An Open Modular Architecture for Effective Integration of Virtual Worlds in the Web

Sergiy Byelozyorov
Saarland University
Saarbrücken, Germany
byelozyorov@cs.uni-saarland.de

Vincent Pegoraro
Saarland University
Saarbrücken, Germany
pegoraro@cs.uni-saarland.de

Philipp Slusallek
DFKI, Saarland University
Saarbrücken, Germany
slusallek@dfki.de

Abstract—The Web and virtual worlds are currently crossing their ways, and although there are some efforts made to integrate them into each other, those typically rely on technologies that are rather esoteric to most web-developers. In this paper, we present a new open architecture that combines several emerging and established technologies to provide convenient tools for developing virtual worlds in the Web. These technologies are easy to learn and understand by the web community and allow for quick prototyping. Overall the modular architecture allows virtual worlds to be developed more quickly and more widely deployed.

Keywords—virtual worlds; the Web; open architecture.

I. INTRODUCTION

The developments made over the last decade have led to a significant impact of the Web not only on business, but also on a social level as people from different backgrounds and cultures communicate with each other using social networks, such as Facebook, Twitter, uniting the real world with a new borderless digital space. The Web has become so significant that Internet itself has become a synonym for it.

At the same time, we have seen a development in graphics in the last few centuries. Modern hardware is able to display real-time highly-realistic environments on the computer screen, making games and other applications an immersive and entertaining experience. On the other hand, an increase in the network throughput has allowed game companies to create a new type of games – massive multiplayer online games, where hundreds of thousands of players can play together in a single shared virtual world.

We are currently at the point where these developments are able to cross their ways by creating next-generation in-browser virtual worlds that are integrated with the Web. In the same way social networks have become a daily part of our society, we expect such shared environments to create another way of interaction and communication in the Web. However, there is currently no widely-accepted and easy way to integrate virtual worlds into the Web.

In this paper, we propose an open modular architecture based on a combination of both emerging technologies, such as XML3D [1], Xflow and Sirikata [2], and established, self-contained and widely-available Web components. Such a combination has a number of advantages.

Firstly, it makes the integration of virtual worlds in the browser much simpler for web-developers, as no special knowledge in the areas of network, graphics or system architecture is required other than Web technologies. In addition, developers can quickly prototype their applications using high-level concepts offered by the underlying technologies.

The approach is highly flexible due to the modular architecture, which allows replacing components easily. In the Web community, this approach is known as mashup, where a developer combines data and functionalities from two or more sources to create new services and applications.

Wider and easier deployment is achieved as users do not have to install additional specialized software, but instead use the browser that is typically shipped with the operating system pre-installed on their computer. An illustration of the world running in different operating systems and using different rendering algorithms is shown in Figure 1.

Finally, an important property of such design is that it is an open system, i.e. it is based on open-source code, open protocols and an open specification. The development of the Web has shown that open systems are more successful than the ones based on closed or proprietary technologies, thanks to a more dynamic response to user feedback via typically shorter integration cycles, and their sustainability beyond the life expectancy of a particular contributing entity.
We describe our approach in the following way. Section II provides an overview of the related work and motivates our choice of technologies. The selected technologies are detailed in Section III, which also presents the prototype library that we have developed. Potential applications of this library are illustrated in Section IV, while limitations and future work are discussed in Section V.

II. RELATED WORK

In this section, we discuss the components needed to integrate virtual worlds into the Web and provide an overview of existing solutions for each of them, including the first subsection that covers 3D graphics technologies and the second one that introduces several 3D animation techniques. In the third subsection, we present networking components for the communication with the virtual world platform, the choice of which is discussed in the last subsection.

A. Graphics

This subsection reviews some of the most widely spread technologies for embedding 3D graphics into web-pages and analyses their pros and cons.

One of the approaches commonly used to integrate graphics into the Web is known as X3DOM [3]. This technology is developed in the Fraunhofer institute and is an adaptation of the ISO-standardized X3D [4] file format to HTML5 [5]. The advantage of this technology is that it is based on the existing standard and thus is backward compatible to existing content. However X3D, which is the successor of the VRML [6], was developed with different underlying technologies than the Web. As such, X3D uses Scene Access Interface (SAI) instead of Document Object Model (DOM), a routing event model instead of a DOM [7] event model, and a material model instead of CSS [5]. One of the most widely-used nodes in existing content is the IndexedFaceSet node, which provides flexible geometry representation, but often requires preprocessing before passing to the hardware, since the latter requires vertex arrays for better performance. Finally, due to the complexity of the X3D standard, the browser needs to be extended with over 70 new nodes, which makes it hard to learn the technology and decreases the chances for potential web-developers to adopt it.

Alternative technology available for 3D graphics in the documents is WebGL [8], developed by the Khronos group. It is highly efficient, because it provides direct JavaScript bindings to the OpenGL ES 2.0 API. It is part of the proposed HTML5 standard and is already implemented by a number of browser vendors. WebGL allows delegating heavy processing to the parallel graphics hardware available on most consumer PCs, such that the central processing unit may perform other tasks. On the other hand, WebGL does not define the way to declaratively represent 3D content and thus requires a JavaScript-based scene graph library, which can only handle scenes of small to medium sizes and significantly limits the visible part of the virtual world in the browser.

Google has tried to address this problem by providing a fast scene graph JavaScript API in their project O3D [9]. While this solved the performance issue and allowed for larger scene sizes, it is still unintuitive for the web-developers to keep the content separate from the DOM.

Finally XML3D, which is developed at Saarland University in our group, defines a format for encoding 3D graphics in XML. However, unlike X3D, it builds on top of adopted Web technologies that are widely available and familiar to Web developers. Specification defines a 3D document as a DOM model in the browser. It reuses ideas from Scalable Vector Graphics (SVG) [10], HTML5, CSS and more. Elements that encode different aspects of the 3D scene provide interfaces to JavaScript that allow an easy and real-time synchronization of the document. XML3D is more efficient because it uses vertex arrays natively to encode geometry, which can be directly mapped to the hardware. This technology is the most adequate for our needs and we present it in more detail in Section III-A.

B. Animation

Without animation, a virtual world is a monotonous and static environment, while we would like to have rich and immersive worlds. Additionally, we need to be able to efficiently synchronize animations over the network using as little bandwidth as possible. Therefore we need an easy, intuitive and efficient way to encode animations.

The most straightforward way to animate an object is to use a script that regularly changes the transformations of objects and vertices. It is also known as DOM-scripting and typically uses JavaScript. While this method, being familiar to web developers from their experience, is intuitive and easy to learn, it is highly inefficient when it comes to complex animations. In order to create a smooth animation of an avatar character, a script would need to move thousands of vertices many times per second. This would typically require a substantial amount of code and expensive hardware to have a smooth realistic animation.

We can solve this problem by moving such modifications to parallel hardware by implementing them as vertex shaders that are executed on the GPU and thus are much more efficient. However, at this point, it becomes highly unintuitive for the web developers who are typically not familiar with shading languages. Learning them requires deep knowledge in graphics and GPU programming, which means that only experts will be able to create such animations.

Instead of writing scripts, we can describe animations declaratively in HTML. Such an approach was suggested by the Synchronized Multimedia Integration Language (SMIL). However, SMIL does not fit well into the DOM concepts and these elements are not very well supported by authoring
tools. Instead, most Web applications use the aforementioned DOM-scripting capabilities for dynamic content.

An alternative is Xflow, which is an extension to XML3D and defines a data flow declaratively in the document. The main idea behind it is that we can represent animations as graphs where only a few data elements are changed, such as bone positions, and influence many more data elements, such as the final mesh vertices of a character. With this approach a programmer defines a flow in the document and only changes a few parameters in JavaScript that define the animation, while the underlying system propagates data through the graph and creates a smooth animation on the screen. Moreover, in order to synchronize such an animation we will only need to send the graph configuration once and send updates to the initial data elements later. Animations will be automatically computed on each computer locally. Due to these advantages, we have selected it as a basis for describing animations in the virtual worlds. A more detailed description can be found in Section III-B.

C. Networking

Aside from 3D graphics and animations, we need a real-time networking technology to create a rich virtual world. This section provides an overview of one existing and one emerging technology that can be used to communicate with server to exchange the data.

An established and convenient way of transferring additional data to the web application after the page is already loaded is Ajax [11], which is a technology based on asynchronous requests to the server. This approach has shown its reliability and usability in thousands of the rich applications in the Web. It is, however, designed for casual data exchange, rather than for continuous data streams that are present in communication-rich applications like virtual worlds. The communication overhead created for each request can easily exceed the transferred data in size, thus making this technology inefficient in terms of the network load.

An emerging technology that was introduced in the upcoming HTML5 standard is known as WebSockets [12]. Similarly to WebGL, it is implemented in most major browsers and supports continuous data streams with only 2 bytes of overhead for each frame. In addition, it can provide similar functionality in the business environments, where all ports other than 80 are blocked, by multiplexing several WebSockets over a single TCP connection, which is important, since we would like virtual worlds to be widely-available for everyone despite the limitations created by local firewalls or network policies. These benefits and tight integration with existing Web technologies make it the most suitable networking technology for virtual worlds in the Web.

D. Virtual World Platform

In addition to the client components, we need a virtual world platform to provide the data and synchronize the state of the world. This subsection presents a number of existing and emerging technologies in the area.

Many popular online games, such as World Of Warcraft, Eve Online, EverQuest and many others, are typically based on a proprietary server and client software. While being extremely immersive and realistic, they typically don’t allow users to create content, because such content typically has to be specially designed for the game’s rendering engine. This type of virtual worlds is not suited for the Web, since they do not allow interoperability, require special client software per world and do not allow users to change the world.

One of the first virtual worlds that allowed to create user content and have gained public attention is Second Life [13] that has existed since 2003. This platform features internal economy, where users can trade their content to each other, which opens more opportunities for business. Second Life is developed by Linden Labs and is run on their private servers. Unfortunately, the system is closed and not designed for interoperability with other worlds, which is an essential part of the Web. Other systems that have similar limitations include Blue Mars, Twinity and HiPiHi.

OpenSimulator [14] is a project that provides open world server software that is compatible with the Second Life client. The implementation inherits Second Life’s server architecture, known as the grid, but also extends it to the hypergrid mode, where external worlds can be connected together. In the grid and hypergrid architecture, the world is divided into square fixed-size regions, each being controlled by a single server. Moreover, when the server is close to its maximum capacity, the users are prohibited to enter the region, which sets an upper bound for the number of users that can visit one region at a time. This creates an issue when an event of interest, e.g. a virtual concert, is happening in a given region and many users are trying to visit it. Servers have to be designed with excessive capacity, which is used at a fraction of its potential most of the time.

The Open Cobalt [15] platform offers a fully decentralized open-source world browser and toolkit, which has embedded tools for collaborative research and education. It is based on the Squeak programming language, which allows to modify the program at runtime, in particular, to extend the world functionality when discovering new object types and sharing their implementation code. Unfortunately, this solution has security limitations as such code can be arbitrary, and thus malicious. Additionally, the current architecture requires each peer to store the entire world and run all downloaded scripts locally, which is not scalable to large worlds.

Red Dwarf [16], a successor to Project Darkstar originally maintained by Sun, is focusing their architecture on scalability and reuse of resources. With their approach, the
simulation of the world regions does not map directly to the servers, but is distributed dynamically depending on the load of each region. This allows resources to be used more efficiently and decreases the cost of running a world. This provides a good solution for a world that is run by a single company. However, in the Web, different companies typically run their web-sites independently and do not share their hardware due to security issues.

Project Multiverse [17] offers an architecture that is very well designed for games. The world is separated into zones and zones into regions. When switching between zones, a user has to wait until the other zone is loaded. However, moving between the regions is seamless as loading of adjacent regions starts in advance when the user is close to the region boundary. Additionally, there are instances that are similar to zones, because they also require a loading screen, but apply when entering a certain closed structure, e.g. a house or a cave. This can be used, for example, to create spaces that are larger than their outer apparent size. All these features are very game-specific and do not fit into the generic definition of the virtual worlds that we would like to see on the Web.

Finally the Sirikata project, which is developed in Stanford, offers an architecture that separates the world architecture into space servers and object hosts, where space servers are responsible for a certain constrained space in the shared world, while object hosts run the objects. This approach offers a better security model, scalable to huge amount of players and offers a federated world, where parts of it can be controlled by different organizations. Additionally Stanford developers provide a ready-to-use implementation of the server and client libraries that help developing the client. Moreover, they support WebSockets that we have identified as a technology that we would like to use for networking. Based on these assets, Sirikata appears as the most appropriate platform. The detailed platform architecture is presented in Section III-C.

III. SYSTEM ARCHITECTURE

This section first describes the XML3D technology, which enables interactive graphics in the Web browser, and subsequently discusses an extension to XML3D for animations, namely Xflow. We then provide an overview of the Sirikata virtual world platform before describing the prototype library that we have created to seamlessly integrate all of the above components into a flexible framework.

A. XML3D

XML3D is based on XML and can be used as a standalone document or embedded into XHTML, using a separate XML namespace. Being inside an XHTML document, certain parts of the HTML specification such as <img> and <video> elements can directly be reused in XML3D as well (e.g. for textures).

The root of an XML3D scene is declared by the <xml3d> element, which creates a 3D canvas at the place of declaration. This canvas can be formatted via built-in browser CSS properties, e.g. borders, size and background. The descendants of the <xml3d> element define the content of the scene by mapping a scene graph onto the DOM tree. Typically, in the scene graph, the content that is stored only once is instantiated many times to save memory. Similarly to SVG, we consequently introduce a definition sub-graph starting with a <defs> element, which includes all the content that is placed into the scene but is not rendered directly (e.g. shaders, vertex arrays, etc.). Its content is referenced by other parts of the scene, such as <mesh> elements that instantiate referenced geometry.

In 3D graphics, shading can be highly complex with many different shaders each having their individual parameters. CSS cannot provide this as it only allows for a fixed set of predefined properties. Instead, a user can declare a shader with parameters using a <shader> element and attach it to the geometry using CSS URI syntax. The code for the shader can be declared using a referenced <script> element that is reused from HTML.

Applied to HTML documents, the style defines the size, position, and appearance of a box-shaped layout element. XML3D translates this concept also to 3D content, where it describes the transformations and appearance properties of <mesh> elements. Transformations and shaders can be assigned to nodes using CSS, in which case all children of the node are affected. For the description of 3D transformations, the CSS 3D Transforms specification is also suitable and for the browsers that don’t support it yet, a <transform> element may referenced using CSS URI syntax.

As XML3D is designed to be a thin additional layer on top of existing Web technology, its functionality is designed to be generic, orthogonal to other technologies, and is kept to the minimum needed to display and interact with 3D content. For example, XML3D is a part of the DOM, which already provides scripting and event services – most additional functionalities can usually be implemented through scripts instead of new special-purpose elements.

Harnessing JavaScript’s built-in timer functions, it is easy to create animations by performing sequential changes of the DOM. For instance, the translation properties of a group can be changed every few milliseconds to create a movement of its content. Generic DOM event listeners reacting to mouse movements and clicks can be connected directly to individual scene elements. We support attaching event handlers statically via element’s attributes, e.g. “onclick” or “onmousemove”, or dynamically via standard DOM methods like addEventListener or removeEventListener.

While the event system described by the DOM events specification of W3C is also applicable to 3D graphics, the predefined event types need to be extended to provide additional information that might be needed for certain
elements. Xflow encodes raw data via array elements that can contain raw data and other elements over references. Some `<data>` nodes may have scripts attached to them, which provide processing capabilities to the node. Xflow defines a number of built-in scripts that define basic processing tasks, such as generating displacement, UV maps, simple geometry or noise as well as more complex tasks, such as skinning, morphing, tessellation, forward kinematics, etc. When custom processing is needed, the user can define their own scripts using the `<script>` element.

Animations are typically driven by one or a few simple parameters in the source `<data>` elements. These parameters can be changed dynamically using JavaScript and the rest of the flow is computed using highly-efficient algorithms that can be directly mapped to the OpenGL vertex shaders on a GPU. While Xflow introduces a new animation concept for web-developers, it frees them from learning shader languages and provides an easy and intuitive way to encode complicated fast animations.

C. Sirikata

Sirikata is an open-source virtual world platform developed at Stanford University. The goal of the platform is to provide a set of open libraries and protocols that can be used to easily deploy a virtual world. The platform design is based on five core principles: scalability, security, federation, extensibility and flexibility.

The scalability requirement is motivated by the number of concurrent visitors in modern virtual worlds and websites. For example, World of Warcraft has reached 1 million concurrent users in China in 2008 [18] and Facebook has over 600 million active users [19].

Security has played an important role in the Internet and virtual worlds, which use publicly available servers enabling many different kinds of attacks. It is important to consider the security from the start and that is why Sirikata has...
outlined this as one of the design principles.

Federation is a principle that is controlling the Web. On the Internet, many different companies control web-pages, but no single company controls the entire network. Instead, these companies have established a set of standard protocols that allow for interoperability. Similar concepts can be applied to virtual worlds, where parts of a big seamless world can be controlled by different entities.

Stanford researchers have designed their platform to support extensible virtual worlds, which means that users and developers should be able to extend the world by creating their own content, scripts or even plugins that can offer the whole world new features.

Flexibility, on the other hand, allows their platform to be used in different scenarios. As in the Web, where different web-sites can be online shops, social networks, news, search engines, etc., developers should be able to create different kinds of worlds: virtual world online games, collaborative worlds, virtual museums, recreations of the real world and others.

All of these principles are reflected in Sirikata’s system architecture by splitting the functionality of the platform into three concerns – space simulation, object simulation and content delivery as shown in Figure 3. This is different from the traditional approach, where all the objects with their scripts and data are simulated on a single server or cluster. Instead, Sirikata separates space simulation on space servers, such as discovery of nearby objects, position synchronization, world physics, etc., from object simulation on object hosts, such as running scripts that define the behaviour of the objects.

This separation is extremely important, because it allows implementing two of the aforementioned principles. Object hosts can be run by different companies on servers that are controlled by each company respectively, effectively allowing them to interoperate through the space server, thus creating a federation. Additionally, it allows to implement a better security model, because the scripts and full object data are controlled by their owner and do not have to be uploaded to the space server or object hosts that are controlled by other organizations.

Additionally, Sirikata suggests to use an independent content delivery network, which synchronizes data that is mostly static or changes rarely, e.g. model meshes, textures. This allows offloading the network load from the space server and provides more room for communication of the frequently-updated information, e.g. object position and orientation. Such design allows a space server to handle many more objects in the world. Furthermore, a given space server can connect to other space servers that simulate other parts of the world, thus effectively increasing scalability for space simulations.

The space server and object host implementation that is provided by Sirikata is designed as a set of modules connected to the core that only implements very basic functionalities that are required to be supported by every participant in the world. By adding to or extending existing modules in the system, developers can provide new features to the world or enhance existing ones therefore providing substantial flexibility. The object host to space server communication protocol allows to register objects dynamically, thus enabling users to create content at the runtime, which makes the world extensible.

Based on the five outlined principles, Sirikata platform is a perfect fit for virtual worlds in the Web.

D. Interconnect prototype library

In order to facilitate the development process, we have implemented a prototype library that can be used to embed virtual worlds into the browser. The library is an extension of KataJS, which is a JavaScript library created by Sirikata developers, to which we have added XML3D and Xflow support. There are several advantages in using KataJS apart from the fact that it was initially designed to work with Sirikata.

Firstly, KataJS implements WebSockets protocol for communication with the Sirikata space server. Moreover, the library is optimized for use on modern multi-core architecture because it uses Web Workers [20], which is part of the upcoming HTML5 standard. Also it has a modular structure, allowing to easily replace components and parts of the system, which is an important feature that allows to quickly integrate latest research results.

In particular, each module has a communication protocol to other modules, which defines its interface, but leaves the programmer a choice for the implementation. The well-defined abstraction for the graphics subsystem allowed us to integrate the XML3D concepts into the system quickly and easily. The current implementation of the XML3D module allows loading an initial world that contains static unmodifiable parts of the world not to be synchronized by Sirikata, e.g. terrain and trees. Additionally, it allows new objects to be dynamically created when information about them arrives from the space server. Such information is first received by the networking module and then translated into a message for the graphic subsystem. When receiving such a message, the XML3D module loads the associated geometric data (mesh) from the content delivery network and once this data is loaded and parsed, a new object is added to the scene, which appears on the user’s screen. Note that loading of the mesh is asynchronous, so it does not block the XML3D module from processing other messages.

Apart from geometry data, a mesh can also contain named animations. The animations are controlled by messages to the graphics subsystem, containing the name of the animation to be played. Each object can have its own mesh, and it can also have its own animations, which can be specific to the object’s structure. For instance, despite the fact that
A browser’s built-in inspector may be used to modify existing or newly created animations for the avatar by selecting them from the library and setting additional parameters. In this case, we have mapped the jumping animation to the idle state in which the avatar would typically stay still. **(right)** A browser’s built-in inspector may be used to modify the scene in a running application, which is a convenient tool for content creators. This screenshot shows how the impact of changing a given color value in the shader is immediately visible on the screen (avatar’s shirt).

KataJS sends the same walking animation message to every avatar, they can behave differently, i.e. a human would walk using two legs, while a tiger would use all four legs and a car would rotate its wheels. In order to integrate this system with Xflow, we have extended the XML3D format with several custom XML nodes that map animation names to the parameters that need to be regularly changed at the source nodes of the flow graph. When the animation message from KataJS arrives to the XML3D module, a small script is triggered that uses the information from these nodes to drive the animation.

**IV. APPLICATIONS**

In order to illustrate the previously outlined benefits of our system, we have extended KataSpace, which is a demo virtual world application that uses the KataJS library, created by the Sirikata developers.

Thanks to its flexibility, our prototype library allows developers to quickly create new features or functionalities needed by a specific application. For instance, let’s assume one wishes to increase the speed of the avatar when the shift key is hold. As a default animation message does not have a parameter to control the animation speed, the latter would remain unchanged, resulting in an artificial look. However, by simply creating a new message and implementing its handling in the XML3D module, the animations may then be easily run at the different speeds. Moreover, exploiting the flexibility of Xflow, it is also possible to add another message that allows creating new animations for the loaded avatar. The user is then able to select an animation from the animation library, set a number of parameters, such as starting frame, length and whether the animation should loop, and map it to the animation name at run-time as shown in Figure 4 (left). This functionality would be non-trivial to achieve in the majority of the existing virtual world architectures, while using our system the feature was implemented in only in couple of hours.

Moreover, the proposed platform exposes the full set of debugging tools provided by modern browsers, allowing the creation of virtual worlds more quickly and easily. We greatly benefited from this feature when designing the prototype library and application, for instance, in order to determine the best approach for animation. These tools can also be used by designers to create virtual world content in a similar way as web-developers design web-pages – instead of changing the source code and reloading the web-page many times, one can modify some parameters in a running browser and once satisfied with the changes, copy them back to the code. An example of such modification in presented the Figure 4 (right).

In addition, the modularity of the architecture provides a great flexibility to the overall system, allowing the substitution of individual components so as to better match the needs of a specific application. For instance, instead of using WebSockets, the Sirikata space server can receive raw messages over the TCP protocol, which one may want to implement as a plugin to the browser as it may be faster and more efficient. Since the initialization and handling of the TCP protocol is similar to WebSockets, the plugin can export an interface to the JavaScript that is identical to the WebSocket object (see Figure 5). Then, in order to replace WebSockets with TCP, one would only need to change a single line of code, where instead of creating a WebSocket object, he or she would create a TCPSocket object. On a larger scale, even the virtual world platform itself can be transparently substituted if features beyond those provided by Sirikata are required, without the need to change graphics, scripting or any other component.

**V. DISCUSSION AND FUTURE WORK**

While increased, the speed of development of the virtual worlds with our prototype library still suffers from the absence of high-level protocols in Sirikata. Only generic messages between different objects are provided by the
system and most of the features that are needed by the application involve adding new message types, which are typically specific to the application and thus complicate the task of creating interoperable worlds. On the other hand, Sirikata is a perfect system for research as one can build any protocol on top of it. As one of the next steps, we are planning to work on a high-level protocol that would provide useful building blocks, simplify the application development and allow for better interoperability.

Also, the library that we have developed does not currently support a WebGL-based XML3D implementation, since the latter does not provide some of the basic functionalities that are crucial for virtual worlds, such as manual setting of the camera location. This limits the deployment to browsers that support XML3D natively, thus making it unavailable to many users. However, this issue can be easily addressed in a future release of our prototype library by implementing the missing functionalities in the WebGL-based XML3D renderer. Moreover, the current version of our library is only able to load worlds of a limited size, because the application always keeps the entire world in main memory. This is not scalable to large shared worlds, which can consist of many millions of objects and billions of triangles. To address this issue, an application should only store the part of the world that is relevant to the current location of the avatar. As such functionality is already provided by the Sirikata platform, we are planning to integrate it soon. Ultimately, our goal is to turn the current prototype into a fully functional library that can be used by developers to create virtual worlds.

Finally, as a next step in promoting the open nature of the system, we are currently starting an incubator group at the World Wide Web Consortium (W3C), which is an international community that develops standards for the Web. In this group, we will work on standardization of declarative 3D graphics in the Web, which is an essential step to integrate virtual worlds into the Web.

VI. Conclusion

In this paper, we have introduced a new open architecture combining well-established web-components with various emerging technologies, such as XML3D, Xflow and Sirikata. The system greatly simplifies the development and design of in-browser virtual worlds. Moreover, the prototype library that we have created serves as a proof-of-concept application and offers a great potential for the virtual world industry by allowing non-experts to quickly and easily create their own virtual worlds. Overall, the modular and flexible design allows the creation of many different types of worlds, such as gaming, social and collaborative environments readily accessible through the browser by the large mass of Internet users.

REFERENCES


