X3DOM – A DOM-based HTML5/ X3D Integration Model

Johannes Behr* Fraunhofer IGD
Peter Eschler† NewMediaYuppies GmbH
Yvonne Jung‡ Fraunhofer IGD/ TU Darmstadt
Michael Zöllner§ Fraunhofer IGD/ TU Darmstadt

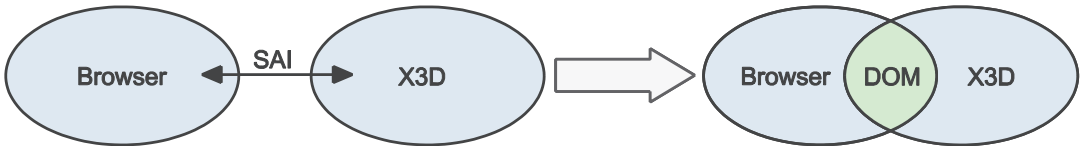

Figure 1: *Moving from a loose plugin-based Scene-Access-Interface (SAI) to the tightly integrated X3DOM model.*

Abstract

We present a model that allows to directly integrate X3D nodes into HTML5 DOM content. This model tries to fulfill the promise of the HTML5 specification, which references X3D for declarative 3D scenes but does not define a specific integration mode. The goal of this model is to ease the integration of X3D in modern web applications by directly mapping and synchronizing live DOM elements to a X3D scene model. This is a very similar approach to the current SVG integration model for 2D graphics.
Furthermore, we propose a framework that includes a new X3D Profile for the DOM integration. This profile should make implementation simple, but in addition we show that the current X3D runtime model still scales well. A detailed discussion includes DOM integration issues like events, namespaces and scripting. We finally propose an implementation framework that should work with multiple browser frontends (e.g. Firefox and WebKit) and different X3D runtime backends.
We hope to connect the technologies and the X3D/ W3C communities with this proposal and outline a model, how an integration without plugins could work. Moreover, we hope to inspire further work, which could lead to a native X3D implementation in browsers similar to the SVG implementations today.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual Reality I.3.6 [Methodology and Techniques]: Standards—Languages

Keywords: X3D, DOM, HTML5, Web integration, Real-time

1 Introduction and Motivation

Right after the first 2D HTML pages went online, people where thrilled with the idea of having 3D content on the net. The ability to visualize and manipulate spatial content in real-time seemed to be the next key-enabler for a wide number of application areas. Various proprietary and a few standard solutions had been developed over the last 15 years and quite some of them introduced a really new and revolutionary model (e.g. online gaming). However, there is still not a single technology widely used, even though we now have rendering and network performance in every phone.

A number of systems and solutions have been implemented in order to overcome this problem, but most of these approaches where heavily technology driven. Nevertheless, people usually detect or believe to detect a specific short-come and add a new feature to an existing environment or standard. Thus a number of rich extensions (e.g. multi-language scripting, distribution, security-layer) have been added to individual solutions confronting technology providers and content distributors with a huge amount of overloaded environments (e.g. X3D, Second Live, Game Engines, etc.). These are all quite mature but at the same time only provide isolated solutions. Some of them offer Browser-Plugins for their engines, however, the application-model and technologies for scripting and distribution are unique to each system and not tightly coupled with todays web-technologies.

On the other hand the web evolved heavily over the last 15 years. We started with static HTML-pages with inline styling, then separated the content from the presentation (CSS), and today we are building dynamic web pages based on JavaScript and AJAX technology. The basis for these changes are developments driven by the XHTML and HTML5 communities as well as browser systems like Mozilla and WebKit.

Those technologies are available and widely used today, and they are the basis for current web-services and the appearing cloud computing. These applications manipulate dynamic content and style of an XML-description represented by a DOM-model. This DOM-model includes a 2D-graph describing the content of a specific application. These standards include methods to manipulate this DOM over specific scripting interfaces, which leads to dynamic changes. There are web-technologies available, which provide distribution, communication and security of content.

All these developments are not directly available for the existing 3D environments since these systems developed their own, system- or standard-specific solutions for e.g. scripting and distribution over the years. Today we have reached the state where we have a 2D web and various 3D environments developed in parallel.

To overcome the current state 3D content must become a first class web media that can be created, modified and shared in the same way as text, images, audio and video is handled on the web right now. Key to this request is a direct integration into the DOM tree.

*e-mail:johannes.behr@igd.fraunhofer.de
†email:p.eschler@newmediayuppies.com
‡e-mail:yvonne.jung@igd.fraunhofer.de
§e-mail:michael.zoellner@igd.fraunhofer.de

2 The current State of 3D in the Web

There has been a large number of systems and proposals over the last 15 years. Most of the systems disappeared over time, and today we have a number of systems following the traditional browser-plugin-based approach as well as a small number of systems, which try to integrate the rendering system directly into the browser architecture or fake 3D renderer by utilizing 2D pipelines

2.1 Rendering with Plugins

All plugin-based systems have two major drawbacks. First, they are plugins and not installed by default on most systems. Therefore, the user has to deal with plugin installation, security and browser or OS incompatibility issues. Second, the presented systems define an application and event model inside of the plugin, which is decoupled from the DOM content. Developers, who try to develop integrated web-applications or web-pages that use both, the DOM/browser and the plugin-model, have to deal with the small plugin-specific interface and it's synchronization capabilities. We believe that these major drawbacks are one reason why plugin-based systems, besides Flash, were not successful over the years.

2.1.1 Flash and PaperVision

Adobe Flash [ado b] is a multimedia platform currently developed and distributed by Adobe Systems[ado c]. Since its introduction in 1996, Flash has become a popular method for adding animation and interactivity to web pages. Flash is commonly used to create animations, advertisements, and various web page components, to integrate video into web pages, and more recently, to develop content rich Internet applications.

Until the latest update of Flash, Version 10 [ado b], there was no real-time 3D support at all. However, authoring tools such as Director [ado a], Cult3D [cul] and Anark [ana] support 3D graphics natively and make it easy to incorporate 3D elements into web-based 2D-movies. Flash, until Version 10, does only know how to display 2D vector shapes on the screen and how to calculate math expressions. Hence people have built simple 3D rendering systems by exploiting these 2D vector shapes, like PaperVision3D[Project]. Even so the results are very impressive, they are limited to simple 3D effects and shapes.

Flash Version 10 [ado b] now includes simple 3D transformations and objects, but they are very limited and only designed for simple 3D composite and GUI effects. However, the PaperVision people started to update their system to utilize the new Version fully and the new system will be called PaperVisionX [Project]. Since they already have produced very impressive results on the 2D render pipeline this is definitely some development to watch.

2.1.2 Silverlight

Microsoft Silverlight [Microsoft] is a programmable web browser plugin that enables features such as animation, vector graphics and audio-video playback that characterizes rich Internet applications. It was developed by Microsoft as Flash alternative and is based on the .Net framework. Even so Microsoft provides Plugins for Windows and Mac, the user and install base is much smaller compared to Flash.

Since Microsoft follows the development of Flash very closely the situation today is very similar. They even dropped the native 3D support of WPF (an successor of Silverlight) since Flash had no 3D support at that time. Until the very last release, Version 3.0, which was unveiled at the MIX09[1] Silverlight did not have native 3D support. People where faking 3D content using similar techniques like in Flash. A new feature in Silverlight 3 are the so-called Perspective Transforms. They allow transforming 2D objects in a 3D coordinate system, but no real 3D shapes – which is again very similar to the last Flash version.

2.1.3 Java, Java3D, JOGL and JavaFX

Even so Sun were pushing Java [Sun a] as client-side technique and were providing browser-plugins for integrating Applets, Java is today mainly used for server-side services and applications. Compared to Flash or even Silverlight it has a much smaller user and install base since it was never officially supported on Windows by Microsoft and early implementations were lacking the performance improvements today's Java runtimes offer.

The Java3D [Sun b] runtime, a scene-graph system that incorporates the VRML/X3D design, was also popular for desktop-based applications but its power was never really utilized for the web and today it is no longer supported by Sun at all. Sun dropped the high-level Java3D library and now provides a lower level interface, which provides direct language-bindings for the OpenGL interface. This interface is called JOGL [Sun d] and e.g. used in the Xj3D runtime. JavaFX [Sun c], officially announced 2008, is the last effort by Sun to build an alternative technology to Flash based on Java. It offers similar media and 2D elements but does not have any official 3D support.

2.1.4 O3D

O3D [Google 2009b] is yet another graphics API for creating rich, interactive 3D applications within a browser. The system consists of two layers. The lower level is implemented using C/C++ as browser plugin and provides a geometry (as vertex buffers) and shader abstraction that is mapped to OpenGL [Khronos 2009b] or DirectX [Microsoft 2009]. The higher level API is implemented in JavaScript and provides a scene-graph API similar to OpenSG [OpenSG 2009] (C++ based), Java3D [Sun b] (Java-based), or C3DL [Bishop 2008] (JavaScript).

The target audience are JavaScript programmers who need the flexibility of a low-level graphics API. Even so the model of the scene-graph is close to existing standards, e.g. X3D [Web3DConsortium 2008], it does not provide a method to define the content in a declarative way. Application developers have to use JavaScript code to build and manipulate the graph-content. However, there are offline processing tools, which allow transcoding declarative 3D data (e.g. Collada files [Arnaud and Barnes 2006]) into a number of JavaScript-calls, which can be used to build the tree structure.

The system also provides basic functions, similar to other scene-graph APIs, for picking and culling. However, they are implemented in the JavaScript-layer and therefore they are slower compared to native implementations provided by high-level runtime abstractions as e.g. included in X3D-browsers. Performance in general may be a problem since the O3D model forces the developer to implement all parts of the application logic and behavior (e.g. physics) in JavaScript. Google hopes that the latest improvements in the V8 JavaScript engine [Google 2009c] will help to minimize this issue but there will always be a potential performance hit.

2.1.5 X3D

X3D is an ISO standard [Web3DConsortium 2008] that describes an abstract functional behavior of time-based, interactive 3D, mul-

[1] MIX09: Las Vegas, March 18-20th 2009

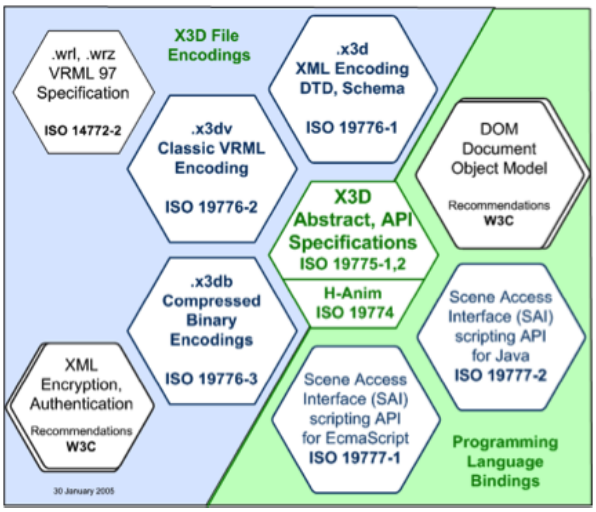

Figure 2: *X3D data-encodings and programming language bindings – DOM is just a recommendation.*

timedia information. It supports a multi-parent scene- and event-graph. The standard is IO-device independent, portable and supports multiple datafile-encodings. It has the ability to encode the scene using an XML syntax, Open Inventor-like syntax derived of VRML97, or by using a binary FastInfoSet[2] based binary encoding. It is the successor to VRML97 and includes a large number of new and extended features and components (e.g. Humanoid Animation, NURBS, GeoVRML and so on).

The X3D specification [Web3DConsortium 2008] includes various internal and external APIs and a full runtime, event and behavior model, and is therefore much more than a simple exchange format. X3D defines several profiles (sets of components) for various levels of capability including X3D Core, X3D Interchange, X3D Interactive, X3D CADInterchange, X3D Immersive, and X3D Full. Browser producers can define their own component extensions prior to submitting them for standardization by the Web3D Consortium.

There is a web-browser integration model right now which allows running plugins inside a browser. The browser holds the X3D scene internally and the application developer can update and control the content using the Scene Access Interface (SAI) [Web3DConsortium a], which is part of the standard. The standard already defines an integration model for DOM-Nodes, as part of SAI, but the current state only imports the DOM-elements a single time. There is currently no update or synchronization mechanism.

A subset of X3D is XMT-A, a variant of XMT, defined in MPEG-4 Part 11. It was designed to provide a link between X3D and 3D content in MPEG-4 (BIFS). The abstract specification for X3D (ISO/IEC 19775) was first approved by the ISO in 2004, whereas the XML and ClassicVRML encodings for X3D (ISO/IEC 19776) were approved in 2005.

2.1.6 MPEG-4 and MPEG-4 Part 11

MPEG-4 is a suite of standards that has many ”parts”, where each part standardizes various entities related to multimedia, such as audio, video, and other content formats. The MPEG-4 Part 11, or Binary Format for Scenes (BIFS) is a binary format for two- or three-dimensional audiovisual content. It is based on X3D and Part 11 of the MPEG-4 standard. BIFS is a MPEG-4 scene description protocol for composing MPEG-4 objects, describing interactions with MPEG-4 objects and for animating 2D and 3D MPEG-4 objects [Pandzic and Forchheimer 2002].

Therefore, there is a 3D subset in MPEG-4 but this fact is neither well known nor supported at all. None of the major MPEG-4 players supports 3D content at all and therefore, today it is unfortunately no available technology for web-application developers.

2.1.7 Collada

Collada [Arnaud and Barnes 2006] is not really a single technology but a 3D-file standard for content exchange now developed by the Khronos group. It was started by Sony and Intel to improve the interoperability of DAE-tools. The main goal was to ease and streamline the game development process and existing game creation pipelines. The specification does not include, unlike the X3D specification, a runtime- or event-model, which would allow defining interactive elements or the behavior and content.

Therefore, it is really designed as intermediate format which can be used in the creation pipeline together with a final deployment format like X3D. The special use case, developing web applications with Collada and X3D, is elaborated by Arnaud and Parisi [Arnaud and Parisi]. There are other fields of cooperation between the Web3D consortium and the Khronos group which led finally to the official liaison [Web3DConsortium c]. Even so Collada is not build as deployment format, there are some tools, e.g. GoogleEarth [Google], which utilize Collada data-files directly to define content for the runtime-environment.

2.2 Rendering without Plugins

There are several proposals available today which utilize existing internal 2D browser techniques to fake a 3D pipeline or even some, which really try to provide an abstraction for the 3D hardware layer directly to the web-application developer without any additional plugin. All these efforts are in an early development state and not standardized in any way right now. However, there is a large interest in these techniques since they allow building more interesting web-applications.

2.2.1 CCS/SVG Rendering Solutions

The current browser techniques like CSS [W3C 2009a], Canvas [W3C 2009c], and SVG [W3C 2009d] do not support 3D at all. With the enormous improvements in browser and JavaScript [ECMA] performance over the last years people started to use these 2D elements to build 3D pipelines based on DHTML techniques. Systems like SVG-VML-3D [Tautenhahn 2006], treebuilder [treebuilder] and especially the latest Google chrome experiments [Google 2009a] are very impressive examples of how powerful the current systems are.

Apple even added some 3D CSS transformations [Apple 2008] to their WebKit engine. But it should be noted, that these 3D transforms only apply to 2D elements. However, the transforms allow a web developer to translate, scale, rotate, skew, and change the perspective of almost any DOM element in 3D space, resulting in some rather spectacular effects. However this is still far from a full 3D scene setup as is for instance provided by X3D.

2.2.2 Hardware Accelerated Rendering

The goal of this proposal is to have an 3D rendering ,which can be hardware accelerated but also a basic feature of any standard web-

[2] http://java.sun.com/developer/technicalArticles/xml/fastinfoset/

browser. Recently, there have been some developments with similar goals, e.g. the Canvas3D [Vladimir Vukicevic 2009] extensions from Mozilla and 3D-Context [Tim Johansson 2007] from Opera. Both more or less wrap OpenGL [Khronos 2009b] or OpenGL-ES [Khronos 2008] and allow web developers to call OpenGL-commands for a specific canvas directly. The Khronos Group even launched an initiative to create an standard for this type of integration [Khronos 2009a] very recently (24th March 2009).

This works quite well in general since OpenGL [Khronos 2009b] is a stable abstraction for 3D graphics subsystems. The standard is utilized for more than 20 years for e.g. games (DOOM III, WOW), Second Live and almost all CAD/CAM systems, and is available on all major and minor platforms. However, it is a very low level interface, where the JavaScript programmer has to deal with every triangle and transformation. This has the advantage that the application developer is free to use whatever visualization model he can think of but it is hard to just visualize a complex 3D model.

People started to develop libraries, like C3DL [Bishop 2008] or X3DomCanvas3D [Excors 2007], to overcome this drawback. These libraries allow loading and displaying more complex data and models more easily, however, performance is a main issue since they have to be implemented in JavaScript, and graphics related algorithms, e.g. culling and collision detection, tend to be complex and runtime intensive.

2.3 The HTML5 Promise

Most people do not realize this important fact but the latest HTML5 specification already explicitly utilizes X3D for declarative 3D scenes. However, the HTML5 specification only states a single line for declarative 3D scenes [W3C 2009b]:

> 12.2 Declarative 3D scenes
>
> Embedding 3D imagery into XHTML documents is the domain of X3D, or technologies based on X3D that are namespace-aware.

Therefore, HTML5 does already define how X3D data should be included into the DOM but does not define how the connection to the runtime system should look like. The HTML5 standard does not specifically define, how the DOM integration should look like or, even more important, how to access the scene-graph content. The specification does not define at all if this 3D imagery should be produced in place, similar to SVG content, or if this DOM elements should just be data containers available to an X3D runtime system that exists elsewhere. So far the current X3D SAI Specification supports the latter one. It allows importing DOM-content from the browser space into an X3D scene living inside of an additional plugin. However there is no update or even live mechanisms and therefore it is a very limited interface.

3 The X3DOM Architecture

As the previous sections show there are a number of activities right now to bring 3D to the web. However they all have different drawbacks which we try, at least partly, to address in our architecture.

Most of the current plugin-based systems and standards offer small interfaces to the runtime- and application-model. These plugins include their own runtime-systems which control all the visualization, interaction, communication and distribution issues internally. A new integration model should avoid new plugins or system interfaces but should rather be accessible through the standard Document Object Model (DOM). This was the same model for the first generation SVG viewers years ago. Today, SVG is an integrated part of a modern Web-browser and the SVG elements can be directly manipulated utilizing the same DOM technology used for HTML content.

Flash and Silverlight are plugin-based and still have very little support for 3D which limits the application developer to visual effects. X3D has a full scene-graph model and already includes some DOM import mechanism but it does not support live DOM data. X3D applications also need plugins to perform scripting, animation and rendering right now. The recently developed JavaScript/OpenGL integration, proposed by Mozilla and Khronos[Khronos 2009a], is a nice abstraction which will run without plugins but really needs an internal scene-graph extension, like X3D, to perform well.

Thus, we propose a more abstract scene-graph layer that should be directly mapped to DOM elements. The scene-graph model is a visualization data structure, which is well established and able to handle large datasets efficiently. And even more important, it allows the application developer to define the scene in a declarative way similar to other web-technologies like SVG or XHTML and not just a programming interface like OpenGL. Libraries like [Excors 2007] try to integrate DOM declaration with a canvas based rendering approach but decouple both which increases the complexity.

Therefore, we present a more integrated and simpler solution. The solution is based on X3D and HTML5. We use X3D as scene graph model for three reasons. First it is a mature and established ISO standard which already defines an XML encoding. Second, as mentioned in section 2.3 the HTML5 specification already explicitly utilizes X3D for declarative 3D scenes. Last but not least, there already is a DOM tree interface described in the X3D-Programming Language binding interface [Web3DConsortium 2008], but an updating mechanism for live and changing DOM content is missing. We use HTML5 because it already utilizes X3D for declarative 3D scenes. But again, the HTML5 standard does not specifically define, how the DOM integration should look like or, even more important, how to access the scene-graph content. The current situation does also, similar to [Excors 2007], decouple the plugin-DOM-object and DOM-data-elements which leads to even more confusion since both are XML based.

We propose to extend this integration model to produce the images not in an extra plugin but rather in place, similar to how SVG works today. To ease the integration and development the model will only support a well defined subset of the full X3D specification and exclude all the scripting and distribution from the X3D execution Context which shares the data with the DOM-tree. The X3D-nodes will be directly integrated as DOM elements. The X3D subsystem will be mainly used to render the scene. All scene-graph manipulation in the DOM execution Context will be done using the standard DOM-based browser scripting interfaces as in conventional DHTML documents.

Thus, we can improve the state of the current 3D-Web integration by not introducing more but *less* technology. The key is to reduce the 3D-system to the visualization component and to borrow web-technology for e.g. dynamics, distribution, security and scripting. A slim environment for a single point of access fusing traditional environments into 3D worlds adopting new strategies based on an appropriate and extended model avoiding ”yet another plugin”.

The proposed architecture, called X3DOM, acts as a connector for the HTML5 and X3D world and content. The goal is to support HTML5 content that includes X3D data in a separate namespace inside of the browser's DOM tree. The foundation for this design is the HTML5 specification [W3C 2009b], which already utilizes X3D for this purpose. The proposed architecture tries to explore this model by providing the the missing link between both worlds.

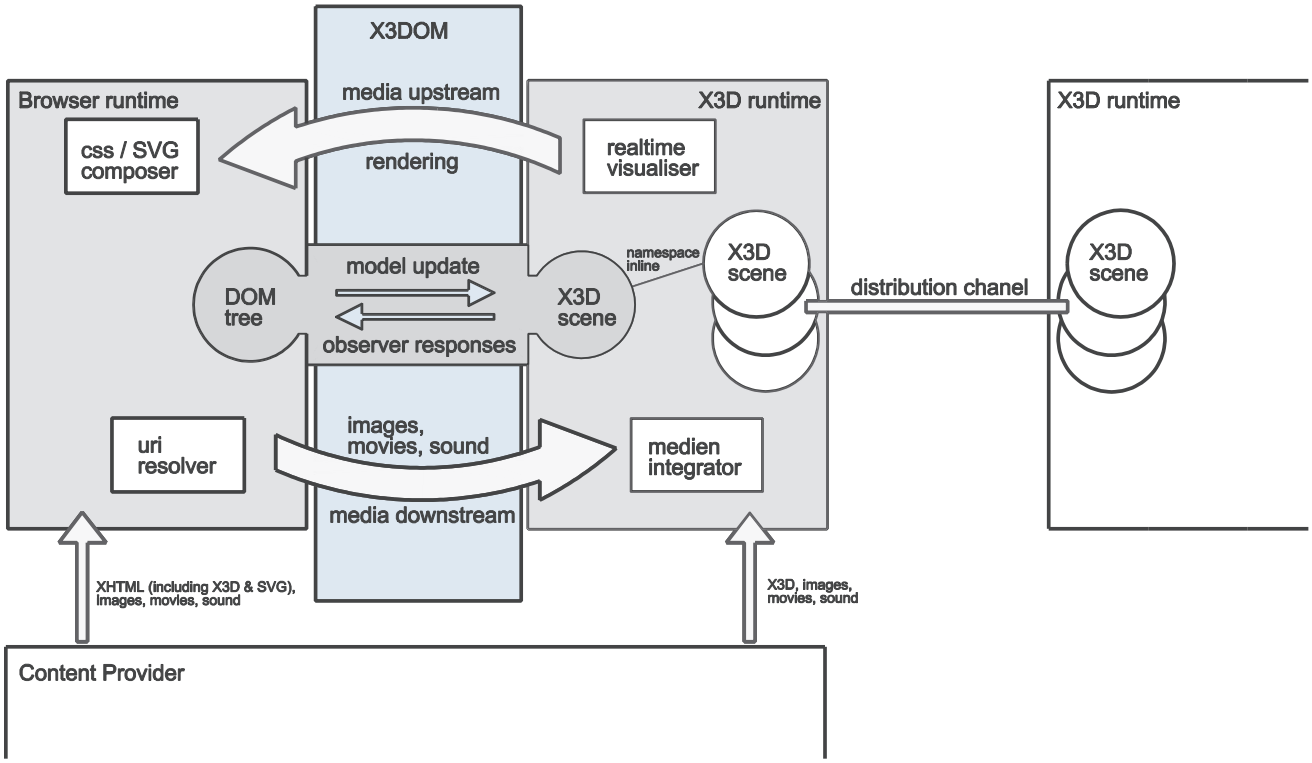

Figure 3: *X3DOM overview showing the components of the proposed integration framework.*

Therefore, it not just connects the actual content but tries to bridge the gap between the two communities.

3.1 Connector

The connector is the inner core of the proposed architecture. It connects the web-browser frontends with the X3D backends and supports mechanisms to communicate changes in the DOM or X3D representation. The architecture does not use the DOM-tree for rendering directly but creates and synchronizes an X3D-tree representation. It should include a DOM/X3D adapter to support different backends and frontends. We plan to have at least two frontend adapters as Firefox- and WebKit-Extensions and two backend adapters for different X3D runtimes (e.g. Instant Reality [**?**], FreeX3D [Stewart]).

The frontend adapter must be able to access the DOM tree content directly. It should not read and parse an XML-data-stream but read and write the DOM representation. Therefore the backend adapter has direct access to the X3D runtime context including the X3D scene, which reflects the DOM tree. The main task of the connector is to keep both representations in sync and distribute changes in both directions.

3.2 Model Updates

Any updates on the DOM tree, like creates, disposes and changes to DOM-elements, must be reflected to the X3D tree using the backend adapter. This includes DOM elements that directly represent X3D nodes but also additional structures like X3D Routes.

3.3 Observer Responses

Depending on the supported Profile the X3D execution-model can create changes in the X3D-tree based on time or user interaction. The connector must be able to distribute those changes back into the DOM representation. This will be enabled by connecting observers to specific X3D-tree elements (e.g. Sensors) which will distribute these changes.

3.4 Media streams

The proposed architecture must be able to handle media down- and upstreams through the connector component.

3.4.1 Media Downstream

Elements in the X3D namespace can reference external media elements like images and movies for texturing and sound. The X3DOM architecture must be able to resolve these external packages using the browser URI/URL streaming mechanism. This should allow to downstream and integrate media through the browser into the X3D representation.

3.4.2 Media Upstream

At least the rendering results from the visualizer must be provided by the X3D-backend. However, usually this is not a pixel-image but a buffer as part of a given graphic-context (e.g. OpenGL or Direct3D). Therefore the architecture must be able to request and transfer an graphics-context once from the browser-frontend to the X3D-backend. Afterwards, synchronized updates and upstreams will be performed by the backend. The resulting image, including color and transparency information, will be available for any further

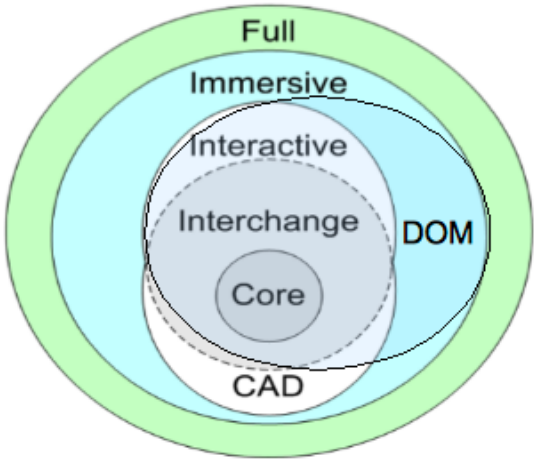

Figure 4: *The proposed DOM Profile shown in relation to the existing X3D Profiles.*

CSS composite requests.

3.5 Scalability and Multi-Profile support

The proposed architecture allows integrating X3D data directly into XHTML content. This enables the developer to incorporate 3D structures into the DOM. This should work well for applications, which only want to utilize a 3D subsystem for composite effects or GUI-elements. For applications which include mega- or even gigabytes of 3D-data, e.g. games or product presentations, this would not work efficiently on the proposed architecture, since all the data must be included within a single XHTML page. However, via the Inline-node [Web3DConsortium b] there is already a mechanism in the X3D-spec which allows including subparts of the scene. This allows application developers to segment the 3D-content in various parts, which could be referenced from the DOM elements using the IMPORT/EXPORT feature. A very nice aspect of the X3D specification is that every inlined scene resembles an own namespace and can support different X3D-profiles. Therefore the extra parts could support Profiles which include e.g. Scripting and Protos

3.5.1 X3D Profiles

There are a number of different X3D-Profiles already included in the current spec (see Figure 4). Each X3D-Profile bundles a number of nodes and features for different spec-components to provide a spec-subset for a specific purpose. This allows X3D tools and applications to work on data that uses a well defined subset of the overall specification. This already well defined method allows us to build a scalable and simple to implement architecture by using the existing X3D Profiles and creating an additional Profile that bundles the features we would like to be directly supported in the DOM-tree.

3.5.2 X3D DOM Profile

The goal is to define a specific Profile that defines all nodes and elements for the DOM integration. This profile must be rich enough to support the intended use-cases but also easy to implement so people can build X3D browser extensions easily. Therefore we propose a Profile that extends the Interactive Profile to include some additional features for animation and event handling. The profile however will not include any support for Prototypes or Scripts (see Table 1).

Component	Level	Level Comment
Core	1	Proto optional
Time	2	
Networking	3	Inline + http/https
Grouping	3	
Rendering	5	
Shape	4	
Geometry3D	4	
Geometry2D	2	
Text	1	
Sound	2	
Lighting	2	
Texturing	3	
Interpolation	5	
Pointing device Sensor	1	
Environmental Sensor	3	
Navigation	3	
Scripting	0	No Scripts
Event utilities	1	
Programmable Shader	1	
Followers component	1	

Table 1: *Components and level of the proposed DOM Profile.*

Therefore all the scripting on DOM elements must be performed from the browser side. However, it will include Inline nodes to support subtrees. We propose some changes to the spec at this point to allow different Profiles in those subtrees. This would support more complex applications, which include Protos and Scripts or event distribution channels to other runtimes using NetworkSensors in any number of subtrees. The only requirement is, that the X3DOM architecture has to provide a list of supported Profiles for the frontend.

3.6 X3D Elements as Single Point of Access

The X3D element in the DOM tree should provide a single point of access for all X3D related interfaces and manipulation issues. The proposed architecture includes two sets for setup and manipulation.

3.6.1 X3D Element Attributes

The X3D element must provide additional attributes to configure and setup the Inline and in-place render and execution engine. This is based on the current SVG-spec and includes generic attributes like 'xmlns', 'x', 'y', 'width', 'height' and especially 'version' and 'baseProfile' to request a specific x3D-runtime version and profile.

3.6.2 SAI Interface

The X3D spec includes a Scene Access Interface (SAI) [Web3DConsortium a]. This abstract specification allows to build bindings for different languages (e.g. Java, JavaScript) to access and manipulate the scene at runtime. Our architecture will include a specific X3D browser object, which allows to interface the scene with the standard interface through the X3D element by using browser-side scripting.

4 DOM Integration Aspects

X3D content can be integrated into any XML and XHTML document using namespaces today. What is missing is a rendering and execution model. The X3DOM architecture proposes an in-place renderer, like SVG, but other integration models should be similar.

In the following we present some code examples on how to integrate an XML encoded X3D world into an XHTML document. We will show several examples, starting with the most explicit markup, and then go on cleaning the markup, thereby showing some complications with the default namespace of attributes according to the XML standard.

The following examples have been tested in Firefox3. For all of the example code you have to make sure that the documents are delivered and interpreted with the correct mimetype. As the document is an XHTML document, the mimetype should be "application/xhtml+xml". Normally the mimetype of the document is given by the server, which must be configured accordingly. When working with local files, it is important to use the ".xhtml" suffix to instruct Firefox (FF) to interpret the document as an XML document. When the ".html" suffix is used, FF will interpret the document as HTML (and namespaces will not work).

4.1 X3D Namespace in DOM Documents

The <html> element in the code example below defines a default namespace for the document. Thus, the <h1> and <p> elements are part of the xhtml namespace. The <x3d> element defines a namespace prefix x3d, which is then used on all nodes belonging to the X3D world. As one can see this is a lot of code to type.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>
X3D DOM integration and manipulation
</title></head>
<body>
  <h1>X3D DOM integration and manipulation</h1>
  <x3d:x3d xmlns:x3d=
   "http://www.web3d.org/specifications/x3d-3.0.xsd">
    <x3d:Scene>
        <x3d:Shape>
          <x3d:Box x3d:size="4 4 4" />
    </x3d:Shape>
    </x3d:Scene>
  </x3d:x3d>
</body>
</html>
```

In order to get rid of the need to prefix every X3D element, we can use a default namespace on the <x3d> element itself, thereby removing the explicit "x3d" prefix. You can find more information about XML default namespaces in [W3C].

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>X3D DOM integration and manipulation</title>
</head>
<body>
  <h1>X3D DOM integration and manipulation</h1>
  <p>This page demonstrates how to integrate an X3D
    world into an XHTML webpage.</p>
  <!-- All elements within the x3d element (and the
       element itself) belong to the x3d namespace -->
  <x3d xmlns=
  "http://www.web3d.org/specifications/x3d-3.0.xsd">
    <Scene>
        <Shape><Box size="4 4 4" /></Shape>
    </Scene>
  </x3d>
</body>
</html>
```

According to the XML specification the "xmlns" attribute of the <x3d> element establishes a namespace for itself and all of its children. Therefore the <Scene>, <Shape> and <Box> elements are part of the x3d namespace. Unfortunately there is one flaw in the code example above. While the "xmlns" established a namespace for all children elements of the containing node, the namespace does not apply to the attributes of the children elements. This means that the "size" attribute is technically not part of the "x3d" namespace. To fix this we need to reintroduce a namespace prefix, which is then used on the attribute.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>X3D DOM integration and manipulation</title>
</head>
<body>
  <h1>X3D DOM integration and manipulation</h1>
  <p>This page demonstrates how to integrate an X3D
     world into an XHTML webpage.</p>
  <x3d xmlns=
       "http://www.web3d.org/specifications/x3d-3.0.xsd"
       xmlns:x3d=
       "http://www.web3d.org/specifications/x3d-3.0.xsd">
    <Scene>
        <!-- The x3d:size and the size attribute belong
             to different namespaces and therefore
             are different attributes.
             Technically the x3d:size is the correct
             one, but it would be cumbersome to add
             the x3d namespace to every attribute.
             Further investigation is needed here.
          -->
        <Shape><Box x3d:size="4 4 4" size="5 5 5" />
        </Shape>
    </Scene>
  </x3d>
</body>
</html>
```

While this last example is technically correct, it would be very cumbersome to add the "x3d" prefix to every attribute of an element in the "x3d" namespace. In fact this is the same problem as with the SVG standard and you can read more about it in [Watt].

In any practical implementation the attribute namespacing can be neglected for the following reasons: First of all, the context of the attribute is the element which is in the "x3d" namespace. Second, the interpreter can then safely assume that the element's attribute is part of that namespace, too. Last but not least, as long as no other XML applications are mixed into the X3D application, no namespace clashes can occur (since only X3D element attributes are interpreted).

4.2 Accessing/Manipulating X3D using DOM Methods

The following example demonstrates how to access and manipulate an attribute of a DOM node. But again: In order to make it work in Firefox you need to use a ".xhtml" file prefix, otherwise the document will be interpreted as HTML. This behavior is based on how FF detects the mimetype of local files (using the file:// protocol). You can learn more about Firefox mimetype detection at https://developer.mozilla.org/En/How_Mozilla_determines_MIME_Types.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
<title>X3D DOM integration and manipulation</title>
</head>
<body>
  <h1>X3D DOM integration and manipulation</h1>
  <x3d xmlns=
      "http://www.web3d.org/specifications/x3d-3.0.xsd">
    <Scene>
        <Shape><Box size="4 4 4" /></Shape>
    </Scene>
  </x3d>
  <script type="text/javascript">
  // The namespace URIs
  var xhtml_ns = "http://www.w3.org/1999/xhtml";
  var x3d_ns =
      "http://www.web3d.org/specifications/x3d-3.0.xsd";
  // Get elements using namespaces
  var h1 =
      document.getElementsByTagNameNS(xhtml_ns, "h1");
  var box =
      document.getElementsByTagNameNS(x3d_ns, "Box")[0];
  // Edit an attribute of the <Box /> element
  alert(box.getAttributeNS(null, "size"));
  box.setAttributeNS(null, "size", "2 2 2");
  alert(box.getAttributeNS(null, "size"));
  </script>
  </body>
```

The example above changes the ”size” attribute of the <Box> element to the value ”2 2 2”. Note the usage of the namespace aware functions (e.g. getElementsByTagNameNS(), getAttributeNS()).

4.3 Naming and Identifying Nodes

While XHMTL uses the ”id” attribute of a node to specify a unique id for an element in the DOM, the X3D standard uses the ”DEF” attribute. It should be considered to change the X3D identifier attribute ”DEF” to ”id” in order to be compliant with the rest of the web.

Besides the unique id of an element, the XHTML standard also defines the ”class” attribute, which associates a given element to the class named in the ”class” attribute. The X3D standard has no concept like ”class”. Strangely the ”class” attribute can be found in the X3D schema, but is not mentioned elsewhere in the X3D ISO standard.

4.4 Events

In order to be able to implement interactive web sites and X3D worlds, both standards ((X)HTML and X3D) define an event model. Unfortunately these two model differ dramatically, due to their different application domains. The following example shows how a possible connection between X3D’s and XHTML’s event model can look like.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>X3D DOM Events</title></head>
<body>
    <h1>X3D DOM Events</h1>
    <x3d xmlns=
        "http://www.web3d.org/specifications/x3d-3.0.xsd">
        <Scene>
            <Transform>
                <Shape><Box size="4 4 4" />
                </Shape>
                <TouchSensor id="ts" DEF="ts" />
            </Transform>
        </Scene>
    </x3d>
    <script type="text/javascript">
    // The namespace URIs
    var xhtml_ns = "http://www.w3.org/1999/xhtml";
    var x3d_ns =
        "http://www.web3d.org/specifications/x3d-3.0.xsd";
    // Get elements using namespaces
    var h1 =
        document.getElementsByTagNameNS(xhtml_ns, "h1");
    var x3d =
        document.getElementsByTagNameNS(x3d_ns, "x3d")[0];
    var ts = x3d.getElementsByTagName("TouchSensor")[0];
    alert("ts=" + ts);
    ts.addEventListener("touchTime", function() {
        alert("clicked");
    }, false);
    </script>
</body>
</html>
```

4.5 Multi-parent relations

While the DOM tree of an (X)HTML document is a single-parent graph, the X3D scene graph is a multi-parent graph (i.e. one node can be the child of more than one parent – the node can be reUSEed). Therefore we propose a special USE-element which will be used and returned for the USEed node.

5 Implementation

The current implementation is just a pre-alpha test to evaluate the interfaces and mechanisms available in the current browser architectures. The first tests run as a Firefox extension, which allows to directly manipulate and monitor the DOM elements. The browser also provides an OpenGL context which is passed to the backend structure. Any downstream media handlers are not yet defined but we are convinced that this is possible with the Mozilla infrastructure. We are evaluating the WebKit framework right now to see if we get the same functionality there. In the end we would like to develop a single system that uses loadable extensions to connect to different front- and backend systems. Furthermore, the plan is to open-source this X3DOM component so every browser or X3D system provider can develop and provide own extensions.

We believe, the approach discussed in this paper will help both worlds or sides respectively to evaluate and develop the integration idea. However, the software framework as well as the extensions have to be installed and therefore the requirements are similar to traditional X3D-plugins. We hope to initiate a process, which is similar to SVG that took place in browser developments during the last years. The first systems were plugin-based and not integrated. Now, SVG is a base functionality of every modern browser. If this project is successful, we have demonstrated a bridge between DOM and X3D, and finally people will start to write native extensions or will directly integrate the techniques into their systems.

6 Conclusion and Future Work

In this paper we have presented a DOM-based integration model for X3D and HTML5. Both specifications already reference each other but the actual integration is currently not sufficiently defined. HTML5 references X3D with a single line of text and X3D only supports simple DOM-Node imports without any support for live data. Our proposed X3DOM design and architecture exploits the current X3D-ISO and W3D-HTML5 standard and tries to provide the missing link and integration model. The model does not try to render the DOM-content directly but synchronizes and updates an

X3D tree automatically. We also propose an in-place renderer similar to the current SVG/XHTML integration model. Furthermore, we have defined a specific X3D-Profile, which excludes Scripting and Protos to ease further implementations but supports Inlines to be scalable and thereby useful for applications utilizing large datasets. The architecture is not bound to a specific browser or plugin but includes a flexible mechanism to support different browser frontends (e.g Firefox and WebKit) and different X3D-backends.

The whole proposal shows how a tight integration of X3D and HTML5 could look like. The architecture and implementation should inspire people to get both communities, around W3C and Web3D, closer together to work on a final integration model or even different official models (e.g. one in-place rendering model like X3DOM and one out-of-place JavaScript/Canvas3D model). The goal is to create a similar effect for X3D that we have seen for SVG. People started with plugins (this is the same situation as for X3D today), and now we have a tight integration and in-place rendering of SVG content. The proposed system can provide the same solution for X3D and finally, the HTML5 promise would be fulfilled – 3D for everyone and everywhere.

References

Adobe director. http://www.adobe.com/uk/products/director/.

Adobe flash. http://www.adobe.com/products/flashplayer/.

Adobe systems. http://www.adobe.com/.

Anark Cooperation. http://www.anark.com/.

APPLE, 2008. 3d ccs-transforms for the webkit. http://webkit.org/specs/CSSVisualEffects/CSSTransforms3D.html.

ARNAUD, R., AND BARNES, M. 2006. *Collada: Sailing the Gulf of 3d Digital Content Creation*, 1 edition ed. No. ISBN-13: 978-1568812878. AK Peters, 2006, August 30.

ARNAUD, R., AND PARISI, T. Developing web applications with collada and x3d. http://www.khronos.org/collada/presentations/Developing_Web_Applications_with_COLLADA_and_X3D.pdf.

BISHOP, C., 2008. Canvas 3d js library. http://www.c3dl.org/.

Cult3d by cycore systmes. http://www.cult3d.com/.

ECMA. Ecma-262, ecmascript language specification. http://www.ecma-international.org/publications/standards/Ecma-262.htm.

EXCORS, P., 2007. Canvax3d. https://labs.mozilla.com/ forum/comments.php?DiscussionID=363.

GOOGLE. Google earth. http://earth.google.com/.

GOOGLE, 2009. Google chrome experiments. http://www.chromeexperiments.com/.

GOOGLE, 2009. O3d; an javascript based scene-graph api. http://code.google.com/apis/o3d/.

GOOGLE, 2009. V8 is google's open source javascript engine. http://code.google.com/apis/v8/.

KHRONOS, 2008. Opengles. Khronos Group. http://www.khronos.org/opengles/.

KHRONOS, 2009. Khronos launches initiative to create open royalty free standard for accelerated 3d on the web. Khronos Group. http://www.khronos.org/news/press/releases/khronos-launches-initiative-for-free-standard-for-accelerated-3d-on-web/.

KHRONOS, 2009. Opengl. http://www.opengl.org/documentation/.

MICROSOFT. Silverlight. http://www.microsoft.com/SILVERLIGHT/.

MICROSOFT, 2009. Directx. http://msdn.microsoft.com/directx/.

OPENSG, 2009. OpenSG. http://opensg.vrsource.org/trac.

PANDZIC, I. S., AND FORCHHEIMER, R. 2002. *MPEG-4 Facial Animation: The Standard, Implementation and Applications*. John Wiley & Sons Ltd, West Sussex, England.

PROJECT, P. Papervision3d. http://blog.papervision3d.org/.

STEWART, J. Freewrl, open-source vrml/x3d runtime. http://freewrl.sourceforge.net/index.html.

SUN. Java. http://java.com/en/.

SUN. Java3d. https://java3d.dev.java.net/.

SUN. Javafx. http://javafx.com/.

SUN. Jogl. https://jogl.dev.java.net/.

TAUTENHAHN, L., 2006. Svg-vml-3d. http://www.lutanho.net/svgvml3d/index.html.

TIM JOHANSSON, O., 2007. Taking the canvas to another dimension. http://my.opera.com/timjoh/blog/2007/11/13/taking-the-canvas-to-another-dimension.

TREEBUILDER. Treebuilder. http://www.treebuilder.de/default.asp?file=206524.xml.

VLADIMIR VUKICEVIC, M. C., 2009. Canvas 3d: Gl power, web-style. http://blog.vlad1.com/2007/11/26/canvas-3d-gl-power-web-style/.

W3C. Namespaces in xml. W3C Consortium. http://www.w3.org/TR/REC-xml-names/.

W3C, 2009. Cascading style sheets. http://www.w3.org/Style/CSS/.

W3C, 2009. Declarative 3d scenes in html5. http://dev.w3.org/html5/spec/Overview.html#declarative-3d-scenes.

W3C, 2009. Html 5 specification, canvas section. http://dev.w3.org/html5/spec/Overview.html#the-canvas-element.

W3C, 2009. Scalable vector graphics. http://www.w3.org/Graphics/SVG/.

WATT, J. Svg authoring guidelines. http://jwatt.org/svg/authoring/.

WEB3DCONSORTIUM. Scene access interface(sai), iso/iec cd 19775-2 ed. 2:200x. http://www.web3d.org/x3d/specifications/ISO-IEC-CD-19775-2.2-X3D-SceneAccessInterface/.

WEB3DCONSORTIUM. X3d inline node from the networking component. http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification_Revision1_to_Part1/Part01/components/networking.html#Inline.

WEB3DCONSORTIUM, K. G. Formal liaison with khronos group. http://www.web3d.org/press/detail/web3d_enters_formal_liaison_with_khronos_group/.

WEB3DCONSORTIUM, 2008. X3D. http://www.web3d.org/x3d/specifications/.