threads, allowing the usage of multicore CPUs. Separating long running JavaScript-code into Web Workers also avoids the slow-script warning that occurs when JavaScript loops for several seconds.[2, p.241]

3.1.8 Cascading Style Sheets 3

Cascading Style Sheets, CSS, enables control over the layout and design of the web page. It works with HTML and is a stand alone style sheet language. In comparison, HTML provides a structure to a web page by organizing context into paragraphs, headers, lists etc; CSS works with the web browser to make HTML look beautiful.[5, p.1]

Before CSS, web designers relied on both styling web pages and context organizing with HTML. By looking at a web page from 1995, the styling was very limited.

CSS offers many advantages over HTML-styling, the following describes some of them.[5, p.2]

- Style sheets offer better formatting choices than HTML. CSS enables formatting paragraphs as they appear in magazines or newspapers.
- CSS styles use much less space than HTML’s formatting options, making the web page look good and load faster.
- By collecting all styles in a single external style sheet, referenced in all pages on the web page, changing that style sheet changes all web pages it is referenced to. Enables an easier way of updating the web site’s layout.

[^8]: http://www.khronos.org/webgl/
CSS was first introduced in 1996 and has since been updated and the current accepted standard is now CSS 2.1. CSS3 is the first version that actually is not one single standard. Due to the increased complexity, the W3C decided to split up CSS3 into separate modules. This enables individual development of each module and there is not a single standard called CSS3. For example, level 3 of the Selectors module is already complete and level 4 is being worked on. CSS3 is in fact a loose collection of a variety of modules at various states of completion. This means that there will not be any CSS4, just new versions of the different modules.[5, p.9]

### 3.1.9 Web Storage

Web Storage enables retaining data on the client side between web requests. Earlier, web servers sent the persistence data back and forth to the client, by using cookies. Web Storage does not only reduce network traffic, it also gives a larger storage volume.[2, p.263]

HTTP Cookies, cookies, have existed far longer than the web storage but is limited due to the fact a cookie header is sent back and forth the client and the server in a HTTP request, regardless if either node needs the stored data. Cookies should provide with a storage capacity of at least 4096 bytes. The capacity could be larger, but due to the continuous transfer in the HTTP requests having larger storage size would increase the bandwidth consumption when it is not necessary.\(^9\)

The Web Storage API enables storage directly on the client side for a set period or unlimited amount of time, even after the browser closes. There are two types of Web Storage, Session Storage and Local Storage.[2, p.264] Local Storage is a key-value storage. At the moment both of which must be stored as strings and other data structures need to be serialized and de-serialized before stored. This can be done by using JSON.stringify() and JSON.parse(). Local Storage can contain up to several megabytes of data, making it more suitable for document and file data that would otherwise exceed the size limit of a cookie. Local Storage data exist until it is deleted by the user or the application. Session Storage is the same as Local Storage except it has a set life span.\(^11\) Web Storage is more secure and faster compared to cookies. The data is only used when asked for and is not included in every server request. The stored data can only be accessed by the user and the website who stored the data. The Web Storage feature does not affect the websites performance.\(^12\)

---

\(^10\)http://www.html5rocks.com/en/features/storage
\(^11\)http://dev.w3.org/html5/webstorage/
\(^12\)http://www.w3schools.com/html/html5_webstorage.asp
3.2 Backend

The backend of the system consist of a web server that supplies clients with HTML document requests over HTTP.\(^\text{13}\) HTML and JavaScript can be run directly on the client with a web browser, with no connection to a web server, but with the lacking functionality of centralized data exchange. The web server’s purpose in the prototype is to receive system tracks from Compact C2 and distribute them to the connected clients. This chapter introduces the chosen solution for the web server along with required tools.

3.2.1 Node

Node.js or just Node is a server-side JavaScript environment. Node uses an event-driven and non-blocking I/O model to become lightweight and efficient in order to serve the purpose of data-intensive real-time applications.\(^\text{14}\) Node has more than 15,000 modules, made by the community, published in Node Package Manager, NPM.\(^\text{[6, p.2]}\)

Node is based on Google’s JavaScript runtime engine named V8 engine and uses it for server side programming. Node and V8 are mostly implemented in C and C++, with focus on performance and low memory consumption. V8 supports mainly JavaScript in the browser, but Node aims to support long-running server processes.\(^\text{[7, p.1]}\) Even though the web server is developed in JavaScript, Node gets a huge boost in performance from V8 due to the source code is compiled into native machine code, instead of executing byte code or using an interpreter.\(^\text{[6, p.3]}\) A Node process does not rely on multithreading to support concurrent execution, but uses an asynchronous I/O eventing model. Node uses the eventing model at the language level. JavaScript supports event callbacks, which is perfect for Node’s approach. For example, when a user click a button or when a browser completely loads a document, the event triggers a callback. Due to JavaScript’s functional nature it is very easy to create anonymous function objects to register as event handlers.\(^\text{[7, p.1]}\)

JavaScript is the most popular programming language in the world, thanks to the web’s tremendous reach. Due to its interoperability, the statement “write once, run anywhere” is fulfilled. It is also becoming faster. After the release of Google Chrome, JavaScript’s performance has increased at a rapid pace due to high competition between browser vendors; Mozilla, Microsoft, Apple, Opera and Google.\(^\text{[6, p.3]}\)

Node has a module in NPM called Socket.io that enables handling WebSocket for both server and client side. According to the Socket.io web site\(^\text{[15]}\) Socket.io handles not only

\(^{13}\text{http://en.wikipedia.org/wiki/Web_server}\\)

\(^{14}\text{http://nodejs.org/}\\)

\(^{15}\text{http://socket.io/#faq}\\)
the WebSocket transport protocol, but also additional features like heartbeats, timeouts
and disconnection support. There are also a variety of different JavaScript frameworks
that can be used with Node development, such as Express or Backbone. Both of which
can be found in the NPM.\footnote{https://npmjs.org/}

Node has a simple module system, where a module exists in one file, a one-to-one corre-
spondence between files and modules. Examples are taken from Node’s documentation.\footnote{http://nodejs.org/api/modules.html}

Foo.js

\begin{verbatim}
1 var circle = require('./circle.js');
2 console.log( 'The area of a circle of radius 4 is ' + circle.area(4));
\end{verbatim}

Circle.js

\begin{verbatim}
1 var PI = Math.PI;
2
3 exports.area = function (r) {
4    return PI * r * r;
5 }
6
7 exports.circumference = function (r) {
8    return 2 * PI * r;
9 }
\end{verbatim}

In Circle.js both area and circumference are visible outside of the module due to the
exports-statement. They are reachable in Foo.js through the require('./circle.js') call.
Local variables to the module will be private, for example PI will be private to Circle.js.\footnote{http://www.json.org}

3.2.2 JSON

JavaScript Object Notation, JSON, is a data interchange format. It is text-based and
both easy for humans to understand as well as easy interpreted for computers. JSON is
derived from JavaScript scripting language for representing simple data structures and
objects. It is language independent, despite the close relation to JavaScript, but uses
conventions that are familiar to programmers of the C-language and its progenies such
as C++, C#, Java, JavaScript Perl, Python etc.\footnote{http://nodejs.org/api/modules.html}

JSON example
Since JSON is derived from JavaScript it is possible to parse JSON text into an object by using JavaScript’s eval() function, but it is not recommended. JSON is often used for serializing and transmitting data over a network, primarily between a server and a web application, providing an alternative to XML.\(^\text{19}\)

JSON is built on two fundamental structures.

- A set of name-value pairs, see Figure 3.3. Might be seen in other languages as an object, record, struct, dictionary, hash table, keyed list or associative array.

  ![Figure 3.3: JSON interpretation of an object](http://www.json.org)

- An ordered list of values, see Figure 3.4. Can be seen as an array, vector, list or sequence in other languages.

  ![Figure 3.4: JSON interpretation of an array](http://www.json.org)

An object is surrounded by “{” and “}”. Name and values are separated with a colon, following pairs are separated with a comma.\(^\text{20}\)

\(^{19}\)http://en.wikipedia.org/wiki/JSON

\(^{20}\)http://www.json.org
3.2.3 MapServer

In order to develop a Command & Control System a geographical map is required for the user. To achieve this and avoid having geospatial data locally on the clients, a distributed approach was chosen with the software called MapServer.

MapServer is an OpenSource geographic data rendering engine written in C and runs on all the major platforms Linux/Unix/Mac OS X and Windows. MapServer was originally developed by the University of Minnesota in the mid 1990’s, but is now a project of The Open Source Geospatial Foundation.

MapServer receives geospatial data as input and renders it to a chosen image format. The input data may be from a local file in vector or raster data, or from a web service, see Figure 3.5. MapServer uses mapfiles to configure where the geospatial data is located and how it is to be drawn. The rendered images can be stored locally or be sent to another destination using the protocol Web Map Service, WMS. \(^{21}\)

One of many input formats supported by MapServer is a Shapefile, which is a geospatial vector data format developed and regulated by Esri. Shapefiles describes for example rivers, water wells and lakes featured as points, lines and polygons.\(^ {22}\)

\(^{21}\) http://mapserver.org/about.html
\(^{22}\) https://en.wikipedia.org/wiki/Shapefile
Figure 3.5: The basic architecture of MapServer applications
http://mapserver.org/introduction.html
3.2.4 Open Layers

OpenLayers is a framework that enables the integration of having a dynamic map in a web page. It can display map tiles and markers from a variety of sources. OpenLayers is an Open Source project written in JavaScript.\textsuperscript{23} OpenLayers receives the geospatial data from MapServer and renders it in the web browser.

OpenLayers has a constant updating JavaScript API for building rich web-based geographic applications. By separating map tools from map data OpenLayers become more interoperable, making it possible for all tools to operate on all the data sources.\textsuperscript{24}

Open Layers Features\textsuperscript{24}

- Support for a variety of data sources
  - WMS; ka-map; TMS; WorldWind; WFS; GeoRSS; Google, Yahoo, Microsoft, MultiMap.
- Support for displaying geographic features, with markers and popups.
- Easy build configuration, designed to help build OpenLayers into other applications.
- JavaScript API to allow full control over OpenLayers-powered map from within JavaScript on a web page.

\textsuperscript{23}http://openlayers.org/
\textsuperscript{24}http://www.osgeo.org/openlayers
Chapter 4

Implementation Choice

In the previous chapter both HTML5, with its new features, and Node were described and explained in some detail. In order to understand this choice, it is necessary to go through some of the other web servers on the market and see to which degree they meet the requirements. This chapter covers the decision process in the choice of web server technology.

The Compact C2 system contains a number of requirements that have to be met, in order to achieve an easier integration with the rest of the system.

- A real-time environment is necessary for the continuous track updates and command execution in a Command & Control System. These cannot be put on hold or be postponed.

- Modularity provides for easy development, deployment and maintenance.

- Scalability is a major issue for real-time implementations. When connections and system complexity increases the web server should still have a solid performance.

- The web server should be compliant with Data Distribution Service, DDS. Compact C2 is aiming to implement DDS and the web server should support it for future use.

- The web server should be able to run on a Linux system since this is the current implementation platform Compact C2 is based on.
4.1 Web Server requirements

One of the requirements for the web server is for it to be real-time, to have real-time data exchange between the server and the client with almost no delay. WebSocket provides the nearest experience of real-time and therefore has to be supported by the web server. There are numerous web servers supporting WebSocket, some with more community support than others.\(^1\)

In order to test various platforms for performance, Evan Weaver conducted a hello world benchmark for a variety of web server platforms including Accept, Jetty, Tomcat, Node, Bottle, Sinatra and Finagle\(^2\).

Tomcat is an open source software implementation of the Java Servlet and JavaServer Pages.\(^3\) The code is written in Java and run in the Java Virtual Machine, JVM, which is responsible for the hardware- and operating system-independence and is the part of the Java platform that ties it all together.\(^8\), p.2] JVM can be run on most platforms, included Linux.\(^4\)

Finagle is a Remote Procedure Call, RPC, system for the JVM and is used for constructing high-concurrency servers. Finagle provides both Scala and Java idiomatic APIs.\(^5\) RPC allows programs to communicate in different address spaces, for example another computer in another network, by executing subroutines or procedures on the other location. The developer writes basically the same code as if the subroutine or procedure were local.\(^6\)

Sinatra is a Domain Specific Language, a language that is only specialized to a special type of domain, for rapid development with minimal effort of web applications in Ruby. Sinatra is written in Ruby and because Ruby is supported by Linux, so is Sinatra.\(^7\)

Weaver used autobench to benchmark the web servers. Autobench is a Perl script running httpperf, a tool for measuring web server performance, on each host and gradually increasing the number of server requests per second.\(^8\) The benchmark uses dynos hosted by Heroku. A dyno is a virtualized lightweight Unix container running a user-specified command.\(^9\) Each dyno have 512 MB of RAM and unclear amount of CPU power. With the given configuration, one single httpperf process could generate up to 25,000

\(^1\)http://en.wikipedia.org/wiki/WebSocket
\(^2\)http://blog.evanweaver.com/2012/02/29/hello-heroku-world/
\(^3\)http://tomcat.apache.org/
\(^4\)http://twitter.github.io/finagle/
\(^5\)http://en.wikipedia.org/wiki/Remote_procedure_call
\(^6\)https://www.ruby-lang.org/en/downloads/
\(^7\)http://www.xenoclast.org/autobench/
\(^8\)https://devcenter.heroku.com/articles/dynos
requests per second and timeouts were set high enough for any intermediate queues to stay flooded.\footnote{http://blog.evanweaver.com/2012/02/29/hello-heroku-world/}

A hello-world benchmark might of course not be the most accurate method to rate different web server platforms, but at least it tells something about handling connections and the performance in the most simple use case. To achieve the best benchmark a more realistic web server application should have been implemented with the different web server platforms, but this would be a very time consuming task, which has not been within the scope of this thesis work.

There were three different benchmarks in the study, where the number of Dyno servers used is the independent factor. In the first benchmark, shown in figure 4.1, Finagle has the best case but it does not show as solid performance as Node. The competing system Sinatra has a quite solid performance, but only the third best. The second benchmark, shown in figure 4.2, leaves Node as the best performer. Apache Tomcat initially appears to have the same performance as Node but the performance gets lower as the amount of requests increases. In the last benchmark, Figure 4.3, Sinatra performs the best, but the other web servers are not far behind. Node drops a bit of performance in the middle of the graph but comes back to the same level as Sinatra at the end. Finagle and Tomcat

\begin{figure}
\centering
\includegraphics[width=\textwidth]{web_server_http_throughput_benchmark_with_1_dyno.png}
\caption{Web Server HTTP Throughput Benchmark with 1 Dyno http://blog.evanweaver.com/2012/02/29/hello-heroku-world/}
\end{figure}
Figure 4.2: Web Server HTTP Throughput Benchmark with 4 Dynos
http://blog.evanweaver.com/2012/02/29/hello-heroku-world/

Figure 4.3: Web Server HTTP Throughput Benchmark with 7 Dynos
http://blog.evanweaver.com/2012/02/29/hello-heroku-world/
perform very much in the same way but hits the bottom at an earlier stage than Node and Sinatra, causing network errors and in the end does not even respond on requests.

These benchmarks is a way of determining which web server best fulfills the requirement of robustness and scalability due to the intense amount of requests per second. In the overall comparison Node stands out as the most solid and best performer, leaving Sinatra, Finagle and Tomcat a bit behind.

By looking at the Tomcat, Finagle and Sinatra are all run in the JVM and are therefore supporting deployment on a linux system. Node does as well support linux.

Node is not just a web server, it’s a server-side JavaScript environment with a webserver feature. Node is very similar to another web server called Nginx. Both use an asynchronous event-driven architecture and focus on being both lightweight and memory efficient. What differs is that Nginx is just a web server and uses the Common Gateway Interface, CGI, to connect with other software, while Node relies on the JavaScript environment.

A few of the mentioned web servers do meet the web server requirements, but the one that does it best is Node. Due to HTML5 and the web’s increased usage over the last decade, JavaScript is a very common language. In addition, the frontend is made in HTML5 and JavaScript and simply having to use the same language for frontend and backend is a good advantage. The next chapter explains in some more details how Node works, in order to investigate the issue of scalability.

### 4.2 Input/Output, Multiple Cores and Single Core

In order to understand why Node can operate with low memory consumption and good throughput performance, the understanding of Input/Output (I/O) and CPU core technology is necessary. Web servers handles a large amount of I/O. A single core system executes only one thread at a time, switching between them after a period of time or when for example an I/O is performed. Although a multiple core system allows execution of multiple threads at the same time, each core executing a separate thread.

There are two different approaches in how to schedule a thread performing I/O, synchronous and asynchronous. A thread performing synchronous I/O starts an I/O operation and enters a wait state until the request is completed. The other variation is

---

11http://nodejs.org/
asynchronous I/O, where a thread sends an I/O request to the kernel. If the kernel accepts the request, the thread switches to another job and processes it in the meantime. When the I/O operation is complete, the kernel signals the thread telling it the I/O operation is complete and the data is to be processed.\textsuperscript{12}

Synchronous and asynchronous I/O has their different use cases. In situations where an I/O request takes long time to complete, the asynchronous approach is better suited. The synchronous technique offers a better performance where I/O requests are relatively short, because of the overhead of processing kernel I/O requests.\textsuperscript{12}

According to Steve Vinoski and Stefan Tilkov[7, p.1] they claim that most developers ”agree that multi-threaded programming is anything but easy”. It consist of problems that are complicated to fix, for example deadlock and defending shared resources amongst threads. Developing for multi-threading causes the developer to lose some control because the Operating System, OS, most commonly decides the run time and what thread to execute. Event-driven programming supplies with a more efficient, scalable and enhanced control over switching between application activities. In the event-driven programming model, the application relies on event notification handled by the OS. Applications tell the OS they are interested in certain events, for example data is ready to be written on a socket, and the OS notifies the application when the event occurs.

There are some problems in event-driven programming with asynchronous I/O. Not all communication in a CPU can be tied to event notifications. As an example, Steve Vinoski and Stefan Tilkov[7, p.2] mention shared memory segments can’t be used for communication between two applications because they can’t register events due to the lack of handles or file descriptors. As they suggest, a workaround would be that ”developers must resort to alternatives such as writing to a pipe or some other event-capable mechanism together with writing to shared memory”.

JavaScript has become a central element of any HTML-based application. Using JavaScript as a server-side programming language is a logical next step, allowing the use of one single language for all aspects of a web-based application. Mozilla and Google have released high-performance, fast and scalable JavaScript runtime implementations.[7, p.2]

I/O, is expensive, especially when the medium is not very close to the CPU. As seen in Table 4.1, the medium consuming most CPU cycles is the network, which of course is a heavily used medium for web servers.

Threads can be described as ”computational workspaces in which the processor works on one task”. If the web server blocks on I/O and there are more requests to handle,
Chapter 4. Implementation Choice

<table>
<thead>
<tr>
<th>Type of memory</th>
<th>CPU cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-1 Cache</td>
<td>3</td>
</tr>
<tr>
<td>L-2 Cache</td>
<td>14</td>
</tr>
<tr>
<td>RAM</td>
<td>250</td>
</tr>
<tr>
<td>Disk</td>
<td>41 000 000</td>
</tr>
<tr>
<td>Network</td>
<td>240 000 000</td>
</tr>
</tbody>
</table>

Table 4.1: CPU cycles on I/O

http://blog.mixu.net/2011/02/01/understanding-the-node-js-event-loop/

the typical approach would be to use multithreading, with one thread for one or several connections. However when the amount of connections increases, the more CPU power and RAM is needed for context switching etc. It can also be a complex task to manage threads in an application.[6, p.7]

Node is an asynchronous event-driven platform, using an event-loop that is abstracted from the developers with the purpose of taking care of all the I/O operations. Node does not need to be multi-threaded due to the event-loop and asynchronous I/O.[6, p.7] The code is run in only one thread but uses callbacks to delegate asynchronous I/O operations to the event-loop. Web servers uses immense communication over the network, it would not be viable to use a synchronous approach due to the immense of CPU cycles it consumes.[14] Callback functions are often used for a programming language to interact with events.[7, p.2]

```javascript
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World');
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

This code creates a web server and tells the server to respond with an html document containing “Hello World” when a user connects. Line 1 shows the example of modularity in Node, the variable http is bound to the module ’http’. The http-module provides with functionality that handles http requests and responses from clients. Line 2 shows a callback-function, function (req, res){ ... }, that is to be called when a client sends a http-request to the server.[15]

---

[6]: http://nodejs.org/about/
[7]: http://blog.mixu.net/2011/02/01/understanding-the-node-js-event-loop/
[14]: http://blog.mixu.net/2011/02/01/understanding-the-node-js-event-loop/
[15]: http://nodejs.org/
Node is available on Linux/Unix/Mac OS X and Windows\textsuperscript{16} and has support for OpenDDS with a module found in the NPM.\textsuperscript{17}

4.3 JSON and XML

Why would you use JavaScript Object Notation, JSON, rather than XML, which might be a more straightforward approach? In order to understand this choice we have to compare the two a little closer.

JSON is simpler than XML because it has a much simpler syntax than XML. It is also more alike the data structures used in modern programming languages with key and value mapping. Both JSON and XML are very alike, although XML has been around longer. However, there are some important advantages of using JSON instead of XML as an exchange format.

- JSON is processed more easily because of its simple structure and is therefore faster.
- JSON needs less specialized software to be integrated with programming languages. JavaScript and Python have the JSON notation built in, while other languages only need a small amount of JSON-specific code.
- Due to JSON’s simple structure, less bytes are needed which consumes less bandwidth.

There is a feature in XML that is not supported by JSON and it’s the ability to contain any imaginable data type, texts, numbers, multimedia objects or even active formats like Java applets or ActiveX components. This feature is used with Character Data (CDATA) which tells the XML-parser to ignore everything inside the CDATA element. JSON does not support this, but there are workarounds.\textsuperscript{18}

\textsuperscript{16}http://nodejs.org/download/
\textsuperscript{17}https://npmjs.org/package/opendds
\textsuperscript{18}http://www.json.org/xml.html
4.4 Alternative geospatial servers

WMS provides a HTTP interface for transferring geospatial map images from geospatial databases.\(^\text{19}\) WMS is a standard that is set by the Open Geospatial Consortium, OGC. OGC is an international industry consortium of several hundreds of companies, government agencies and universities collaborating to develop public interface standards.\(^\text{20}\)

Because of WMS’s global acceptance it will be used for serving map images between the map server and the client in the prototype. There are a few open source geospatial server except for MapServer that supports WMS, called Mapnik and GEOserver. Due to the fact that SAAB have previous expertise in MapServer and the focus of this thesis is not on this specific topic, I chose MapServer.

4.5 Frameworks

Developers can do much in Node as it is, but to fastened the developing process and add maintainability to large applications the usage of different frameworks is very helpful. Node supports a lot of frameworks with different niches and community support.\(^\text{21}\)

Express.js is a framework that gives a Model-View-Controller structure of the Node web server. Express.js also has comes with some features to simplify routing HTTP requests.\(^\text{22}\) Backbone.js is used by many larger sites\(^\text{23}\) and gives the developer an API of handling data as models, which can be created, validated, destroyed or saved to the server. Models are listed in views. When a User Interface, UI, action causes a model to change, it will trigger a change-event and all the views displaying the specific model are alerted of the change. Ember.js is another framework and provides both routing, like express.js, and data templates, as backbone.js.\(^\text{24}\)

\(^{19}\)http://www.opengeospatial.org/standards/wms

\(^{20}\)http://www.opengeospatial.org/ogc

\(^{21}\)https://github.com/joyent/node/wiki/modules#wiki-web-frameworks-full

\(^{22}\)http://expressjs.com/

\(^{23}\)http://backbonejs.org/

\(^{24}\)http://emberjs.com/
Chapter 5

Prototype

5.1 TDLH

Tactical DataLink Handler handles interactions with the communication link called Link 16 in Compact C2, enabling communication with other systems over the same link. This module has been modified in order to provide this thesis project with system tracks in a simple way and not altering the original implementation.

5.2 Structure

![Diagram of Prototype Structure]

*Figure 5.1: Structure of the prototype*
Following explains the reference numbers in 5.1

1. TDLH releases new system tracks to the web server via an UDP socket. The data being sent is in JSON format, which is minimal and easy to access in JavaScript.

2. The web server is written in Node.js with the framework Express.js and the API Socket.io. Express is used for giving a MVC-structure and also comes with routing features. Socket.io is an API that uses WebSocket, and other transport protocols, but with certain features like heartbeats, timeouts and disconnection support that are not provided by the WebSocket API.

3. The client connects to the web server via HTTP that is later upgraded to a WebSocket connection and is asynchronously updated with new data.

4. MapServer gets the geospatial data from a shapefile, supplied by SAAB. MapCache is used alongside MapServer and serves as a cache for the tile map images, for faster response to the client. The image are sent via WMS to the client and MapCache is used to speed up the requests.

5. The client, web browser, uses the OpenLayers API to provide the map view with icons for the system tracks. At the top there is a menu, completely made in CSS3. When the Free Text Message menu entry is pressed a Free Text Message dialogue box shows, with drag and drop support.

The overall impression of the prototype is that it is pretty responsive on an iPad 4 with Google Chrome as web browser. The map is built by several map tile images, depending on the number of zoom levels. At first load the map shows some blank tiles but they are retrieved much faster on second load. The menu is a bit buggy on the iPad and the drag and drop feature is not optimised for touch screens. Everything works better in a desktop environment, except for the map which works about the same. The menu can be optimized for mobile devices.
Chapter 6

Conclusions

The prototype provides with a lag free and extremely stripped Command & Control System for desktops. It can be viewed on a mobile device, with a responsive map implementation but with some bugs in the menu and drag and drop features. It is a prototype and with more work to optimize the prototype for mobile devices, the user experience will increase.

Node is an excellent choice for building a Command & Control System. It is a JavaScript server side environment with event-driven and asynchronous I/O for building web servers. Supports WebSocket, DDS and can be developed on a variety of platforms, including Linux. Node is built up by different modules, each with a specific set of functions, making it more modular. Node is very robust and scalable due its asynchronous I/O and event-driven design, leading to low memory consumption but yet able to handle a large amount of connections. There are many JavaScript API’s and Frameworks for example Express.js, Socket.io, Angular.js, to increase productivity and a more structured development. MapServer can be used to serve the client with geospatial data in order to generate a world map.

HTML5 comes with a large set of new features for web applications. WebWorkers enabling more complex computing without interfering with the user experience, WebSockets to enable real-time data exchange between server and client, WebGL to provide with complex 3D rendering, CSS3 for improved design and layout and WebStorage to store data locally and reduce bandwith consumption.

HTML5 can be used as a HMI in a Command & Control System and should perhaps start with a smaller set of features and test thoroughly. To actually implement all features in the Java version into a HTML5 version would take a considerable amount of
time. Instead of replacing everything, a proposal would be to use it as a complement to the already existing HMI with limited functionality.
Chapter 7

Future Work

This thesis brings up what tools to use in order to create a Command & Control System in HTML5, along with the development of a prototype. The prototype only shows a fraction of features compared to the existing system in Java. If it is decided to take the prototype development further, some interesting knowledge would be how long would it take to implement it. The time amount is relative to what features should be considered to be implemented, but is something worth looking into.

The development of the prototype was mainly focused on a desktop version leading to some bugs on the mobile platforms. Further development and optimization should be considered if mobile devices should be a part of the end product.

OpenLayers 2 was used in the prototype but OpenLayers 3 is on its way with new interesting features, such as support for Cesium, a spinning 3D globe API, and also WebGL. OpenLayers 3 will be faster, prettier, with improved API and more\(^1\) that make it worth updating to when it is released.

\(^1\)http://openlayers.org/
Bibliography


Chapter 7. Future Work


