MARLi - Molly Alternate Realities Language *interactive*: A New XML Markup Language for Defining Virtual and Augmented Reality

Ronald P. Vullo, Ph.D.
Department of Information Sciences and Technologies
Rochester Institute of Technology
Rochester, New York 14623
rpv@mail.rit.edu

Christopher A. Egert, Ph.D.
School of Interactive Games and Media
Rochester Institute of Technology
Rochester, New York 14623
caeics@rit.edu

Andrew Phelps, M.S.
College of Art & Design
Rochester Institute of Technology
Rochester, New York 14623
amp5315@rit.edu

***Abstract*— Over the last few years, virtual reality and related technologies have seen a resurgence with the advent of easily accessible headsets and adapters for PC, mobile devices, and tablets. However, developers and content creators have struggled with the means of authoring for these systems for mass-consumption and web deployment. This paper examines the historical state of web authoring frameworks for VR and describes the authors' approach to the problem.**

Keywords— Virtual Reality, Web Virtual Reality, Web Multimedia Content

I. INTRODUCTION

Virtual Reality/Augmented Reality/Mixed Reality (heretofore abbreviated as VR) has once again made a reappearance on the consumer front. Readily available headsets and adapters for mobile devices have lowered the bar for consumer entry and has re-sparked the imagination for the potential of this technology. However, in order to harness the power of VR, authoring tools must allow easy entry for developers while still allowing reasonable complexity in an VR application. This is apparent in the current state of VR for the web, where potential solutions try to balance the ease of HTML markup approaches with the need for scripting access to control interactivity and dynamic state management within scenes and virtual worlds. This is not unusual in the natural evolution of new technologies, but also not unusual is the subsequent development of authoring tools and approaches that abstract the complexity and so speed the further development and adoption of those technologies. The authors' background in web development led us develop an elegant yet powerful VR markup language to allow us to take advantage of the rapidly improving hardware. Together, the authors combined experience includes web XML parsing engine development, game engine development, game development, and VR development going back over two decades. Thus the decision was made to move forward with the development of a VR markup-based language syntax, server-side parsing engine, and client-side rendering engine. This paper introduces the first of these three components - the new VR language.

II. BACKGROUND

Before beginning to develop our new VR language, we decided to look back at the language we all first used to build VR scenes: Virtual Reality Markup Language (VRML) [1]-[2]. As we dusted off and began paging through our twenty year old VRML books, we realized that it would be valuable to explore the domain of VR markup languages for web delivery. What we found was admittedly limited, but proved instructive in our decision to move forward. What follows is a brief summary of the key features of the various languages that have been developed over the years.

A. Summary of Previous VR Languages

1) VRML

Virtual Reality Markup Language [1]-[3] was the first attempt at creating a dedicated VR development language back in the early days of the web and Hypertext Markup Language (HTML). Despite its name being similar to (likely derivative of) HTML, VRML bears no similarity to what we have come to consider markup languages to be. It does not share HTML's original SGML ancestry, or modern XML incarnations. It is more of a "bare metal" programming language-like node specification syntax employing braces for blocks instead of traditional markup language tags. As the language evolved and attempts were made to build VR environments with VRML, content creators increasingly relied upon IndexedFaceSet objects rather than geometric primitives to create their scenes, thus reducing the readability of the markup. To those reading VRML code, these index sets read as long lists of numbers which quickly became impenetrable when coding. While some WYSIWYG authoring environments were available [4] and so made VRML essentially dependent on the use of a WYSIWYG authoring tool. This is the antithesis of a "markup language" and, we believe, eventually led to its failure as a functional standard for building VR. In the end, it just felt more complex than it needed to be.

```
#VRML V2.0 utf8
#Color example: a pyramid
Shape {
          appearance Appearance {
                   material Material { }
                   }
          geometry IndexedFaceSet {
                   coord Coordinate {
                    point [
                                       # bottom
                                       -1.0 -1.0 1.0,     #vertex 0
                                       1.0 -1.0 1.0,      #vertex 1
                                       1.0 -1.0 -1.0,     #vertex 2
                                       -1.0 -1.0 -1.0,    #vertex 3
                                       # top
                                       0.0 1.0 0.0                #vertex 4
                    ]
                  }
          colorPerVertex FALSE #so each face will have one of the colors
          color Color {
                    color [
                        1.0 0.0 0.0,#color 0
                        0.0 1.0 0.0,#color 1
                        0.0 0.0 1.0,#color 2
                        1.0 1.0 0.0,#color 3
                        0.0 1.0 1.0,#color 4
                    ]
                  }
          colorIndex [ 2 3 4 0 0 ] #the first face will have color 2, the 2nd color3...
                   coordIndex [
                            #bottom square
                             0, 3, 2, 1, 0, -1,
                            #side1
                             0, 1, 4, -1,
                            #side2
                             1, 2, 4, -1,
                            #side3
                             2, 3, 4, -1,
                            #side1
                             3, 0, 4, -1,
                            ]

          }
    }
```

Fig. 1. Sample VRML Syntax [5]

2) xVRML

In 2003 a former colleague of ours began a project to re-implement VRML using an XML-based notation and a schema-based definition. The resulting work was named xVRML [6] and the approach attempted to address the concerns that content developers expressed when using VRML. However, there was no attempt at re-engineering the underlying structure and assumptions of VRML, merely to develop an XML compliant syntax that would be synonymous with web markup standards. Work seems to have focused primarily on formalization of the schema, not the functionality of the language itself and development stalled more-or-less permanently in mid 2006 at the beginning of the "demonstration implementation" (in Java) phase of the project.

```
<?xml version="1.0" encoding="UTF-8"?>
<World xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.xvrml.net/schemas/core"
xsi:schemaLocation="http://www.xvrml.net/schemas/core
http://www.xvrml.net/schemas/xVRML.xsd" name="" >
  <WorldInfo name="" >
    <title>Coffeetable</title>
    <info>Converted by Michael Dana Murphy</info>
  </WorldInfo>
  <children>
    <Viewpoint rotation="0 0 -1 0" description="Default" position="0 1 10"
name="Default" />
    <Transform rotation="0.0 0.0 -1.0 0.0" scale="0.1 0.1 0.1" translation="0.0 -
1.93787e-08 0.0" >
      <children>
        <Shape>
          <appearance>
            <Material ambientIntensity="0.0" name="mat1" >
              <diffuseColor alpha="1.0" red="0.13725" blue="0.18431"
green="0.13725" />
            </Material>
          </appearance>
          <geometry>
            <IndexedFaceSet ccw="false" creaseAngle="3.14" >
              <coord>-28.8 13.7999 20.8312 -28.8 13.8014 -36.7687 28.8 13.8014 -
36.7687 28.8 13.7999 20.8312 28.8 14.4014 -36.7687 -28.8 14.4014 -36.7687 -28.8
14.3999 20.8312 28.8 14.3999 20.8312 24.2122 13.8 15.819 -23.7878 13.8013 -32.181
-24.2121 13.8013 -31.7567 23.7879 13.8 16.2433 24.2122 1.02074e-05 15.8186 23.7879
1.93787e-07 16.243 -24.2121 0.00127001 -31.757 -23.7878 0.00127001 -32.1814
24.2121 13.8013 -31.7567 23.7878 13.8013 -32.181 -24.2122 13.8 15.819 -23.7879
13.8 16.2433 24.2121 0.00127001 -31.757 -23.7879 1.93787e-07 16.243 -24.2122
1.02074e-05 15.8186 23.7878 0.00127001 -32.1814</coord>
              <coordIndex>2 1 0 -1 3 2 0 -1 6 5 4 -1 7 6 4 -1 6 0 1 -1 5 6 1 -1 1
2 4 -1 5 1 4 -1 2 3 7 -1 4 2 7 -1 3 0 6 -1 7 3 6 -1 10 9 8 -1 11 10 8 -1 14 13 12
-1 15 14 12 -1 14 15 9 -1 10 14 9 -1 10 11 13 -1 14 10 13 -1 8 12 13 -1 11 8 13 -1
9 15 12 -1 8 9 12 -1 18 17 16 -1 19 18 16 -1 22 21 20 -1 23 22 20 -1 21 19 16 -1
20 21 16 -1 17 23 20 -1 16 17 20 -1 22 23 17 -1 18 22 17 -1 21 22 18 -1 19 21 18 -
1</coordIndex>
            </IndexedFaceSet>
          </geometry>
        </Shape>
      </children>
    </Transform>
    <DirectionalLight name="aDirectionalLight" />
  </children>
    </World>
```

Fig. 2. Fig. 2 Sample xVRML Syntax [7] (See: Examples > Prowl the vrmLab > Coffeetable_vrc.xwrl)

3) X3D/X3DOM

X3D [8] is a royalty-free ISO standard for declaratively representing 3D computer graphics and is sometimes noted to be the official successor to VRML [9]. That said, it doesn't really feel to us like a complete VR language, but more a syntax/tool for embedding 3D objects in regular web pages. The syntax of X3D requires several layers of nested tags to display even the simplest scene, making it verbose and less than intuitive. Although the JavaScript runtime script for implementing X3D was last updated in 2016, it still functions at the time this paper was authored.

```
<html>
   <head>
    <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
     <title>My first X3DOM page</title>
     <script type='text/javascript' src='https://www.x3dom.org/download/x3dom.js'>
</script>
     <link rel='stylesheet' type='text/css'
href='https://www.x3dom.org/download/x3dom.css'></link>
   </head>
   <body>
     <h1>Hello, X3DOM!</h1>
     <p>
       This is my first html page with some 3d objects.
     </p>
          <x3d width='500px' height='400px'>
            <scene>
                  <shape>
                     <appearance>
                            <material diffuseColor='1 0 0'></material>
                     </appearance>
                     <box></box>
                  </shape>
                  <transform translation='-3 0 0'>
                    <shape>
                            <appearance>
                              <material diffuseColor='0 1 0'></material>
                            </appearance>
                            <cone></cone>
                    </shape>
                  </transform>
                  <transform translation='3 0 0'>
                    <shape>
                            <appearance>
                              <material diffuseColor='0 0 1'></material>
                            </appearance>
                            <sphere></sphere>
                    </shape>
                  </transform>
            </scene>
          </x3d>
   </body>
</html>
```

Fig. 3. Sample X3D Syntax [10]

4) 3DMLW

3D Markup Language for Web (3DMLW) [11] is described as an open-source XML-based Markup Language for representing interactive 3D and 2D content on the World Wide Web. While this project had potential, it seems its creators transitioned the project into an authoring application around 2012 and it appears to have been abandoned in 2015. From the minimal examples we found it seems to have been a reasonable start at a syntax, but the lack of available documentation makes it difficult to make a definitive determination.

```
<?xml version='1.0' standalone='no'?>
<document>
        <content2d>
                <area width='200' height='100' color='#C0C0C0FF'
texture='flower.png' />
        </content2d>
        <content3d id='content' camera='{#cam}'>
                <camera id='cam' class='cam_rotation' y='10' z='40'
viewy='10'/>
                <box name='ground' width='100' height='2' depth='100'
color='green' class='ground' />
                <box name='dynamic' y='20' width='10' height='10' depth='10'
color='blue' />
        </content3d>
</document>
```

Fig. 4. Sample 3DMLW Syntax [12]

5) A-Frame

Originally developed by Mozilla A-Frame [13] is now maintained by its co-creators under the auspices of Supermedium (https://supermedium.com). We were excited early on by A-Frame as they seemed to be headed in the right direction - which is to say, developing a tag language from content authors' perspective. However, A-Frame has been undergoing revisions that seem to be moving toward a scripting focus, placing most of the functionality for interaction and events in the code rather than in the markup. A-Frame has the distinct advantages of being designed for modern VR hardware and web browsers and is actively developed. It also provides some of the modern amenities of lower level web graphics programming APIs while maintaining a reasonable level of abstraction. However, A-Frame does present a challenge with its continuing evolving specification that sometimes breaks backwards compatibility along with its name, which presents a challenge when trying to search for examples and tutorials within modern search engines.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Dynamic Lights - A-Frame</title>
    <meta name="description" content="Dynamic Lights - A-Frame">
    <script src="../../../dist/aframe-master.js"></script>
    <script src="https://unpkg.com/aframe-randomizer-components@3.0.2/dist/aframe-
randomizer-components.min.js"></script>
    <script src="https://unpkg.com/aframe-entity-generator-
component@3.0.1/dist/aframe-entity-generator-component.min.js"></script>
    <script>
      AFRAME.registerComponent('random-material', {
        init: function () {
          this.el.setAttribute('material', {
            color: this.getRandomColor(),
            metalness: Math.random(),
            roughness: Math.random()
          });
        },
        getRandomColor: function () {
          var letters = '0123456789ABCDEF'.split('');
          var color = '#';
          for (var i = 0; i < 6; i++) {
            color += letters[Math.floor(Math.random() * 16)];
          }
          return color;
        }
      });
      AFRAME.registerComponent('random-torus-knot', {
        init: function () {
          this.el.setAttribute('geometry', {
            primitive: 'torusKnot',
            radius: Math.random() * 10,
            radiusTubular: Math.random() * .75,
            p: Math.round(Math.random() * 10),
            q: Math.round(Math.random() * 10)
          });
        }
      });
    </script>
  </head>
  <body>
    <a-scene background="color: #111">
      <a-assets>
        <a-mixin id="light"
                 geometry="primitive: sphere; radius: 1.5"
                 material="color: #FFF; shader: flat"
                 light="color: #DDDDFF; distance: 120; intensity: 2; type:
point"></a-mixin>
        <a-mixin id="torusKnot"
                 random-torus-knot
                 random-material
                 random-position="min: -60 -60 -80; max: 60 60 40"></a-mixin>
      </a-assets>

      <!-- Use entity-generator component to generate 120 entities with the
torusKnot mixin. -->
      <a-entity entity-generator="mixin: torusKnot; num: 120"></a-entity>

      <!-- Lights. -->
      <a-entity animation="property: rotation; to: 0 0 360; dur: 4000; easing:
linear; loop: true">
        <a-entity mixin="light" position="30 0 0"></a-entity>
      </a-entity>

      <a-entity animation="property: rotation; to: 360 0 0; dur: 4000; easing:
linear; loop: true">
        <a-entity mixin="light" position="0 0 40"></a-entity>
      </a-entity>
    </a-scene>
  </body>
    </html>
```

Fig. 5. Sample A-Frame Syntax [14]

6) VR Languages In General

All of the languages reviewed seem to share a similar trait. They each have fallen into the trap of having their syntaxes optimized for the back-end programmer, more than the content creator (be that a human or a script). None of the XML (or XMLish) systems seem to have properly used XML namespaces to avoid tag name collisions. Strangely, the most modern of the languages, A-Frame, prefixes the names of all their tags with "a-" as a pseudo namespace, rather than declaring an actual XML namespace. Other than A-Frame, none of the languages are implemented for modern VR use with headsets and phone accessories. Rather, they create 3D environments for display and interaction for 2D viewing. This is not entirely surprising as they all were essentially abandoned before the advent of modern VR headsets just a few years ago.

Because of these shortcomings, the authors felt that a fresh attempt at exploring a language from the vantage of the HTML markup author was warranted. The authors wanted to focus on the core of what was important - an easy to understand tagging language, clear operational semantics, and mechanisms to support commonly used interactive engagements with 3D content while not limiting advanced content authors.

B. Molly, MAML, MAGE, and MARLi

Molly [15] is a general purpose web development framework, the core of which is a system for parsing XML and generating web pages dynamically based on that XML. Molly, initiated by Vullo in December, 2000, is an ongoing research project at the Rochester Institute Technology, designed to simplify the process of creating dynamic web sites. Molly is an open source system and architecture that allows web site developers to build dynamic web sites using HTML and MAML (Molly Active Markup Language) tags. MAML tags provide the web author with components that can be used to build sophisticated server-side functionality without programming. Molly has been used together with A-Frame [16] to build dynamic web-based VR environments.

Using Molly with A-Frame was not the first time the system was used to generate graphical content instead of HTML. Molly's architecture early on allowed the deployment of the optional MAGE (pronounced "Maggie") module which implements the Molly Automated Graphics Engine tags that create SVG-based charts using MAML tags to incorporate data from databases.

We have now embarked on the development of the MARLi module to implement the Molly Alternate Realities Language *interactive* tags described in this paper.

III. APPROACH

The Molly Alternative Realities Language *interactive* (MARLi) is based upon the premise that web-based VR markup should be simple and approachable to the content creator while still being powerful and flexible enough to accommodate for advanced world creators. The development of the MARLi language was guided by a number of design principles.

The first principle is that MARLi should embrace strong web standards and abide by modern markup techniques and best practices that are currently available. As such, MARLi properly utilizes namespaces and supports syntactical validation in its design. The design embraces proper separation of tags and attributes to differentiate between instances in a 3D world and modifications to those objects. References such as ids, nesting of elements, as well as modularity and reuse of components was well thought out in advance of creating the tagging language. Reasonable defaults were also chosen to ensure immediate viewability and interaction when authors introduce new and unfamiliar markup into a 3D world.

Second, MARLi is also designed to be human readable and human editable at its core, providing a lower barrier to entry for new developers. Tag and attribute names were chosen to mirror the languages familiar to 3D content creators. The authors tried to avoid names that are often associated with the inner workings of graphics API so as to reduce the syntactic barrier to entry. It should be noted while the naming conventions do not exclude notations that speak more to the graphics API, it is the authors' hope that by using such naming conventions, those who are interested in extending the language will employ similar techniques.

Third, MARLi is not dependent upon scripting to perform common interactions within a 3D environment. The authors had as a goal that typical content creators should not have to employ JavaScript or other client-side scripting language constructs to create interactivity. Simple triggers such as selection, proximity, timing, collision, and property change can all be modeled within the tagging system. As MARLi is distributed across client and server, transformations of the tagging system can be presented to the client in an appropriate scripting format as needed. When scripting is present, it should augment the tagging system such that interactions and behaviors can be built upon and interact with the default tags in a seamless manner.

Fourth, MARLi is designed to be adaptable as content creators' needs change. While flexible and easy to understand out of the box, MARLi is adaptable to complex scenes and interactions while still maintaining ease of readability. MARLi is also adaptable to automation, as its formatting lends itself to database and generated markup that still can remain readable and adaptable. MARLi can also be adapted for use with WYSIWYG and/or visual programming environments if and when necessary.

Finally, MARLi is designed to be modular in nature. Too often, systems allow for the construction of content, but do not address reuse and connection to create interconnected experiences. MARLi can provide for seamlessness in both the client and the server to maximize reuse and recontextualization of experiences. This is meaningful in that it allows users to quickly learn from existing and relevant content while still providing ease of access. As a Molly module, MARLi is completely integrated with tag-based database access as well.

IV. EXAMPLE

```
<marli:world radius="6" color="#ECECEC"
         xml:lang="en"
         xmlns="http://www.w3.org/1999/xhtml"
         xmlns:maml="http://interpersonalnet.com/maml/"
         xmlns:marli="http://mollyar.org/marli/">

         <marli:viewpoint position="0, 1.6, 0" interocular="6" field_of_view="80"
direction="0,0" current="true" />

         <marli:override category="units" attribute="light" value="percent" />
         <marli:light intensity="100" type="ambient" color="#bbbbbb" shadow="100"
shadowcolor="#000000" />
         <marli:light intensity="60" type="directional" position="-0.5 1 1"
rotation="0,0,0" color="#ffffff" shadow="100" shadowcolor="#000000" />

         <marli:cube width="1" height="1" depth="1" color="#4CC3D9" position="-1
0.5 -3" rotation="0 45 0" shine="0" />

         <marli:sphere radius="1.25" color="#EF2D5E" position="0 1.25 -5"
rotation="0, 0, 0" shine="0" />

         <marli:cone color="#FFC65D" radius=".5" tradius=".5" height="1.5"
position="1 0.75 -3"  rotation="0, 0, 0" />

         <marli:plane position="0 0 -4" rotation="-90 0 0" width="4" height="4"
style="color: #7BC8A4;" />

</marli:world>
```

Fig. 6. Sample MARLi Syntax

V. CONCLUSION

With the heightened interested in hardware to provide consumer level VR experiences, it is increasingly important to present content creators access to languages and frameworks that support the rapid deployment of 3D worlds and scenes. The authors' work with MARLi provides an approach that attempts to adhere to the ideals of web-based markup languages. As we continue to work with MARLi and the underlying Molly technologies, we will explore the boundaries between responsibilities of the content author and the systems that support the deployment of web VR content. By employing such an approach, the authors hope to return to content primacy instead of focusing upon the language technology that drive such experiences.

REFERENCES

[1] R. Carey and G. Bell. *The Annotated VRML 2.0 Reference Manual*. Essex, UK: Addison-Wesley Longman Ltd., 1997

[2] J. Hartman and J. Wernecke. *The VRML 2.0 Handbook: Building Moving Worlds on the Web*. Redwood City, CA: Addison Wesley Longman Publishing, 1996.

[3] "The Virtual Reality Modeling Language Specification." Internet: http://www.graphics.stanford.edu/courses/cs248-98-fall/Assignments/Assignment3/VRML2_Specification/, Aug. 4, 1996 [Apr. 24, 2019].

[4] M. Lawton. "How to be a Virtual God: Constructing Worlds in VRML 2.0 on a PC." Internet: https://www.developer.com/tech/article.php/602251/How-to-be-a-virtual-god-constructing-worlds-in-VRML-20-on-a-PC.htm, Mar. 5, 1998 [Apr. 24, 2019].

[5] S-M Zoltan. "Creating Complex Geometry." Internet: http://www.c3.hu/cryptogram/vrmltut/part5.html, Aug. 27, 1997 [Apr. 24, 2019].

[6] J. Sonstein. "XML-based 3D: Content Creators, the Web, and xVRML Are Ready for Each Other." in *Proceedings of E-Learn 2005-World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*, 2005, pp. 154-159.

[7] J. Sonstein. "xVRML Web Site (archive)." Internet: https://web.archive.org/web/20090727082021/http://www.xvrml.net/, Jul. 27, 2009 [Apr. 24, 2019].

[8] "x3dom - Instant 3D the HTML Way!." Internet: https://www.x3dom.org/, [Apr. 24, 2019].

[9] P. Festa and J. Borland. "Is a 3D Web More Than Just Empty Promises?" Internet: http://news.zdnet.co.uk/internet/0,1000000097,39199121,00.htm, [May 19, 2005 [Apr. 24, 2019].

[10] "Hello, X3DOM!" Internet: https://doc.x3dom.org/tutorials/basics/hello/HelloX3DOM.html, [Apr. 24, 2019].

[11] "3DMLW.com (archive)." Internet: https://web.archive.org/web/20080731010934/http://www.3dmlw.com/, Jul. 31, 2008 [Apr. 24, 2019].

[12] Wikipedia. "3DMLW: 3DMLW File Format." Internet: https://en.wikipedia.org/wiki/3DMLW, [Apr. 24, 2019].

[13] A-Frame - Make WebVR." Internet: https://aframe.io, [Apr. 24, 2019].

[14] "A-Frame Dynamic Lights Example.", Internet: https://github.com/aframevr/aframe/blob/master/examples/showcase/dynamic-lights/index.html, Sep. 4, 2018 [Apr. 24, 2019].

[15] R. Vullo. "About Molly." Internet: https://molly.rit.edu/, [Apr. 24, 2019].

[16] R. Vullo and M. Catalfamo. "Dynamically Generating Virtual Reality Scenes Using Molly and A-Frame." in *International Conference on Internet Computing and Internet of Things*, 2017, pp. 21-24.