Hybrid client-server and P2P network for web-based collaborative 3D design

Caroline Desprat
University of Toulouse
118 Route de Narbonne
Toulouse, France
desprat@irit.fr

Hervé Luga
University of Toulouse
118 Route de Narbonne
Toulouse, France
luga@irit.fr

Jean-Pierre Jessel
University of Toulouse
118 Route de Narbonne
Toulouse, France
jessel@irit.fr

ABSTRACT

Our proposed research project is to enable 3D distributed visualization and manipulation involving collaborative effort through the use of web-based technologies. Our project resulted from a wide collaborative application research fields: Computer Aided Design (CAD), Building Information Modeling (BIM) or Product Life Cycle Management (PLM) where design tasks are often performed in teams and need a fluent communication system. The system allows distributed remote assembling in 3D scenes with real-time updates for the users. This paper covers this feature using hybrid networking solution: a client-server architecture (REST) for 3D rendering (WebGL) and data persistence (NoSQL) associated to an automatically built peer-to-peer (P2P) mesh for real-time communication between the clients (WebRTC). The approach is demonstrated through the development of a web-platform prototype focusing on the easy manipulation, fine rendering and light update messages for all participating users. We provide an architecture and a prototype to enable users to design in 3D together in real time with the benefits of web based online collaboration.

Keywords

WebRTC, WebGL, collaborative, Peer-to-peer, Applications, Web

1 INTRODUCTION

As Ortiz [OJ10] questioned if "3D is finally ready for the web?", the Internet responds with a large amount of creation, transmission, storage and access solutions for 3D contents [ERB14]. 3D CVEs (Collaborative Virtual Environments) are representative of this increasing popularity in industry and makers communities ; due to the competition and the mobility nowadays, people are turning faster toward the optimal resources. A good example of this trend is the emerging market of 3D collaborative modelers, server-based such as GradCAD, ThinkerCAD, Sunglass.io, Clara.io [HLL+13], Verold Studio or cloud-based like AutoCAD360. The collaborative aspect in 3D modeling CVEs shows the need of an efficient cooperation over the network between the users. Even if they are geographically far and have different points of interest, they have the same shared goal: the manufacture of the product.

This research is led by the desire of working collaboratively and sharing 3D scenes across the network. Large scenes or complex models are very likely to be constructed and reviewed by more than one person, especially in the context of 3D design, PLM (Product Lifecycle Management) or BIM (Building Information Modeling) solutions. The new usages and the increasing mobility of workers are pointing to web based solutions. Moreover, in small designing teams, we can observe that the design process is conducted with direct communication channels. The mimic of direct communication in computing is peer-to-peer (P2P): why to pass through a proxy when the team members are so close? We can also observe that the need of persistent communications is mandatory with this running. Since the network speed can be a limiting factor in collaborative design, one of the main criteria for our system is to spread and display only relevant information between the users without overloading the server.
The contributions of this work are multi folds:

- consider the solutions for plug-in-less visualization of 3D scenes on the web,
- use efficiently the local client resources for visualization and storage,
- allow small and asynchronous message system,
- overcome the difficulties related to interactive collaboration across the network with shared access to 3D models with bandwidth limits of the actual connections.

The reminder of this paper is organized as follows: the related work in distributed collaborative modeling is in section 2, our model architecture is described in section 3, the implementation of the web editor, the server architecture with the storage mechanisms and the P2P

 ISBN 978-80-86943-65-7

communication layer with the synchronization are explained in section 4. Then we introduce some examples of collaboration on 3D scenes with our model including a discussion about the network and display optimizations of the system regarding the user experience (field-speciality, pieces that "matters" to the user, client resources, device used...) in section 5. Finally, conclusion and future works are given in section 6.

2 RELATED WORK

CAD is an essential tool for 3D models production in industry. During the last years, considerable time and resources were spent on CAD as well as in the improvement of the computers power. These two features associated with a increasing need of team working and professional mobility, enabled the development of several Internet based collaboration tools.

2.1 Web-based visualization and collaboration

A wide range of standards and technologies have emerged the last decade for web-based and mobile 3D visualization. With HTML5 and more powerful clients, solutions that do not require the installation of a software are now well admitted on the web (unlike Flash, Unity3D[1]). Two predominant pluginless approaches exist: imperative with WebGL[2] supported by the W3C[3] and declarative [SKR+10] with X3D [JBDW12]. Mouton and al. [MSG11] sum up a coverful analysis of current systems and trends in CVEs [Fle12] arguing that web applications have major benefits over desktop applications because they are available for all major platform guaranteeing a cross-platform compatibility (including mobile devices) and do not require any software or libraries (except the web browser). They highlighted the need for new applications to reduce their bandwidth consumption by using local client resources to increase performances (interactivity). Web-based collaboration is particularly present in scientific visualization [JFM+08][GGCP11][CSK+11], cultural heritage [DBPM+14] and CAE (Computer-Aided Engineering) applications [CCW06]. This last one offers many online collaborative and distributed modelers like GradCAD, ThinkerCAD, Sunglass.io, Clara.io [HLL+13] or Verold Studio. However, most of these popular systems are client-server based and rely on full transfer of large 3D data for each client.

2.2 Web-based networking

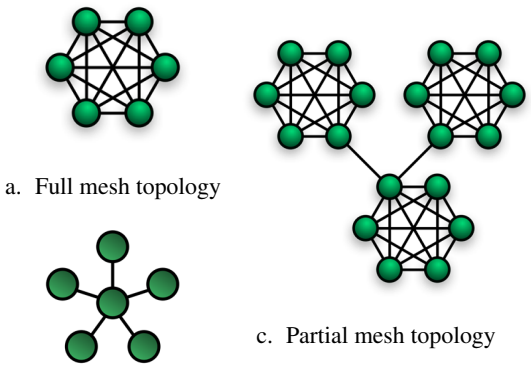
2.2.1 *Client-Server*

The client-server network topology puts the different clients in relation via the server that manages, transforms and stores the modifications and the data using a persistent database. This type of network offers security and easy management of information.

In 2011, Gutwin and al. [GLG11] has exposed the increasing role of web browsers as *"a platform for delivering rich interactive applications"* and details the different web-based networking approaches. They are all client-server types: HTTP-based communication (sending/getting server requests), AJAX with XHR (requesting without page reloading) and WebSockets [Rak14] (keeping an open connection with the server and communicate through messages). The works of Marion [MJ12] and Grasberger and al. [GSWG13] are based on the WebSocket protocol [FM11]. [MJ12]'s proposition transfers scientific data to and between clients to visualize with WebGL. The users can work on the data concurrently but they cannot edit it unlike in [GSWG13] where a BlobTree functional representation method requiring small memory footprint messages is also used to store, transfer, visualize and edit data.

2.2.2 *Peer-to-peer*

The P2P topology network allows each peer to be client and server simultaneously and to communicate directly with each other. The P2P network topology offers better resilience in case of a system crash or incongruous network disconnection using the autonomy of the peer (and data replication in the mesh). This induces higher efficiency in communication between team members (direct communication like in the real world).



a. Full mesh topology

b. Star Topology

c. Partial mesh topology

Figure 1: Peer-to-peer topologies

Full mesh topology (Figure 1.a) connects each nodes to every other one. It requires that every time a new peer join the network, the other peers establish a connection with the new peer. The increase in the number of connection is exponential and it does not scale well, saturating the bandwidth.

Star topology (Figure 1.b) uses a star node that distributes the data to the others. That node is a very

[1] http://unity3d.com/5

[2] http://www.khronos.org/webgl/

[3] World Wide Web Consortium – http://www.w3.org/

high bandwidth consumer and could be a dedicated server. This client-server like topology removes the advantage of P2P distribution but keeps the benefit of having reliable messages.

Partial mesh topology (Figure 1.c) connects the nodes "indirectly" to each others: one device maintains multiple connections to others without being fully meshed. Partial mesh topology provides redundancy by having several alternative routes and needs good recovery mechanism to maintain the data transmission in the mesh.

The development of P2P communication between browsers arrived in 2011 with the drafts of WebRTC (Web Real-Time Communication) API [BBJN12] of the W3C and IETF®[4]. Many projects with WebRTC technology [BBJN12] are interested in the MediaStream API (audio and video streaming) but only a few are using the DataChannel part. Services like PubNub[5], very popular to set up real-time applications with WebSockets, are just starting to support WebRTC. [WV14] presents the WebRTC architecture foundations for decentralized content-publishing facility between browsers with concerns about security and privacy.

ROCCAD [CT07] is a prototype providing a 2D/3D graphics interchanges in real-time during a development process for synchronous design collaboration. It offers distributed mechanisms to handle data transmission, data access policy and conflict resolutions, users management based on Tree First P2P overlay network over TCP/IP. The communication architecture exposed in [KVaD14], is very powerful and scalable using a client manager to abstract and synchronize the different devices communicating in P2P, where the servers are supporting the LODs (Level Of Details) management.

Chen and al.[CH14] presented an asynchronous online collaboration for BIM generation using hybrid client-server and P2P network based on a hierarchical topology: a peer team appointed a local server to transmit data to the global server. This architecture offers a good scalability in collaboration with parallel modeling (intra-disciplinary) to achieve a single multi-disciplinary task. Moreover, design team members can share their work in modeling and cooperate while working concurrently. The critical points are the servers: the local server could be overloaded (it is the only proxy to reach the global server) and if the global server suddenly goes down, the inter-collaboration is broken due to conflict generation between sub-models avoiding (teams) to communicate.

[4] The Internet Engineering Task Force – https://www.ietf.org/

[5] http://www.pubnub.com/

3 HYBRID ARCHITECTURE FOR 3D MODELING COLLABORATION

Our collaborative design environment (CoDE) requires an appropriate network model. The two main types of communication networks on the web are client-server and P2P. Even if client-server is more common, P2P is coming more and more attractive because of its characteristics of decentralized control and self-organisation although its web standards are still on progress.

To set up our CoDE, we developed a full web-based communication architecture for a 3D modeling platform. This work is lead in the context of a small amount of users (small teams max 7/8 people) which means full mesh topology is adapted (direct reachability) and its exponential growth is negligible. Indeed, with more users a partial mesh topology should fit better to relieve the network congestion. In such a virtual workspace, the contributions of each user is directly transmitted to others and they can observe the doings of others in real-time. The network model is mixing conventional client-server architecture, mostly used for persistence, and a full mesh P2P network for the real-time data transmission between the clients.The users are working together on a scene where they can add, remove and update 3D models.

3.1 Web-based 3D editor

The 3D rendering framework is based on WebGL which is pluginless. The framework is able to handle 3D data stored locally and on an external server for persistence and synchronization. The 3D viewport editor allows users to view and interact with the model. The interactions offered to the user are:

Viewing, navigating and using transformations tools The user can lean on commons commands from known CAD programs to interact with the view and the camera. It uses classic handles for object translation, rotation and scaling, and conventional CAD navigation.

Uploading 3D models, textures The editor handles the most used of open 3D file formats in CAD [Bou12] (OBJ, PLY, DAE, JSON...) via user friendly interaction: drag and drop importation.

Referential modification The modification of the reference coordinate system from local to global for transformation can be helpful for designer.

Grip snapping The definition of the grid can be modified by the user to get a specific resolution. It is also possible to use it to align models on the grid points.

Switching point of view The user can switch from his camera to other's users point of view.

 ISBN 978-80-86943-65-7

Assembled 3D models (textured or not) are available to every users (viewers, collaborators, editors). The interface is designed to be easy and clear to provide a better user experience (non-repulsive) particularly for the non-experts users for enhancing the accessibility of the application.

A content access policy for the objects is necessary to avoid modification conflicts during the collaboration. We use a lock/unlock mechanism with a visual feedback associated to represent that the object is in use (selection state) by a user to prevent concurrent edition of the same object.

3.2 Server: RESTful architecture

The client-server part is based on a REST (Representational State Transfer) architecture [TV10] that benefits from distributed hypermedia systems such as our. The responsibilities are separated between the client (user interface) and the server (data storage interface). Each request from client to server contains all the necessary information to let the server understand the request without context dependency stored on the server. With its uniform interface, each resource is unitarily identified with defined representations and auto descriptive messages. Also, the caching exempts many client-server interactions.

Table 1: REST architecture summary

Pros	Cons
Easier to maintain; No need to keep an open connection permanently; Web context: HTTP protocol, URI as resource representative, caching.	Bandwidth increasing and latency: client needs to keep locally all the necessary data to send the request.

Table 1 resumes the advantages and the drawbacks of a RESTful system. It fits well for web distributed systems even if mobile devices should have limited performance due to back and forth energy consuming requests.

NoSQL database

The rise of the web as a platform encourages the change in data storage for new needs like supporting large volumes of data (such as 3D data). NoSQL database provides dynamic schema and a rich query language API for data manipulation. Therefore the records can add new information on-the-fly facilitating the enrichment of the (3D) objects. In our application the NoSQL database is mainly used to maintain persistence of the state of the scenes while a user is absent. When the user returns, he/she receives the entire scene document. It provides robustness to the system and better experience to the user.

3.3 P2P communication

3.3.1 Topology

We propose to automatically connect users on a scene with a WebRTC connection. As each user send their ID to the database at their arrival, they also retrieve those which where already present on the scene. We are able to create a full mesh topology network in order to make them communicate the updates.

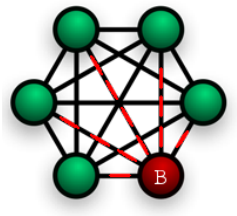


Figure 2: P2P topology of our model: a star node for message broadcasting inside a full mesh network.

Even if we have a full mesh topology, the P2P message layer is more similar the star topology. The Figure 2 shows the path of a sent message operation on the connection and it is only sent to the one-degree neighbors of the original broadcaster (“B” node on the Figure 2).

3.3.2 WebRTC and DataChannels

Web Real-Time-Communication (WebRTC) is a collection of standards, protocols (Figure 3) and JavaScript APIs specifying media transport and data sharing between browsers (peers)[Gri13]. P2P communication with *“WebRTC still needs servers for signaling”* and *“to cope with NAT and firewall”* [Ris14]. The signaling mechanism (Figure 4) allows peers to send control messages to each other in order to establish the communication protocol, the canal and the connection API.

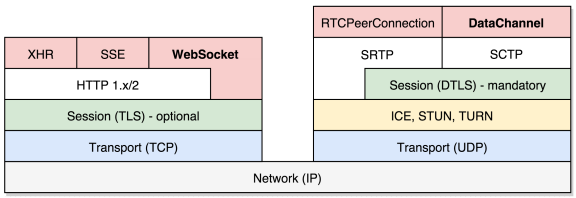


Figure 3: WebRTC protocol stack [Gri13]

We use the DataChannel protocol through the RTCDataChannel API to exchange arbitrary data between peers with customizable delivery properties (reliable or partially reliable, in-order or out-of-order) of the underlying transport [Gri13]. We choose to keep reliable and in order delivery for now. The RTCDataChannel API supports many data types (strings, binary types: Blob, ArrayBuffer...). These types are helpful in a 3D multi user environment to broadcast messages including the objects and their transformations. We tried to limit the amount of sent data with granularity choices (see Section 4) to prevent channel overfeeding.

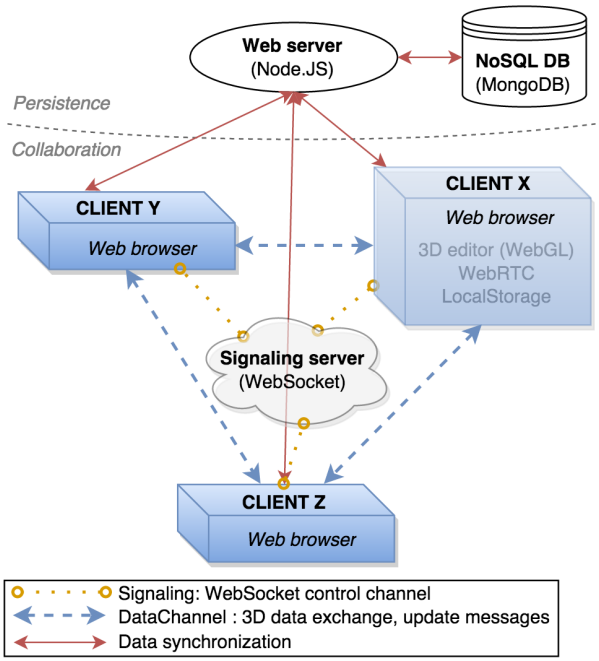


Figure 4: System overview

Some issues remain in RTCDataChannel API: the compatibility and interoperability is still not complete between browsers[6], some browsers (like Chrome) impose a send limit (about 6MB) for the data transmitting through DataConnections and the security of the communication is still vulnerable[7]. The system overview (Figure 4) illustrates the communication architecture topology between the peer clients, the web server and database (plus signaling server).

4 IMPLEMENTATION

As illustrated in the Figure 5, when a user arrives on the workspace editor the clients retrieves the scene from the server database. Each object is added to the Three.JS scene graph and rendered in the viewport. The connection to the P2P network with WebRTC is initiated by the assignation of an ID to the peer client which corresponds to the signaling mechanism. With this ID, the server automatically builds the full mesh topology by creating bi-directional connections between the new peer and the others. This action updates the list of connected users and their relations. Now that the scene is loaded and the P2P mesh is built, the user can freely interact with the scene with CRUD (Create, Read, Update, Delete) operations and synchronize the updates with the server and the other peers. For each operation, the type and the data are stored in a message according to the granularity of the transmission defined as follow:

- on import: all meshes and materials (such as textures) are sent.
- on transformation (translation, rotation, scale): the id of the transformed object and matrix of the transformation are sent.
- on delete: the id of the object to delete is sent.
- on lock/unlock: the id of the object is sent.

Once the message sent through the P2P mesh, the other peers can update their scene graph with the new values. It is also sent to the server through XMLHttpRequest for database persistence. When the client leaves the workspace editor, a flag is raised on the WebSocket server therefore it can broadcast the peers to update their list of connected peers to avoid useless sendings.

The implementation has a strong dependence on our architecture and the browser-based rendering constraints. WebGL is a JavaScript API provided by the Khronos Group. It is completely integrated into all the web standards allowing the browsers to use the GPU accelerated usage of image processing and effects as part of the web page. The choice of the 3D rendering framework has been oriented on an imperative solution: **Three.JS** [cab10]. It is a cross-platform solution that has already been widely adopted by the 3D community [MR10].

Our application is event-driven because of the nature of the manipulations of the users in a 3D environment. The event model is characterized by the event loop, event handlers and asynchronous processes. We based the message layer for user interface on a custom event/messaging system library called ***js-signals***[8]. Each signal has its own controller, which allows easy control of the event broadcaster and subscriber, avoiding the wrong objects reacts to the event. When a *Signal* instance is defined, procedures can be added to it. The signal will be intercepted anywhere in the scope of the application, the associated procedures will be triggered as well. A very interesting property of the event loop model is that JavaScript (unlike a lot of other languages) never blocks. A *Signal* is typically performed via events and callbacks, therefore when the application is waiting for a WebRTC message or an asynchronous XHR request to return, it can still process things like clicks.

As an asynchronous server-side run time environment, Node.JS replicates this model by using continuations: it keeps a stack of functions waiting to be run when the right event comes along. It is ideal for a data intensive real-time application that runs across distributed machines or a fast file upload client. Furthermore, with Node.JS we have JavaScript both client and server side, simplifying the understanding and the maintenance of

[6] WebRTC compatibility between web browsers from `http://iswebrtcreadyyet.com/`

[7] `https://github.com/diafygi/webrtc-ips`

[8] `http://millermedeiros.github.io/js-signals/`

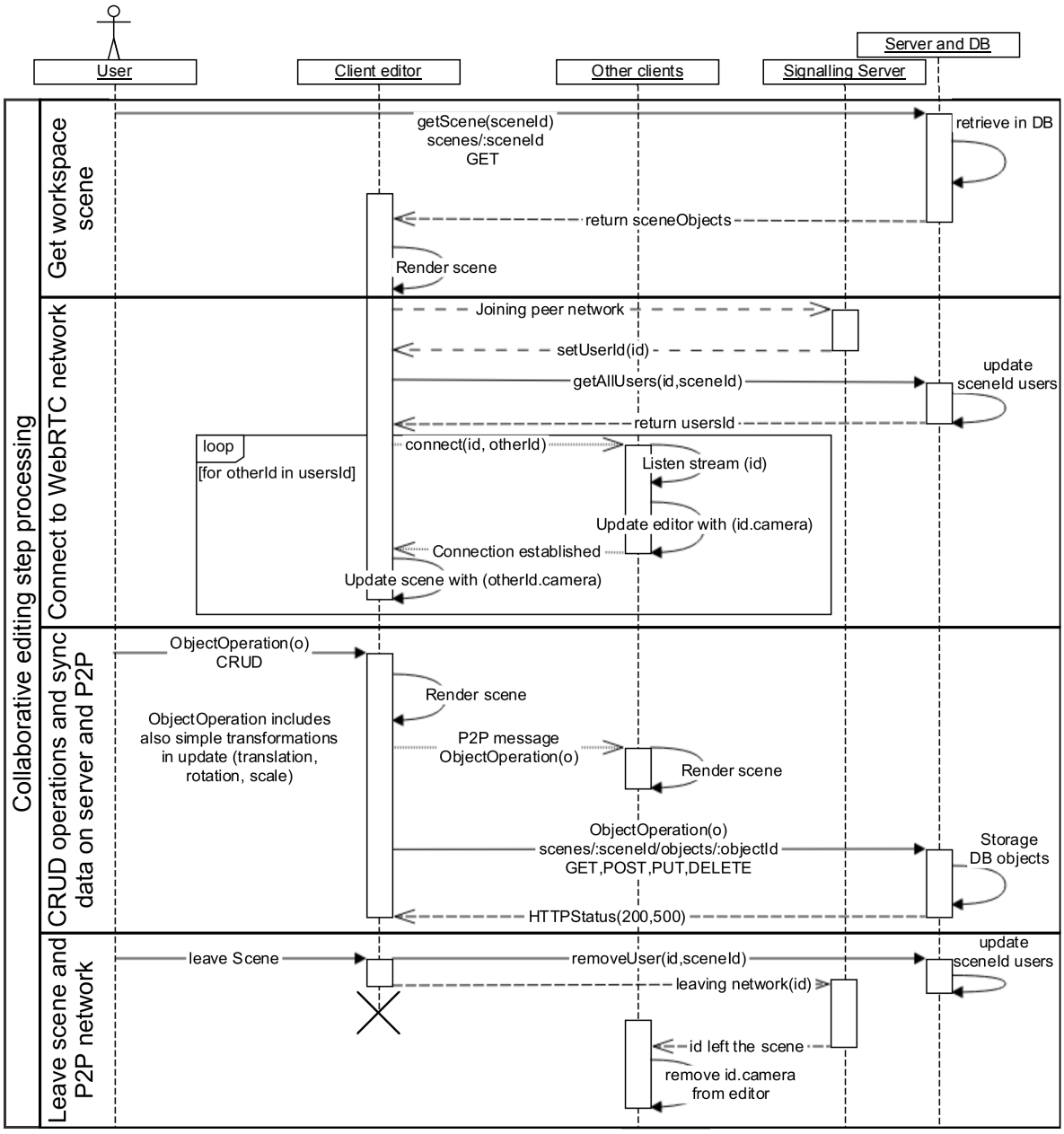


Figure 5: Sequence diagram of collaborative communication process

the environment. We use the micro-framework **ExpressJS**[9] to build the Node.JS web application.

To ensure a persistent track of the world state of the user's scene modifications, we choose the NoSQL (Not Only SQL) database **MongoDB**[10] over a conventional relational database. MongoDB is based on the *document database* technology[11] that stores and retrieves *documents* (relevant data stored together). A *document* is self-describing and can nest values in a hierarchical tree data structure. A collection is a grouping of documents, the equivalent of a relational database table. Our database contains two collections: *scenes* and *objects*. The *scenes*'s collection contains the scenes descriptions with their IDs and their metadata of the virtual workspace including a label and the connected users. This last information is crucial for the creation of the P2P mesh described in the Section 3.3. In the *objects* collection, the database stores the object with the ID of the scene it is attached to, its own ID and the full 3D object export in JSON format. The query parameters for the fundamental database operations (CRUD) on collections are fully supplied by the REST request.

[9] http://expressjs.com/
[10] http://www.mongodb.org/
[11] http://www.mongodb.com/nosql-explained

Because of the youth of WebRTC, browser's compatibility is still incomplete and some features are not yet implemented. That is why our application is only compatible with Chrome(v42+) and Firefox (v39+). **PeerJS** [MB13] is an open source library that wraps the browser's WebRTC implementation to provide a peer-to-peer connection API. The peer client, equipped with a clientID by the signaling server, can connect to a remote peer. In any case to establish a WebRTC session, a signaling protocol is needed such as WebSocket Protocol [LPR12]. We use the **PeerJSServer** implementation, based on a WebSocket server, provided by PeerJS to help broker connection between *PeerJS* clients.

Table 2: Summary of implementation choices

	Platform/Service	Library (version)
Client	WebGL Rendering	Three.JS (r69)
	Event manager	signal-js (v1.0.0)
	WebRTC	PeerJS (v0.3.9)
Server	Node.JS (v0.10.32)	ExpressJS (v4.9.0)
	WebSocket	PeerJSServer (N/A)
	NoSQL database	MongoDB (v2.6.8)

We use many different technologies in our model, as reflected in the technical choices for implementation resumed in Table 2. The modules of our implementation are communicating through APIs which is a benefit for modularity and maintenance. These choices were done with scalability perspectives.

As a result of this implementation we proposed a web platform for users where they can access to the list of the scenes' links and then the scene editor where the 3D collaboration starts.

3D Editor interface

The scene editing is done with the editor presented in Figure 6. The user can access to the list of the scenes from the menu and the "Back to scenes" link. The scene editor is titled with the scene's name, and the users connected to the scenes are shown by their ID (the bold one is the active user's). The editing part contains the tools, the viewport and the relative info. The tools are represented by the actions buttons *translate*, *rotate* and *scale* that trigger the associated helper. The *grid integer input* is for changing the resolution of the grid helper; the checkboxes: *snap* is for snapping the selected object to the grid; *local* is for changing the referential from world to local; *show grid* is for showing/hiding the grid helper. The key *s* allows to switch to the other users' cameras. The user can see the point of view shown by the camera helper representing the other's one. The viewport is where the 3D scene is represented. We can see that the user has selected the wheel of the plane and intents to translate it on the X,Y axis. The viewport info contains

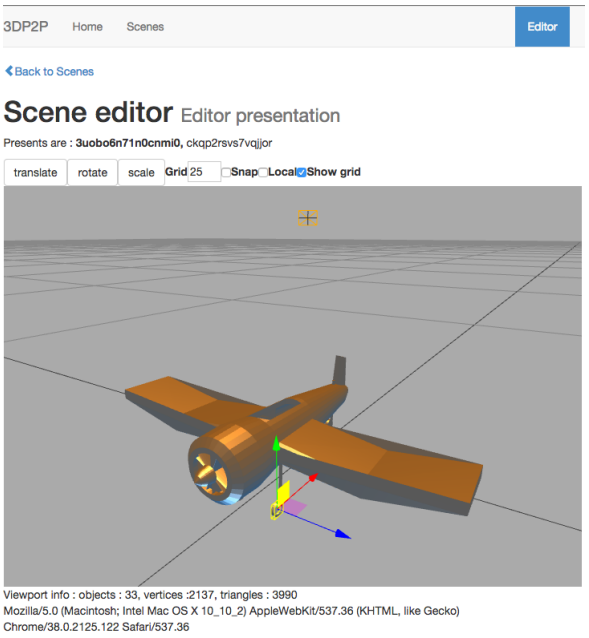


Figure 6: Editor interface

the information relative to the scene such as the number of objects (including light, cameras. . .), the number of vertices and triangles. Finally, the client information are displayed.

5 EXPERIMENTAL SETUP

We developed a prototype of the the web-based multi-user collaborative modeling to demonstrate the feasibility of our model architecture focusing on the user experience.

Table 3: Model descriptions for the experiments

Experiment	objects	size	users
Wind turbine	6	1.0 MB	2
Pick up	8	1.3 MB	4
Castle from *server*	35	1.3 MB	4
Castle from *peer*	35	1.3 MB	4

In one hand, the visualization was based on the WebGL technology using Three.JS to visualize 3D models online and offline without plugins. On the other hand, the real-time interactive collaboration relied on the hybrid architecture model exposed in the previous section. The Node.JS server platform allowed us to run a WebSocket server that handled the signaling mechanism for the WebRTC user connections creating the P2P mesh. The resources of the client were used in terms of graphics, storage, WebRTC capabilities in order to share the scene information between the users. We propose four experiments with three different models experiments (Table 3) to evaluate our system in the following criteria: user-friendly interface and the quality of the collaboration mechanisms (feedbacks, robustness and latency). A the end of each experiment, qual-

Table 4: Form distributed for each experiment and global results for 14 forms

	Questions	Answers* (results in %)			
General	Do you understand the goal?	No (0%)	**Yes (100%)**		
	Did you reach the goal?**	0 (0%)	1 (0%)	2 (14%)	**3 (86%)**
	Collaboration satisfaction ?	0 (0%)	1(14%)	**2 (72%)**	3 (14%)
	Type(s) of communication?***	None (0%)	**Oral (100%)**	Virtual (15%)	
User interface quality	3D interface expertise	0 (7%)	1 (14%)	**2 (50%)**	3 (29%)
	Tools usage	0 (0%)	1 (0%)	**2 (71%)**	3 (29%)
	Object manipulation	0 (14%)	1 (14%)	**2 (57%)**	3 (14%)
	Global quality	0 (7%)	1 (21%)	2 (64%)	3 (7%)
	The collaboration is:	Interactive (21%)	**Real-time** (79%)		
Open questions	Practice (define your practice: difficulties or frustration).				
	Collaborative rendering (define :latency, consistency, recovery)?				
	What improvements should you suggest?				

*Rates: 0 (bad), 1 (poor), 2 (good), 3 (very good). **Not asked for Castle experiments. ***Could use more than one channel.
Results have been rounded to the unit.

itative feedback was asked to the users via a form (see Table 4). For the experiments, the users were on the same local network as the server.

The *Wind Turbine* and *Pick Up* experiments had the same goal: assemble a model with multiple pieces apart. We showed a picture of the final assembly to the users, showing them the different parts of it. We distributed the pieces arbitrarily between the users and they had to import them in the viewport scene. Using the editor tools, real time information from others (application updates or any real communication), they had to manipulate the pieces (select, translate, rotate, scale) collaboratively to match the final assembly in a coherent way. The difference between the *Wind Turbine* and the *Pick Up* was the number of simultaneous users connected to the scene.

In the *Castle from **server***, the goal was slightly different: a castle kit (towers, walls, stairs...) was uploaded first on the server. At connection, the users retrieved (automatically from the server) the objects and they had about 10 minutes to creatively, but still collaboratively, build a castle. A variant, *Castle from **peer*** was introduced with the importation of another castle kit by a peer into the scene, broadcasting the new objects.

5.1 Results

Each experiment lasts about 5/10 minutes. We compiled the qualitative feedbacks of the users (see Table 4) and our observations and deductions. The users where not all very familiar with 3D interfaces but very familiar with computer: we observed mutual aid between expert users and beginners. Users were globally satisfied of the collaborative and visual results of the experiments (see Figure 7) because the goals were achieved: they succeeded in the assembly of the proposed 3D models without (too much) frustration. They even had fun on the castle experiment because they were free to create and they wanted to stay longer on the scene.

We noticed during the experiments that, on the lock system, we forgot to indicate which object was used by which user. Consequently to this lack of visual feedback, the external collaboration channel was mostly oral to exchange about object prehension: what they were doing on which piece.

The user interface was well appreciated, but maybe too simple for expert users. The manipulation of objects had a good evaluation except for the reception of a new imported model caused by the size of the message and the processing. A window frozen once during the *Castle from **peer*** on Chrome browser. The user had to quit and come back to the scene to refresh the viewport. This was a robustness test because the user appreciated to retrieve all the data and the other peers connections from the server at his/her return. The same remark was done for the fluidity of the application on the collaboration aspect.

The users did not feel latencies due to transformation operations during the experiments so they qualify the quality of the collaboration as real-time more than interactive. The variation of the number of users between experiments has not altered the rendering and networking quality of the user experience in terms of latency.

6 CONCLUSION

This paper proposed web-based 3D modeling collaboration based on a hybrid communication architecture client-server and P2P network.

The client is responsible for 3D rendering and handling the user interactions on a scene. It also hosts the peer connection to be able to communicate updates to other peers. The server is used to link the client with the NoSQL database in order to store the modifications, and manage the users presence on a scene and automatically create a P2P full mesh topology network between them. The P2P connection relies on a WebRTC communication that transmits information directly between

Figure 7: Results of 3D editor's scenes during the experiments: Wind Turbine, Pick Up and Castle kit

browser with update messages, broadcasting according to the P2P star topology, using a signaling server to establish the communication between two peers.

The qualitative evaluations of the experiments were conclusive overall even if some points should be improved. On model import, the broadcast causes latency issues on client peers receivers (camera freeze). To reduce latency with larger scenes we consider using progressive rendering and making a better use of peer-to-peer mesh to stream the model relying on seed peers like in the partial mesh topology. An improvement of interface features and visual feedback of collaborative manipulations was asked by the users. To start we have set focus on user experience.

The evaluation will be supplemented in future works with a quantitative evaluation to compare our hybrid architecture to others by selecting metrics (server logs, FPS in client, throughput, bandwidth requirements, number of connections supported...). We are now investigating experimental WebRTC tools[12] which provides statistics and graphs on the data exchanged between peers' browsers and automation tools (web automation like SeleniumHQ[13]), to evaluate on a set of scenarii the global performance and scalability of the system.

7 REFERENCES

[BBJN12] A Bergkvist, D Burnett, C Jennings, and A Narayanan. Webrtc 1.0: Real-time communication between browsers. w3c working draft. *World Wide Web Consortium*, 2012.

[Bou12] Rozenn Bouville. *Interopérabilité des environnements virtuels 3D: modèle de réconciliation des contenus et des composants logiciels*. PhD thesis, INSA de Rennes, 2012.

[cab10] Three.js - javascript 3d library, 2010.

[CCW06] Chih-Hsing Chu, Ching-Yi Cheng, and Che-Wen Wu. Applications of the web-based collaborative visualization in distributed product development. *Computers in Industry*, 57(3):272–282, 2006.

[CH14] Hung-Ming Chen and Chuan-Chien Hou. Asynchronous online collaboration in BIM generation using hybrid client-server and P2P network. *Automation in Construction*, 45:72–85, 2014.

[CSK+11] John Congote, Alvaro Segura, Luis Kabongo, Aitor Moreno, Jorge Posada, and Oscar Ruiz. Interactive visualization of volumetric data with webgl in real-time. In *Proceedings of the 16th International Conference on 3D Web Technology*, pages 137–146. ACM, 2011.

[CT07] Hung-Ming Chen and Hung-Chun Tien. Synchronous design collaboration in a peer-to-peer network. 2007.

[DBPM+14] Marco Di Benedetto, Federico Ponchio, Luigi Malomo, Marco Callieri, Matteo Dellepiane, Paolo Cignoni, and Roberto Scopigno. Web and mobile visualization for cultural heritage. In *3D Research Challenges in Cultural Heritage*, pages 18–35. Springer, 2014.

[ERB14] Alun Evans, Marco Romeo, and Arash Bahrehmand. 3D Graphics on the Web: a Survey. *Computers & Graphics*, 2014.

[Fle12] Cédric Fleury. *Modèles de conception pour la collaboration distante en environnements virtuels distribués: de l'architecture aux métaphores*. PhD thesis, INSA de Rennes, 2012.

[FM11] Ian Fette and Alexey Melnikov. The websocket protocol. 2011.

[GGCP11] Daniel Ginsburg, Stephan Gerhard, John Edgar Congote, and Rudolph Pienaar. Realtime visualization of the connectome in the browser using webgl. *Frontiers in Neuroinformatics*, 95, 2011.

[GLG11] Carl A Gutwin, Michael Lippold, and TC Graham. Real-time groupware in the browser: testing the performance of web-

[12] Chrome: `chrome://webrtc-internal`; Firefox: `about:webrtc`

[13] `http://www.seleniumhq.org/`

based networking. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, pages 167–176. ACM, 2011.

[Gri13] Ilya Grigorik. *High Performance Browser Networking: What every web developer should know about networking and web performance*. " O'Reilly Media, Inc.", 2013.

[GSWG13] Herbert Grasberger, Pourya Shirazian, Brian Wyvill, and Saul Greenberg. A data-efficient collaborative modelling method using websockets and the blobtree for over-the air networks. In *Proceedings of the 18th International Conference on 3D Web Technology*, pages 29–37. ACM, 2013.

[HLL+13] Ben Houston, Wayne Larsen, Bryan Larsen, Jack Caron, Nima Nikfetrat, Catherine Leung, Jesse Silver, Hasan Kamal-Al-Deen, Peter Callaghan, Roy Chen, et al. Clara. io: full-featured 3d content creation for the web and cloud era. In *ACM SIGGRAPH 2013 Studio Talks*, page 8. ACM, 2013.

[JBDW12] Yvonne Jung, Johannes Behr, Timm Drevensek, and Sebastian Wagner. Declarative 3d approaches for distributed web-based scientific visualization services. In *Dec3D*, 2012.

[JFM+08] Sebastien Jourdain, Julien Forest, Christophe Mouton, Bernard Nouailhas, Gerard Moniot, Franck Kolb, Sophie Chabridon, Michel Simatic, Zied Abid, and Laurent Mallet. Sharex3d, a scientific collaborative 3d viewer over http. In *Proceedings of the 13th International Symposium on 3D Web Technology*, Web3D '08, pages 35–41, New York, NY, USA, 2008. ACM.

[KVaD14] Timo Koskela, Jarkko Vatjus-anttila, and Toni Dahl. Communication architecture for a p2p-enhanced virtual environment client in a web browser. pages 1–5, 2014.

[LPR12] Salvatore Loreto and Simon Pietro Romano. Real-time communications in the web: Issues, achievements, and ongoing standardization efforts. *IEEE Internet Computing*, 16(5):68–73, 2012.

[MB13] Eric Zhang Michelle Bu. The peerjs library, 2013.

[MJ12] Charles Marion and Julien Jomier. Real-time collaborative scientific webgl visualization with websocket. In *Proceedings of the 17th international conference on 3D web technology*, pages 47–50. ACM, 2012.

[MR10] Anna Maria Manferdini and Fabio Remondino. Reality-based 3d modeling, segmentation and web-based visualization. In *Digital Heritage*, pages 110–124. Springer, 2010.

[MSG11] Christophe Mouton, Kristian Sons, and Ian Grimstead. Collaborative visualization: current systems and future trends. In *Proceedings of the 16th International Conference on 3D Web Technology*, pages 101–110. ACM, 2011.

[OJ10] Sixto Ortiz Jr. Is 3 d finally ready for the web? *Computer*, 43(1):14–16, 2010.

[Rak14] Shruti M Rakhunde. Real time data communication over full duplex network using websocket. 2014.

[Ris14] Dan Ristic. Webrtc data channels for high performance data exchange, 02 2014.

[SKR+10] Kristian Sons, Felix Klein, Dmitri Rubinstein, Sergiy Byelozyorov, and Philipp Slusallek. Xml3d: Interactive 3d graphics for the web. In *Proceedings of the 15th International Conference on Web 3D Technology*, Web3D '10, pages 175–184, New York, NY, USA, 2010. ACM.

[TV10] Stefan Tilkov and Steve Vinoski. Node.js: Using javascript to build high-performance network programs. *IEEE Internet Computing*, 14(6):0080–83, 2010.

[WV14] Max Jonas Werner and Christian Vogt. Implementation of a browser-based p2p network using webrtc. *Hamburg University of Applied Sciences, Technical Report, January*, 2014.