



TAMPEREEN TEKNILLINEN YLIOPISTO  
TAMPERE UNIVERSITY OF TECHNOLOGY

KASHIF IHSAN

## **DEVELOPMENT OF USER INTERFACE FOR THE MANAGEMENT SERVER OF HTML5 BASED MOBILE AGENT FRAMEWORK**

Master of Science Thesis

Examiner: Prof. Kari Systä  
Examiner and topic approved on  
March 1<sup>st</sup> 2017



## ABSTRACT

**KASHIF IHSAN:** TUT Thesis Template

Tampere University of technology

Master of Science Thesis, 53 pages

22<sup>th</sup> May 2017

Master's Degree Programme in Information Technology

Major: Pervasive Computing

Examiner: Professor Kari Systä

Keywords: HTML, Mobile Agents, Management Server, User Interface

The emergence of World Wide Web as a widely used content-sharing environment and a rich software platform has revolutionized the life style of people across the globe. The Web is an archetypal commodity used by millions of people on daily basis, for a variety of online services that range from photos, music, videos, and online games to online shopping, online banking, e-marketing, e-communication, online business, etc. A web browser is the most frequently used application to access the above-mentioned services. The appearance of HTML5 has enabled more and more applications to run in the web browsers. Latest research in the field of Web and computer networks has given a rapid upsurge in the usage of web-based services for data storage and information exchange. This has resulted in challenges to develop efficient web-based systems, which can handle huge amount of information flow. One of the solutions to these challenges is *mobile agents*. A mobile agent is a software program that is able to migrate from one host to another to continue its execution.

This thesis presents the development of a User Interface for the management server of an HTML5-based mobile agent platform. This platform was developed in 2012 at Tampere University of Technology and its second iteration was completed in 2013 at the same campus. The management server of the framework was without a user interface and could to be managed only from the command line. In this thesis, a web-browser based User Interface for the management server of the framework is developed. The management server exposes a Web API to manage the mobile agents and the agent servers through an HTTP interface. So, the API is leveraged by using JavaScript to make Ajax calls. A full fledged User Interface has been developed by using Web technologies that includes the JavaScript, JQuery, ReactJS, and Bootstrap in particular.

The developed User Interface is able to display all the running servers and the mobile agents. There are two views of the UI: AgentServers view and Agents view. In the AgentServers view, all the agents are displayed under their respective servers. In the Agents view, all the agents are displayed, irrespective of whether they are running in a server or in a browser. The UI also helps in moving the agents from one server to another and deleting the agents in graphical mode.

## **PREFACE**

In this thesis, a User Interface has been developed for the Management Server of the HTML5 based Mobile Agent Framework. The framework was originally developed at the Department of Pervasive Computing, Tampere University of Technology.

I would like to start by offering gratitude to my supervisor Professor Kari Systä for providing me an opportunity to work on this framework. I thank him for his consistent guidance, motivation and technical help, throughout my thesis work. I have been doing a part time job, which somehow affected my pace of working on this thesis. However, Professor Kari showed great patience and allowed me with flexible schedule.

I would like to extend my gratitude to all my friends in Tampere, who have made my stay here; easy and memorable. I am also thankful to Finland for giving me a prospect to experience a different life outside my home country.

Last, but not the least, I am eternally indebted to my parents for their unconditional love and boundless support in my life.

Kashif Ihsan

Tampere, 22nd May 2017

# CONTENTS

1.	INTRODUCTION .....	1
1.1	Background .....	1
1.2	Objectives of This Thesis .....	2
2.	WEB AS AN APPLICATION PLATFORM .....	4
2.1	Background .....	4
2.2	Evolution of Web .....	7
2.2.1	Web as a Document Environment .....	7
2.2.2	Web as an Application Environment .....	7
2.2.3	Web as The Application Environment.....	8
2.3	Ecosystem of HTML5 Web Applications.....	8
2.3.1	Client-side Features.....	9
2.3.2	Server-side Features .....	10
3.	MOBILE AGENTS.....	11
3.1	Background .....	11
3.2	Mobile Agent.....	11
3.3	Characteristics of a Mobile Agent.....	12
3.3.1	Autonomy.....	12
3.3.2	Mobility.....	12
3.3.3	Communication.....	12
3.3.4	Reactivity .....	12
3.3.5	Proactivity .....	12
3.4	Benefits of Using Mobile Agents.....	12
3.4.1	Reduction of the Network Load.....	13
3.4.2	Asynchronous & Offline Execution.....	13
3.4.3	Robustness & Fault Tolerance .....	13
3.4.4	Scalability.....	13
3.4.5	Task Delegation .....	13
3.5	Reference Models for Mobile Agent Systems .....	14
3.5.1	Agent Model .....	14
3.5.2	Life Cycle Model .....	14
3.5.3	Computational Model .....	15
3.5.4	Navigation Model .....	15
3.5.5	Communication Model .....	15
3.5.6	Security Model.....	15
3.6	Existing Mobile Agent Platforms.....	16
3.6.1	Concordia.....	16
3.6.2	Grasshopper .....	17
3.6.3	Java Agent Development Framework (JADE) .....	17

3.6.4	Telescript.....	18
3.6.5	Aglets .....	18
3.6.6	Mole .....	19
3.6.7	Voyager.....	19
3.7	Critique of Mobile Agent Systems.....	19
4.	HTML5 BASED MOBILE AGENT FRAMEWORK .....	21
4.1	Overview of The Framework .....	21
4.2	Models in HTML5 Agent.....	22
4.3	Architecture of the Framework .....	24
4.3.1	Configuration of HTML5 Agent.....	24
4.3.2	Generic Agent .....	25
4.3.3	Application Agent .....	27
4.3.4	Agent-to-Agent Communication.....	28
4.3.5	Agent Server .....	29
5.	DEVELOPMENT OF BROWSER-BASED USER INTERFACE FOR MANAGEMENT SERVER .....	31
5.1	Management Server and its API.....	31
5.2	The Management Server API of Our Agent Framework .....	31
5.2.1	Agent Interface.....	32
5.2.2	Management Interface.....	33
5.2.3	AgentServers Interface.....	34
5.3	Technologies Used in the UI Development .....	34
5.3.1	ReactJS.....	34
5.3.2	jQuery.....	35
5.3.3	Bootstrap .....	35
5.3.4	Babel .....	35
5.3.5	Noty.....	35
5.3.6	Sortable .....	35
5.4	Management Server User Interface & React Components .....	35
5.4.1	App.....	36
5.4.2	ServerList.....	37
5.4.3	AgentList.....	37
5.4.4	AgentListSmall .....	37
5.5	Features of the Management Server User Interface .....	39
5.5.1	Displaying Agents & Servers.....	39
5.5.2	Displaying Agent's Information .....	39
5.5.3	Migration of An Agent Between Servers.....	40
5.5.4	Deletion of An Agent From Its Server.....	41
5.6	Screenshots of The Management Server User Interface .....	42
6.	EVALUATION AND CONCLUSION .....	49
6.1	Evaluation of the Current Framework.....	49

6.1.1	Advantages of the Current Framework vs Other Agent Platforms	49
6.1.2	Improvements Needed in the Current Implementation.....	49
6.2	Evaluation of the User Interface Developed in This Thesis.....	50
6.2.1	Evaluation Against the Core Objectives of This Thesis .....	50
6.2.2	Heuristic Evaluation of the Developed User Interface .....	50
6.3	Conclusion.....	52
6.4	Potential Future Improvements .....	53
REFERENCES.....		54

## LIST OF SYMBOLS AND ABBREVIATIONS

API	Application Programming Interface
AJAX	Asynchronous JavaScript and XML
CSS	Cascading Style Sheets
DOM	Document Object Model
GPS	Global Positioning System
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
REST	Representation State Transfer
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WebGL	Web Graphics Library
XML	Extensible Markup Language



# 1. INTRODUCTION

## 1.1 Background

Since its invention in early 1990s as an information management platform, the World Wide Web has evolved into a popular content sharing environment and rich software podium. It has become a quintessential commodity for various online services such as online banking, e-marketing, e-newspapers, online shopping, e-communication, etc. The Web has transformed our lives in different ways. Today, everyday objects and services such as photos, music, videos and documents are available on the Web [1]. People are now using it to store their personal data online and retrieve it from anywhere in the world. It is widely used as a communication medium through emails, chat messengers and social media platforms.

Recent times have seen exciting developments in the fields of computer networking and the World Wide Web. There is a swift rise in the usage of web-based services for data storage and information exchange. This has given challenges to the web developers to come up with efficient ways of handling huge amount of data and information flow. One of the possible ways to address these challenges consists of *mobile agents*. A mobile agent is a computer software which is able to move (migrate) from one computer to another and continue its execution in new destination [2]. A mobile agent framework allows different agents to move between different hosts and communicate with other agents. HTTP is a well-known, widely accepted mechanism that contains all the necessary primitives to support a mobile agent framework.

HTML is the core markup language in World Wide Web. It was developed by Tim Berners-Lee in 1990 and has gone under massive evolution since then. The latest version of it is the HTML5 that provides new features necessary for modern web applications [3]. It extends the applicability of web technologies to develop RIAs (Rich Internet Applications) that run on a browser. RIAs combine the lightweight distribution architecture with the interactivity and computation power of desktop applications [4]. The RIAs do not require local installation on client side. For the end user, a web application does not differ from a conventional desktop application that is installed locally. Web applications avoid the liability of deployment in every client machine and can be accessed from anywhere. Also, the web applications are platform-independent, so they are readily available to use from different platforms. The web applications are very easy to manage and update as compared to the traditional desktop applications.

## 1.2 Objectives of This Thesis

This thesis presents the development of User Interface for the Management Server of HTML5 based mobile agent framework. This framework was developed by Prof. Kari Systä at Tampere University of Technology in spring 2012. The second iteration of this framework was done by Laura Järvenpää as her Master's Thesis in 2013 [5]. The primary objective of this thesis is the development of a web browser-based User Interface for the management server of our framework. Currently, all functionalities of the management server can only be implemented using command line. These functionalities consist of displaying the information of all agents and agent servers running in the framework, information about a specific agent, moving an agent from one host to another host and deletion of an agent.

The UI is expected to graphically display the list of mobile agents and agent servers on the webpage of the management server. The UI should also enable a user to move an agent from one server to another and to delete an agent, through a graphical interface. Besides the development of the UI, a personal objective of this thesis was to get a hands-on experience to use some of the latest Web technologies.

Prior to the development of the UI for the management server of the framework, the whole framework was carefully studied. The framework has been developed using Node.js which is server-side platform for building fast and scalable applications [6]. It is an open source, platform-independent runtime environment to develop server-side applications. The Node.js model embraces an asynchronous, event-driven, non-blocking I/O approach, which makes it an efficient and lightweight platform to create applications for distributed networks.

The management server of this framework exposes some Application Programming Interfaces (APIs) which are used to manage the mobile agents and the agent servers through an HTTP interface. The APIs are leveraged by using JavaScript to make Ajax calls. The agent servers register themselves with the management server. This UI has been developed by using several components. These components include ReactJS, JQuery, Bootstrap, Babel, Noty and Sortable. *ReactJS* is a JavaScript library that is used to create interactive JavaScript applications. It offers features such as state management which let us to change the parts of UI by simply updating the state-model of the application. *Babel* is a transpiler that is used to convert the ReactJS templates (JSX) to pure JavaScript. *Jquery* is a no-brainer for JavaScript based projects. We are using it for DOM manipulation and plugins that are required in this UI development. *Bootstrap* is a user-interface library, which provides organized Cascading Style Sheets (CSS) components that are interpreted by the browser. *Noty* is a notification library, which is used to create alerts, error, warnings, success and confirmation message dialogs in the UI.

The rest of the thesis is organized as following:

Chapter 2 describes Web as an application platform, how the Web was evolved and some recent developments in Web. Chapter 3 describes Mobile agents, their characteristics and some existing mobile agent platforms. Chapter 4 describes the Framework of HTML5 mobile agents, which is under consideration in this thesis. It comprises the architecture, a generic agent, agent-to-agent communication, an agent server, etc. Chapter 5 describes the development of Web Browser based User Interface for the Management Server of the HTML5 based Mobile Agent Framework. Chapter 6 discusses evaluation and conclusion.

## 2. WEB AS AN APPLICATION PLATFORM

### 2.1 Background

In this chapter, we will discuss the evolution of web as a brief history. In addition, we will discuss some common web technologies that are relevant to our thesis work.

To start with, let us understand one thing very clearly that World Wide Web is not the same thing as the Internet. The Internet is a huge collection of physical communications links, made operable by a standard set of protocols, that interconnect computer all over the world [7]. In reality, the Internet is regarded as a network of networks, which connects millions of computer devices worldwide. It all started as ARPAnet, a research project that was started by U.S. Defense Agency in 1970s, with an objective to connect government and industry researchers who worked on military projects. Until 1991, the commercial use of Internet was barred due to its funding by the government. However, different companies started to add “commercial links” to the Internet for the sake of its own marketing purposes. Eventually, the Internet completed a switch from government-funded entity to an all-commercial infrastructure by 1995 [8], [9].

On the other hand, the World Wide Web or simply Web is a way to access information through the medium of the Internet. It is regarded as one of the services that is provided by the Internet [7], [9]. The Web is a massive collection of interconnected Documents or Contents that helps the communication between different devices (computers). The core concept of invention of the Web was to have a common information space for communication and sharing information. However, it eventually went on to revolutionize the ways in which we connect to other people, share news and information and store personal data online. The Web has evolved from a simple document sharing system to a massively popular, general purpose application and content distribution environment [10].

Furthermore, Web was meant to be well documented, open and free available for anyone to use. Tim Berners-Lee, the creator of web, realized the need of an independent organization that can make universal standards for Web. It led to foundation of World Wide Web Consortium (W3C) in 1994 as an organization for the development of rules and guidelines for Web [11].

Currently, most of the web applications heavily rely on technologies that are fundamental components of the web browser [10]. Some of these components comprise Hyper Text Markup Language (HTML), JavaScript, Cascading Style Sheets (CSS), JavaScript Object Notation (JSON), Document Object Model (DOM), Asynchronous JavaScript & XML

(Ajax), Hyper Text Transfer protocol (HTTP), Extensive Markup Language (XML), Uniform Resource Identifier (URI) and Uniform Resource Locator (URL). All the above components are briefly described below:

### **Hyper Text Markup Language (HTML)**

It is a standard markup language devised for creating Web pages [12]. The basic units of HTML are the elements, which define a certain way to present the data contained in a webpage.

### **JavaScript**

JavaScript is a programming language used to make web pages interactive. It is a scripting language that is interpreted by the browser engine when the web page is loaded. Syntactically, it resembles C++ and Java with object-oriented capabilities [13]. It was originally developed by Netscape. The JavaScript code is usually written in HTML page but it may also be written in a separate JavaScript file. Later on, source path of external JavaScript files should be added in the HTML page. JavaScript is also used to implement the event handlers that are invoked, when a user interact with a webpage.

### **Cascading Style Sheets (CSS)**

It is a style sheet language used to illustrate the presentation of a web page that is written in a markup language. CSS describes how HTML elements are displayed on screen and can control the layout of different web pages simultaneously [14]. The latest version of the Cascading Style Sheets is the CSS3, which offers additional features like animations, round corners, different text effects, rotations, shadows, etc. All the latest web browsers are integrated with support for CSS3.

### **JavaScript Object Notation (JSON)**

It is a lightweight plain-text format used for the data interchange [15]. It is used to serialize different data structures into strings [16]. The data in JSON is in name-value pairs, separated by commas and is meant to be easily read by both human and computers.

### **Document Object Model (DOM)**

It is a platform- and language-independent mode of representing a collection of objects which comprises a web page [17]. It defines the logical structure of web document that is similar to a tree, hence enabling a developer to access, modify, navigate and delete different elements of that document. DOM is a programming interface for HTML and XML documents which provides an organized logical representation of the documents and how they are accessed and manipulated [18]. It is used to dynamically access and update the style, structure and content of HTML and XML. When a browser loads a

webpage, it creates a DOM of the page. It is like a tree of documents that is rendered by a browser.

### **Asynchronous JavaScript and XML (AJAX)**

It is a collection of web development techniques used to asynchronously update web pages through exchange of small amount of data with server [19]. The basic theme of AJAX approach is to use JavaScript to send HTTP requests to the web server and update the page contents dynamically, without reloading the whole page. With Ajax application, the browser loads an Ajax engine that is responsible for rendering the interface to the user and communicating with the server. It allows the user's interaction with the application to occur asynchronously i.e. independent of the communication with the server. AJAX is typically used to access remote services and is very successful because of its non-blocking and asynchronous functioning.

### **HyperText Transfer Protocol (HTTP)**

It is an application level protocol for distributed, hypermedia, collaborative information systems [20]. It is an object-oriented, stateless protocol that is primarily used for negotiating data type and representation, so that to allow systems to be built independently of the data being transferred. It is a connection-less protocol, which means that whenever a request is sent to web server, the connection between client and server is disconnected. It requires a new connection between client and server for each request.

### **Extensible Markup Language (XML)**

It is a simple, flexible, software- and hardware-independent text-format tool that is used for giving structure, storing and transferring data [21]. Developers can create their own tags using XML, as it was created by World Wide Web Consortium (W3C) to overcome the limitations of HTML.

### **Uniform Resource Identifier (URI)**

It is a sequence of characters used to identify a logical or physical resource [22]. The uniform means that URI can be used to identify different resources, irrespective of their types. Resource is anything that possesses an identity, like an image, a service or some other resources. Identifier means a pointer object which can refer to a resource [23].

### **Uniform Resource Locator (URL)**

It is a subset of URI that recognizes different resources in the Web, i.e. their locations. The URL contains the name of the resource on the Web and the name of the protocol to be used to access that resource [24].

## 2.2 Evolution of Web

This thesis will discuss the evolution of Web from a simple document-sharing platform to an application platform in three different periods as discussed in [1]. In the first era – the Web as a document environment, the Web was limited to document sharing environment. In the second era – the Web as an application environment, capabilities of Web started to rise with the commence and competition of different technologies. In the third era that is unfolding currently, it is argued that the transition towards the Web as *the* application environment will fundamentally affect the landscape of the software industry [1].

### 2.2.1 Web as a Document Environment

As the Web began its life in early 1990s, the web pages were truly pages, which contained page-structured documents [1]. Those documents were non-interactive and without any animations, predominantly consisted of text. A new webpage would open from the Web server every time, whenever a user clicked on the hyperlinks in the Web page. Few pages were presented as *forms* with simple text fields and basic interaction points e.g. buttons.

In mid 1990s, Dynamic HTML (DHTML) was introduced as a combination of HTML, CSS, JavaScript and DOM. It enabled to create more interactive web pages with advanced graphical contents [25]. Towards late 1990s, the web pages were becoming increasingly interactive which caught the attention of different companies. They understood to use web pages for advertising their products and services, only to pave way to commercialize the Web. Web plugins such Flash and Real Player were introduced to create webpages with interactive multimedia. The navigation within different web pages was no more limited to the hyperlinks.

Furthermore, the development of web editors enabled the creation of webpages without having excessive knowledge about HTML, which increased the user created contents in Web. This gave rise to Web 2.0, which does not have a hard boundary to be defined [26]. It has emerged, since then, as the standard for cutting-edge web development. Web 2.0 applications and services make possible, even more dynamic interactions between clients and servers, more enchanting webpage displays and more direct, interactive user-to-user communication [27].

### 2.2.2 Web as an Application Environment

The introduction of built-in support in web browser for Ajax (Asynchronous JavaScript and XML) recognized the initial signs of a web browser being used an application platform. The idea behind Ajax was to make use of asynchronous exchange of data between the server and the client, so that the UI (User Interface) updates are separated from the network updates. Traditionally, the information retrieval in web applications involved of

loading the whole web page when a user would click a hyperlink or submit data to the server [28]. As the server returns a new page every time the user submits some input, the conventional web applications tends to run slowly. On the other hand, an Ajax web page can load and transmit data to the web server dynamically, while the web page does not need to be reloaded as a whole.

The increased use of Ajax-based web applications, however, brought forth the shortcomings of web browser as an application platform. The lack of APIs (Application Programming Interfaces) in web browser to access the proficiencies of underlying devices made it very hard to write complex applications that are portable among different web browsers. Hence, it paved way for the rise of RIAs (Rich Internet Applications) which implanted the use of desktop programming environment into a Web environment by using Web plugins e.g. Adobe Flash, Microsoft Silverlight etc.

Mobile software developers also found Web as an exciting prospect, using Web Widgets to replace the web browser technologies. The Web Widgets are small applications with specific functionality that can be installed inside a web page.

### **2.2.3 Web as The Application Environment**

Since its inception, the HTML has been constantly worked upon by World Wide Web Consortium (W3C) and in the beginning of 2010, the latest version of it “HTML5” was published. The emergence of HTML5 facilitated the development of RIAs based on web browser technologies. New characteristics and different APIs has been added into HTML specification to meet the ever-demanding flexibility of web applications.

Another technology by the name WebGL appeared to meet the insufficiencies of graphical representation of Web [5]. It is a cross platform standard of Web for 3D hardware accelerated graphics. It helps in rendering 3D graphics natively in the browser, without support of plugins.

## **2.3 Ecosystem of HTML5 Web Applications**

A software (application) ecosystem is defined as “a set of software solutions that enable, support and automate the activities and transactions by the actors in the associated social or business ecosystem and the organizations that provide these solutions” [29]. The Web 2.0 is considered as a kind of application ecosystem that qualifies to create latest web applications.

The HTML5 is particularly suitable to create websites and applications for the client-side. The separation of styling and functionality has made the job easy for the developers. Conventionally, the web applications have been primarily the user interfaces for the applications that run on certain server. However, HTML5 tends to allow the creation of



independent client-side applications [30]. While most of the web applications run on a server, HTML5 based web applications are also intended to run in offline mode. Once installed, they do not essentially depend on any server.

Following are described few features provided by HTML5 technologies, in the context of client-side and server-side:

### 2.3.1 Client-side Features

In this sub-section, we discuss only those client side features of HTML5 that are in the scope of this thesis, hence the following features are not the complete list of HTML5 features.

**Canvas:** It is one of HTML5 significant feature that allows developers to create graphics, animations and videos on web pages. It helps to render the graphics locally, which result in faster rendering of heavy-graphical web pages.

**Geo-Location Services:** are offered through an API with support for mobile browsers. It is done by enabling interaction with GPS based data. Hence, there is no need to use any external library to find GPS-based location.

**Media Elements:** A new feature in HTML5 is the built-in support for audio & video playback in the web browser. Preceding versions of HTML did not have multimedia tags. Hence the customary browsers depend on plugins like Adobe Flash, Microsoft Silverlight and Quicktime to play the multimedia content; however, mobile web browsers lack those plugins. HTML5 gives solution to this problem by providing `<audio>` and `<video>` tags to be used in web pages.

**Scalable Vector Graphics (SVG):** SVG specification is an open standard that is developed by the World Wide Web Consortium. Before the HTML5, the web browsers would require additional plugins for 3D-graphics i.e. Flash, Quicktime, etc. However, HTML5 provides built-in scalable vector graphics that are used to draw graphics or animations. The HTML elements `<svg>` is a container for graphics. It has many methods for drawing different boxes, circles, paths, images, etc.

**Semantic Elements:** The introduction of new semantic elements e.g. `<nav>`, `<article>`, `<section>`, etc. has been supportive of describing the structure of a web page more precisely. Semantic HTML5 elements bring meaning to a webpage, rather than just presentation. A semantic element has a key meaning that is conveyed to both the developer and the browser. For example, `<span>` and `<div>` are the non-semantic elements of HTML, which do not reveal anything about their contents. On the other hand, semantic elements of HTML i.e. `<article>`, `<form>`, `<table>`, `<code>` `<aside>`, etc. not only present but

also define their content. The addition of semantic elements in a document provides additional information about that document, which is helpful in communication.

**Working Offline:** HTML5 enables web applications to keep running and executing, even without a network connection. It allows web applications to store data locally by providing an application cache, so as to work like a desktop application, even without an internet connection. Conventional web browsers are only able to cache the visited webpages and is usually at the disposal of user and web-browser to clear the cache for new webpages. In HTML5, a developer can define the resources, which are required to be cached.

**Drag & Drop and Local Storage:** The drag and drop functionality of HTML5 in web applications allows a smooth integration of those applications with users' devices. HTML5 also supports offline caching which is important for storing the data for future use, instead of sending it to server.

**Web Worker API:** HTML5 provides a JavaScript Web Worker API that runs in the background and does not affect the performance of the webpage. Prior to HTML5, the webpages would need more time to load, because of the rigorous JavaScript code that would run alongside the browser in the same thread. However, the Web Worker API of HTML5 runs the JavaScript code in a separate thread, therefore enabling a seamless user interaction with the front-end of the webpage. Hence, HTML5 is said to be a non-blocking technology.

### 2.3.2 Server-side Features

There are many server-side features provided by different technologies for development of web applications. However, the HTML5 based framework focused in this thesis, mainly include Node.js.

According to the official documentation of Node.js, “it is a JavaScript runtime built on Chrome’s V8 JavaScript engine. It uses a non-blocking I/O, event-driven model that makes it efficient and lightweight”[31]. Node.js is a server-side technology that uses the JavaScript to build server-side web applications. Traditional programming performs I/O operations in the same style as a local function call does. It needs a user to wait (block) for a certain task to be completed before continuing to do other tasks. A better alternative to this style of executing programs is the event-driven programming, which is also known as asynchronous programming. It is one of the significant feature of Node.js, which means that a process doing I/O operations will not block [32].

## 3. MOBILE AGENTS

### 3.1 Background

The outburst of advancements in the fields of networking and software engineering has resulted in introducing different technologies, devices, operating systems and platforms that possess diverse architectures and protocols. The ever-rising popularity of the Internet has changed the scope of using computers. The trends are shifting from traditional network communication to distributed network communication. People are using a number of interconnected devices to access different applications and services from Internet [33]. There are challenges in organizing information and simplifying its efficient retrieval across such heterogeneous networks. Also from network point of view, the developers have to be considerate of bandwidth management.

Many researchers believe [34] that the mobile agent paradigm could be one with solutions to the above-mentioned challenges. Mobile agent paradigm is an emerging model that is specifically targeted by mobile computing, Internet applications, network management and telecommunication systems. Mobile agent based systems differ from the orthodox client-server systems because there is no explicit distinction between a server and a client [35].

### 3.2 Mobile Agent

A software agent can be defined as a computational object that acts on behalf of a user or another object, is autonomous and is able to perceive and learn from its environment [35]. A *mobile agent* is a software agent (program) or code that can migrate autonomously from one host to another in a heterogeneous network [35], [36]. The internal state of the program is saved, then migrated to a new host and restored which means that the program starts its execution from where it left.

In traditional network applications, the clients require data and services from the servers. Conversely, in mobile agent network applications, a mobile agent migrate through different hosts and interact with other agents (resources) and typically returns to its original location to deliver the collected data (completion of certain task) to the user.

### **3.3 Characteristics of a Mobile Agent**

The characterization of what exactly establishes a mobile agent has been hugely debated in the research communities [37] and it continues, however, there is a communal understanding that a software object must exercise certain characteristics to qualify as a mobile agent. Following are listed few of the main characteristics of a mobile agent:

#### **3.3.1 Autonomy**

A mobile agent should be able to operate in accordance to a pre-plan that is created with user given task. An agent should be able to make its own decisions and should not consult its owner (e.g. user) for every step [37][38].

#### **3.3.2 Mobility**

It is a characteristic of a mobile agent, which enables its migration from one host to another host. An agent should be able to suspend its execution, migrate itself to another host (location) and restarts itself from where it halted.

#### **3.3.3 Communication**

A mobile agent should be able to communicate with the user, other agents and its environment of execution.

#### **3.3.4 Reactivity**

It refers to the capability of mobile agents to perceive its environment and react to events happening in that environment. A mobile agent should be able to accordingly adapt its functionality to the varying execution environment [38].

#### **3.3.5 Proactivity**

It is a distinctive characteristic of a mobile agent, which depicts that it not only reacts to changes in the environment, but also able to take initiative and dynamically plan [37]. It is the capability to initiate and coordinate new interactions with other agents.

### **3.4 Benefits of Using Mobile Agents**

Emergence of the paradigm using mobile agent based communication has also evoked the comparative studies against other available paradigms e.g. client-server paradigm. Mobile agents as viewed as a tool for the implementation of distributed applications with aim to improve the performance of existing distributed applications. Following are listed the few benefits of using mobile agents as discussed in [39], [40]:

### **3.4.1 Reduction of the Network Load**

When the application has to deal with a huge volume of data, it usually requires a high bandwidth on network. The concept of mobile agents fits the best in this scenario. The fundamental idea is amazing, albeit simple: the code is moved near to the data instead of bringing the data to the code. It helps in reducing the number of interactions and amount of data that is sent through the network.

### **3.4.2 Asynchronous & Offline Execution**

Mobile agent based application can send certain tasks to be executed on another host where the resources are available. The application can go offline after dispatching the task in a mobile agent while the mobile agent continues to execute its task asynchronously on its own. Later on, the application can connect back to the network and fetch the mobile agent with results of its execution.

### **3.4.3 Robustness & Fault Tolerance**

Web applications using traditional client-server approach are vastly dependent on the availability of a network. If a network breaks down, the application usually fails and needs to be resumed from very start. However, mobile agents exhibit higher robustness in such scenarios. If the network is unavailable, the mobile agent based applications are still able to continue their execution until the network is online. The mobile agent platform should also provide support for fault tolerance so that the applications are more reliable. This way, the mobile agents have the ability to perceive their environment of execution and react autonomously to varying circumstances and failures in the network.

### **3.4.4 Scalability**

Mobile agents allow the distribution of functionalities throughout a network and tend to decentralize the executing tasks among several hosts in a network. In comparison to a client-server paradigm, where most of the execution process is completed in a centralized server, the mobile agent paradigm delivers a more scalable solution. Scalability of mobile agent applications delivers better results in the environments where conventional network approaches would produce bottlenecks.

### **3.4.5 Task Delegation**

Mobile agents help to avoid the information overload, so that a user can focus on other things while the mobile agents perform different tasks assigned by the user.

### 3.5 Reference Models for Mobile Agent Systems

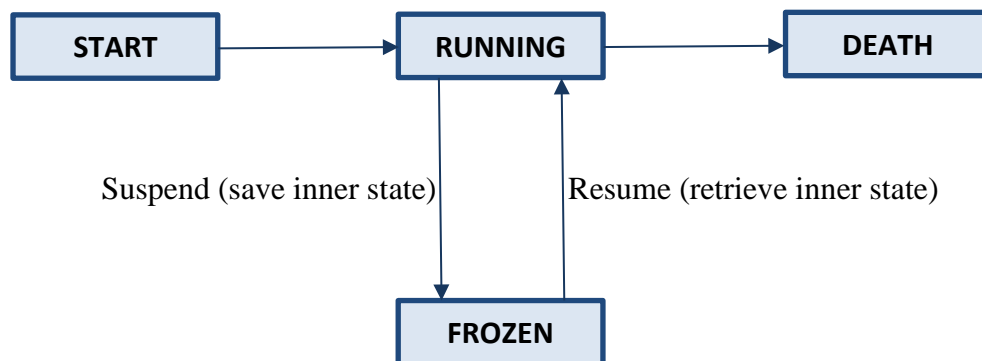
In this section, we discuss different models in a mobile agent system. The main reference material is [41], which debates an agent model, a life-cycle model, a computational model, a navigation model and a security model for a mobile-agent system.

#### 3.5.1 Agent Model

It defines the internal assembly of the intelligent part of a mobile agent. It explains the autonomous nature and cooperative properties of an agent. It also postulates the reactivity and proactivity features of a mobile agent.

#### 3.5.2 Life Cycle Model

It defines the different execution states of a mobile agent along-with the events, which cause the shift from one state to another. The only life cycle model that is relevant to mobile-agent framework in this thesis is the *persistent process based life cycle*. A generic structure of the persistent process based life cycle is described in the following figure.



**Figure 3.1** *Persistent Process Based Life Cycle* [41]

The *start* state indicates a mobile agent being initialized for the first time and then moved to the *running* state where it continues to execute. When the execution is completed, the mobile agent goes to *death* state and perishes. During the *running* state, if the mobile agent needs to migrate to another host, the internal state is saved and the agent enters the *frozen* state. When the agent is migrated to another host, the internal state is resumed and it enters again into the *running* state to continue its execution.

### 3.5.3 Computational Model

The computational model describes how a mobile agent is executed during the *running* state. The computation occurs in an environment that is facilitated by some processor or more abstract processor e.g. one found in Java Virtual Machine. The computational model is relevant to almost all other models because it specifies an instruction set for a mobile agent, which determines the computational capacities of an agent.

### 3.5.4 Navigation Model

This model is all about the *mobility* of a mobile agent. It holds the information about all potential destinations (hosts) of a mobile agent. It also defines the way of transmitting mobile agents through different hosts, how to suspend a mobile agent, saving its inner state, retrieval of inner state and resumption of the execution of a mobile agent. The navigation model of mobile agents is related from discovering of destination hosts to the manner in which these agents are transported. Overall, the following points establish the navigation model for a mobile agent platform [41]:

- All the entities in the mobile agent platform (e.g. hosts, agents, etc.) should have some naming conventions.
- Access to information about a remote environment, where the mobile agents are anticipated to be migrated or to be retrieved from.
- Ability to *freeze* (suspend) a mobile agent, get it ready for migration to another host and its successful migration.
- Ability to receive a *frozen* (suspended) mobile agent from a remote environment and reconstruct it to execution in its destination environment.
- Information about the network topology can be helpful to mobile agents to plan their autonomous migration in the underlying network.

### 3.5.5 Communication Model

A mobile agent would of minimal use if it is not able to communicate with other entities e.g. other mobile agents, different hosts, etc. in the computational environment. This model is used when the mobile agent requires services outside of itself and when other entities in the computational environment need services of the mobile agent.

### 3.5.6 Security Model

The security of mobile agent platforms can be broadly categorized into two main aspects. The first aspect is the protection of hosts from malicious mobile agents. The malicious mobile agents can possibly acquire and alter data on the host or can use resources that are not dedicated to them. The other aspect is the protection of mobile agents against vicious

hosts. In order to run on a host, the mobile agent exposes its state and code to the host, which makes it viable to be changed or killed by the host.

All the models that are briefly described above are closely related to one another. For an instance, the computational model can have effect on the structure of other models. The navigation and communication models highly rely on the security model. The navigation model needs security model to ensure a secure migration of agents from one host to other. The communication model needs security model to prevent mobile agents to be corrupted by the other mobile agents or hosts.

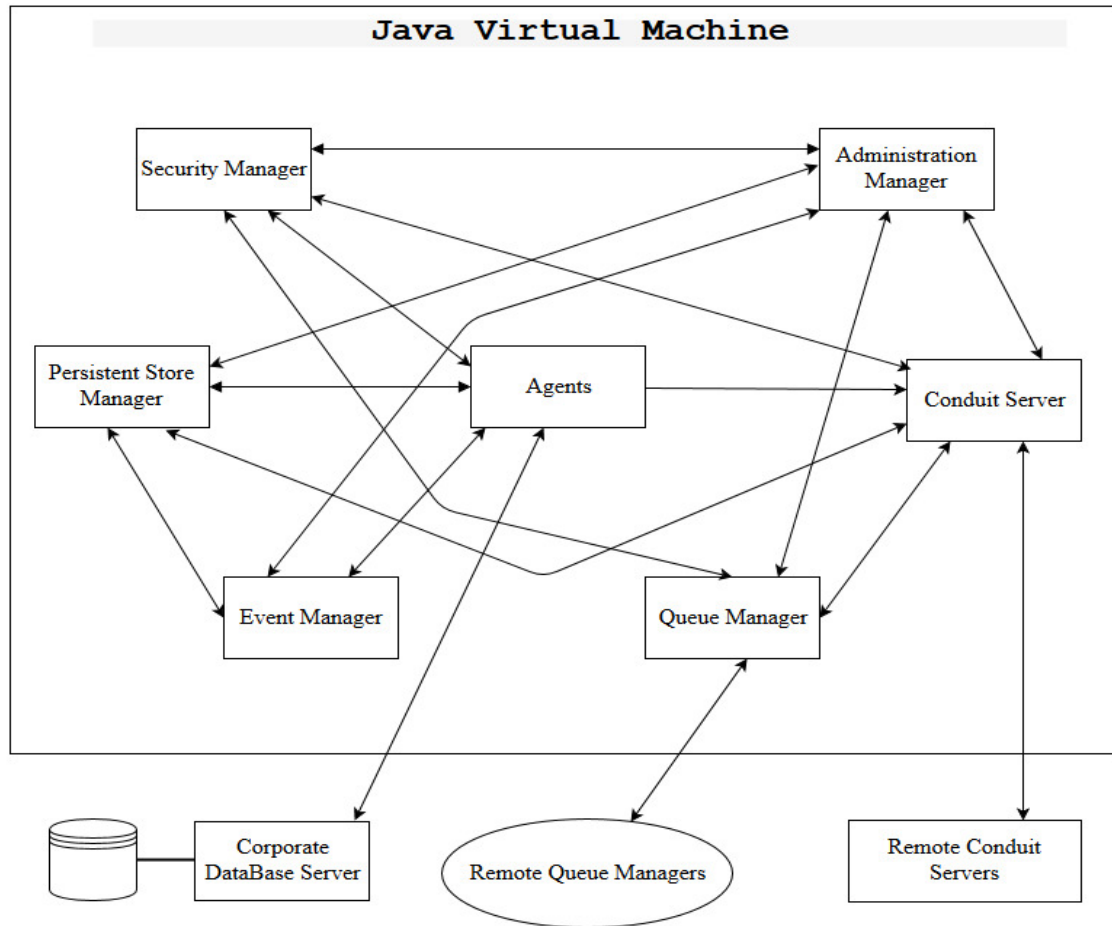
## **3.6 Existing Mobile Agent Platforms**

Since the conception of the idea of mobile-agent based applications, many mobile agent platforms exist. We will discuss only few of the mobile agent platforms below. Some of the platforms do not exist anymore but they were crucial along the timeline of research in this field.

### **3.6.1 Concordia**

The Concordia is a mobile agent platform used for the development of network-efficient mobile-agent applications. It is a Java-based mobile agent platform built on Web but also needs a Java Virtual Machine. In its simplest form, a Concordia system consists of a server, at least one agent and a Java Virtual Machine [42]. The design goal of development of Concordia in Java is to guarantee platform-independence among its agent applications. It provides an ample support for agents' collaboration, reliable communication and security. Concordia offers two forms of agent communication: asynchronous distributed events and agent collaboration [43]. Concordia mobile agents can extend their mobility to a number of local area networks as well as wide area network.





*Figure 3.2 : Concordia System: Data Flow Diagram [43]*

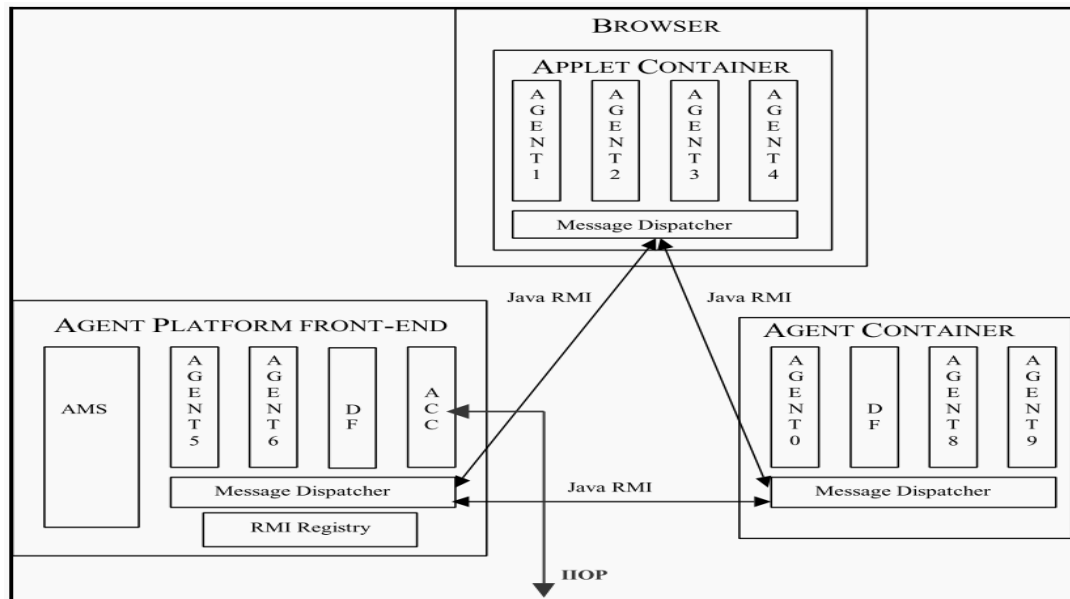
### 3.6.2 Grasshopper

Grasshopper is a Java based distributed mobile-agent platform. It has a user-friendly graphical interface but its overall performance is said to be below expectations [44]. There are two types of agents in Grasshopper, the mobile agents and stationary agents. The Grasshopper system provides the following services [45]: Management service, Security service, Communication service and Registration service. The management service is used to monitor and control the agents. The security service is meant for securing the components of Grasshopper platform. Communication service offers support for communication between agents and distributed regions. The registration service is used to store all the information of the agents. Grasshopper platform is no more available and new versions will not appear in future [46].

### 3.6.3 Java Agent Development Framework (JADE)

JADE is an open-source Java based software framework for the development of mobile agent systems that are meant to manage networked information resources. JADE complies with specifications of FIPA (Foundation for Intelligent Physical Agents) to develop

interoperable multi-agent systems [47]. It also extends its capabilities to be used on such devices, which have limited resources e.g. mobile phones and PDAs. A typical JADE platform consists of Java processes that act as agent containers. These containers provide runtime services needed for hosting and executing mobile agents. JADE development is still in continuation and further enhancements in the implementations have been planned as specified by FIPA [48]. The main disadvantage is that the key element of JADE is not the mobility, rather it emphasizes more on other functionalities that are relevant in the development of multi-agent systems [46].



**Figure 3.3:** Architecture of a JADE Agent Platform [47]

### 3.6.4 Telescript

Telescript is an object-oriented programming language developed for implementing the mobile agent applications in 1991 at General Magic, Inc. Initially, the Telescript mobile agents were used in creating infrastructure for e-marketplace [5] but due to the rise of World Wide Web as a foundation, the Telescript agents were combined with Web, eventually to be used by web browsers. One of the important aspect of Telescript is the integrated support for autonomous process migration. A Telescript agent needs a *ticket*, which contains the address of another place where the agent would move. The *ticket* also holds protocols for communication, defining how an agent will move from one place to another. Telescript also has a mechanism of *permits* for reducing resource consumption [49].

### 3.6.5 Aglets

Aglets is a Java-based mobile agent platform and library that facilitates the development of agent-based applications. It was developed at IBM Tokyo Research Laboratory in 1996 and is sustained by an open source community *sourceforge.net* since 2001. All aglets

agents are derived from an abstract Aglet base class and each agent is an object. An *aglet* is a Java agent, capable of autonomously migrating from one host to another. It is able to migrate to any aglet-enabled host in a network, it is autonomous and reactive because of its capability to react to incoming messages [50].

### 3.6.6 Mole

It is the first Mobile Agent System developed in Java. It first appeared in 1995 and has been constantly improved. Mole offers a stable distributed application environment for the development of mobile agents. When an agent is create in Mole, a unique identifier is created that is used to identify that agent globally. There is a concept of Strong Migration and Weak Migration in Mole [51]. During *Strong Migration*, the entire state (execution state and data) is moved along with its code to a new host. During *Weak Migration*, only the state and data of the agent is moved.

### 3.6.7 Voyager

Developed by *ObjectSpace* in 1997, Voyager is a distributed computing platform for the simplification of management of remote communications of RMI protocols. The mobile agents communicate using conventional remote method invocations. However, its key disadvantage is that Voyager is a commercial product and is not available for free [46].

## 3.7 Critique of Mobile Agent Systems

This section presents a critical analysis of mobile agent systems from different outlooks. Despite various advantages of using mobile agent systems, there are certain challenges and problems that these systems come across. Some of those challenges and problems are discussed in [5][52][53][54] as follows:

- The existing mobile agent systems save network bandwidth and latency, but at the cost of higher loads on service machines. This problem ascends particularly due to the fact that most of agents are written in relatively slow interpreted language e.g. Java, for security and portability reasons.
- It is hard to have a complete certainty that any Internet service will be keen to change from client-server paradigm to a mobile agent paradigm.
- One of the key criticisms on mobile agent systems is their security. Since an agent moves freely and autonomously in a heterogeneous network, there are chances that either the agent could contaminate the server's code or vice versa. Also, many malwares can arrive in disguise of mobile agents.

- Authentication is another issue related to the security critique of mobile agent systems. It is argued that how to be sure of who an agent is what it says it is. How to be convinced of the fact that a mobile agent is not being altered, while arriving after navigating different networks.
- Privacy of the mobile agents is another debatable challenge in this field. It refers to how to ensure that an agent is keeping its privacy and someone else is unable to read the data and the code of that mobile agent.
- Another challenge is the performance issues, which indicates that how to know the effect of increasing number of mobile agents on certain network.
- A well-known problem with self-modifiable scripting languages like JavaScript is that it can modify itself to the extent that it becomes very hard to detect the action of an agent when it is executed. Hence, it disables the hosts to find out if the agent is harmful, beforehand.

## 4. HTML5 BASED MOBILE AGENT FRAMEWORK

In this chapter, we discuss the overall structure of our mobile agent framework, for which we developed the user interface of its management server. Since this framework has been developed by Prof. Kari Systä and later on, its second iteration is done by Laura Järvenpää [5], so our main reference will be that. The research papers [36] [33] [55] are also used as the main references in this chapter. The main objective of HTML5 is to allow the development of complete client-side web applications [55]. HTML5 enables more client-side applications to run in browser and these applications can be installed over network by using the standard Web technologies. Primarily, these applications are network-connected; however, they are also enabled to run in off-line mode to continue executing their tasks.

### 4.1 Overview of The Framework

The underlying framework in this thesis consists of two parts: an HTML5 agent base class and an agent server. The agent refers to an HTML5 application that is capable to run in two different modes: with user interface inside a browser and in headless mode (without user interface) in a server. The state of an agent is preserved, whenever it migrates between a server and a browser. The framework implementation uses *code-on-demand* paradigm [56] for getting the static files of the application when an agent moves in network. The agent server represents the mobile agent environment in server side while the browser represents the mobile agent environments in client side.

An HTML5 agent on this framework is composed of two parts:

- Description of the user interface of an agent in HTML and CSS files
- The executable content and dynamic aspect of the user interface resides in the JavaScript files

The HTML5 is used as an implementation tool in accordance to [55] which gives the following explanations:

- HTML5 technologies have a strong ecosystem and are widely used in practice
- A user can run the agent application in the UI mode in browser, can transfer it to the agent server and pull it back in another perspective. This whole scenario put the user in firm control and the whole platform becomes more user-oriented.

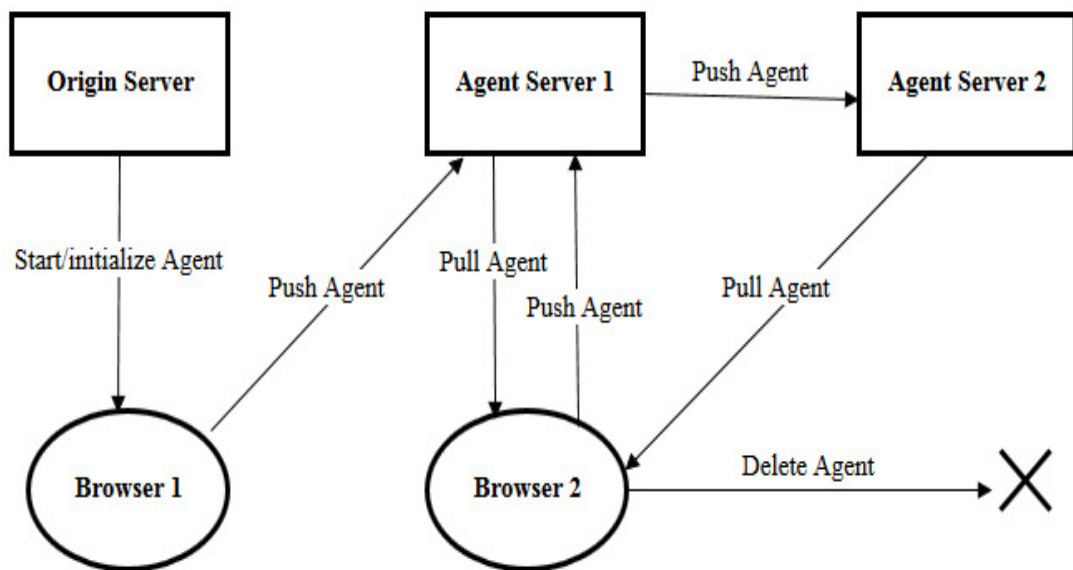
The use of Node.js for implementing the server-side environment for the execution of agents gives a prospect of using JavaScript as the same programming language to develop both client-side and server-side execution environment.

## 4.2 Models in HTML5 Agent

According to latest iteration [5] of our HTML5 based mobile agent framework, only the agent model, life-cycle model and computational model have complete implementations. The communication model, security model and the navigation model are in the initial stages and needs to be further developed to serve the purpose of the platform in even higher degrees.

**Agent Model** consists of the intelligent part of the agent that is capable of making autonomous decisions. It mainly includes the management of the inner state of the mobile agent.

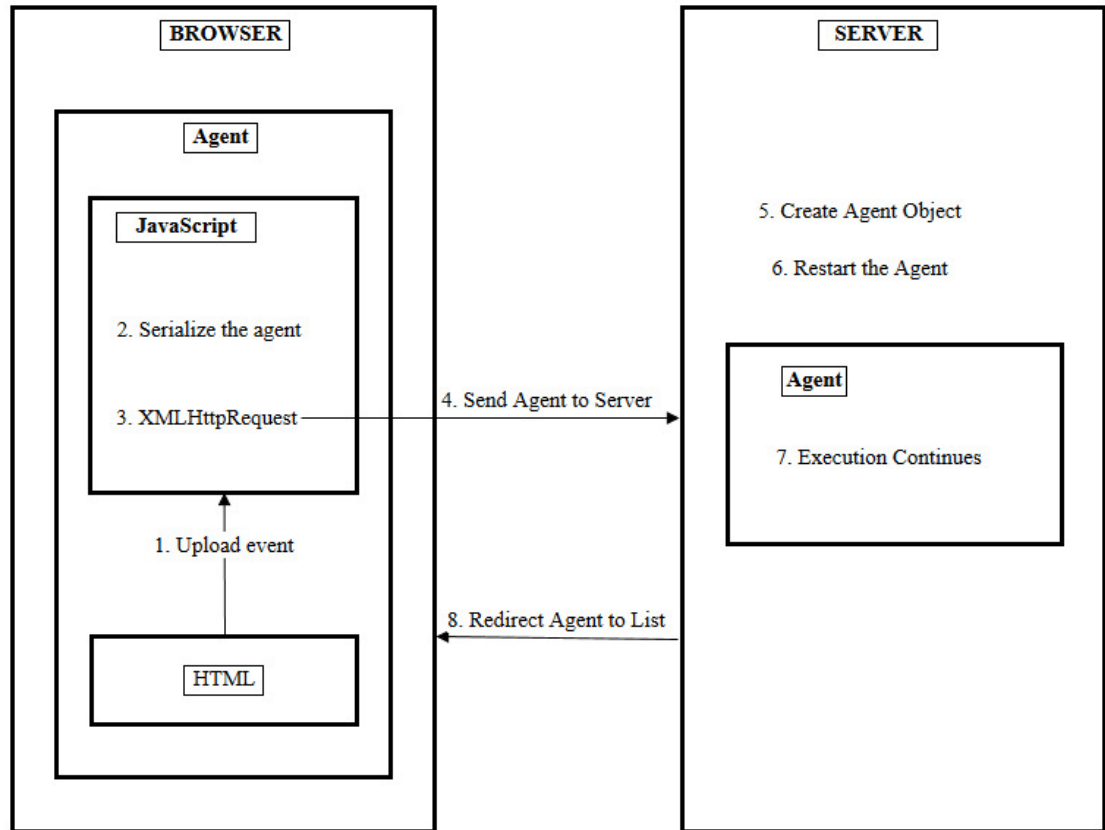
**Life-Cycle Model** follows the persistent process life cycle [41]. An agent is created and *started*, when it is fetched from its origin server (file server). Inside the browser, the agent is in *running* state with user interface that is rendered by the browser. When that agent is uploaded to an agent server, it is set to *frozen* state, serialized and then restarted from where it was frozen in the server. When the agent is again fetched from the agent server, a description of the agent is sent to the browser to enable the agent to restart itself in UI mode in the browser. The life cycle of an HTML5 agent is shown in the following figure:



**Figure 4.1** Example Life-Cycle of HTML5 Agent [36]

**Computational Model** is implemented in JavaScript and is based on event-handlers. It is needed to run in both the browser and the server. In browser, it should be non-blocking the event loop of the browser. In the server, it should be able to execute the agent without its UI events.

**Navigation Model** consists of a configuration file that is downloaded with the agent. In this implementation [5] of the framework, the navigation model comprises of connections only to one agent server and the origin server of the mobile agent. It also contains the serialization of an agent and its transfer management, which is illustrated in the following figure.



*Figure 4.2 Serialization & Transfer of the Mobile Agent [5]*

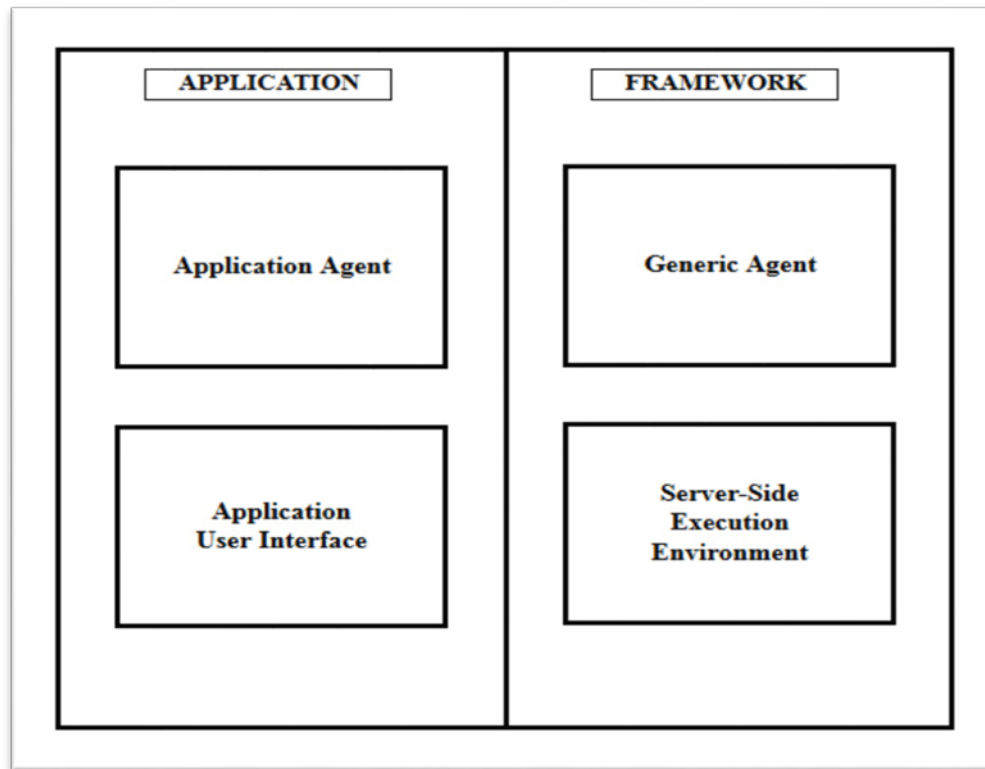
**Communication Model** includes the communication with user, agent server, another agent application and third-party services. An agent is able to communicate with the third-party service through the APIs provided by the service. Agent-to-agent communication is achieved by the following three cases [5]:

- 1) Both agent applications are in separate servers
- 2) Both agent applications are in an agent server
- 3) One agent is in browser and another agent is in server

**Security Model** in this implementation depends on the standard security mechanisms of HTML5 applications in browser.

### 4.3 Architecture of the Framework

The basic architecture on abstract of this HTML5 based agent framework is shown in Figure 4.3. The Figure 4.3 depicts the framework and its correlation to the application specific implementation.



*Figure 4.3 Basic Architecture of the Framework [5]*

The agent part in the architecture consists of two parts: a generic agent and an application agent. In the server, the agent is accessed through generic interface, delivered by generic agent. In the browser, the agent is accessed through application specific user interface.

#### 4.3.1 Configuration of HTML5 Agent

In the current implementation [5], all the agents are configured by using *configuration.js* file. This file contains the URLs to origin server of an agent, list view of agents in agent server, the agent server that is used by agents to communicate and where the agents are uploaded. A configuration object is created by calling constructor of **configurationClass**, which creates an object that includes the specified URLs. Following are the functions specified in *configuration.js* file to access different URLs:



***listUrl***: it returns URL to the list view of all the agents, executing in agent server

***uploadUrl***: it returns URL to the agent server where an agent is to be uploaded.

***applicationUrl***: it return URL to the origin server (file server) of the application.

***communicationUrl***: it returns URL to the agent server, which is used by the application for communication.

***managementUrl***: it returns URL to the management server of the framework, where the agents are handled.

At present, the origin server is same for all the agents as they use the same *configuration.js* file. This file also contains a function **downloadAgent** that downloads the JavaScript file of generic agent.

### 4.3.2 Generic Agent

It serves as the base class for every mobile agent application. It provides the generic parts of an agent to all the agent applications. The application agent inherits from the generic agent through functional inheritance [57], [58]. The fundamental structure of the generic agent is given as follows:

```
function Agent(src,html) {
    var that = {};
    that.id = (new Date()).getTime() % 1000000;
    that.auri = src;
    that.huri = html;
    that.method = function(args) {...}
    return that;
}
```

In the above code generic for generic agent, **that** is a variable, which stores all variables and functions of application agent. The initialization parameter **src** specifies the URL to the functionality of the application agent i.e. the JavaScript file. The other initialization parameter **html** specifies the URL to the user interface of the application agent i.e. the HTML file, which has a reference to the CSS file.

The generic agent also provides functions that are required for the state management of an application agent. Following are the functions in generic agent, which facilitates to manage different states of an application agent:

**createVar()**: this function is used to create a variable in the list of variables of an agent.

**registerVar()**: this function registers the specified variable to the memory, so that it can be recovered later on.

**setVal()**: this function is used to change the value of a specified variable.

**getVal()**: this function is used to get the value of a specified variable.

**setMemory()**: this function resets the memory of an agent to the given variable list. It clears all the previous saved memory variables and their data.

The current implementation of the framework [5] proposes two approaches of saving the inner state of an agent. The first method is the use of **setVar()** and **getVar()** functions of the Generic Agent class to set or get specific variables of the agent. The second method uses the **registerVar()** function, which adds registered variable to the agent's memory. Serialization of the state of an agent depends on the approach of saving the inner state of the agent.

### **Serialization of an Agent**

Serialization is the process of converting the state information of an object into binary or textual form for the purpose of its storage in memory or transportation over a network. The main objective of serialization is to save the state of an object, so it can be recreated, when needed [59], [60]. In the current implementation of the framework [5], an agent is serialized by constructing a JSON string, which include the agent's description based on the inner data in the variables of the agent. The transfer of the agent is done by HTTP. Following are the functions in generic agent base class, which are needed for serializing and transporting the agent:

**serialize()**: this function returns a JSON-string (agent's description) i.e. all the necessary information required to preserve the state and continue execution of the agent in new location.

**preupload()**: this function is only required, if the application needs to do some job before it is being uploaded to the server. The execution of agent is stopped before a call to this function.

**upload()**: this function calls the **serialize** and transfers the description of agent to the server.

## Execution Management of the Agent

The generic agent contains the following functions that are needed to manage the execution of the agent:

**continueWork()**: when the agent arrives in a new location, this function initializes the agent and returns the variables and memory of the agent. Generic agent **continueWork()** has to be called in the start of application specific parts [5].

**work()**: this function defines what an agent should be doing in defined intervals. Generic agent does not have an implementation of this function, since it is application-specific.

**setRunInterval()**: this function specifies the interval for the calls of *work* function.

**start()**: this function initializes an agent.

**stop()**: this function is used to stop an agent.

**getRunningStatus()**: this function returns information about the status of the agent.

**isInServer()**: this function is used to check whether an agent is in server or not. It returns *true* if the agent is in server and returns *false* if the agent is not in the server.

### 4.3.3 Application Agent

An application agent is the actual implementation of the mobile agent application, which is capable of migrating between a browser and a server. The following code snippet illustrates the inheritance of an application agent from the generic agent, discussed in section 4.3.2:

```
function ExampleAgent(src, html) {
  var that = new Agent(src, html);

  /* user-defined methods */

  return that;
}
```

In the application agent code, the variable **that** must include all the variables, which are saved to the memory; else the generic agent will not be able to find those variables. The following are the main functions for an application agent:

**work()**: this function is the concrete implementation of the abstract function, which is declared in the Generic Agent class. It is supposed to contain all the functionalities that an agent is supposed to do.

**continueWork()**: this function initializes or resets those application specific parts, which are not saved to memory or variable list of agents.

**createAgentObject()**: this function is used create an instance of a specific agent. It is needed both in browser and in server. This function takes in parameters, the location of html file and JavaScript file of the agent.

**initExecution**: this function initializes the agent in browser. The **createAgentObject()** is called in this function and after that, the agent is started by calling **continueWork()**.

The **initExecution()** function is recommended to be called as onload-function in the body element of the HTML file [5] so that the agent is initialized and started, prior to the UI of the application agent is displayed for a user.

#### 4.3.4 Agent-to-Agent Communication

In the current version of the framework, the communication component is generic. The communication component consists of two main functions, which are as follows:

**initIO()**: this function initializes the connection to agent server, creates application-specific namespace; thus enabling other agents to join in the same communication namespace for sharing information.

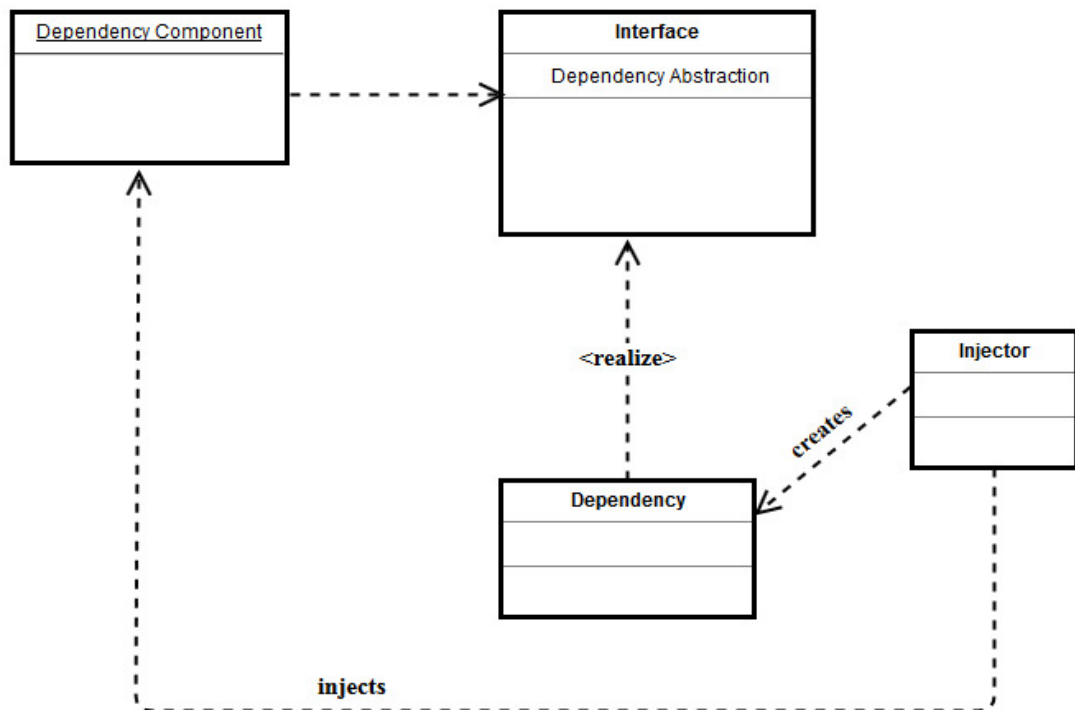
**sendMessage()**: this function takes in parameters the data that is to be sent. It is used for sending information to the other agents, which share a common namespace.

The exchange of information between agents is done through JSON strings, which helps in transferring the static data and the functions. The JavaScript Object Notation (JSON) is a lightweight, format-friendly data interchange format [15], [16]. It is easy to read and write, more importantly, it is easier to generate and parse data as compared to XML which helps the Ajax-based application to do more smooth data presentation [61]. Following are the few reasons of using JSON strings:

- It is widely used and well-known format in the field of Web development.
- It is “native” to JavaScript.
- It is lightweight, text based and easy to parse.
- It is easily readable by humans and by machines.
- Serialization of objects into JSON strings and paring JSON strings back to objects is implemented as built-in feature in commonly used browsers, thus a separate parser is not needed.

### 4.3.5 Agent Server

The agent server in this framework [5] comprises of three parts: (1) server, (2) router and (3) handlers. For the sake of flexibility in the implementing agent server, a dependency injection design pattern is used [62]. The core concept of dependency injection is to define separate injector component, which injects the dependencies on the dependent components. It allows flexibility to a client of being configurable. The dependent component consists of only the abstraction of the dependency. The concrete implementation of the dependency is injected into the dependent component by an entirely separate injector class. This phenomenon facilitates the changing of dependency class with another implementation whenever it is required. The following Figure 4.4 is envisioning the functionality of dependency injection:



**Figure 4.4** Sketch of Dependency Injection [5]

In our implementation, the injector is a script file *agentserverindex.js*, which creates and initializes the agent server and its relevant parts. Dependent parts are the server, which is dependent on the defined router and the router, which is dependent on the handlers. The server part is liable for receiving request from agent server, parsing the path in requested URL and passing that request and its path to the router. Job of the router is to route various requests to different handlers. The handler functions are provided for the use of other modules. Following are listed the handler functions and those functions, which are used to initialize the agents in agent server [5]:

**upload( )**: this function is called when an upload request is made.

**list( )**: this function displays the list of all agents on the server, along with their current status

**getSpecificAgent( )**: this function is used when a specific agent is called for execution in browser (UI mode). Hence, it removes the agent from the server.

**getAgentDescrList( )**: this function is used when the descriptions of all agents running in a server is required in JSON string.

**getSpecificAgentDescr( )**: this function also removes the agent from its server. It is used to get the description of a specific agent in JSON string.

**createAgent( )**: this function takes a serialized agent as a parameter and creates the agent description and a concrete instance of the agent for the server.

**startAgent( )**: this function restarts the execution of the agent in server. It takes the agent description as parameter.

In the current implementation of our framework, the injector script *agentserverindex.js* creates and initialize an agent server with port number 8891. However, during this thesis, we have edited this script file to allow us to run different agent servers on different ports, in order to be able to have multiple agent servers running simultaneously. To start the default agent server, we would execute a command as follows:

```
$ node agentserverindex.js
```

The output of the above command is that an agent server starts running on port 8891. In our implementation, after editing the *agentserverindex.js*, the following command is executed, in order to run multiple agent servers, on different ports of our choice:

```
$ AgentPort=XXXX node agentserverindex.js
```

where XXXX is any port number that we choose for the agent server to run on.

## 5. DEVELOPMENT OF BROWSER-BASED USER INTERFACE FOR MANAGEMENT SERVER

This chapter discusses the original work done in this thesis for the development of browser-based user interface for the management server of our framework. During the development of current framework [5], an API (Application Programming Interfaces) is included for the management server. However, there was not a GUI-based deployment of the management server. This thesis embark on using that API and develop a full-fledged web browser based user interface.

### 5.1 Management Server and its API

The Management Server is responsible to know the location and the status of every agent. Also, the agents are to be controlled by the Management Server. It exposes an API to manage agents and agent servers through an HTTP interface. According to [33], the management protocols are made compatible with the overall scheme of the underlying framework. The Management Server of this agent framework implements a REST (Representational State Transfer) API, which is used to control the migration of agents and the application. Web services are meant to support the needs of application programs. So, the client applications use the APIs to communicate with those web services [63] [64]. An API exposes a set of data and functions to facilitate connections between client programs and enable them to exchange the information. Modern age web services are designed by using the REST architecture style, which is defined to help in creating and organizing the distributed systems. REST emphasizes on the roles of components, constraints on their interaction with other components, and their interpretation of different data elements [65]. The client-server communication in REST architectural style is stateless; it implies that no client context is stored on the server during client-server requests.

### 5.2 The Management Server API of Our Agent Framework

The Management Server of our framework has implemented a RESTful API. The most important part of this API comprises of two types of REST calls [33]: *ImHere*, for the agents who arrive in a new location and *Status*, a call that is sent to the Management Server on regular intervals to indicate the agent's running status. The response to these calls contain an instruction to the agent to migrate to another host (location). The current implementation also includes instructions for the client applications to exit or change the values of different variables. This is a lightweight management server framework, which assumes the agents to collaborate and has no effect on the non-participating agents [33].

The current framework relies on the HTTP protocol and only those agents comply with the received instructions that reside in server.

The underlying API of the Management Server has three distinct interfaces, which are briefly described as following:

1. Agent Interface
2. Management Interface
3. AgentServers Interface

### 5.2.1 Agent Interface

The *Agent* interface is used to track and manage different agents in the framework. The following commands are included in the Agent interface of the API:

**/ImHere:** When an agent migrates to a new location, it should send a message **ImHere** to the Management Server.

Management/**ImHere**(PUT)

Payload example:

from server: {"id":"230698","server":"http://ubuntu:8891"}

from browser: {"id":"230698","client":"homepcHost"}

**/Status:** An agent should send **Status** message to the Management Server, every time after the execution of **Work** function.

Management/**Status**(PUT)

Payload example:

{"id":"230698","status":"the counter is 27"}

**/ImDying:** When an agent is about to die (exit), it should send **ImDying** message to the Management Server.

Management/**ImDying**(PUT)

**/Error:** It is recommended that agent should send an **Error** message, in case of error. This is helpful in debugging.

Management/**Error**(POST)



Payload example:

```
{“Message”:string, <error report>}
```

The agent server may send some instructions to the agent, as a response to the messages like /ImHere and /Status. For example,

- Request to exit     {“\_kill” , null}
- Instruction to move to new location (server) {“\_goto” , address}

## 5.2.2 Management Interface

The Management Server renders the responses in JSON strings, while these responses are rendered to end user by the client application. The management interface includes the following important commands:

**/Agents/list:** is used to get a list of running agents.

```
/Agents/List (GET)     {id, id, id, id, id}
```

The response to it would be, e.g.

```
[“230698”, “85412”, “634120”, “78454”, “224151”]
```

**/Agents/id:** is used to query a detailed information about a specific agent.

```
$ curl localhost:8896/Agents/230698
```

The response is, e.g.

```
{“id”:“230698”,“status”:“counter=87”,“state”:“running”,“server”:null,
“client”: “127.0.0.1:74321 (homepcHost)”}
```

There are also some other commands, e.g. /Agents/id/history is used to get the history of an agent with specific id. /Agents/id/todo is used to give instructions to the agent e.g. “goto”, “kill”.

### 5.2.3 AgentServers Interface

In the current implementation [5], the Management Server gathers information about the different agent servers. The API command to get all the known agent servers is /AgentServers. e.g.

```
$ curl localhost:8896/AgentServers
```

The response to the above command is, e.g.

```
[“http://ubuntu:5898”,“http://ubuntu:8897”, “http://ubuntu:7762”]
```

## 5.3 Technologies Used in the UI Development

In this section, the technologies used in the development of the User Interface for our framework are discussed.

### 5.3.1 ReactJS

It is an open-source JavaScript library used for making reusable UI components. It was developed by Facebook in 2013. ReactJS is not a framework, but acts as only the “View” part of MVC (Model-View-Controller) architecture of web applications. It is a concept that is more dedicated on component-driven development. It enables the developers to create applications with reusable UI components, which can be re-used anywhere in the applications. The core function of ReactJS is to bind HTML elements to the data; once there are changes in that data, ReactJS takes care of updating the interface.

When a webpage is uploaded in a browser; the DOM of that webpage is created, which represents the webpage in a tree-like structure. A web browser provides the JavaScript DOM API for manipulating the DOM with JavaScript [18]. However, manipulating the DOM with JavaScript has some major issues, as discussed in [66]. The programming style becomes imperative, if it is decided to use the JavaScript DOM API and the DOM mutations are slower. ReactJS readily offers a solution called the “virtual DOM” to above-mentioned problems. A virtual DOM is a fast, in-memory representation of the real DOM. Whenever the state of the data-model changes, the virtual DOM and React renders the UI to a virtual DOM representation [67]. React calculates the difference between two virtual DOM representation: virtual DOM representations before and after the changes in the data-model. The difference between those two virtual DOM representations is then changes in the real DOM.

ReactJS helps in dividing the user interface into reusable, independent pieces called the components. Conceptually, components are similar to functions in JavaScript.

### **5.3.2 jQuery**

It is a platform-independent, lightweight and feature-rich JavaScript library, which is used to simplify the client-side JavaScript programming. Web analytics show that jQuery is the most widely used JavaScript library [68]. jQuery is designed to make the navigation of DOM elements and handling different events more easy.

### **5.3.3 Bootstrap**

It is a powerful open source, front-end development framework, originally developed at Twitter. It is one of the most popular HTML, CSS and JavaScript framework to develop responsive websites. Its main advantages include responsive features, consistent design, compatibility with modern browsers and being free of cost.

### **5.3.4 Babel**

It is a JavaScript compiler that is used to convert ReactJS templates to pure JavaScript, so that it can be rendered by web browsers.

### **5.3.5 Noty**

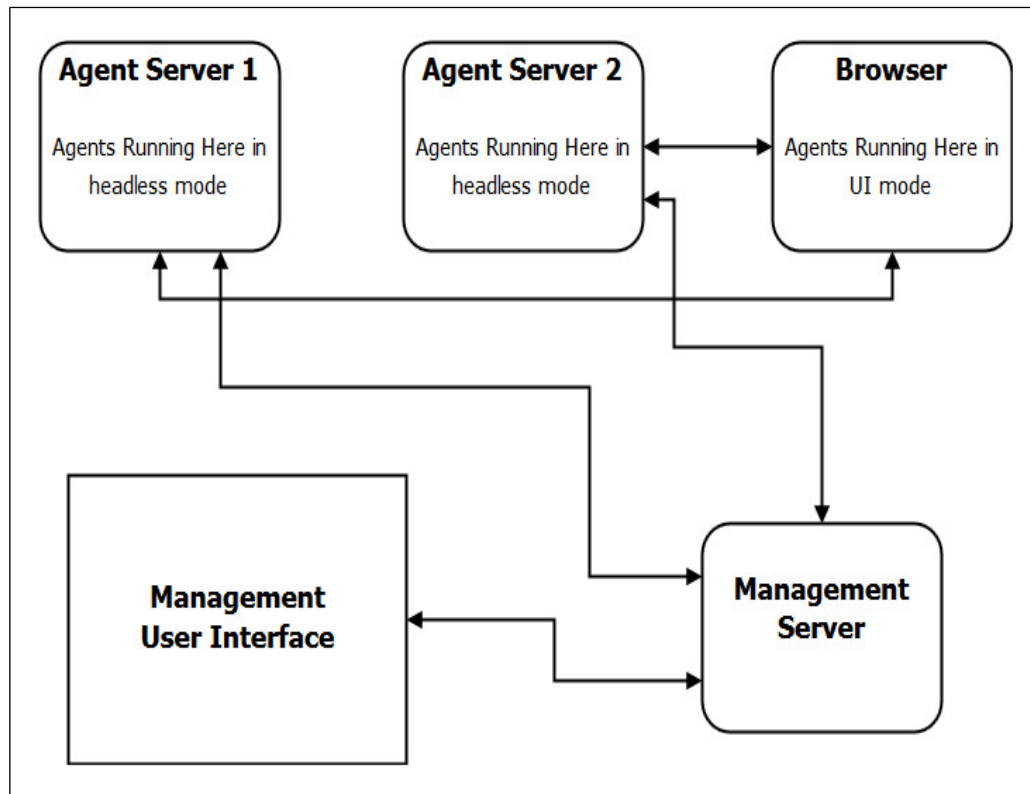
It is a dependency-free notification library; used for creating alert, success, error, warning, information, confirmation messages as an alternative to the stand alert dialog boxes [69].

### **5.3.6 Sortable**

It is a discreet JavaScript library, which is used for the drag-and-drop functionalities in the user interface.

## **5.4 Management Server User Interface & React Components**

The User Interface application of the management server has been developed in such a way that the application itself does not directly interact with agent servers. Rather, it only deals with the management server, which in turn is communicating with the agent servers. The following Figure 5.1 is abstract level view of the framework, in which the Management User Interface has been integrated:



*Figure 5.1: Abstract View of the Framework with Management UI*

All the React components used that eventually renders the User Interface of the Management Server, are discussed as follows:

#### 5.4.1 App

It is the main ReactJS component that hosts the state of the Management User Interface and renders all the sub-components. The component “app” contains the following two functions:

**getAgentList:** this function gets the information of all agents from the management server and passes it to the other React components *ServerList* and *AgentList*.

**getServerList:** this function gets the information of all running servers that are registered with the management server.

The “app” components refreshes every 5 seconds, in order to receive latest information about the servers and the agents through **getAgentList** and **getServerList** functions.

### 5.4.2 ServerList

This component renders the list of all servers on the *app* component of the user interface. It comprises the code to display each agent in a small box along with agent's id on it. The *ServerList* component also facilitate the migration and deletion of the agents.

### 5.4.3 AgentList

This component receives the list of agents from the *app* component and display that list on the UI. All the agents are displayed with its relevant information: irrespective of whether the agents are running in the browser or in any server. The information of an agent rendered by this component consists of Agent ID, server/client (context-dependent), state and buttons for moving or deletion of the agent. Apart from displaying the list of agents on the UI, this component has the following two other important functions:

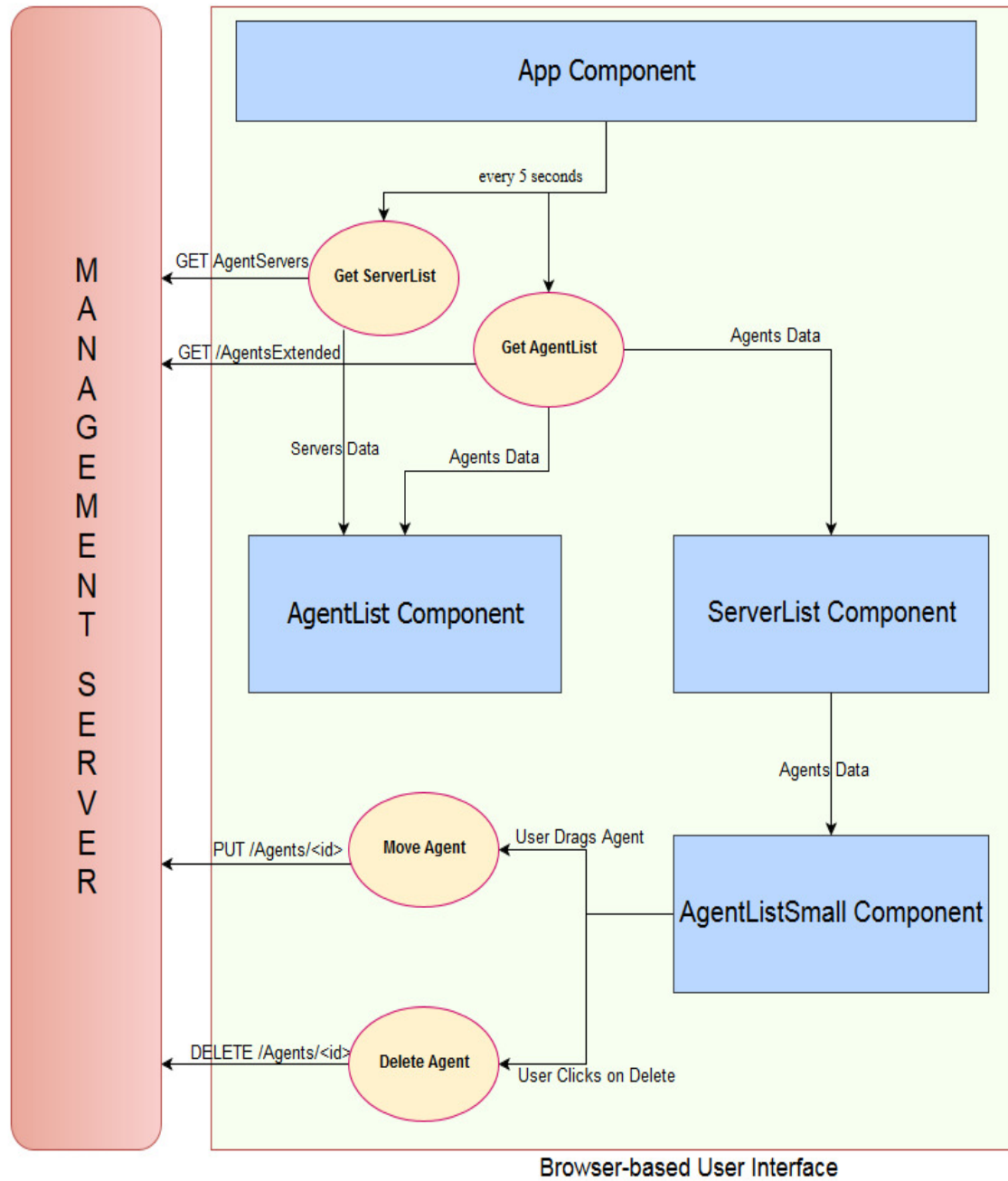
**moveAgent:** this function is used to move (migrate) an agent from one server to another.

**destroyAgent:** this function kills (delete) a running agent from any given server.

### 5.4.4 AgentListSmall

This component displays the agents as tiny rectangular cards, under their corresponding servers. The tiny cards (agents) can be dragged-and-dropped from one server to another among all the running servers. An agent (the tiny card) has the agent ID and two buttons: one for displaying more information about that agent and second button for deleting that agent. The component *AgentListSmall* also make use of the JavaScript library “*sortable*” to enable the drag-and-drop feature in this implementation of the User Interface. The drag-and-drop feature enables us to move an agent from one server to another, merely by dragging it from the current server (host) to the destination server.

The following figure shows the overall interaction of the different components of the user interface:



**Figure 5.2:** Interaction of Different Components of the User Interface

## 5.5 Features of the Management Server User Interface

The User Interface developed in this thesis offers the following features for the HTML5 based Mobile Agent Framework developed in [5]:

### 5.5.1 Displaying Agents & Servers

This browser-based User Interface displays all the running agents and the servers on the homepage of the management server. This information displayed on the UI is divided into two sections;

1. There is a list of all the running Agent Servers. If there are some agents that are running on any server, their information is also displayed, separately under the corresponding server.
2. After the agent server, there is a list of all the Agents, irrespective of whether they are running in browser or in some server.

### 5.5.2 Displaying Agent's Information

The information of an agent that is displayed consists of an Agent ID, state of the agent, address of its server or its client and status of the agent. Displaying agent's information is enabled in both; the Agent Servers view and the Agents view.

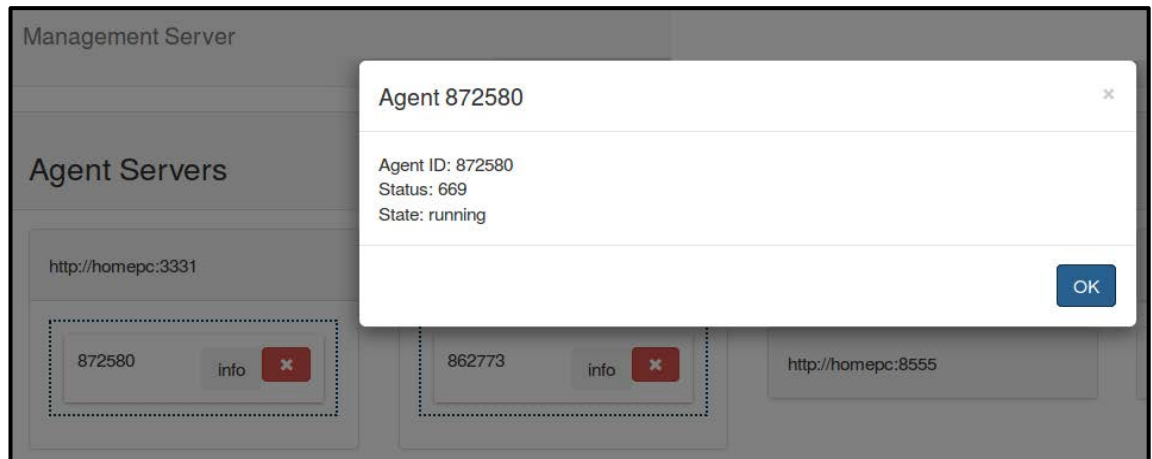
Agents				
Agent ID	State	Server	Client	Status
842839	running		::ffff:127.0.0.1:46392 (xhost)	475
862773	running	http://homepc:8891		Clock is 456

Remove

Move

**Figure 5.3:** Agents Information in Agents-view of the UI

The above Figure 5.3 shows the information of two running agents in the Agents-view of the UI. The first agent is running in a browser, so there is no server address for that agent. The second agent is running on a server, so it has its server information.



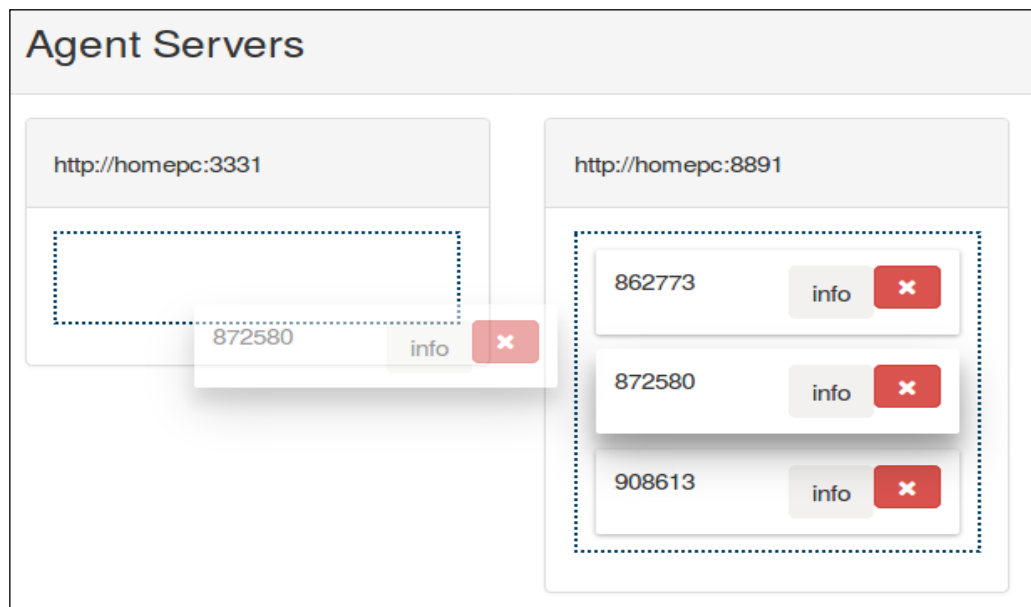
**Figure 5.4:** Agent Information in AgentServers-view of the UI

The Figure 5.4 shows the information of an agent with “Agent ID = 872580” running on a server “http://homepc:3331” in Agent Servers view, after a user click on the small “info” button on the agent.

### 5.5.3 Migration of An Agent Between Servers

It is an important feature of the Management Server UI. There are two modes of migrating an agent from one server to another which are as follows:

1. **Drag-and-Drop:** This feature is available in the Agent Servers view. An agent can be easily dragged from one server and dropped to another server.

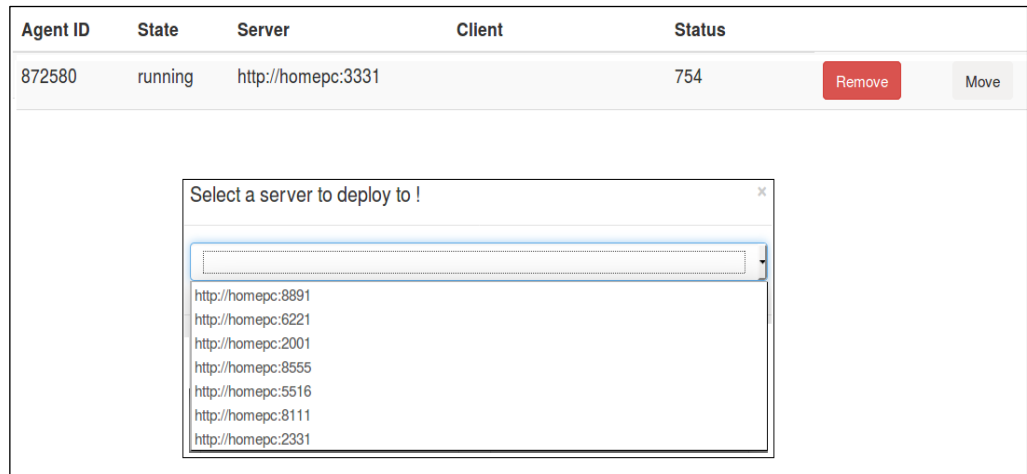


**Figure 5.5:** Drag-and-Drop migration of an agent between servers

In the above Figure 5.5, an agent with ID “872580” is being dragged from the server “http://homepc:8891” to another server “http://homepc:3331”.



2. **Click on the “move” Button:** In the Agents view, there is a button “move” for every agent that is running on some server. When that button is clicked, a dropdown list appears in the center of the web page, which contains the list of all servers. The user can choose whichever server he wants the agent to move by simply clicking on that server from the dropdown list.



**Figure 5.6:** Migration of an agent via ‘Move’ button

In the above Figure 5.6, the migration of an agent between servers via “Move” button is shown. When the Move button on an agent is clicked, a dropdown list of all the running servers appears. A user can click on the server, to which he wants the agent to be migrated to.

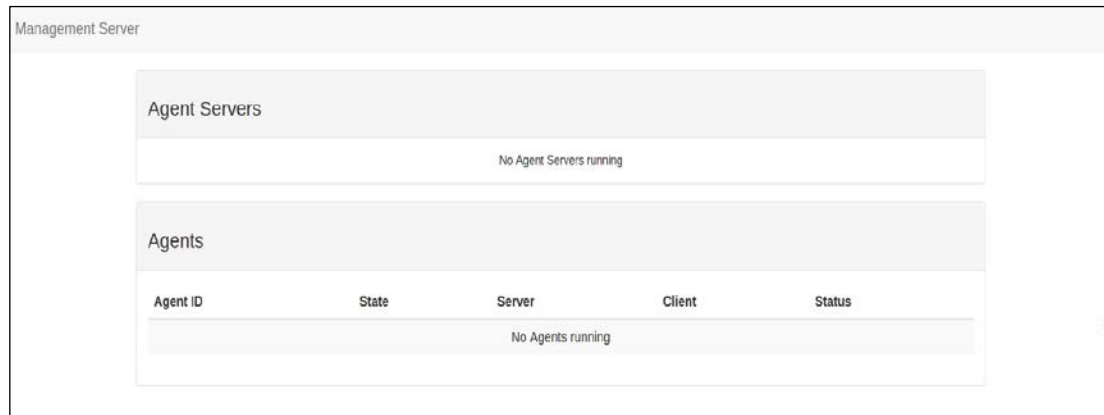
#### 5.5.4 Deletion of An Agent From Its Server

This feature is available on every agent that is running in some server. There is a “remove” button on every agent, which is used to delete (kill) the agent. A confirmatory box, still, confirms from the user about its action, before eventually deleting the agent. After the deletion, the information of that agent is removed from the Agents view, with only its status point to being “killed”.

## 5.6 Screenshots of The Management Server User Interface

This section intends to give an overview about the developed user interface, by providing screenshots during various functionalities of the management server.

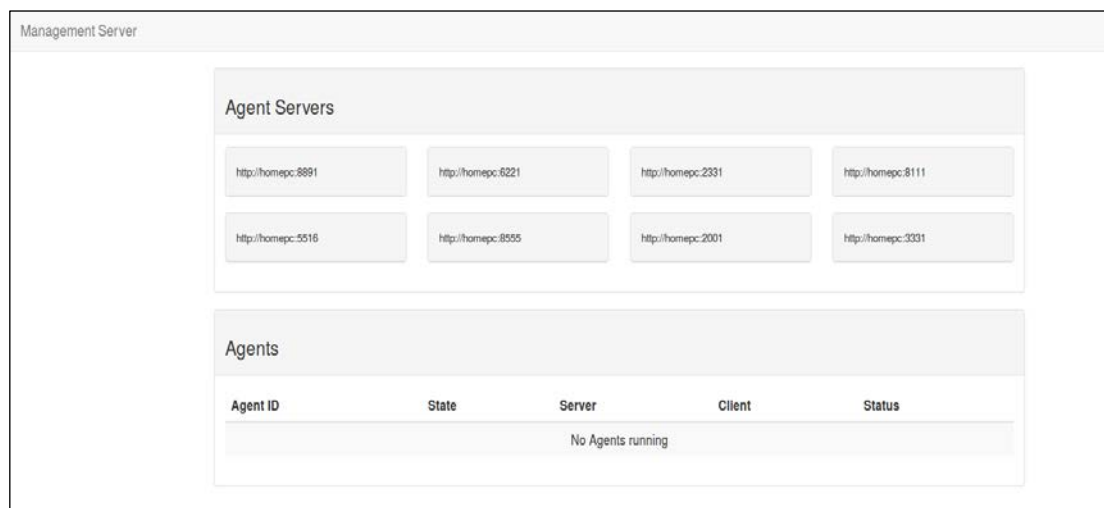
### 1. When There Are No Servers & No Agents Running



**Figure 5.7:** Main homePage of UI, no agents & servers running

This is the main homepage view of the management server UI. At this point of time, there are no agent servers or any agents running. So it displays the respective messages.

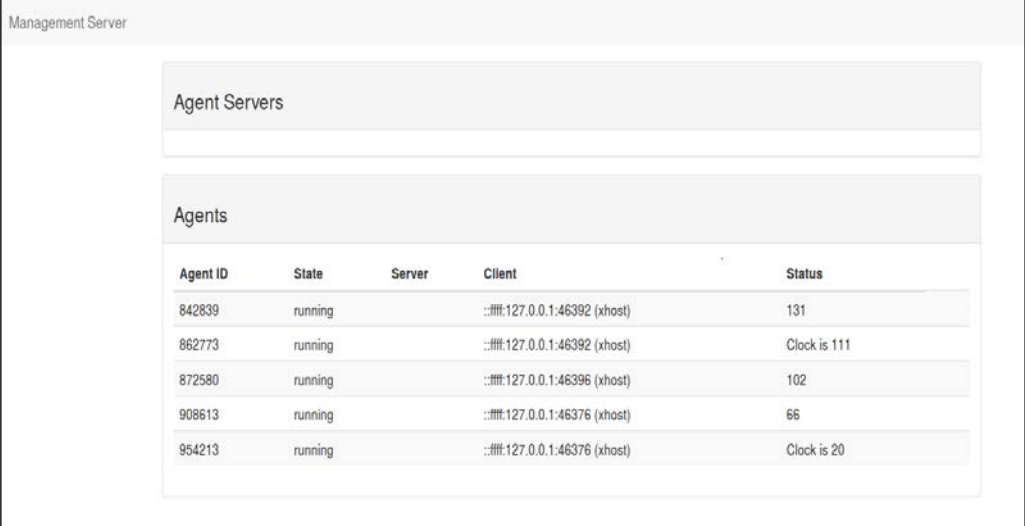
### 2. When There Are Several Agent Servers Running



**Figure 5.8:** UI showing several servers running

This is the view of the UI when several agent servers are actively running but still there are no agents running on the framework.

### 3. When There Are Different Agents Running, All in Browser (UI) Mode



Management Server

Agent Servers

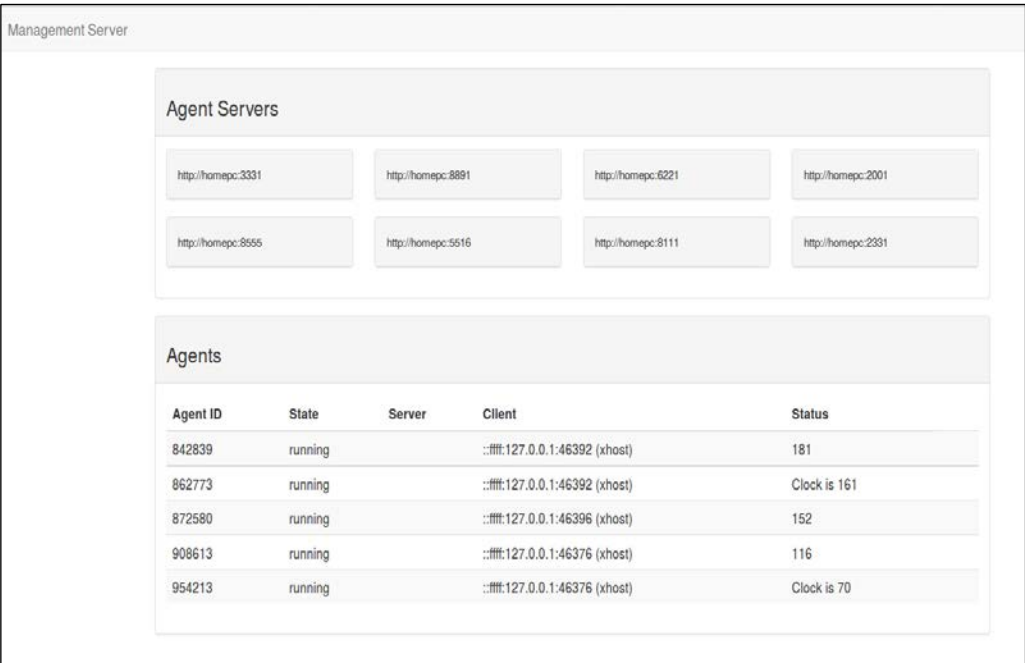
Agents

Agent ID	State	Server	Client	Status
842839	running		::ffff:127.0.0.1:46392 (xhost)	131
862773	running		::ffff:127.0.0.1:46392 (xhost)	Clock is 111
872580	running		::ffff:127.0.0.1:46396 (xhost)	102
908613	running		::ffff:127.0.0.1:46376 (xhost)	66
954213	running		::ffff:127.0.0.1:46376 (xhost)	Clock is 20

**Figure 5.9:** UI showing different agents running in browser

This view of the UI shows different agents running. All these agents are running in browser, hence, they are not in headless (in server) mode. At this point, there are no Agent Servers started.

### 4. When Agent Servers and Agents Are Running Simultaneously



Management Server

Agent Servers

Agents

Agent ID	State	Server	Client	Status
842839	running		::ffff:127.0.0.1:46392 (xhost)	181
862773	running		::ffff:127.0.0.1:46392 (xhost)	Clock is 161
872580	running		::ffff:127.0.0.1:46396 (xhost)	152
908613	running		::ffff:127.0.0.1:46376 (xhost)	116
954213	running		::ffff:127.0.0.1:46376 (xhost)	Clock is 70

**Figure 5.10:** UI showing different AgentServers & agents running

This view displays two views; the Agent Servers view which shows all the running agent servers in small rectangular lists.

## 5. When Some of The Agents Are Running In Agent Servers While Others Are Running in Browser

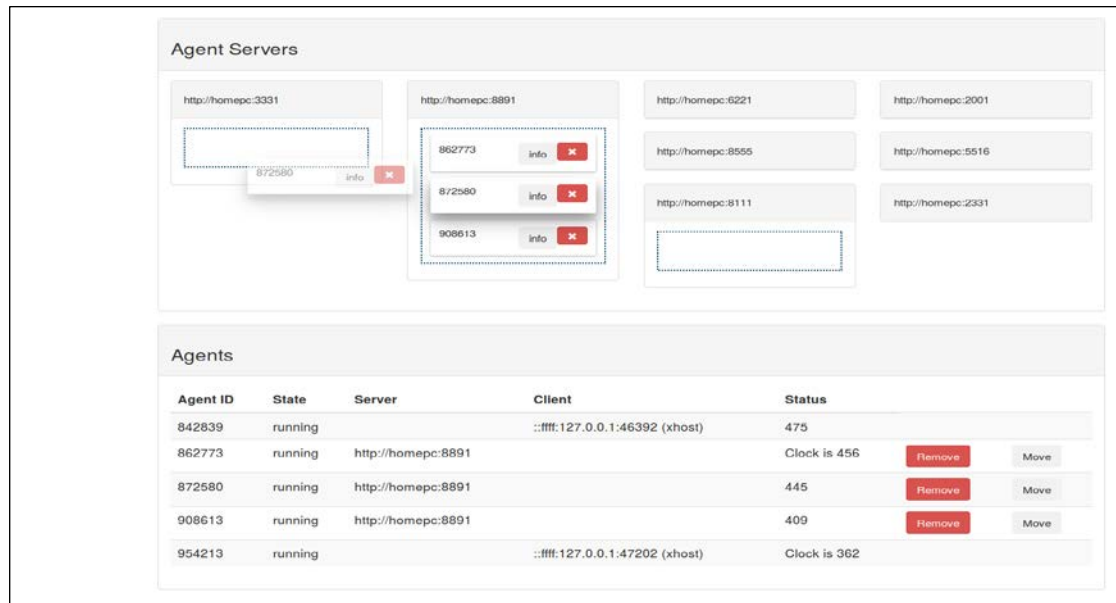
The screenshot shows the Management Server interface. The top section is titled "Agent Servers" and displays a grid of server cards. The server card for "http://homepc:8891" is expanded, showing two agents: 872580 and 954213, each with an "info" button and a red "x" button. Below the "Agent Servers" section is the "Agents" section, which contains a table of running agents.

Agent ID	State	Server	Client	Status	
842839	running		::ffff:127.0.0.1:46392 (xhost)	395	
862773	running		::ffff:127.0.0.1:46392 (xhost)	Clock is 375	
872580	running	http://homepc:8891		365	Remove Move
908613	running		::ffff:127.0.0.1:46376 (xhost)	330	
954213	running	http://homepc:8891		Clock is 283	Remove Move

**Figure 5.11:** UI showing agents running in browser and in servers

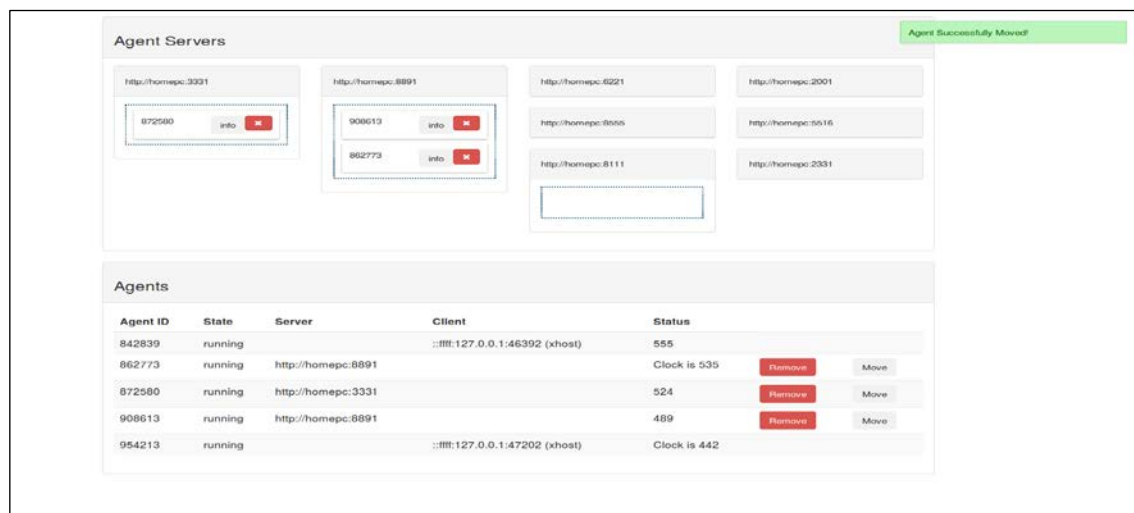
In the above view of UI, it can be seen that there are 5 agents running in the framework. Two of those agents are running in a server “*http://homepc:8891*” while the other three are running in browser. Consequently in the Agent Servers view, only those two agents are shown under the server “*http://homepc:8891*”. The other running servers can also be seen, a couple of them are expanded while the rest are not expanded; but all of them have no agents running on it.

## 6. Drag-and-Drop Feature



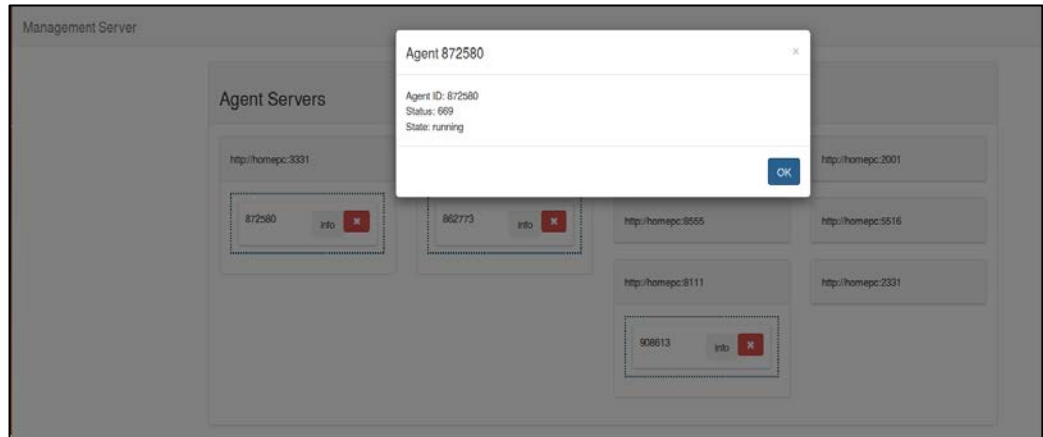
**Figure 5.12:** UI showing drag-and-drop feature

In the above UI view, it can be seen that one (middle one) of the three agents that are running on agent server “`http://homepc:8891`” is being dragged by mouse towards another agent server “`http://homepc:3331`”. Now, the following view of the UI shows a successful completion of the drag-and-drop of an agent from one server to another. The agent with id “872580” can be seen that it is now running on the new server. The server address in the Agents view is also changed for the migrated agent.



**Figure 5.13:** UI showing drag-and-drop feature

## 7. When The “INFO” Button On An Agent in AgentServer View is Clicked

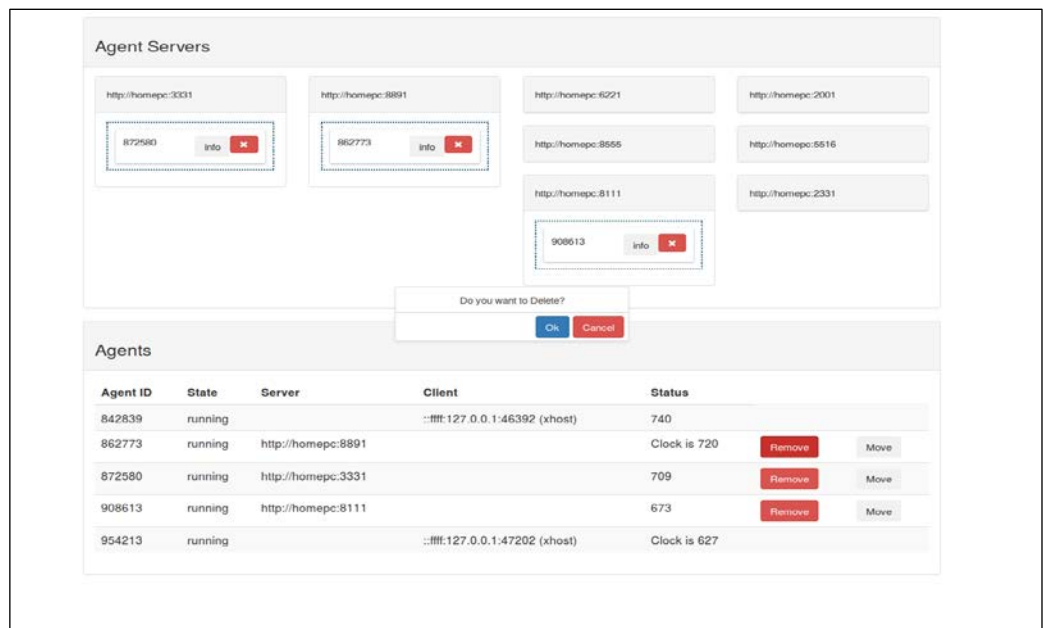


**Figure 5.14:** UI showing info of an agent in AgentServer view

When the small “info” button on the agent is clicked in the Agent Servers view, the main page view is shadowed and disabled, while a pop-up screen appears with the agent’s name at its top, giving the information about the agent, which include agent ID, its status and its state.

## 8. Agent’s Deletion

To delete (kill) an agent that is running on any server, the UI provides the “cross-sign” and the “remove” red-color buttons in the AgentServers and Agents view respectively. When either of the above mentioned buttons is clicked, a dialog box appears in the middle of the screen, which confirms from the user about the deletion of the agent.



**Figure 5.15:** UI showing agent’s deletion

When the user confirms the deletion task of an agent by clicking the “OK” button in the dialog box, the following view of the UI appears, in which, the agent is deleted from the server and the management UI now contains an updated information of all the agents. The status of the deleted agent becomes “*Killed*” and all other information of that agent is deleted.

The screenshot displays two main sections: 'Agent Servers' and 'Agents'.

**Agent Servers:** This section shows a grid of server cards. Each card represents a server with a URL (e.g., http://homepc:3331) and a list of agents. The agent 872580 is shown with a red 'X' icon, indicating it has been deleted. The agent 908613 is also shown with a red 'X' icon.

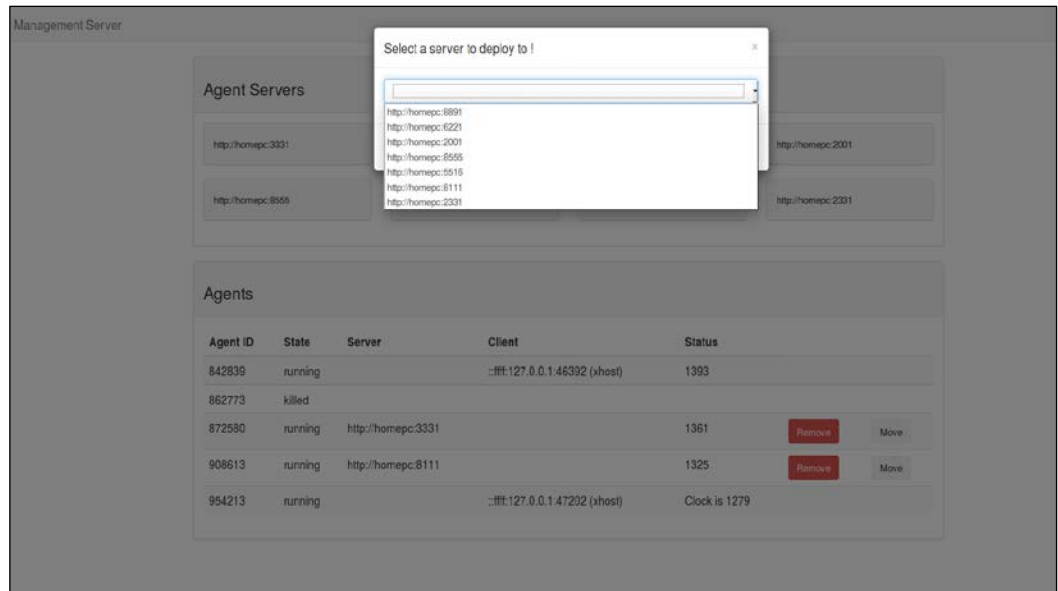
**Agents:** This section contains a table listing all agents. The table has columns for Agent ID, State, Server, Client, and Status. The agent 872580 is listed with a status of 'killed'.

Agent ID	State	Server	Client	Status
842839	running		::ffff:127.0.0.1:46392 (xhost)	785
862773	killed			
872580	running	http://homepc:3331		754
908613	running	http://homepc:8111		718
954213	running		::ffff:127.0.0.1:47202 (xhost)	Clock is 672

**Figure 5.16:** UI showing updated view after agent's deletion

## 9. Moving Agent Between Server From Agents' View

Earlier, we discussed the drag-and-drop feature of the UI in Agent Servers view, which is meant to drag an agent from one server and drop it to another server. However, a user can also move an agent from one server to another, from the Agents view of the UI.



**Figure 5.17:** UI showing migration of agent in Agents view

The above Figure 5.17 shows the scenario, after clicking the “move” button on an agent. A dropdown list appears with the rest of the webpage disabled. The dropdown list is a list of all the agent servers that are running in the framework. The user can select the desired server, to move (migrate/deploy) the agent to.



## 6. EVALUATION AND CONCLUSION

In this chapter, the HTML5 mobile agent framework is briefly evaluated and then the conclusion is presented for this thesis work.

### 6.1 Evaluation of the Current Framework

The HTML5 mobile agent framework is evaluated on the basis of overall development of the framework and against the relevant existing mobile agents systems. Following is the summarized assessment of our HTML5 Mobile Agent Framework, in accordance to [5]:

#### 6.1.1 Advantages of the Current Framework vs Other Agent Platforms

- Our current framework does not need a dedicated installation of any software or environment because it is fully capable of running on any modern web browser.
- There is no need of plugins, since the agents run natively in the web browser.
- HTML5 applications already have a strong ecosystem.
- The web technologies that are used in the development are widely familiar and standardized.

#### 6.1.2 Improvements Needed in the Current Implementation

Apart from the advantages as mentioned above, [5] also proposes improvements needed in the current implementation of the framework:

- The current agent-server implementation of the framework should be re-factored to make it more scalable and to make the addition of new features smoother.
- In the current implementation, there exists only communication and navigation model in a preliminary phase. For example, if it is need to track back the route travelled by an agent in the framework, we need an advance flexible navigation model to replace the current model.
- The communication model of the current version of the framework needs to be improved. At present, there is no communication channel for different servers to communicate to one another.

- In the current implementation, there is no specific method of informing all the agents about a specific agent that just perished.
- Strong mobility of the agents is not fully implemented because the agent code is transported as URLs instead of binding it to the transferrable state of the agent.

## 6.2 Evaluation of the User Interface Developed in This Thesis

### 6.2.1 Evaluation Against the Core Objectives of This Thesis

In this sub-section, the developed User Interface is evaluated against the objectives that are stated in the beginning of this thesis. The main objective is accomplished by developing a browser-based UI that functions as a front end to the management server of our framework. The UI is capable of displaying all the agents and the agent servers that are running in the framework. The information about the agent servers and the agents are displayed in two different views in the developed UI. The first view is the *agent servers view* that lists all the running agent servers along with the agents that belong to their corresponding servers. The second view is the *agents view* that displays the information of all the agents running on the framework, irrespective of whether they are running in a browser or in a server.

Through the developed UI, a user is able to move a specific agent from one server to another in the list of running servers. Among the list of running servers in agent server view of the UI, a user is also able to move the agent across different servers by the drag-and-drop feature. There is also a functionality in the UI for deleting a running agent. Both, the agent servers view and the agent view in the UI offers the facility of migration and deletion of an agent.

The developed UI does not come into a direct interaction with the agent servers. Instead, it only interacts with the management server for all the information that it needs regarding the agent servers and the agents. In our point of view, it is not a good idea to expose a direct access of the UI to the agent servers and the agents. The UI does not need to know information of agent servers e.g. their locations, their topology, etc. since there is the management server to deal these information.

### 6.2.2 Heuristic Evaluation of the Developed User Interface

In this sub-section, we evaluate the developed User Interface for the management server of our framework [5] in the light of the usability principles (heuristics) for user interfaces. Generally, the UI experts use two main techniques to evaluate a user interface: the first is *Empirical Evaluation* that is the evaluation with actual users and the second is *Heuristic*

*Evaluation* that is the evaluation based on certain set of rules. Since there are no broad groups of users available in our framework at the moment, therefore, we evaluate our developed User Interface as following, according to the *Nielsen Usability Heuristics of User Interface*, which are proposed by Jakob Nielsen [70], [71].

- **Visibility of System Status**

This heuristic can simply be called “Feedback”. The system should always keep the users informed about what is going on.

In our case, the developed UI informs the user, whenever an agent is moved to another location (server) or an agent is deleted. Also, the status of every running agent is displayed to let the user know about the status of each agent.

- **Match Between System & Real World**

This heuristic confers that a system should use real-world conventions, words and language that is familiar to a user. This whole match would make the information to appear in a logical order.

In our UI, the information about agents and agent servers appear in a simple list, which is easy to understand. A user would be able to see different servers in the agent servers view while the complete list of all the agents would appear in the agent view.

- **User Control and Freedom**

This heuristic means that the users should not be trapped by the interface. Any dialog box should have a *cancel* button, so that it can be used to *undo* an action if the user make mistakes.

In the developed UI in this thesis, if a user clicks on *delete* button to kill an agent, there is still a confirmatory dialog box appearing, asking the user “*Do you want to delete the agent?*” with two options: *Ok* and *Cancel*.

- **Consistency and Standards**

This heuristic determines the fact that user interface should not surprise the users with the way a command works. Similar things should act and look the same.

In our developed UI, the color scheme and pattern of the objects that appear on the UI are kept consistent.

- **Error Prevention**

There is a more chance that the users will make errors if given an opportunity, the solution to this is the error prevention. One way of error preventions in the UIs is to allow the users to *select* rather than *type*.

In our UI, when user wants to move an agent from one server to another, either it is done by drag-and-drop feature or there appears a list of available servers, from which the user can *select* by simply clicking on it.

- **Recognition Rather Than Recall**

This heuristic is concerned to minimize the user's memory load by making the actions, options and object visible. The users should not have to remember the information of the interface, from one part of the dialogue to another.

In our UI, this heuristic is not used because (a) there is not an overloaded amount of information and (b) the UI takes care of the information flow, when the user does an action. For example, in case of deleting an agent, when the user clicks on *delete* button, the UI simply prompts a confirmation dialog box for this action, the user simply does not need to remember, for example, the ID of that agent to be deleted and simply have to click *Ok* button to complete the deletion task. Hence the user recognize the *delete* button and does not need to recall what it is supposed to do.

- **Aesthetic and Minimalist Design**

This heuristic is about simplicity of the UI design. It states that the UI should not be overloaded with extra and irrelevant information, which may confuse the user. Our user interface is aesthetically a minimalist design because of its simplicity. There is very simple and straightforward information on the UI, which helps the user to understand and use it.

## 6.3 Conclusion

This thesis has presented the development of a web-browser based User Interface for the HTML5-based Mobile Agent Framework. This User Interface has been developed using well-known and widely used Web technologies, e.g. ReactJS, JavaScript, jQuery, AJAX and Bootstrap. This development has been done from the scratch, since there was no graphical user interface implemented in the HTML5 Mobile Agent Framework.

The main objective of this thesis was to implement the functionalities of the management server of our framework in a browser-based User Interface. The original framework had the management server developed but its functionalities could only be used through a command line. This development of the User Interface is done by exposing the web API of the Management Server of the underlying framework through an HTTP application interface.

Through working on this thesis, we wanted to get a hands-on experience on some of the latest Web technologies. The framework's implementation is mainly done by using Node.js. The User Interface is developed by using Web technologies like ReactJS, JQuery, Bootstrap, etc., which are learnt during this thesis and tried to be used in practical.

The strength of the developed User Interface is the implementation of main functions of the management server, which includes the list view of agents and agent servers, to view information of a specific agent (ID, state, status, etc.), enabling the migration of the agents between different servers graphically and the deletion of an agent through UI. HTML5 technologies have been used that owns a strong web-ecosystem.

The insufficiencies in this developed UI relates to the shortcomings in the management server of the framework. Lack of security model and inflexible navigation model of the original framework makes the management UI incapable of aiding agent communication. The developed UI has a lot of scope in terms of aesthetics, since the current version of the UI is more focused on bringing the functionality of the management server to a web browser based application interface.

## **6.4 Potential Future Improvements**

As the User Interface for the management server of our framework is developed with several functionalities that are discussed earlier, the following considered improvements in future may also prove useful in the overall operational scope of the developed UI:

- The UI might facilitate a user to upload an agent in client application e.g. browser to server, directly from the User Interface. Currently, agents can only be uploaded to a server, only from the browser.
- Some access restrictions on different agents and agent servers can be implemented through UI, e.g. to ensure the security aspects of the whole framework.
- It might also be a good idea to include a feature in the current UI to give a user more insight into what a particular agent is doing e.g. description of an agent's functionality.

## REFERENCES

- [1] A. Taivalsaari and T. Mikkonen, “The web as an application platform: The Saga continues,” *Proc. - 37th EUROMICRO Conf. Softw. Eng. Adv. Appl. SEAA 2011*, pp. 170–174, 2011.
- [2] D. Rus, “Mobile agents for mobile computing Technical Report PCS-TR96-285 1 Introduction,” *Network*, 1996.
- [3] M. Pilgrim, *HTML5 : up and running*. O’Reilly, 2010.
- [4] P. Fraternali, G. Rossi, and F. Sánchez-Figueroa, “Rich Internet Applications,” *IEEE Internet Comput.*, vol. 14, no. 3, pp. 9–12, May 2010.
- [5] L. Järvenpää, “Development and evaluation of HTML5 agent framework - Master of Science Thesis, Tampere University of Technology,” no. February, 2013.
- [6] “Node.js Introduction.” [Online]. Available: [https://www.tutorialspoint.com/nodejs/nodejs\\_introduction.htm](https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm). [Accessed: 21-Apr-2017].
- [7] A. C. Weaver, “The Internet and the World Wide Web,” in *Proceedings of the IECON’97 23rd International Conference on Industrial Electronics, Control, and Instrumentation (Cat. No.97CH36066)*, vol. 4, pp. 1529–1540.
- [8] “The Differences Between the Internet and the Web - Webopedia.” [Online]. Available: [http://www.webopedia.com/DidYouKnow/Internet/Web\\_vs\\_Internet.asp](http://www.webopedia.com/DidYouKnow/Internet/Web_vs_Internet.asp). [Accessed: 24-Apr-2017].
- [9] “Internet vs World Wide Web - Difference and Comparison.” [Online]. Available: [http://www.diffen.com/difference/Internet\\_vs\\_World\\_Wide\\_Web](http://www.diffen.com/difference/Internet_vs_World_Wide_Web). [Accessed: 28-Apr-2017].
- [10] T. Mikkonen and A. Taivalsaari, “Web Applications - Spaghetti Code for the 21st Century,” *2008 Sixth Int. Conf. Softw. Eng. Res. Manag. Appl.*, 2008.
- [11] “W3C Mission.” [Online]. Available: <https://www.w3.org/Consortium/mission>. [Accessed: 23-Apr-2017].
- [12] “Introduction to HTML.” [Online]. Available: [https://www.w3schools.com/html/html\\_intro.asp](https://www.w3schools.com/html/html_intro.asp). [Accessed: 23-Apr-2017].
- [13] D. Flanagan, *JavaScript : the definitive guide*. O’Reilly, 2006.
- [14] “CSS Introduction.” [Online]. Available: [https://www.w3schools.com/css/css\\_intro.asp](https://www.w3schools.com/css/css_intro.asp). [Accessed: 23-Apr-2017].
- [15] “Introducing JSON.” [Online]. Available: <http://www.json.org/>. [Accessed: 23-

Apr-2017].

- [16] M. Nottingham and P. Bryan, “JavaScript Object Notation (JSON) Patch,” pp. 1–18, 2013.
- [17] “W3C Document Object Model.” [Online]. Available: <https://www.w3.org/DOM/>. [Accessed: 23-Apr-2017].
- [18] “Introduction to the DOM - Web APIs.” [Online]. Available: [https://developer.mozilla.org/en-US/docs/Web/API/Document\\_Object\\_Model/Introduction](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction). [Accessed: 23-Apr-2017].
- [19] “AJAX Introduction.” [Online]. Available: [https://www.w3schools.com/xml/ajax\\_intro.asp](https://www.w3schools.com/xml/ajax_intro.asp). [Accessed: 23-Apr-2017].
- [20] R. Fielding, R. Fielding, H. Frystyk, T. Berners-Lee, J. Gettys, and J. C. Mogul, “Hypertext Transfer Protocol - HTTP/1.1,” 1996.
- [21] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, “Extensible markup language (XML),” *World Wide Web J.*, vol. 2, no. 4, pp. 27–66, 1997.
- [22] L. Masinter, T. Berners-Lee, and R. T. Fielding, “Uniform Resource Identifier (URI): Generic Syntax.”
- [23] T. Clark, “Uniform Resource Identifier (URI),” pp. 2319–2320, 2013.
- [24] “What is URL (Uniform Resource Locator)? - Definition from WhatIs.com.” [Online]. Available: <http://searchnetworking.techtarget.com/definition/URL>. [Accessed: 23-Apr-2017].
- [25] D. Goodman, *Dynamic HTML : the definitive reference*. O’Reilly, 2002.
- [26] H. M. Donelan, K. L. Kear, and M. Ramage, *Online communication and collaboration : a reader*. Routledge, 2010.
- [27] T. M. Harrison and B. Barthel, “Wielding new media in Web 2.0: exploring the history of engagement with the collaborative construction of media products,” *New Media Soc.*, vol. 11, no. 1–2, pp. 155–178, 2009.
- [28] J. Kluge, F. Kargl, and M. Weber, “The Effects of the AJAX Technology on Web Application Usability,” *Int. Conf. Web Inf. Syst. Technol.*, pp. 289–294, 2007.
- [29] J. Bosch, “From Software Product Lines to Software Ecosystems,” *Proc. 13th Int. Softw. Prod. Line Conf.*, no. Splc, pp. 111–119, 2009.
- [30] K. Systä, T. Mikkonen, and L. Järvenpää, “HTML5 agents: Mobile agents for the web,” in *Lecture Notes in Business Information Processing*, vol. 189, Springer, Berlin, Heidelberg, 2014, pp. 53–67.
- [31] “Node.js.” [Online]. Available: <https://nodejs.org/en/>. [Accessed: 27-Apr-2017].

- [32] S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to Build High-Performance Network Programs," *IEEE Internet Comput.*, vol. 14, no. 6, pp. 80–83, Nov. 2010.
- [33] J. Voutilainen, A. Mattila, K. Systä, and T. Mikkonen, "HTML5-based Mobile Agents for Web-of-Things," vol. 40, pp. 43–51, 2016.
- [34] V. A. Pham and A. Karmouch, "Mobile software agents: An overview," *IEEE Commun. Mag.*, vol. 36, no. 7, pp. 26–37, 1998.
- [35] A. Bieszczad, B. Pagurek, and T. White, "Mobile Agents For Network Management," *IEEE Commun. Surv. Tutorials*, vol. 1, no. 1, pp. 2–9, 1998.
- [36] L. Jarvenpaa, M. Lintinen, A. Mattila, T. Mikkonen, and K. Systä, "Mobile agents for the Internet of Things," *Int. Conf. Syst. Theory, Control Comput.*, pp. 763–767, 2013.
- [37] P. Braun and W. Rossak, *Mobile agents : basic concepts, mobility models, and the Tracy toolkit*. Elsevier, 2005.
- [38] A. Genco, *Mobile agents : principles of operation and applications*. WIT, 2008.
- [39] L. Silva and L. Almeida, "The Advantages of Using Mobile Agents in Software for Telecommunications," *Proc. Int. ...*, 1999.
- [40] D. Lange and M. Oshima, "Mobile agents with Java: the Aglet API," *World Wide Web*, vol. 1, pp. 1–18, 1998.
- [41] S. Green, L. Hurst, B. Nangle, and P. Cunningham, "Software agents: A review," TCD-CS-1997-06, 1997.
- [42] "Mobile Agent Computing, A White Paper." [Online]. Available: [https://www.cis.upenn.edu/~bcpierce/courses/629/papers/Concordia-WhitePaper.html#\\_Toc381690642](https://www.cis.upenn.edu/~bcpierce/courses/629/papers/Concordia-WhitePaper.html#_Toc381690642). [Accessed: 07-May-2017].
- [43] D. Wong, N. Paciorek, T. Walsh, J. DiCelie, M. Young, and B. Peet, "Concordia: An infrastructure for collaborating mobile agents," Springer, Berlin, Heidelberg, 1997, pp. 86–97.
- [44] L. M. Silva, G. Soares, P. Martins, V. Batista, and L. Santos, "The performance of mobile agent platforms," in *Proceedings. First and Third International Symposium on Agent Systems Applications, and Mobile Agents*, pp. 270–271.
- [45] C. Bäumer and T. Magedanz, "Grasshopper — A Mobile Agent Platform for Active Telecommunication Networks," Springer, Berlin, Heidelberg, 1999, pp. 19–32.
- [46] R. Trillo, S. Ilarri, and E. Mena, "Comparison and Performance Evaluation of Mobile Agent Platforms," in *Third International Conference on Autonomic and Autonomous Systems (ICAS'07)*, 2007, pp. 41–41.
- [47] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE—A FIPA-compliant agent framework," *Proc. PAAM*, pp. 97–108, 1999.



- [48] S. Kumar and U. Kumar, "Java Agent Development Framework," *Int. J. Res.*, vol. 1, no. 9, pp. 1022–1025, 2014.
- [49] J. Tardo and L. Valente, "Mobile agent security and Telescript," in *COMPCON '96. Technologies for the Information Superhighway Digest of Papers*, pp. 58–63.
- [50] D. B. Lange, M. Oshima, G. Karjoth, and K. Kosaka, "Aglets: Programming mobile agents in Java," Springer, Berlin, Heidelberg, 1997, pp. 253–266.
- [51] P. V. Rajguru and S. B. Deshmukh, "Analysis of Mobile Agent," *J. Glob. Res. Comput. Sci.*, vol. 2, no. 11, pp. 6–10, 2011.
- [52] H. Paulino, "A mobile agent systems' overview," *Department of Informations, Faculty of Science & Technology, Nova University of Lisbon*. pp. 1–29, 2002.
- [53] H. K. Tan and L. Moreau, "Trust Relationships in a Mobile Agent System," in *Mobile Agents - 5th International Conference Proceedings, MA, Atlanta, USA*.
- [54] W. Jansen and T. Karygiannis, "Mobile Agent Security," *Nist Spec. Publ.*, vol. 323, no. September, pp. 3–10, 2000.
- [55] K. Systä, T. Mikkonen, and L. Järvenpää, "HTML5 Agents – Mobile Agents for the Web," *Proc. 9th Int. Conf. Web Inf. Syst. Technol.*, pp. 37–44, 2013.
- [56] A. Carzaniga, G. Pietro Picco, and G. Vigna, "Designing distributed applications with mobile code paradigms," in *Proceedings of the 19th international conference on Software engineering - ICSE '97*, 1997, pp. 22–32.
- [57] D. Crockford, *JavaScript: The Good Parts*, vol. 44. 2008.
- [58] N. Heinle and B. Peña, *Designing with Javascript : creating dynamic Web pages*. O'Reilly, 2002.
- [59] "What is Serialization?" [Online]. Available: <https://www.techopedia.com/definition/867/serialization-net>. [Accessed: 08-May-2017].
- [60] "Serialization - Microsoft Docs." [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/articles/csharp/programming-guide/concepts/serialization/>. [Accessed: 08-May-2017].
- [61] B. Lin, Y. Chen, X. Chen, and Y. Yu, "Comparison between JSON and XML in Applications Based on AJAX," in *2012 International Conference on Computer Science and Service System*, 2012, pp. 1174–1177.
- [62] "Inversion of Control Containers and the Dependency Injection pattern." [Online]. Available: <https://martinfowler.com/articles/injection.html>. [Accessed: 10-May-2017].
- [63] M. Massé, *REST API design rulebook*. O'Reilly, 2012.
- [64] F. Doglio, *Pro REST API Development with Node.js*. .

- [65] R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” University of California, Irvine, 2000.
- [66] A. Fedosejev and A. Bush, *React.js Essentials : a fast-paced guide to designing and building scalable and maintainable web apps with React.js*. .
- [67] Vipul A. M; Prathamesh Sonpatki., *ReactJS by Example - Building Modern Web Applications with React*. Packt Publishing, 2016.
- [68] “Usage Statistics and Market Share of JavaScript Libraries for Websites, May 2017.” [Online]. Available: [https://w3techs.com/technologies/overview/javascript\\_library/all](https://w3techs.com/technologies/overview/javascript_library/all). [Accessed: 15-May-2017].
- [69] “NOTY - a notification library.” [Online]. Available: <http://ned.im/noty/>. [Accessed: 15-May-2017].
- [70] Jakob Nielsen, “10 Usability Heuristics for User Interface.” [Online]. Available: <https://www.nngroup.com/articles/ten-usability-heuristics/>. [Accessed: 21-May-2017].
- [71] J. Nielsen and R. Molich, “Heuristic Evaluation of User Interfaces,” in *Proceedings of the SIGCHI conference on Human factors in computing systems Empowering people - CHI '90*, 1990, pp. 249–256.