A Virtual World Web Client Utilizing An Entity-Component Model

Toni Dahl, Timo Koskela, Seamus Hickey and Jarkko Vatjus-Anttila
Center for Internet Excellence
University of Oulu
Oulu, Finland
Email: firstname.lastname@cie.fi

***Abstract*—The popularity of virtual worlds has increased considerably in recent years. Currently, many service providers are trying to make it easier for users to access their virtual worlds. However, accessing a virtual world typically requires a client application that needs to be versioned for each device platform and operating system configuration. Using WebGL, an interactive 3D environment can be used on a cross-platform supported web browser. In this paper, we present a system architecture that utilizes an entity-component model, and a prototype implementation of a WebGL-based virtual world client to provide a plug-in free, extensible and open source web client for 3D virtual worlds. The performance of the web client was evaluated in terms of frame rate, CPU load, memory consumption and scene processing speed. Based on the results, the performance of the web client was good on a desktop PC, but mobile hardware specific optimizations are required to provide a good user experience on mobile devices.**

***Keywords*—WebGL; web browser; performance; realxtend Tundra; 3D**

I. Introduction

In recent years, the popularity of virtual worlds has increased considerably [1] and they are evolving to become part of our reality. Currently, many virtual world services and software development kits (SDKs) for creating virtual world services are being provided, such as realXtend Tundra[1], Second Life[2], Open Wonderland[3] and Sirikata[4]. All of these virtual worlds are trying to improve the accessibility of their services to users. Accessing a 3D virtual world typically requires the installation of a standalone client application that needs to be versioned for each supported device platform and operating system configuration. This is not only an expense for service providers, but also a nuisance for end-users hindering the adoption of 3D virtual worlds. The rapid evolution of web 3D standards and technologies have made it possible to develop plug-in free 3D applications that run on a cross-platform supported web browser. Using WebGL, the interactive 3D environment can be integrated to be a part of a web page. Many web browsers and devices support WebGL already and it will also be supported by mobile devices in the near future.

Currently, there are no plug-in free and open source 3D virtual world clients available for web browsers that implement an extensible architecture for 3D virtual worlds. Also, the demand for easily deployable web clients has increased in the virtual world industry in recent years. In this paper, we present an architecture and a prototype implementation of a WebGL-based web client (hence called the web client) that uses an extensible entity-component (EC) model [2]–[4] for managing the object hierarchy in a 3D virtual world scene. The EC model makes the definition of the 3D scene very flexible, because the 3D scene consists only of entities that can be customized dynamically by modifying their components. By using the EC model, the virtual world architecture can be more simple and readily extensible. In order to evaluate the performance of the implemented web client, framerate, memory, CPU load and scene processing times were measured on a desktop PC and a tablet device.

The paper is organized as following. Section II introduces the related work and web technologies utilized in the implementation of the web client. In section III, the architectures of client-side and server-side implementations are presented. In sections IV and V, experimental setup and results are explained. In section VI discussion and future work are presented and finally, in section VII conclusions are made.

II. Related Research

Component-based scene architectures for Web 3D applications have been studied in few papers. Dachselt, Hinz and Meißner implemented an XML based architecture for component-oriented 3D applications utilizing X3D as the scene graph basis [5]. Lescinsky et al. describe a Virtue3D system [6] and Schiefer, Ullrich and Fellner present a software architecture that integrates a RESTful web service into an OpenSG scene graph [7]. These two approaches allows dynamically rearranging the scene at run time-time similarly to our EC model architecture. However, aforementioned architectures do not utilize the Web 3D technologies currently supported by the popular web browsers.

Byelozyorov et al. implemented an open modular virtual world client library for integrating virtual worlds in a web browser using XML3D and other technologies. [8] However, the problem with their implementation is that it requires a custom version of Google Chrome web browser. Additionally, they do provide a modular client framework.

Chen and Xu evaluated WebGL-based multiplayer online game framework [9]. Similarly to our approach, Chen and Xu utilized WebSocket, WebGL and three.js technologies in the implementation and they analytically showed that WebGL and WebSockets are feasible for creating multiplayer online games. However, their implementation is mainly intended for simple multiplayer online games.

[1]https://github.com/realXtend/naali
[2]http://secondlife.com/
[3]http://openwonderland.org/
[4]http://www.sirikata.com/

978-0-7695-5090-9/13 $26.00 © 2013 IEEE
DOI 10.1109/NGMAST.2013.11

III. System Architecture

Based on WebSocket[5] and WebGL[6] technologies, a virtual world web client for viewing an extensible virtual world was developed. WebGL provides a JavaScript API for 3D rendering for the web browser. Plenty of WebGL frameworks have been created, such as three.js, PhiloGL or GLGE, to simplify the development of WebGL applications. Three.js framework[7] was chosen for the implementation, because it is a very mature open source WebGL framework providing many useful features that can be utilized in the web client. Examples of these features are: a WebGL renderer, a scene graph, perspective and ortographic cameras, morph and keyframe animation, lights and shadows and 3D math utilities.

The WebSocket Javascript API define a two-way connection over a TCP socket between a web-based client and a remote host. A WebSocket can be used to implement a real-time synchronization of data between multiple web browser users over the Internet. This is particularly useful for a virtual world web client, where the network utilization is high.

In figure 1, the top level architecture of the virtual world system is presented. In the system, 3D asset data are stored in the *data server* which can be an ordinary HTTP server. Additionally, the web client code itself is delivered to the web browsers by the data server. The *simulation server* distributes the 3D scene to the clients and simulates all the physics and scene entity changes and transmits the results to the clients. It downloads the required asset data from the data server and passes references for the assets used in the scene to the clients. In the system, realXtend Tundra (hence called Tundra) virtual world server was used as a simulation server, because it is an open source and easily customizable platform. A WebSocket module that manages the synchronization of web clients, was implemented in the Tundra server. Tundra and the WebSocket module will be presented in more detail in Section III-A.

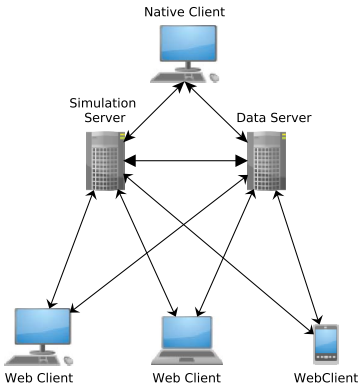

Fig. 1. The top level architecture of the virtual world system.

Native clients and *web clients* are connected to the simulation server and the data server. Both the native and the web clients communicate with the simulation server in a similar manner, the difference being that web clients use WebSockets and native clients use ordinary TCP and UDP-sockets for communication. The clients request the needed assets from the data server by using the references provided by the simulation server. The EC model is used by the clients and by the simulation server for managing the entities in the virtual world.

[5]http://www.w3.org/TR/websockets/
[6]https://www.khronos.org/registry/webgl/specs/1.0/
[7]http://threejs.org

A. RealXtend Tundra

Tundra implements an EC model based architecture, which provides a mechanism for building easily extensible virtual worlds. It was chosen as the simulation server because it can be flexibly extended by creating additional modules and by customizing or creating new component implementations for entities. Also, the behaviour of entities, for example avatars, on the server or on the native Tundra client can be easily scripted using the integrated JavaScript scripting environment. [2]

The EC model attempts to simplify the scene object system by dividing the different object functionalities in different components making the management of the objects and the maintenance of the system architecture easier. At a general level, virtual world objects or entities are only unique identifiers which have no data. Components can be added to entities and they can be any type and contain any type of data. Each scene object consists of predefined components which are managed by the component manager. [3], [4]

An example of the EC model is given in figure 2. Objects 1, 2, and 3 are entities that consist of different components that each implement a certain functionality. An example of basic components in Tundra are Mesh, Placeable, Light, Sky and Camera. All the basic entity-components that are used in Tundra, are implemented in the web client to provide good interoperability between the server and the client.

For web client synchronization, a WebSocket module was implemented in the Tundra server. WebSocket++ library was used in the implementation of the WebSocket module because it provides an easy event-based API and because Tundra is written in C++-language. In the module, Tundra scene manager provides data of the changes in the virtual world to the WebSocket module. The data are the same for native Tundra clients and web clients. The WebSocket module processes this data and converts them to JSON (JavaScript Object Notation) formatted event messages that are sent to web clients through a WebSocket.

The JSON format is used as the message data format between the server and web clients to reduce the processing time needed for parsing on the web client side. JSON is a lightweight data-interchange format which is easy for machines to parse and generate. However, because JSON is not a compressed data

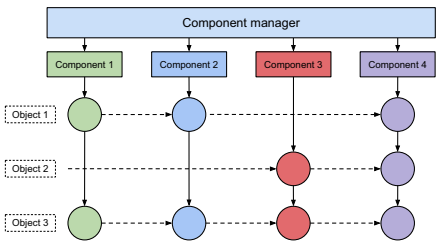

Fig. 2. An example of the EC model.

format, using it increases the amount of data transferred via a WebSocket and it is not optimal for use cases where the network bandwidth is limited.

B. Web Client Architecture

The relationship between the modules of the web client is illustrated in figure 3. The web client framework provides an easily accessible API for the application developers by which they can define settings for each core module. Next, the modules of the web client will be described in detail.

1) WebSocket Manager: The WebSocket manager handles the WebSocket connection with the simulation server and parses the JSON event messages that come through the WebSocket. An event name and related data are extracted from the JSON message and a function corresponding to the event name is executed using the data as a function argument. These events are utilized mainly in the EC manager.

2) EC Manager: The EC manager is responsible for creating entities and their specific components from the parsed JSON messages. It reacts to specific events triggered by the WebSocket manager, which are ”EntityAdded”, ”EntityRemoved”, ”ComponentsRemoved”, ”ComponentsAdded”, ”AttributesRemoved”, ”AttributesAdded” and ”AttributesChanged”, of which the most important are EntityAdded and AttributesChanged events.

The data contained in the EntityAdded event define a whole entity which is existing in the simulation server scene. This data include, for example, unique entity id, entity name, and its component data. AttributesChanged events are sent from the simulation server when some of the attributes of components in the virtual world scene, have been changed. An example of an attribute synchronization event is shown in figure 4. In the figure 4, a JavaScript script running on the simulation server moves an entity by giving a new transform value to its Placeable component. The propagation of the transform value is shown by a solid line arrow and the dotted arrow represents a network event message sent through a WebSocket.

3) Asset Manager: The asset manager requests asset data from the data server if a specific component of an entity requests

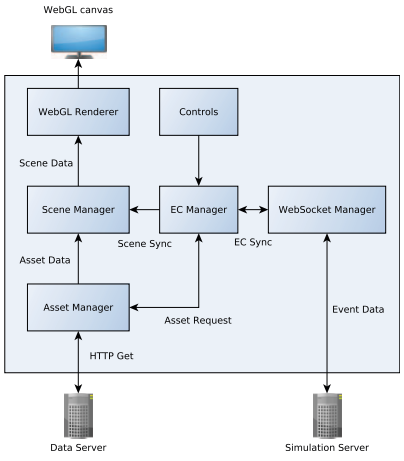

Fig. 3. Basic architecture of the web client.

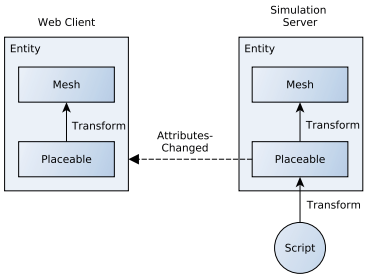

Fig. 4. An example of a synchronization event between two entities of which one is located in the simulation server and the other one is in the web client.

an asset. A request queue functionality was implemented in the asset manager, so an asset with the same name would be requested only once. The asset manager caches all the loaded assets in the memory and indexes them by their name. Assets are categorized by their types, which are textures, materials and 3D meshes, so different types of assets could be easily searched when needed locally in the web client. When a request for an asset is made, the asset manager will check if the asset is already downloaded and returns a reference to the asset if it was found, otherwise it will request the asset from the data server.

After downloading the asset, it will be parsed. In our implementation, support for Ogre mesh format was implemented because Tundra uses it internally. Various texture formats are supported such as DDS, JPEG, GIF or PNG. New parsers for different 3D file formats and texture formats can be added easily to the asset manager if needed.

4) Controls: Entities can be controlled by using the controls module and a Controller component that can be added into an entity, as illustrated in figure 5. The controller component can request a control script from the controls module. The control script implements a specific control type such as free look controls which allow a user to use keyboard and mouse to move an object around the scene freely. If a control component is added to an entity it will apply the 3D transformation changes caused by the control script to the entity. In the web client, Placeable component stores the main transform information of the entity and applies the transformation to its children, which in the figure 5 is a mesh component.

5) Scene Manager: The scene manager is paired with a WebGL renderer. Components can request their 3D data to be

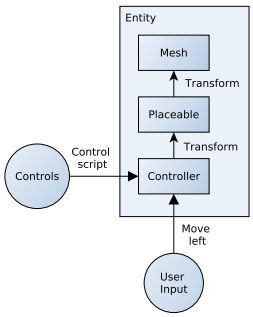

Fig. 5. An example of a controller component as a part of an entity in the web client. Transforms caused by the controls are propagated to the Placeable component and its children.

added in the scene through the scene manager. The WebGL renderer manages the cameras, sky box and lighting of the scene and it runs the rendering loop that calls the three.js WebGL renderer periodically. All the 3D data added to the scene are rendered by the WebGL renderer which feeds the three.js scene-graph data to the GPU.

IV. Experimental Setup

Performance measurements were conducted in order to evaluate the feasibility of the web client. Network performance was omitted from the experiments, because the main focus in our research is on the performance of the web client.

A. Devices and tools

In table I, the test devices are presented. Because the tablet device used in the testing has a quite limited memory, it is important to know how much free RAM for the web client is actually available while conducting the measurements. This is not relevant for a PC, because the test scenes will never exceed the memory limitations.

For performing the CPU and the memory measurements, tools provided by Google Chrome web browser are used. The CPU load is inspected by using the CPU profiler tool and the memory consumption is measured by using the heap snapshot tool. For measuring the frame rate, an additional functionality was added to the web client because Google Chrome did not inherently provide a good way for measuring the frame rates. The approximate processing times of the test scenes is measured by using a stopwatch.

B. Methodology

The web client is evaluated in terms of

1) *CPU load.* For measuring the CPU load, a pre-calculated circular path was added on the XY-plane around the test scenes and the radius of the path was given a large enough value to make the whole scene visible in the camera frustum. The camera is moved four laps along the path to get an average result of the measurements and it looks always to the centre of the scene. The most heavy-weight software routines of the web client are then extracted and categorized.
2) *Frame rate.* The frame rate the web client is measured using the same setup as in the CPU load measurements.
3) *Memory consumption.* The memory consumption of the test scenes on the web client are measured after the scene has finished loading. For comparison, the memory consumption of the web client without any test scene loaded is measured.
4) *Scene processing time.* For measuring the processing time of each test scene the network delay of loading the scene is omitted from the measurements.

In table II, the test scenes used in the evaluation are listed. The 3D City High scene is a model of a nine block area at downtown Oulu, Finland. It serves as a good stress test scene and the 3D City Low scene is used for testing how the reduction of polygons affects the performance of the web client. The 3D City High scene is rendered using high rendering settings. For the settings, the shader program precision was set to high and the size of the rendering area was set to full screen. The Outdoor scene is a medium quality scene and represents a usual Tundra scene in terms of the scene size and the number of the entities. For comparing the web client performance on the PC and on the tablet device, both Outdoor scene and the 3D City Low scene were tested only with low rendering settings. The low rendering settings were used so that measurable frame rates could be achieved with the tablet and the same settings were used on PC so the results would be comparable. For the low rendering settings, the precision of the shader programs were set to the lowest value and the rendering resolution was halved. Also, the dimensions of the rendering area on the PC were set to the same values as on the tablet.

V. Results

In this section, the experimentation results are presented and analysed.

A. CPU load

The CPU measurement results are presented in figures 6 and 7. Differences were observed with the CPU idle time and the other processing time between the tablet and the PC. Interestingly, the more heavy the test scene was, the more the CPU idle time increased. Because rendering a lower quality scene takes less time for the GPU to render, the web client can call the render function and other related functions more often. This increases the CPU load and reduces the CPU idle time. With a higher quality scene, the rendering takes longer on the GPU causing the the CPU to wait for rendering to complete, thus decreasing the CPU load and increasing the CPU idle time.

However, on the tablet, other tablet processes seem to dominate the overall CPU usage and reduce the web client performance. Also, because the tablet CPU is weak compared to

TABLE I. Test devices

Device	CPU	GPU	RAM	Free RAM	Operating System
Tablet: High-end tablet	NVIDIA Tegra 3 Quad Core Q8500, 1.2 GHz	NVIDIA Tegra 3 ULP GeForce, 416 MHz	1 GB	408 MB	Android 4.1.1
PC: Lenovo Think-Center	Intel Quad Core Q8500, 2.66 GHz	Quadro 600, 640 MHz	4 GB	Not relevant	Ubuntu 12.04

TABLE II. Test scenes

Name	Polygons	Size on Disk	Files	Enti-ties
3D City High	315833	260.77 MB	Meshes: 59 files (130MB) Materials: 61 files (67 kB) Textures: 71 DDS files (131 MB)	74
3D City Low	30395	16.51 MB	Meshes: 58 files (15.8 MB) Materials: 60 files (65.8 kB) Textures: 70 DDS files (649 kB)	73
Outdoor	100877	36.3 MB	Meshes: 78 files (33.6 MB) Materials: 53 files (52.5 kB) Textures: 5 PNG files and 5 JPEG files (1.8 MB)	90

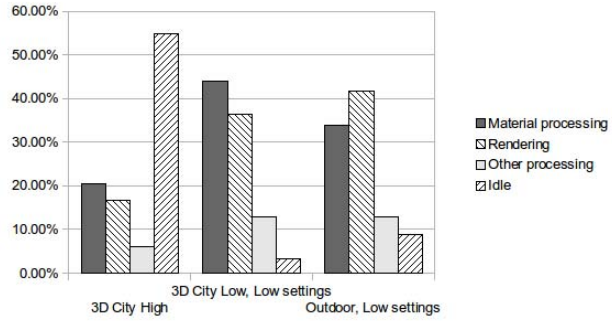

Fig. 6. The CPU load of each test scene on the PC.

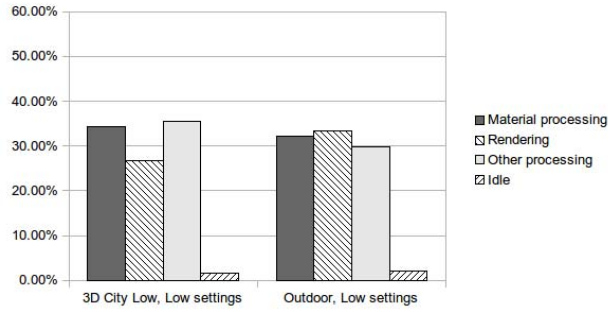

Fig. 7. The CPU load of each test scene on the tablet.

the PC CPU, the idle-time on the tablet is zero in practice. The effect of the weak CPU can be seen in the frame rate differences between the PC and the tablet, as shown in the next section.

B. Frame rate

In figure 8, the average frame rates for each test scene are illustrated. The 3D City Low scene has slightly higher FPS (Frames per Second) value than the Outdoor scene on the PC, but smaller FPS value than the Outdoor scene on the tablet. Also, it can be seen in figure 8 that the web client performs quite well on the PC even with the 3D City High scene.

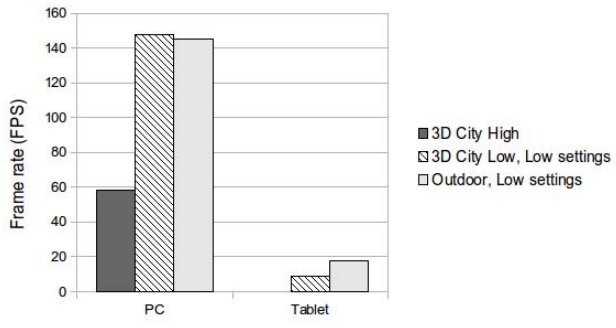

Fig. 8. Measured average frame rates for each test scene.

The Outdoor scene has more meshes and they are more complex than in the 3D City Low scene. However, the 3D City Low scene uses much more textures and more materials than the Outdoor scene. Zorrilla et al. concluded that WebGL frame rate on mobile hardware is limited the most by the number of the objects rendered simultaneously. [10]. Here, we observe that also the amount of the materials has also an effect on the frame rate on mobile devices. Because the material processing is CPU oriented and the 3D City Low scene has more materials and more complex surfaces compared to the Outdoor scene, the 3D City Low scene has lower FPS. Because the CPU is not a bottleneck on the PC, the frame rate difference between these scenes is minimal.

C. Memory consumption

In figure 9, the memory consumption of each test scene loaded in the web client is shown. The memory footprint of the 3D City High scene was not measured on the tablet because it had not enough processing power and memory to load the test scene. The Outdoor scene was not measured on the tablet, because it had not enough free memory to generate the heap snapshot, although it had enough capacity to load the scene.

There was a small difference of 16 megabytes in memory footprints between the tablet device and the PC. The 3D City Low scene consumed a bit more memory on the PC than on the tablet. We infer that this is caused by more aggressive garbage collection on the mobile version of Google Chrome, because mobile devices have more limited memory than desktop computers.

D. Processing time

Figure 10 shows the time consumed by the web client in processing the test scenes. The processing time includes the time used in parsing the meshes, parsing the material files, processing the texture files and creating the actual scene objects. As shown in figure 10, processing the Outdoor scene on PC took about the same time as processing the 3D City Low scene. On the tablet, the difference is much greater. This difference is presumably related to the memory consumption of the scene objects. Objects of the Outdoor scene consume much more memory than the 3D City Low scene and the memory space in the tablet is very limited, so processing the Outdoor scene on the tablet may invoke more browser garbage collection cycles thus increasing the processing time of the scene.

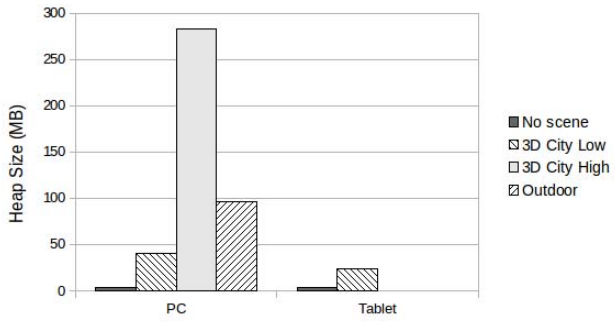

Fig. 9. The memory consumption of the web client for loading a test scene.

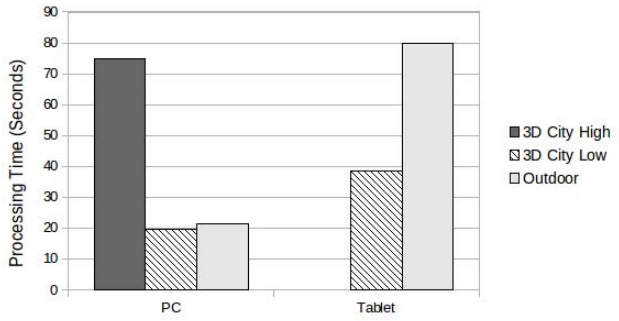

Fig. 10. Processing time of the test scenes.

VI. Discussion

The measurements show that virtual world web clients have a bright future on PCs, but further development must be done on mobile devices to achieve tolerable performance. Optimization must be done on the web client side, but also the mobile hardware must evolve so it is more capable of processing larger 3D scenes on a web browser. For larger 3D scenes the weak CPU and the lack of RAM memory are the obvious bottlenecks on mobile devices.

The web client can be optimized for mobile devices for example by improving the scene management and rendering by more advanced algorithms, such as oct-trees, occlusion culling or tiled rendering. If a mobile device has better GPU than CPU, some CPU intensive calculations could be possibly migrated into GPU shader programs. On mobile devices, the limitations in terms of performance are still GPUs, CPUs and memory, bus speed and FPUs (Floating Point Unit). Evolution of these hardware components are limited by energy consumption issues and advances in mobile battery technology [11], [12], but however these components have been improved a lot in recent years. Nevertheless, also the WebGL performance must be improved in the mobile web browsers to achieve maximum efficiency for rendering [10].

The EC model was very helpful for dynamically managing the 3D scene. Because we implemented the basic components that are used in the Tundra server, most of the basic Tundra scenes work on the web client. If a scene uses a component that has not been added into the web client yet, it can be easily implemented and added into the web client. Also, if one makes changes to the scene entities running on the simulation server, the changes are delivered to the web client and easily applied to the scene entities.

Our web client works on every web browser that properly supports WebGL and WebSockets, thus making it available for many users both on mobile and desktop devices, compared to other approaches that require installing a custom web browser or a plug-in. Our intentions are that the web client could be used and extended easily by web developers for creating web clients for various virtual world servers. For the future development, we aim to evaluate the effect of network quality on the performance of the web client. The web client implementation can be found from: https://github.com/Chiru/Chiru-WebClient .

VII. Conclusion

In this paper, we have presented an extensible architecture for a virtual world web client that utilizes WebGL and WebSocket technologies and the EC model for managing the virtual world entities. The EC model makes the web client scene model easily extensible and dynamic, because the scene is defined by its entities. The architecture allows developers to easily extend their virtual worlds and the web client by implementing more components. Altogether, the web client will serve as a good foundation for creating virtual world web clients for different virtual world servers and it is easily accessible for many users through a modern web browser. As shown by the performance measurements, the web client has a good performance on PC, but more work must be done on mobile devices to achieve a good performance with web-based 3D virtual worlds.

Acknowledgement

The authors would like to thank the staff at the Intel and Nokia Joint Innovation Center for their support. This work has been carried out in the Tekes Chiru Project.

References

[1] KZero Worldswide, "Kzero universe chart q4 2012 (accessed 28.2.2013.)," 2012, url: http://www.slideshare.net/nicmitham/kzero-universe-q4-2012.

[2] T. Alatalo, "An entity-component model for extensible virtual worlds," *Internet Computing, IEEE*, vol. 15, no. 5, p. 30, Sept.-Oct. 2011.

[3] M. West, "Evolve your hierarchy (accessed 13.11.2012)," 2007, url: http://cowboyprogramming.com/2007/01/05/evolve-your-heirachy/.

[4] C. Stoy, "Game programming gems 6." Charles River Media, 2006, pp. 393–394.

[5] R. Dachselt, M. Hinz, and K. Meissner, "Contigra: an xml-based architecture for component-oriented 3d applications," in *Proceedings of the seventh international conference on 3D Web technology*, ser. Web3D '02. New York, NY, USA: ACM, 2002, pp. 155–163.

[6] G. Lescinsky, C. Touma, A. Goldin, M. Fudim, and A. Cohen, "Interactive scene manipulation in the virtue3d system," in *Proceedings of the seventh international conference on 3D Web technology*, ser. Web3D '02. New York, NY, USA: ACM, 2002, pp. 127–135.

[7] A. Schiefer, R. Berndt, T. Ullrich, V. Settgast, and D. W. Fellner, "Service-oriented scene graph manipulation," in *Proceedings of the 15th International Conference on Web 3D Technology*, ser. Web3D '10. New York, NY, USA: ACM, 2010, pp. 55–62.

[8] S. Byelozyorov, V. Pegoraro, and P. Slusallek, "An open modular architecture for effective integration of virtual worlds in the web," in *International Conference on Cyberworlds (CW)*, 2011, pp. 46–53.

[9] B. Chen and Z. Xu, "A framework for browser-based multiplayer online games using webgl and websocket," in *2011 International Conference on Multimedia Technology (ICMT)*, 2011, pp. 471–474.

[10] M. Zorrilla, A. Martin, J. R. Sanchez, I. Tamayo, and I. G. Olaizola, "Html5-based system for interoperable 3d digital home applications," in *2012 Fourth International Conference on Digital Home (ICDH)*, 2012, pp. 206–214.

[11] A. Mulloni, D. Nadalutti, and L. Chittaro, "Interactive walkthrough of large 3d models of buildings on mobile devices," in *Proceedings of the twelfth international conference on 3D web technology*, ser. Web3D '07. New York, NY, USA: ACM, 2007, pp. 17–25.

[12] D. Nadalutti, L. Chittaro, and F. Buttussi, "Rendering of x3d content on mobile devices with opengl es," in *Proceedings of the eleventh international conference on 3D web technology*, ser. Web3D '06. New York, NY, USA: ACM, 2006, pp. 19–26.