3drepo.io: Building the Next Generation Web3D Repository with AngularJS and X3DOM

Timothy Scully*
3D Repo Ltd
London, UK

Jozef Doboš
3D Repo Ltd
London, UK

Timo Sturm
Fraunhofer IGD
Darmstadt, Germany

Yvonne Jung
Hochschule Fulda
Fulda, Germany

Abstract

This paper presents a novel open source web-based 3D version control system positioned directly within the context of the recent strategic plan for digitising the construction sector in the United Kingdom. The aim is to achieve reduction of cost and carbon emissions in the built environment by up to 20% simply by properly managing digital information and 3D models. Even though previous works in the field concentrated mainly on defining novel WebGL frameworks and later on the efficiency of 3D data delivery over the Internet, there is still the emerging need for a practical solution that would provide ubiquitous access to 3D assets, whether it is for large international enterprises or individual members of the general public. We have, therefore, developed a novel platform leveraging the latest open web-based technologies such as AngularJS and X3DOM in order to define an industrial-strength collaborative cloud hosting service 3drepo.io. Firstly, we introduce the work and outline the high-level system architecture as well as improvements in relation to previous work. Next, we describe database and front-end considerations with emphasis on scalability and enhanced security. Finally, we present several performance measurement experiments and a selection of real-life industrial use cases. We conclude that jQuery provides performance benefits over AngularJS when manipulating large scene graphs in web browsers.

CR Categories: I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics; I.3.4 [Computer Graphics]: Graphics Utilities—Application packages

Keywords: AngularJS, X3DOM, 3D Repo, version control, BIM

1 Introduction

Within the Architecture, Engineering and Construction (AEC) industry, the idea of digital design and virtual data management was popularised by concepts such as Virtual Building by Graphisoft [Cyon Research 2003], Building Information Modelling (BIM) by Autodesk [Autodesk, Inc. 2003] and Integrated Structural Modelling by Bentley Systems [Kuhfeld 2010]. In order to benefit from these process changes, the UK Government mandated the use of collaborative 3D technologies on all public construction by 2016 [BIM Industry Working Group 2011]. The aim is to change the behavior of the construction supply chain and to adopt novel digitalized ways of collaborating based on 3D models enriched with technical specifications. Ultimately, the data formats and classifications are meant to be standardized across different levels of detail and attribute layers. The core premise is to lower the built environment capital expenditure and carbon emissions during the construction and operation by up to 20%. In support of this mandate, a Common Data Environment (CDE) as a single source of digital information was defined [The British Standards Institution 2013]. This entails design and construction documentation such as engineering drawings, 3D models, plan of works, bills of quantities, etc. As of February 2015, the UK Government unveiled BIM Level 3 Strategic Plan which further aims to move away from static file asset management systems to dynamic cloud-based hosted environments supported by databases [HM Government 2015].

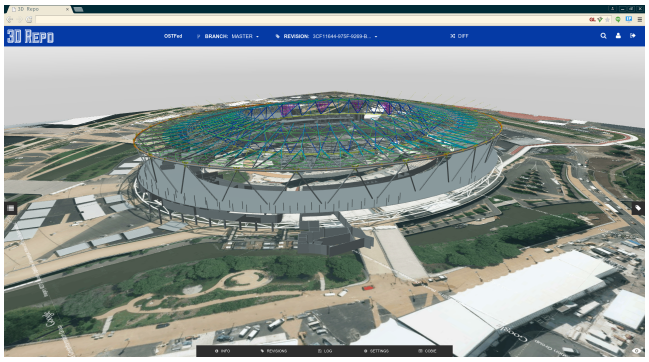

Figure 1: *Screenshot of the `3drepo.io` front-end showing the London Olympic Stadium Transformation federated 3D model. The visualisation is composed of 3 individually version controlled projects; 2 roof sections and a bowl. The total of 323,940 scene graph nodes was optimized to 880. Thanks to X3DOM, this solution runs on desktops, tablets and mobiles alike. Model courtesy of Balfour Beatty Construction Services UK.*

Yet, deliberate vendor lock-in and lack of interoperability between various instances of competing editing software are still a serious issue. Many times, even different offerings by the same vendor suffer from data loss on import due to often rapid growth by acquisition. This problem has been only partially resolved by the introduction of the Industry Foundation Classes (IFC) ISO 16739 [2013] open 3D data format which in turn is based on the Standard for the Exchange of Product Model Data (STEP) ISO 10303 [2014] dating back to 1994. Since IFC can represent some but not all aspects of the construction process, an ongoing *IFC Alignment* extension proposes new definitions and data models for infrastructure projects. Nonetheless, IFC data representation is convoluted without strict hierarchical composition joined by weak links making it prone to errors as demonstrated in the sample IFC files by the buildingSMART alliance, e.g. *Office Building* shown in Fig. 6.

Thus, we embark on a quest to fulfil the vision of the BIM Level 3 as mandated by the UK Government and create a file-format independent version control system in the cloud. The inspiration comes from XML3DRepo [Doboš et al. 2013], a Representational State Transfer (REST) Application Programming Interface (API) written as a combination of XML3D [Sons et al. 2010] front-end and 3D Repo [Doboš and Steed 2012] back-end. In order to devise a robust

*tim.scully@3drepo.org

enterprise-level solution, this paper introduces significant database (DB) and server architecture improvements. The new system has been successfully tested on several large-scale engineering projects presented throughout the paper and in the supplemental materials.

The associated client application, depicted in Fig. 1, is written in AngularJS [Hevery et al. 2009] and X3DOM [Behr et al. 2009]. Angular, as it is commonly known, provides two-way data binding in order to automatically synchronise client views with models, thus decoupling the direct Document Object Model (DOM) manipulation from the application logic. The system works by parsing an HTML web page on the client-side for custom tags to be interpreted as directives that provide bindings with a locally maintained model of the application data. Notable is the client routing which takes away processing burden from the web server. Instead of requesting fixed page templates pre-populated with data, the server provides JSON-based information which is requested asynchronously by Angular to prevent refreshing the whole web page. Working off the principle that declarative programming is suitable for connecting software components while imperative programming is better at managing application logic, we attempt to obtain the best of both worlds. Thus, we devise an Angular-based user interface (UI) logic with rendering via declarative 3D represented by X3DOM.

Contributions. Firstly, we begin by describing the existing open source and proprietary solutions. Next, we explore the benefits of unifying AngularJS with X3DOM. Hence, our contributions can be summarized as follows:

1. A practical solution to 3D version control on the web.
2. Improved NoSQL database schema for commercial 3D assets.
3. Novel REST API for X3D data serving.
4. Demonstration of case studies in an industrial setting.
5. Comparative evaluation of managing large scene graphs using jQuery and AngularJS.
6. Open source implementation available on GitHub.

2 Related Work

Central to this work is the aforementioned CDE as specified by the British Standards Institution. Its role is to act as a single source of truth for the construction project holding different types of 3D models and associated attributes together with metadata such as author, timestamp, etc. Thus, a closely related work is that of Doboš et al. [2013] which defines XML3DRepo REST API as well as performance measurements on different encoding formats. In this paper, we take a similar approach but with significant considerations for the usability of the proposed solution as well as its actual application to a commercial environment. Instead of basing the front-end on XML3D, we utilise X3DOM by Behr et al. [2009]. Previously, the benefit of XML3D over X3DOM was its fine-grained resource compositing which was not possible with the `<Inline>` functionality in X3D alone. Large-scale industrial models tend to encompass tens of thousands of components, thus, lowering the number of XmlHTTPRequest (XHR) calls becomes crucial. Fortunately, with the recent introduction of the Shape Resource Container (SRC) [Limper et al. 2014], we are able to minimize the number of requests by interleaving geometry and normals with textures inside a single server response, see §3.2. In contrast, the GL Transmission Format (glTF) [Robinet and Cozzi 2013] draft specifies an imperative encoding that might become one of the supported containers, cf. [Coughlin 2014], although without the benefits of interleaving or progressive streaming [Limper et al. 2014]. In addition, XML3D puts emphasis on animation, skinning and dataflows [Klein et al. 2012] none of which are important in architectural visualisation. Nevertheless, Schubotz and Harth [2012] also devised a prototype server with XML3D rendering, although, without external referencing of resources. Furthermore, XHR requests in combination with a NoSQL database were previously used by Olbrich [2012] on Extensible 3D (X3D) [2013] communication.

Another closely related work is that of Mouton et al. [2014]. Their system establishes a scalable model-driven web service architecture targeting 3D Computer Aided Design (CAD) in web browsers. Models are stored using macro-parametric approach in a traditional Product Data Management (PDM) system with the addition of a *3D server* which executes complex CAD operations remotely. Similarly, one of the most recent systems, still in beta, is Onshape.com, a CAD editor developed by the original SolidWorks team. This system, backed by $64m venture funding, aims to deliver radical change in collaborative 3D editing directly in web browsers à la Google Docs. Other notable browser-based editing tools include Clara.io, WebGL Studio [Agenjo et al. 2013] and Verold acquired by Box. It is our belief that a solution for ubiquitous access to 3D must support the widest range of modelling tools, hence we do not attempt editing. Instead, we rely on a domain-specific version control system [Doboš and Steed 2012] and build a solution on top.

Several solutions target the IFC data format directly. BIMServer [Beetz et al. 2010] by TNO and Eindhoven University of Technology is a dedicated open source IFC model server based on a Berkeley DB key/value data store. This system supports a basic query language [Mazairac and Beetz 2012] and automated IFC validation [Zhang et al. 2014]. Versioning is achieved on a per-object basis such that each instance in a model is assigned a unique key as a composite of an *object ID*, *project ID* and a *revision number*. However, the absence of *globally unique identifier (GUID)* values in many parts of the IFC specification makes delta change detection incomplete. In addition, the system supports subprojects that can federate multiple IFC components within the same 3D space. IFCWebServer is another open source solution that provides on-the-fly parsing and retrieval of objects from IFC files. Its associated IFCWebViewer renders models converted to COLLADA [Barnes and Finch 2008] using SpiderGL [Di Benedetto et al. 2010] library.

On the other hand, proprietary platforms such as Autodesk BIM 360 or Sunglass.io are based on three.js library and serve json-based meshes while storing attributes in a *flat file*. Graphisoft BIM Server acts as a document store with built-in version control. Component-level locking enables multi-user collaboration by sharing updates via a centralised DB. This, however, does not scale well and supports only a limited number of proprietary file formats. In addition, version control is limited to fixed assets rather than actual delta changes. Some of this functionality is also provided by commercial solutions such as GrabCAD, Trimble Connect (formerly BIMShare, then GTeam) or TeamPlatform, amongst others.

3 Architecture Overview

The aim of the `3drepo.io` Software as a Service (SaaS) platform is to provide an enterprise-level version control system in the cloud that would scale to complex 3D models without the need for installing plug-ins in web browsers. What is more, the security of the infrastructure has to be of utmost importance as high-profile construction projects are being stored and accessed remotely. Fig. 2 depicts the architecture of the overall system. On the server side, the persistent data storage is provided by 3D Repo. The *Bouncer* service enables secure access to and from the database with two tier authentication as shown in Fig. 3. Long lasting computations such as model transcoding are passed onto a queue of dedicated *Compute Nodes*. On the client side, the rendering is achieved using X3DOM while the UI interaction is handled by AngularJS, see §7.

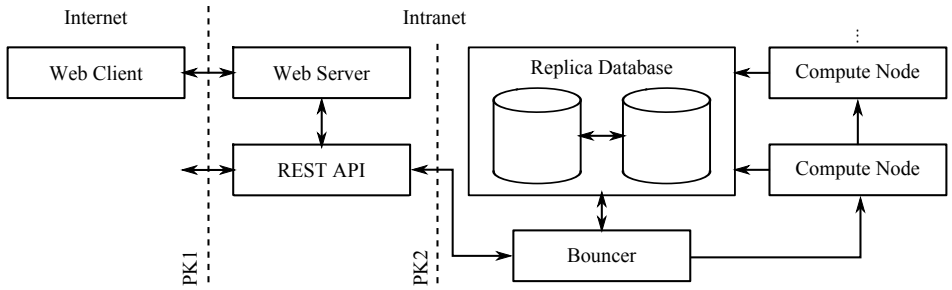

Figure 2: *Architecture overview diagram. Web request is routed from Web Server through the publicly accessible REST API. Behind a second private-key protected firewall, a Bouncer service decides on validity of the request before serving data directly from the database or processing heavy-weight computations via Compute Nodes before storage.*

3.1 3D Repo

3D Repo is an open source non-linear version control system for 3D models based on a NoSQL database MongoDB [Doboš 2015]. Various types of assets are decomposed into their constituent scene graph components and stored as individually version controlled documents in a database. These polymorphic documents consist of key/value pair properties that are transmitted and stored as Binary JSON (BSON) entities. Each document is assigned a revision tuple which consists of a *Unique ID (UID)*, a functional requirement of the database, and a *Shared ID (SID)* common to corresponding components across different revisions. Further compulsory fields include the node `type`, level of the `api` and all `paths` from the root node to itself. In order to provide a truly file-format independent version control, models are loaded using the Open Asset Import Library (Assimp) [Schulze et al. 2014] and transcoded into a unified 3D Repo specific scene graph representation. This is then uploaded to the database for storage. The scene graph is by design data agnostic and the only requirement it poses is for its nodes to be organised in a directed acyclic graph (DAG) data structure. This design enables version control on over 40 popular 3D formats including COLLADA, IFC, FBX, OBJ, DWG, etc. Additional attributes, loaded from comma separated files, can be attached as child nodes into the scene graph itself. In 2014, 3D Repo system won the MongoDB World Innovation Award in the open source category.

3.2 X3DOM

X3DOM [Behr et al. 2009] is a JavaScript-based open source framework for declarative 3D graphics on the web which relies on a polyfill implementation to include the missing parsing capability in web browsers. Based on the open ISO standard X3D [Web3D Consortium 2013] with its XML encoding, the framework seamlessly integrates interactive 3D elements into HTML that can then be easily modified at runtime via DOM scripting. Despite this, including the whole 3D scene into an HTML document, even if indirectly using the X3D `<Inline>` node comes along with its own problems. Raw ASCII-encoded 3D mesh data with position and other vertex attributes should not be part of the HTML document as it is potentially massive. Therefore, with the `<BinaryGeometry>` node, a method and a container format was presented that allows for efficient splitting of geometry from the main scene-graph. This is comparable to other popular web multimedia elements such as images or videos [Behr et al. 2012]. A further optimization that allows for progressive streaming based on criteria such as camera distances or screen sizes was presented by Limper et al. [2013a]. Following this, they integrated both the geometry and appearance into a single container called *SRC* [Limper et al. 2014]. However, with this approach the whole `<Shape>` node is no longer accessible for online modifications. Hence, the suitability of different X3D transmission formats is analysed in §6. Nevertheless, rendering is much more efficient when having flattend the scene graph to reduce the number of draw calls. This should be done before data delivery as explained by [Aderhold et al. 2013]. Yet, to maintain the ability for online modifications of individual parts of complex 3D models, the `<MultiPart>` node [Behr and Sturm 2015] was introduced in X3DOM. This builds on an offline optimization process which creates large homogenous chunks but provides an online API to manipulate the visualization based on the original data structure. During the optimization process, a single JSON-based file is generated which holds the whole information to map between the converted data chunks and the original structures. This information is later used by the API to preserve the ability to manipulate individual parts of the scene.

3.3 AngularJS

Over the past few years, Angular has become an increasingly popular choice for rapid development of dynamic HTML pages. This open source framework simplifies the design and implementation of single-page websites by providing two mechanisms; *data binding* and *client-side routing*. Their implementation is designed in such a way as to supercede the use of more established dynamic frameworks such as jQuery, even though a subset of it exists within Angular. The community actually discourages that the two methodologies be mixed, and the consensus is that everything should be implemented in one or the other. This lends itself to a dichotomic view of the client-side design which presents an interesting conundrum. On the one hand, there is the need for a fluid UI consisting of a single project page. On the other, it has to render complex 3D models with associated attributes that need to be optimized through lazy fetching for performance reasons.

Data binding. In Angular, the dynamic parts of a web page are attached to a specific *controller*. This defines the *scope* containing a set of variables that control the dynamic output. When one is updated, the output is automatically changed to reflect the new state, even for user input fields. This method of linking means that there are no direct DOM manipulations like in jQuery. If still being unavoidable, Angular has introduced the concept of *directives*. These are attached to an element in the DOM using the common HTML tag structure. Thus, directives define a connection between the element, its attributes, and the controller scope. Typically, a *watch* is set up to monitor the scope so when it changes, the expression activates and the element attributes are changed accordingly. In jQuery, in contrast, the dynamic page manipulation is performed using events and functions. If the user interacts with a page through an input element, an event is fired. A listener is attached

to this event that activates a set of functions in response. Creation of dynamic content then becomes a case of first identifying the element in the DOM, and then changing its attributes through a call to the `setAttribute()` function. Thus, a key difference between jQuery and AngularJS is the requirement for prior knowledge of the DOM. When a page is manipulated by Angular, it must first know which tags in the DOM have an attached directive. Hence, it must perform a compilation step where it loops over all the elements in DOM to parse the directives. In jQuery, the code already knows which elements to manipulate, or uses the browsers' built-in query selectors to find the elements in DOM. Typically, the direct DOM access is implemented using a tree structure, hence retrieving individual elements can be more efficient. In the case of 3D scenes, this can have serious performance implications, see Fig. 5 for examples.

Client-side routing. When the user navigates to another page, Angular can utilise client-side routing in order to dynamically switch the contents of a page without requiring a refresh. This happens by switching the website template that controls the layout of the content, and the controller that is in charge. If the viewer needs to update, AngularJS will call JavaScript code on a specially designed viewer object that controls loading of X3DOM elements efficiently without the need for a refresh.

4 Database Schema

We extend the previous work of Doboš et al. [2013] to offer both commercial and performance improvements. The first step is to switch the database conceptually from per project to per account. This provides greater security while simplifying the backup procedure at the same time. The second step is to introduce a separate pre-processed X3DOM cache into DB. This allows for a specific scene to be served up quickly, without the need of repeated transcoding. In the final step, new types of scene graph nodes are introduced that support the addition of attributes and metadata, federation of different projects and visualisation of map tiles.

4.1 Access Control

In XML3DRepo, the design of the system was based around establishing functionality, thus did not include security features. In a commercial offering, however, data security and access control are of paramount importance. Engineering models and semantic data can offer opportunities to nefarious groups that might wish to exploit them. To mitigate these, the notions of *users*, *groups* and *companies* are introduced. Each account is given a separate DB to store their projects creating a natural Chinese wall on the same instance. This means sensitive information can be protected not only from other companies or organizations, but groups within the same organization. Thus, in each DB, there is a new set of collections storing information on users/groups detailing their access privileges. Previously in 3D Repo, each repository represented a project containing two collections, a history and a scene. The history, stored as a DAG of revisions, contained the lists of *added*, *deleted*, *modified* and *unmodified* components. The scene collection, on the other hand, contained the actual 3D assets represented as a scene graph, also a DAG. In this paper, the notion of a DB is modified from representing a single project to a *company* which stands for a user, an organization, or a business unit. Each project then becomes a set of collections stored in a database, namely `db.project.scene` and `db.project.history`. In addition, the `db.project.users` collection is added that details the users and their permissions on a specific project. Furthermore, projects can be made *public* in which case anyone has read but not write permissions. In the particular use-case of a NoSQL data store, a further advantage is achieved by preserving each DB as a separate file on a disk. Thus, backing up a particular account's data is as simple as backing up the file. Hence, accounts can be stored in different geographical locations, while still using the same "cloud".

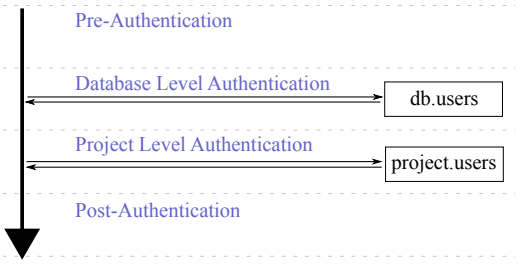

Figure 3: *Two step authentication process at DB and project levels.*

4.1.1 User Authentication

To facilitate access control on specific projects, the system needs to store a set of user credentials. These contain the username and password plus additional attributes such as email, first and last name, etc. To simplify and strengthen the implementation, these are stored using the DB's built-in user management system which provides several distinct advantages as follows:

1. Firstly, the database automatically protects the user details with its own role-based system. Hence, the ability to read and write user credentials can be tightly controlled. For example, a web-based front-end can be given permissions to change other users' passwords, but not its own. The same applies to roles. This means that an intruder pretending to be the front-end user would have great difficulty in overtaking the account or gaining super-user permissions. In addition, the safe storage of passwords is controlled through a well tested mechanism of the DB itself.
2. Secondly, it allows the inclusion of enterprise-level features such as Kerberos [Neuman et al. 2005] or Lightweight Directory Access Protocol (LDAP) [Sermersheim 2006]. Using a unified authentication method means that collections, and therefore projects, can be locked at a database level. Thus, a company that decides to situate our solution on premise rather than in the cloud can have full control over access. This is especially important for those users who have raw access to the database, and may have restrictions imposed through legal entities. Such a situation is prevalent in consultancies, where different teams within the same company may be working for competing clients.

Once the user has been authenticated, the system checks their permissions against a specific project. Due to their sensitive nature, the permissions are stored in the `db.project.users` collection within a company database. In the enterprise version, this allows read/write permissions to be restricted per project through DB collection roles. This completes a two-step authentication process, shown in Fig. 3, controlling access to project resources. As a convenience measure, the users store a set of bookmarks in their profile to keep track of projects. These are separate from the permissions stored at the project-level, thus can be links to data the users do not necessarily have permissions on. If trying to access it, however, they will be denied access at the point of entry.

4.2 Additional Node Types

A set of scene graph node types was defined in [Doboš 2015], namely `cameras`, `comments`, `materials`, `meshes`, `revisions`, `textures` and `transformations`. We extend

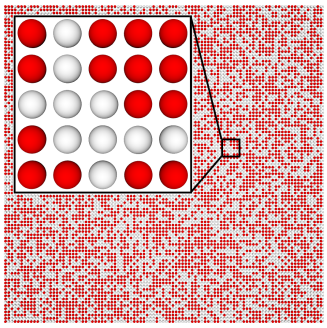

Figure 4: *Test scene with 10,000 spheres being randomly colored.*

this list by 3 new nodes. The first is a `metadata` node containing information about a particular sub-tree of the scene graph. The second is a `reference` node that allows the collation of multiple projects into a single scene graph. The third is a `map` node for longitude and latitude values that point to map tiles from systems such as Google Maps, e.g. Fig. 1. See GitHub for detailed definitions.

Metadata node. The metadata node adds the ability to store descriptive information, i.e. attributes, about its parent in the scene graph. Although it is not the only application, this is particularly of interest in 3D architectural projects. It is common that these models have engineering descriptions beyond what can be represented in 3D such as construction materials, quantities, technical specifications, etc. This information is often stored directly in common file formats such as IFC and be extracted and imported into the 3D Repo system. The structure of document-based NoSQL DBs mean that storing metadata is very straightforward as it can be stored and queried as key/value pairs natively. Thus, scene nodes can be retrieved based solely on their meta properties. A query would consist of obtaining a set of metadata documents and returning their parents. The DB can even be queried without any reference to 3D data. This allows for statistical analysis using the *Big data* abilities of NoSQL.

Federation node. Analogously to a submodule in Git, federation provides a way of linking projects from the same or another DB. This gives the users the ability to join specific revisions of projects into the same 3D scene. This is of great utility to users who may wish to swap variations on a single design quickly. In addition, it allows them to continue working on a particular model while others still access some previous reversion concurrently. The federation node shares the same common attributes with other nodes as specified in §3.1. Additional fields specify the DB and project along with the required branch and revision. Typically, it will be a child of a transformation that aligns the subproject with the rest of the scene.

Map node. The map node allows us to include any map tile from the Google Static Maps API into the scene as a set of textured polygons. The center of the tiles corresponds to a specific latitude and longitude at Google zoom level. Around this point, a set of tiles is generated such that there are $\texttt{width}^2$ squares in total, where each tile has geometric world size of `worldTileSize` in the scene. To support alignment with the rest of the scene, the map tiles can rotate around the up vector, i.e. the y-axis, by `yrot` radians.

5 AngularJS vs jQuery

In general, for complex scene graphs and X3DOM manipulation, jQuery should have a distinct advantage over Angular, whereas for simple UI design Angular tends to be much simpler to implement. Therefore, to test the performance implications of using one or the other, we perform the same set of three experiments in both libraries and compare the differences in time it takes to render a single frame. These experiments are based on manipulating a set of N^2 spheres laid out in a square grid. Typically, the size of an architectural 3D model can approach many thousands of nodes, and therefore we base the experiments on grid sizes ranging from a few to approximately 10,000 nodes as shown in Fig. 4.

A unique ID is assigned to each sphere representing its position in the grid. In jQuery, the spheres are manipulated directly using the ID with the query selector. In AngularJS, on the other hand, a directive is attached to each sphere element which is used to manipulate it. These directives have a watch expression that monitors a boolean flag in an array attached to a controller scope. This then enables the manipulation of spheres based on their ID, too. The first method, in most modern browsers, is implemented using a hashmap enabling constant time access to an element by ID. Similarly, for the second test the directive activates only when the switch is flipped. This provides a one-to-one mapping between activation and elements in the scene. Thus, both tests are agnostic to scene graph complexity, as there is no graph traversal. The only factor is the mechanism of manipulating scene elements, and therefore the results are equally applicable to other scenes, e.g. buildings, cities, etc., regardless.

In order to eliminate time required to create the grid scene, two ready-made X3D files were prepared beforehand. For jQuery, this means an X3D file with ID fields on the HTML elements which was then imported into X3DOM using its built-in `<Include>` tag. This reads the file into the scene. For Angular, this means an X3D file with directives attached to each element. As Angular is required to parse and compile the file separately, we cannot use the X3DOM loader directly. Instead, a two stage loading process has to be performed using a new separate directive that combines `ng-include` from Angular, and `<Include>` from X3DOM. The first stage loads the file and adds the elements to the DOM. This then automatically triggers X3DOM to start adding these elements to the scene. Finally, Angular compiler attaches the directives to the elements in DOM. Once ready, the following experiments were performed:

1. The first experiment consists of repeatedly changing the color of a random subset of nodes in the grid. At each iteration, subset using a random number generator is selected. The colors of any previously changed spheres are reset, so that the new colors can be set. As described previously, jQuery can access the DOM elements directly via their IDs. In Angular, however, a boolean in an array that is watched by a particular element directive in the DOM is flipped. Thus, the performance was tested across increasing sizes of the selected subset in the grid. This evaluated the speed at which each system can manipulate DOM elements.
2. The second experiment consisted of switching on the rendering of a particular sphere in a variable size grid. At each iteration, a node is randomly selected and any previously selected node is switched off. By only switching on a single node, the rendering time of X3DOM can be kept constant. This, therefore, tests the efficiency of each library's ability to select a single node from a pool of nodes by varying the size of the grid.
3. In the final experiment, the relative startup time of each library was measured. For jQuery, it was the time that is required to add the elements to DOM using X3DOM internals. This is the same for both libraries, therefore dismissed. For Angular, however, the overhead was the time it takes to compile the scene to add directives, before the elements can be added.

Fig. 5 (a) shows the results of varying the number of changed nodes for a fixed grid size. Initially, the performance of jQuery is higher than that of Angular. However, after approximately 500 and more nodes being switched, the performance advantage swaps. This

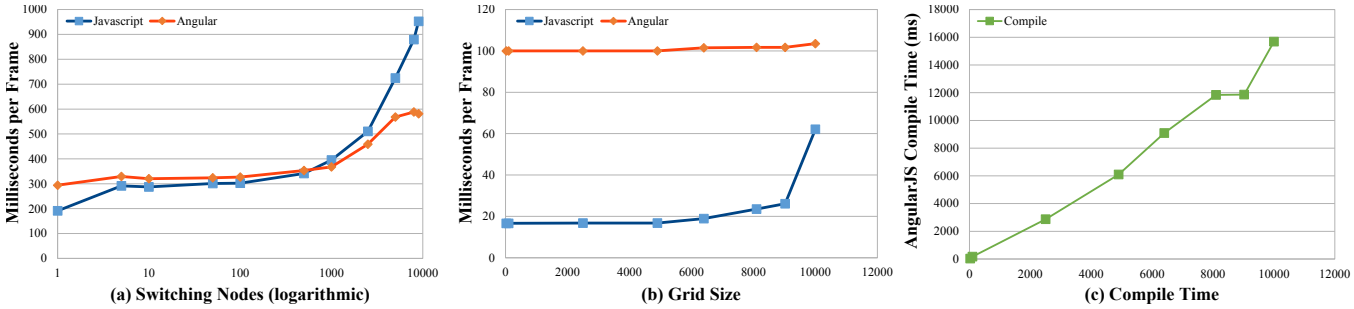

Figure 5: *Performance measurements. (a) Number of milliseconds to render each frame against the number of nodes switching on and off in an iteration. Note that X-axis is logarithmic. (b) Number of milliseconds to render each frame for a single node out of a random selection from a given grid size. (c) Overall compile time of AngularJS in milliseconds required to initialize a grid size in (b).*

Figure 6: *Office federated from the Architectural, Structural, and MEP models provided by the buildingSMART alliance initiative. Note that the second storey was hidden to reveal the interior.*

shows that from that point onwards Angular performance increases due to its internal functionality of watching switch variables, rather than looping over an increasing number as in jQuery. However, Fig. 5 (b) demonstrates that when rendering a fast scene, the Angular's internal DOM manipulations mean that the performance is severely rate-limited. This, coupled with the increasing compile times shown in Fig. 5 (c) demonstrate that AngularJS is a poor choice for manipulating X3DOM scenes.

6 Transmission Formats

X3DOM offers a selection of different sub-formats for delivery of meshes with associated attributes such as vertices, normals, indices and texture coordinates. The first is the XML encoding of X3D where attribute arrays are presented as UTF-8 encoded floating point arrays. The second is BinaryGeometry [Behr et al. 2012], where each of the attributes is retrieved through a separate request such that each response is the raw binary data to be uploaded to GPU directly. The final is SRC [Limper et al. 2014] which comprises all binary attributes in a single request preceded by a JSON header. Similar to SRC is Blast [Sutter et al. 2014], although this was not evaluated as it is based on XML3D, not X3DOM. However, due to the similarities, comparable results are likely to be obtained.

To test the different bandwidth requirements of each of the formats, the same 3D model, shown in Fig. 6, was evaluated. This is an unoptimized example IFC project by the buildingSMART alliance and comprises an office building with Architectural, Structural and Mechanical, Electrical and Plumbing (MEP) information present. The total size of the downloaded data was measured both as uncompressed and GZip compressed, just like in [Doboš et al. 2013] and [Limper et al. 2013b]. However, our 3D model has 5,844 scene components which is several orders of magnitude larger than in either of the previous papers. This, in our opinion, demonstrated the advantages and disadvantages of each of the encodings much better.

As shown in Tab. 1, the ASCII format is improved the most by compression, since it is the least bit-packed. However, it is non-intuitive that the compressed ASCII is the most bandwidth efficient. The large difference in bandwidth comes from the number of requests each format requires. In ASCII, there is only one request for each sub-scene that is part of the federation. This single request contains all the information required to reconstruct the scene and so no further requests are required. In BinaryGeometry, each of the mesh attributes is a separate request to an external resource, thus there are multiple requests per object in the scene. In SRC, however, there is only a single request per object. Furthermore, for each request that is sent, the size of the response header of ∼500 B has to be included, too. Once the scene reaches many thousands of nodes, as the chosen example 3D model does, this becomes a large overhead. In addition, for SRC, every request includes a header describing the format of the binary data, compression on which does not have as large effect as it does not ASCII.

Despite the increased bandwidth usage due to a large number of requests, the advantages of such an implementation have to be considered. By splitting the scene, web browsers can utilize their native parallelization abilities. Additionally, a scene can be lazily loaded, as described in §7.3, according to visibility and spatial position. Thus, a compromise would be reached by using the SRC format but reducing the size of its header. One further disadvantage of BinaryGeometry is that the number of requests is the number of attributes multiplied by the number of shapes. In office federation, this overwhelmed the browser which would shut down after too many requests, remedied only by reducing the model that was being visualized simultaneously, thus incrementing the requests slowly. This highlights not only a performance degredation in BinaryGeometry, but also a functional one.

Format	Raw [MB]	Gzip [MB]	#Requests
ASCII	36.5	4.0	9
Bin Geometry	28.7	23.4	15,870
SRC	25.3	21.1	4,729

Table 1: *Comparison of different data transmission formats for their speed of delivery and bandwidth requirements.*

7 Implementation

This section describes the two different variations on the system implementation. The first is a *cloud-hosted* SaaS, while the second is an enterprise *on premise* solution. Both are based on the same architecture of the MEAN stack, i.e. a combination of MongoDB, Express, AngularJS and node.js [Dickey 2014]. This was selected as it provides scalability and fits well with the 3D Repo system architecture on top of which this platform is built. The public facing *Web Server* and *REST API* services in Fig. 2 are implemented in node.js and JavaScript. This enables a seamless transition of the same language between the client and the server. The *Bouncer* and *Compute Node* services derive directly from *3D Repo Core* C++ lib.

7.1 Security

Base architecture of the system provides security features that are shared by both cloud-hosted and on premise flavors as follows. Firstly, there are the layered firewalls depicted as public-key PK1 and PK2 interfaces in Fig. 2. PK1 faces the Internet directly. This allows HTTP and HTTPS connections to pass through redirected ports on the server. To connect via a Secure Shell (SSH), public-key encryption on a non-standard port has to be used. This enables access to the Intranet where the web services reside. Once inside, the initial zone user is unable to query the DB directly and must make requests through the *Bouncer* service with which other services interact. In order to gain access to the second level behind PK2, the user must create a tunnel through the first zone. Therefore, to access the already password protected DB directly, the intruder would have to obtain two different private keys. Yet, maintenance of the services can continue with only the first key. Furthermore, the servers communicate via Secured Socket Layers (SSL) providing a minimum of 128-bit encryption on data transmission. In order to provide additional layer of security, the certificate is verified by a Certificate Authority for the Web Server and the REST API equally. The two flavors of the system provide user authentication in slightly different ways. Both access the DB through the Bouncer service that performs authentication and controls permissions to accounts and projects. The REST API, however, logs onto the DB with a single account, and controls permissions manually. This is necessary due to limitations of MongoDB which only allows up to 100 concurrent connections simultaneously precluding the use of a connection per user. In a cloud-based system, this is sufficient as users will not have direct access to the DB hidden behind two firewalls. In contrast, for an enterprise on premise solution, collection level locking is added alongside the permissions already described. This enables a small set of users to run Bouncer services directly from their desktop 3D Repo GUI application rather than from the server.

7.2 REST API

This section defines a REST API that provides access to 3D assets. In [Doboš et al. 2013], the API was based around a two-way encoding as a combination of `type` and `id` variables. Depending on their ordering, the API supported addressing of collections of resources, single resources and even their individual attributes such that the `id` was either the UID or SID depending on context, see §3.1. Although novel, it was not necessarily user friendly. Therefore, we build upon this by the inclusion of access control and security features, and by significantly simplifying the respective calls.

All requests are pre-pended with `/account/project` for the server to reject any calls to sub-functions if the user has not been validated for the specific account and project. In the case of a public project, all reads are allowed but writes are still restricted on a user by user basis. In addition, calls to the API are followed by a file extension which represents the format that the data is to be returned in. For instance, if `/account.json` is requested, the account information in JSON format is returned. If, however, a `.jpg` extension is used, then the avatar associated with the account is returned instead. For 3D assets, the extensions `.x3d.[src|bin|txt]` are used to request assets in X3D SRC, BinaryGeometry and XML respectively. In the future, other formats will be added accordingly. To enforce user permissions, all calls to the API are authenticated via cookies. Further major enhancements where blue color denotes variables are as follows:

`/login` [POST] Allows a user to log in via a POST request containing a JSON wrapped username and password. If validated the response contains a time-limited cookie to be used onwards.
`/login` [GET] Returns a HTTP status code OK (200) if logged in, Unauthorized (401) otherwise.
`/account/project` [GET] Returns the whole project page.
`/account/project/uid.ext` [GET] Returns a specific asset with a Unique ID `uid` where `ext` specifies the file format.
`/account/project/revision/branch/rid` [GET] Returns a scene by `branch` name or UID and revision ID `rid`.
`/account/project/revision/branch/rid/sid` [GET] Returns an asset by `branch` name or UID, revision ID `rid` and Shared asset ID `sid`.

7.3 Front-end

Based on the results of the experimentation in §5 and §6, the client web application UI was written in Angular while all of the X3DOM scene graph manipulation was achieved via jQuery. The Angular controller switches the contents of the page dynamically by requesting JSON information from the server. As shown in Fig. 1, this entails various tabs such as general project information, comments, revisions, logs and settings. At the base of the system, there is a X3DOM viewer which is loaded by Angular from a URL. This makes the viewer embeddable on other web pages without the need to rely on Angular as it is fully self-contained. A user profile page, on the other hand, lists the available projects for the user as well as settings to change their own account details including password. Further examples are available in the supplemental materials.

Lazy fetching. The federation node, described in §4.2, combines multiple projects into a single visualization. This is achieved when transcoding the scene into X3DOM. The algorithm recurses over the scene graph and outputs the equivalent X3D nodes as it progresses. To represent a federation, an `<Inline>` node is generated which calls the REST API to return another X3D file of the subproject being federated. This use of a recursive inlining structure offers several distinct advantages over producing a single monolithic file. For every subproject, its bounding box is attached. This allows X3DOM to optimize lazy loading of the federated scene. For a project outside the view, or too small to see, X3DOM can cull and therefore delay the loading of external resources. In addition, the use of inlines optimizes the caching on the client side. Each call to the REST API encapsulates the branch and revision IDs of the scene, and so is cached accordingly. If the same revision is used in another federation, then the X3D file is retrieved from the local cache, rather than being queried from the REST API again. For a single base project used across multiple revisions, this represents a significant saving in bandwidth/cost for users and providers alike.

8 Applications

In order to demonstrate the feasibility of the proposed solution, presented here are a number of real-life use case studies that are supported by the `3drepo.io` system.

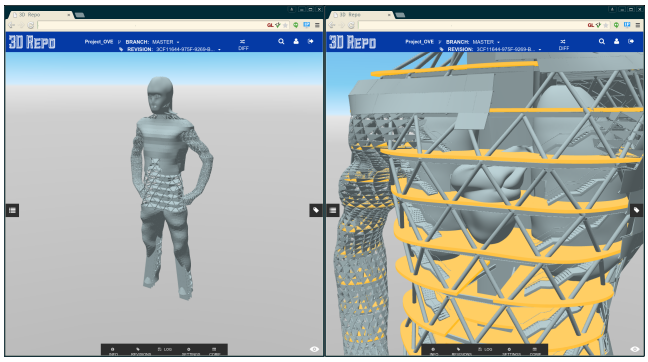

Figure 7: Left: *Project OVE federated in* `3drepo.io` *with over 23m triangles.* Right: *Zoom in on chest cavity showing internal stairs and structures. Model courtesy of Arup Associates.*

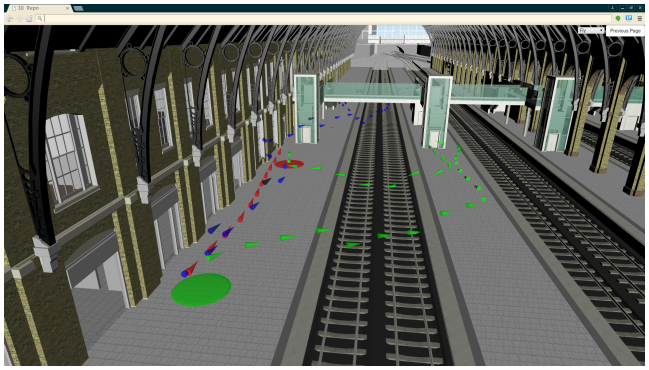

Figure 8: *Wayfinding visualisation in King's Cross Station. Glyphs represent recorded position and camera orientation with 1sec spacing. Model courtesy of Network Rail.*

Online visualisation. *Olympic Stadium Tranformation (OST)* project by Balfour Beatty is a £190m conversion to host Rugby World Cup in 2015 and become the permanent home of West Ham United in 2016. Currently, there are over 20 different companies and subcontractors working on various aspects of the project. Unfortunately, no one software has been contractually specified, thus different companies rely on different solutions throughout the build. What is worse, structural difficulties in the roof design and tight schedule make the task even more complex. Therefore, the ability to share and visualise the model directly in web browser even on low end devices such as tablets is very useful. Fig. 1 shows the federation of the main bowl with 2 separate roof sections. In order to position the 3D model within the context of the surroundings, employed are also the newly introduced map nodes. The heavily optimized model consists of over 8m polygons and 450 shapes.

Systems federation. *Project OVE* by Arup Associates is a trial testing the potential of BIM. This 170m tall building in the shape of a human, a laser scan of one of the employees, is an example of different architectural components working in synchrony. These include a full MEP model to represent the respiratory and circulatory systems as well as steel structure in place of skeleton and cladding for skin. The resulting 3D representation consists of multiple individual sub-projects exported as FBX from Autodesk Revit. Fig. 7 shows the final heavily optimised federated model in our system.

Wayfinding user study. The aim of this user study is to evaluate the signage before being physically built. By doing so, the design team is able to test the usability and navigability of the station in virtual space what significantly reduces the costs in comparison to detecting and fixing errors post build. To devise this solution, the location and orientation of the user's camera is recorded every second. Such recordings are double buffered per each 10s client-side before being posted via REST API as JSON. Afterwards, the recordings can be visualised within the same 3D model using glyphs to denote the position and orientation as demonstrated in Fig. 8. Apart from signage, such information can be used to evaluate advertising spaces and shop locations within the BIM environment. Although this particular project is already built, our solution is now being applied on live projects such as the MTR Admiralty Station in HK.

9 Conclusions

We have presented a novel web-based architecture for management and visualisation of version controlled 3D assets. This was built using AngularJS and X3DOM with a cloud-based repository provided by 3D Repo. Significant improvements have been made to the database schema in order to devise a commercially viable solution with enhanced security features. These include the introduction of users, groups and accounts into the system, two layer authentication process and new scene graph node types for meta attributes, references and maps. The DB schema for attributes can be easily restricted to specific templated keys on insertion. In addition, a new REST API was introduced which is simplified in comparison to previous work. In order to support large-scale federated 3D models, lazy fetching via X3D inline functionality was utilised client-side. Several experiments compared the performance of scene graph manipulation with varying sizes using jQuery and AngularJS. The conclusion is that due to the very long parsing times in Angular, it becomes impractical with the increasing complexity of a 3D scene. Furthermore, Angular suffers from a fixed overhead on a per-frame manipulation regardless of the scene complexity when comparing to jQuery. Nevertheless, once compiled and running, on very large scenes Angular is able to outperform jQuery. What is more, to compare the relative advantages of different encoding strategies for X3DOM, formats such as XML, BinaryGeometry and SRC were evaluated on a test scene with a large number of components. Even though XML can be compressed well, it cannot reference individual meshes. In contrast, SRC and BinaryGeometry come with a large number of requests and additional overhead for response headers and JSON format description in the case of SRC. This would be the most preferred option, nevertheless. Finally, three industrial use case studies demonstrated the feasibility of the proposed solution.

Future work. In its current implementation, the system provides a read-only access to the contents of the repository. Therefore, in the very near future, we will add the ability to upload and process various types of 3D assets via web browsers in order to remove the need for the 3D Repo GUI desktop application which currently provides the write access. Although 3D Repo can already heavily optimise scene graph structures, this still needs to be married with X3DOM MultiPart on the client. The `3drepo.io` portal is in beta testing with select clients and will be opened to the public in July.

Acknowledgements

We would like to thank Maciej Gryka and Anke Pohl for their work on AngularJS as well as Neil Thompson, Casey Rutland and Carl Collins for access to the industrial projects featured in this paper. This work has been funded by the Breakthrough Information Technology Exchange and the Innovate UK grants. 3D Repo is supported by the EIT ICT Labs and the Open Data Institute.

References

ADERHOLD, A., JUNG, Y., WILKOSINSKA, K., AND FELLNER, D. W. 2013. Distributed 3d model optimization for the web with the common implementation framework for online virtual museums. In *Proceedings Digital Heritage 2013*, vol. 2.

AGENJO, J., EVANS, A., AND BLAT, J. 2013. Webglstudio: A pipeline for webgl scene creation. In *Proceedings of the 18th International Conference on 3D Web Technology*, Web3D '13.

AUTODESK, INC. 2003. Building information modeling. White paper, Autodesk Building Industry Solutions, San Rafael, CA.

BARNES, M., AND FINCH, E. L. 2008. Collada - digital asset schema release 1.5.0. Tech spec, Khronos Group.

BEETZ, J., VAN BERLO, L., DE LAAT, R., AND VAN DEN HELM, P. 2010. Bimserver.org–an open source ifc model server. In *Proceedings of the CIP W78 conference*.

BEHR, J., AND STURM, T. 2015. *MultiPart - Offline creation and online API*. Fraunhofer IGD.

BEHR, J., ESCHLER, P., JUNG, Y., AND ZÖLLNER, M. 2009. X3dom: A dom-based html5/x3d integration model. In *Proceedings of the 14th International Conference on 3D Web Technology*, ACM, Web3D '09.

BEHR, J., JUNG, Y., FRANKE, T., AND STURM, T. 2012. Using images and explicit binary container for efficient and incremental delivery of declarative 3d scenes on the web. In *Proceedings Web3D 2012: 17th Intl. Conference on 3D Web Technology*.

BIM INDUSTRY WORKING GROUP. 2011. Government construction strategy. Policy paper, The UK Cabinet Office.

COUGHLIN, B. 2014. 3d for the modern web: Declarative 3d and gltf. Tech. rep., GMU CS-752.

CYON RESEARCH. 2003. The building information model: A look at graphisofts virtual building concept. White paper.

DI BENEDETTO, M., PONCHIO, F., GANOVELLI, F., AND SCOPIGNO, R. 2010. Spidergl: A javascript 3d graphics library for next-generation www. In *Proceedings of the 15th International Conference on Web 3D Technology*, ACM, Web3D '10.

DICKEY, J. 2014. *Write Modern Web Apps with the Mean Stack: Mongo, Express, AngularJS, and Node.js (Develop and Design)*, first ed. Peachpit Press. ISBN-10: 0133930157.

DOBOŠ, J., AND STEED, A. 2012. 3d revision control framework. In *Proceedings of the 17th International Conference on 3D Web Technology*, ACM, Web3D '12.

DOBOŠ, J., SONS, K., RUBINSTEIN, D., SLUSALLEK, P., AND STEED, A. 2013. Xml3drepo: a rest api for version controlled 3d assets on the web. In *Proceedings of the 18th International Conference on 3D Web Technology*, ACM, Web3D '13.

DOBOŠ, J. 2015. *Management and Visualisation of Non-linear History of Polygonal 3D Models*. EngD thesis, UCL.

HEVERY, M., MINÁR, I., AND JÍNA, V., 2009. Angularjs. Google.

HM GOVERNMENT. 2015. Digital built britain: Level 3 building information modelling - strategic plan. Policy, The UK Department for Business Innovation and Skills. URN BIS/15/155.

ISO 10303-242. 2014. Industrial automation systems and integration – Product data representation and exchange – Part 242: Application protocol: Managed model-based 3D engineering. ISO.

ISO 16739. 2013. Industry foundation classes (ifc) for data sharing in the construction and facility management industries. ISO, buildingSMART International Ltd.

KLEIN, F., SONS, K., JOHN, S., RUBINSTEIN, D., SLUSALLEK, P., AND BYELOZYOROV, S. 2012. Xflow: Declarative data processing for the web. In *Proceedings of the 17th International Conference on 3D Web Technology*, ACM, 37–45.

KUHFELD, R., 2010. Bentley's integrated structural modeling brings structural engineers into integrated project workflows.

LIMPER, M., JUNG, Y., BEHR, J., AND ALEXA, M. 2013. The pop buffer: Rapid progressive clustering by geometry quantization. *Computer Graphics Forum (Pacific Graphics 2013) 32*, 7.

LIMPER, M., WAGNER, S., STEIN, C., JUNG, Y., AND STORK, A. 2013. Fast delivery of 3d web content: A case study. In *Proceedings of the 18th International Conference on 3D Web Technology*, Web3D '13.

LIMPER, M., THÖNER, M., BEHR, J., AND FELLNER, D. W. 2014. Src - a streamable format for generalized web-based 3d data transmission. In *Proceedings of the Nineteenth International ACM Conference on 3D Web Technologies*, Web3D '14.

MAZAIRAC, W., AND BEETZ, J. 2012. Towards a framework for a domain specific open query language for building information models. In *EG ICE*.

MOUTON, C., PARFOURU, S., JEULIN, C., DUTERTRE, C., GOBLET, J.-L., PAVIOT, T., LAMOURI, S., LIMPER, M., STEIN, C., BEHR, J., AND JUNG, Y. 2014. Enhancing the plant layout design process using x3dom and a scalable web3d service architecture. In *Proceedings of the Nineteenth International ACM Conference on 3D Web Technologies*, Web3D '14.

NEUMAN, C., YU, T., HARTMAN, S., AND RAEBURN, K. 2005. The kerberos network authentication service (v5). RFC 4120.

OLBRICH, M. 2012. Accessing http interfaces within x3d script nodes. In *Proceedings of the 17th International Conference on 3D Web Technology*, ACM, Web3D '12.

ROBINET, F., AND COZZI, P. 2013. gltf - the runtime asset format for webgl, opengl es, and opengl. Tech spec, Khronos Group.

SCHUBOTZ, R., AND HARTH, A. 2012. Towards networked linked data-driven web3d applications. In *Dec3D*.

SCHULZE, T., GESSLER, A., KULLING, K., NADLINGER, D., KLEIN, J., SIBLY, M., AND GUBISCH, M., 2014. Assimp.

SERMERSHEIM, J. E. 2006. Lightweight directory access protocol (ldap): The protocol. RFC 4511, Network Working Group.

SONS, K., KLEIN, F., RUBINSTEIN, D., BYELOZYOROV, S., AND SLUSALLEK, P. 2010. Xml3d: Interactive 3d graphics for the web. In *Proceedings of the 15th International Conference on Web 3D Technology*, ACM, Web3D '10.

SUTTER, J., SONS, K., AND SLUSALLEK, P. 2014. Blast: A binary large structured transmission format for the web. In *Proceedings of the Nineteenth International Conference on 3D Web Technologies*, ACM, Web3D '14.

THE BRITISH STANDARDS INSTITUTION. 2013. Pas 1192 specification for information management for the capital/delivery phase of construction projects using building information modelling. .

WEB3D CONSORTIUM. 2013. Extensible 3d (X3D). Specification.

ZHANG, C., BEETZ, J., AND WEISE, M. 2014. Model view checking: automated validation for ifc building models. ECPPM.