



Università degli Studi “Roma Tre”

Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di laurea magistrale

*Progettazione e sviluppo di plugin personalizzati
per il framework web Metior in scenari reali*

Laureando

Stefano Perrone

Relatore

Prof. Alberto Paoluzzi

Co-relatori

Dott. Enrico Marino, Dott. Federico Spini

Anno Accademico

2015/2016

Dedicato alla mia famiglia

Indice

Ringraziamenti	v
Introduzione	vi
1 BIM e Web	1
1.1 Building Information Modeling	1
1.2 Applicazioni BIM su Desktop	4
1.3 Tecnologie Web per il BIM	6
2 Metior	8
2.1 User Experience	9
2.2 User Interface	11
2.2.1 Viewer 2D	13
2.2.2 Viewer 3D	14
2.2.3 Plugin Catalog	15
2.3 Stack Tecnologico	16
2.3.1 React	16
2.3.2 Threejs	18
2.4 Stato dell'applicazione	20
2.5 Architettura	24
3 Metior Plugin	25
3.1 Definizione	25

3.2	Tassonomia	26
3.3	Proprietà Specifiche	28
3.4	Generazione modelli server-side	29
3.5	Processo implementativo	30
3.5.1	Studio della geometria	30
3.5.2	Definizione funzione di render 2D	31
3.5.3	Definizione funzione di render 3D	32
3.5.4	Instanziazione nel catalogo	33
3.5.5	Testing nella scena	33
4	Custom Plugin	34
4.1	BaM	34
4.2	Virtual CED	40
4.3	Deconstruction	43
5	Conclusioni e sviluppi futuri	46
5.1	Conclusioni	46
5.2	Sviluppi futuri	47
5.2.1	IoT	47
5.2.2	Progettazione Collaborativa	48
5.2.3	Fotorealismo	51
Bibliografia		53

Elenco delle figure

1	Fasi coinvolte nel BIM	vi
1.1	Schermata Autodesk Revit	5
2.1	Schermata con visuale 3D	9
2.2	Schermata con visuale in prima persona	10
2.3	Schermata interfaccia utente	11
2.4	Schermata viewer 2D	13
2.5	Schermata viewer 3D	14
2.6	Dettaglio Plugin: (a) Vista dei plugin nel catalogo, (b) inserimento oggetto dopo la selezione nel catalogo	15
3.1	(a) Foto modello reale, (b) Modello 3D esploso in elementi geometri semplici, (c) Modello 3D virtuale	30
4.1	Vista 3D del modello di un'aula	35
4.2	Dettaglio Plugin: (a) appendiabiti, (b) armadietto, (c) attacca- panni, (d) portaombrelli	36
4.3	Dettaglio Plugin: (a) banco, (b) cattedra, (c) lavagna, (d) lim	37
4.4	Dettaglio Plugin: (a) condizionatore, (b) termosifone, (c) ce- stini differenziata	38
4.5	Dettaglio Plugin: (a) libreria, (b) computer, (c) finestra con tenda, (d) finestra con veneziana	39

4.6	Vista 3D di un Virtual CED	40
4.7	Dettaglio Plugins: (a) estintore, (b) metal detecton, (c) rilevatore fumo, (d) cassetta naspo, (e) porta antipanico, (f) rilevatore di fumo	41
4.8	Dettaglio Plugin: (a) quadro elettrico, (b) rack server, (c) router wifi, (d) telecamera	42
4.9	Vista 3D di un modello per il Deconstruction	44
5.1	Plugin con informazioni IoT visualizzate in real-time	47
5.2	Esempio di un ambiente realizzato con il fotorealismo	51
5.3	Esempio di visione di un ambiente attraverso VR	52

Ringraziamenti

Grazie a...

Introduzione

Negli ultimi decenni, il settore delle costruzioni è stato coinvolto nel processo di evoluzione tecnologica. Questo ha portato ad un cambio di prospettiva e metodologia operativa, introducendo il *Building Information Modelling* (BIM). Esso fornisce un insieme di informazioni geometriche, visive, dimensionali, ambientali, tecniche e di processo, rendendo il processo di progettazione “sostenibile”. Il trend è di portare gli applicativi BIM già presenti nel panorama Desktop sul Web, per renderli disponibili anche sui dispositivi portatili (smartphone e tablet). L’obiettivo è cercare di portare dei benefici in termini di disponibilità, affidabilità, scalabilità, facilità di implementazione, manutenzione e aggiornabilità.



Figura 1: Fasi coinvolte nel BIM

Il lavoro presentato in questa tesi, svolto presso il *CVDLAB*, dell’Università di Roma Tre, è consistito nello studio delle tecnologie e nello sviluppo del framework *Metior*. Questo applicativo consente di modellare edifici in ambiente Web, secondo una metodologia che coniuga le potenzialità dell’approccio BIM con la semplicità di utilizzo. In particolar modo si è approfondita l’implementazione dei *Plugin*, oggetti posizionabili all’interno della scena, creando una procedura formalizzata del processo. L’obiettivo principale è stato portare il concetto di BIM sul Web, al fine di modellare i diversi ambienti a seconda del contesto in modo personalizzato. Durante il percorso di tesi sono state di grande importanza, in termini di crescita personale e professionale, le collaborazioni con SOGEI (Società Generale d’Informatica S.p.A. la società IT del Ministero dell’Economia e delle Finanze) e CNG (Comitato Nazionale Geometri). La collaborazione con SOGEI ha riguardato la modellazione di un *Virtual CED* (Centro Elaborazione Dati), realizzato con l’intento di avere un modello 3D navigabile, che consenta lo sviluppo di un sistema per la generazione di Indoor Interactive Virtual Mapping per il supporto ad applicazioni di Indoor-Location, Indoor-Navigation ed IoT-Mapping. L’altra collaborazione è stata quella con il *CNG* per il progetto *BaM* (Building and Modelling), facente parte del progetto nazionale *Georientiamoci*, per l’orientamento degli alunni nella scelta del percorso di studi, per il passaggio dalle scuole medie alle superiori. Il progetto nasce con lo scopo di far realizzare agli alunni, “*l’aula che vorrei*”. Questa iniziativa si prefigge di sensibilizzare gli studenti nell’utilizzo di materiali ecosostenibili e rispettare l’ambiente. Infine il progetto *Deconstruction*, il quale nasce con l’idea di fare delle previsioni sui costi di demolizione e smaltimento degli edifici, al fine di ottimizzare i processi e far diventare questi materiali delle risorse per la ricostruzione.

Nella tesi gli argomenti trattati sono stati sviluppati in quattro capitoli. Nel primo capitolo si descrive il concetto di BIM, nell’ambito della modella-

zione propondendo una possibile soluzione per portare questa metodologia in modo “semplicato” sulle piattaforme Web. È stato fatto uno studio sullo stato dell’arte, per fornire un overview sugli applicativi Desktop disponibili sul mercato. Si descrive inoltre come è possibile portare la modellazione sulle piattaforme Web e attraverso quali scelte tecnologiche. Nel secondo capitolo si introduce la scelta fatta per portare il BIM sul Web, introducendo *Metior*. Il framework implementa le funzionalità del BIM attraverso l’utilizzo di librerie software, in particolar modo React 2.3.1 e Threejs 2.3.2. L’utente dispone di un applicativo paragonabile a quelli Desktop come potenzialità, ma con una maggiore facilità di utilizzo, consentendo l’interazione del modello creato tramite una visualizzazione 2D e 3D. Nel terzo capitolo si fornisce una descrizione completa di un *Plugin* delle proprietà caratteristiche e la loro tassonomia ad alto livello, fino a definire in modo formalizzato la procedura implementativa all’interno del framework *Metior*. Nel quarto capitolo si descrivono i contesti di applicazione, facendo un overview sul progetto *BaM* 4.1 realizzato in collaborazione con il CNG, la modellazione di un *Virtual CED* 4.2 all’interno di SOGEI, ed infine il progetto *Deconstruction* 4.3.

Nelle conclusioni si è fatto un resoconto sul progetto sviluppato, facendo una panoramica sulle possibili strade da intraprendere per degli sviluppi futuri, come l’integrazione dei dati estratti da dispositivi *IoT* 5.2.1, l’introduzione nel framework del concetto di *Progettazione Collaborativa* 5.2.2 e l’utilizzo della tecnica del *Fotorealismo* 5.2.3 per la modellazione di un ambiente.

Capitolo 1

BIM e Web

In questo capitolo si descrive il concetto di BIM, nell’ambito della modellazione, proponendo una possibile soluzione per portare questa metodologia in modo “semplicato” sulle piattaforme Web. La nella prima sezione si fa uno studio sullo stato dell’arte, per fornire un overview sugli applicativi Desktop disponibili sul mercato. La seconda sezione descrive come è possibile portare la modellazione sulle piattaforme Web e attraverso quali scelte tecnologiche.

1.1 Building Information Modeling

Una tendenza a minimizzare l’umanizzazione di nuovi territori e di spingere per il riutilizzo di alloggi già costruiti e caduti in disuso è diventata una necessità pressante nelle società avanzate. La direzione intrapresa è di integrare il modello “zero energy” (ogni costruzione è stata prodotta con il consumo della stessa energia) con il modello “zero waste”, nuovo paradigma di progettazione dove i materiali di riufito della demolizione diventano risorse per ricostruire[2]. I processi di costruzione e progettazione necessitano di essere rinnovati per tener conto delle preoccupazioni ambientali. Per ridurre l’impatto dei progetti di costruzione sull’ambiente, il progetto ha bisogno di prendere in considerazione la questione dei materiali di costruzione. Le am-

ministrazioni pubbliche hanno bisogno di strumenti adeguati per il calcolo e il controllo dei materiali riutilizzati o smaltiti. I nuovi strumenti dovrebbero gestire l'elaborazione digitale dei materiali in tutto il ciclo di vita del progetto, supportando i requisiti del nuovo processo come: progettazione per la Demolizione, Progettazione per il Riciclo e per i Rifiuti. In particolare, un ciclo di vita dell'edificio è sostenuto da un processo di costruzione che prevede cicli allineati con i fenomeni naturali. In questo progetto di tesi si propongono delle soluzioni che servono a chiudere il cerchio del ciclo di vita dell'edificio, allontanandosi dalla tradizionale risposta lineare con tassi di energia ad alto consumo e verso il riutilizzo dei materiali in decostruzione/ri-costruzione, supportato da un processo computerizzato di demolizione selettiva. Tutti i cicli di ristrutturazione degli edifici dovrebbero prevedere passi di de-costruzione e ri-costruzione, mirati verso la sostituzione dei materiali al fine di ottenere una maggiore efficienza. Il trattamento di questi materiali richiede la codifica appropriata sia per lo smaltimento, secondo i codici del CER (Catalogo Europeo dei Rifiuti), sia per la pianificazione e la progettazione di nuovi edifici, seguendo la metodologia BIM (Building Information Modeling). A questo scopo si necesita di scene georeferenziate di realtà aumentata sulla base di modelli 3D veloci, facilmente navigabili e misurabili. Si è a conoscenza dei costi di costruzione da zero, ma poco si sa circa i tassi di sostituzione di parti di un edificio da realizzare tramite la demolizione selettiva. Un moderno processo di demolizione selettiva richiede l'intervento umano, con costi assicurativi elevati a causa della pericolosità per coloro che lavorano in queste attività. Quest'ultimo punto richiede un'alternativa alla sforzo umano in questi processi. È auspicabile che i robot automatici sostituiscano il lavoro umano; ad esempio i droni potrebbero operare in un contesto semanticamente familiare e dare aggiornamenti in tempo reale dell'ambiente rilevato. C'è, quindi, bisogno di un moderno framework di facile utilizzo per la modellazione da usare per la costruzione/decostruzione nel

settore AEC (Architecture, Engineering and Construction), per fornire un modello di realtà aumentata attraverso il riconoscimento semantico tramite computer vision e fotogrammetrico con precisione centimetrica.

Tali strumenti di realtà virtuale/aumentata richiedono sia una veloce modellazione della costruzione 3D e aumento del contenuto semantico, per poter essere controllata con tempi quasi realtime: questa è la vera sfida richiesta anche dal futuro sviluppo con sistemi IoT.

1.2 Applicazioni BIM su Desktop

Il panorama delle applicazioni Desktop che implementa le funzionalità BIM e di modellazione è vasto. Il software più noto e l'attuale leader di mercato BIM nella progettazione architettonica è Autodesk Revit. Questo è il motivo per il quale si è scelto questo framework come parametro di confronto.

Autodesk Revit

Autodesk *Revit* è un programma CAD e BIM per sistemi operativi Windows, creato dalla Revit Technologies Inc. e comprato nel 2002 dalla Autodesk per 133 milioni di dollari, che consente la progettazione con elementi di modellazione parametrica e di disegno. Revit negli ultimi sette anni ha subito profondi cambiamenti e miglioramenti. Prima di tutto, esso è stato modificato per poter supportare in maniera nativa i formati DWG, DXF e DWF. Inoltre, è stato migliorato in termini di velocità ed accuratezza di esecuzione dei rendering. A tal fine, nel 2008 il motore di rendering esistente, AccuRender, è stato sostituito con Mental Ray. Tramite la parametrizzazione e la tecnologia 3D nativa è possibile impostare la concettualizzazione di architetture e forme tridimensionali. Questo nuovo paradigma comporta una rivoluzione nella percezione progettuale, poiché questa si sostanzia in termini non più cartesiani ma spaziali, con i vantaggi che questa può apportare alla progettazione [29]. Revit, come programma BIM, (come si vede in Figura 1.1) è da intendersi come un approccio più vicino alla realtà percettita dall'uomo. Uno dei punti di forza di Revit è quello di poter generare con estrema facilità viste prospettiche o assonometriche, che richiederebbero notevoli sforzi nel disegno manuale; un esempio è la creazione di spaccati prospettici ombreggiati. Altra caratteristica di estrema importanza è quella di costruire il modello utilizzando elementi costruttivi, mentre in altri software analoghi la creazione delle forme è svincolata dalla funzione costruttiva

e strutturale. Elemento portante di Revit è lo sfruttamento della quarta dimensione, cioè il tempo. Si possono infatti impostare le fasi temporali: ad esempio, Stato di Fatto e Stato di Progetto. Ogni elemento del modello può essere creato in una fase e demolito in un'altra, avendo poi la possibilità di creare viste di raffronto con le opportune evidenziazioni. I punti deboli del programma sono rappresentati, invece, dall'interfaccia talvolta poco intuitiva e dalla qualità dei rendering, che, pur utilizzando il motore radiosity, che fornisce un illuminazione globale del modello, non è paragonabile a quella ottenibile con software di rendering dedicati.

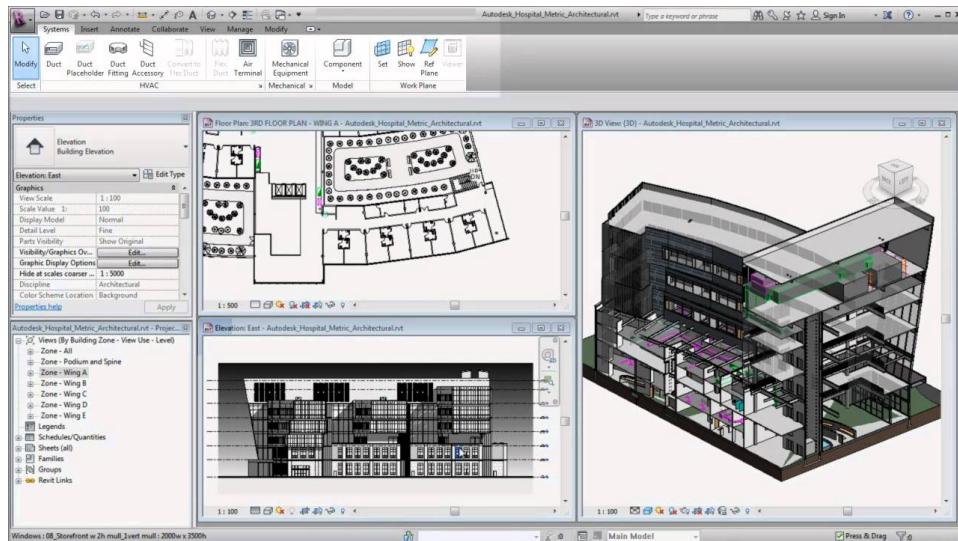


Figura 1.1: Schermata Autodesk Revit

1.3 Tecnologie Web per il BIM

Il mondo dei prodotti software sta assistendo ad una migrazione inarrestabile verso i servizi accessibili attraverso il mezzo Web. Ciò è dovuto principalmente ai benefici innegabili in termini di accessibilità, usabilità, manutenibilità e diffusione concessi dal mezzo Web stesso. Tuttavia questi benefici non arrivano senza un costo: le prestazioni e la complessità dello sviluppo sono fattori chiave nell'ambiente Web. In particolare, a causa dell'introduzione di diversi livelli di astrazione, non è sempre possibile “portare” le applicazioni Desktop nel dominio Web, un aspetto da prendere in considerazione anche per le differenze rilevanti tra gli hardware di tutti i dispositivi dotati di un browser Web. Può essere ancora più arduo affrontare un'architettura con software distribuito (almeno un client/server) indotta dalla piattaforma Web. Tuttavia sono sempre più ricche e complesse le applicazioni Web che sono comparse, sostenute dalle API arricchite da HTML5, insieme a WebGL [14] (che consente l'accesso diretto alla GPU), Canvas [23] (API raster 2D) e SVG [15] (API di disegno vettoriale), che hanno aperto la strada per l'ingresso di applicazioni Web Graphic. L'obiettivo è stato la definizione di uno strumento di modellazione di edifici basato sul Web che supera le suddette difficoltà prestazionali e di sviluppo basandosi su un modello di progettazione del flusso di dati unidirezionale e su un'architettura serverless. Un'architettura serverless, al contrario di ciò che il nome può suggerire, in realtà si avvale di molti server specifici diversi, togliendo l'onere del funzionamento e della manutenzione allo sviluppatore del progetto. Questi diversi server possono essere visti come servizi di terze parti (tipicamente cloud-based) o funzioni eseguite in contenitori effimeri (possono durare solo per una invocazione) per gestire lo stato interno e la logica server-side. L'interazione in realtime tra gli utenti che lavorano congiuntamente sullo stesso progetto di modellazione è ad esempio ottenuta tramite API di terze parti che rendono possibile la collaborazione tra utenti remoti. L'interfaccia

utente, interamente basata sul modello a componenti Web, è stata mantenuta il più semplice possibile: è richiesto all'utente interagire principalmente con segnaposto simbolici bidimensionali, che rappresentano parti dell'edificio, evitando così interazioni complesse in 3D. La complessità del modello è quindi spostata dal modellatore allo sviluppatore, il quale deve compilare un *catalogo* di *building elements* estendibile e personalizzabile. Il modellatore deve solo selezionare l'elemento richiesto, il luogo e parametrizzarlo a seconda delle esigenze. È ovvio che un gran numero di building elements deve essere fornito per garantire l'adempimento delle maggior parte delle esigenze di modellazione.

In questo capitolo si è descritto il concetto di BIM (Building Information Modelling), nell'ambito della modellazione 3D su piattaforme Web. Nel prossimo capitolo verrà descrita la scelta fatta per portare il BIM sul Web attraverso *Metior*.

Capitolo 2

Metior

In questo capitolo si descrive la scelta fatta per portare la “filosofia” del BIM sul Web, tramite il framework sviluppato *Metior*. Questo applicativo consente di modellare edifici in ambiente Web, secondo una metodologia che coniughi le potenzialità dell’approccio BIM con la semplicità di utilizzo. La semplificazione alla base del progetto sta nel fatto che l’utente non deve interfacciarsi con il classico programma di CAD con complesse procedure dalla elevata curva di apprendimento, ma, attraverso un approccio parametrizzato, realizza un modello in modo semplice e veloce. Si descrive il framework Metior partendo dall’esperienza ad alto livello dell’utente, per scendere in profondità a basso livello fino a conoscerne l’architettura del framework. Le funzionalità “semplificate” del BIM vengono implementate attraverso l’utilizzo di librerie software open-source, in particolar modo React 2.3.1 e Threexjs 2.3.2, dando all’utente un applicativo paragonabile a quelli Desktop, che consente l’interazione del modello creato tramite le visualizzazione 2D e 3D, sotto le quali è presente un architettura *serverless*.

2.1 User Experience

Nello sviluppo del framework *Metior* ci si è concentrati nel fornire all'utente una *User Experience* tale da semplificare il compito di modellazione di un edificio. Il framework richiede che l'utente utilizzi il più possibile un'interfaccia bidimensionale la quale permette di definire il piano e posizionare gli elementi architettonici complessi (chiamati *building elements*) su di esso. Tali *building elements* sono presenti all'interno in un catalogo pre-caricato, e possono essere configurati e personalizzati attraverso un pannello laterale. Questo approccio di modellazione sposta la complessità verso lo sviluppatore dei *building elements* personalizzabili, lasciando all'utente finale il compito di posizionare e configurare gli elementi utilizzati. Un ricco catalogo di elementi è quindi fondamentale per rispondere alle esigenze di modellazione degli utenti. Una volta che la scena è stata definita in base all'approccio *place-and-configure*, il sistema può automaticamente generare un modello 3D che può essere esplorato esternamente (Figura 2.1) o in prima persona (Figura 2.2).

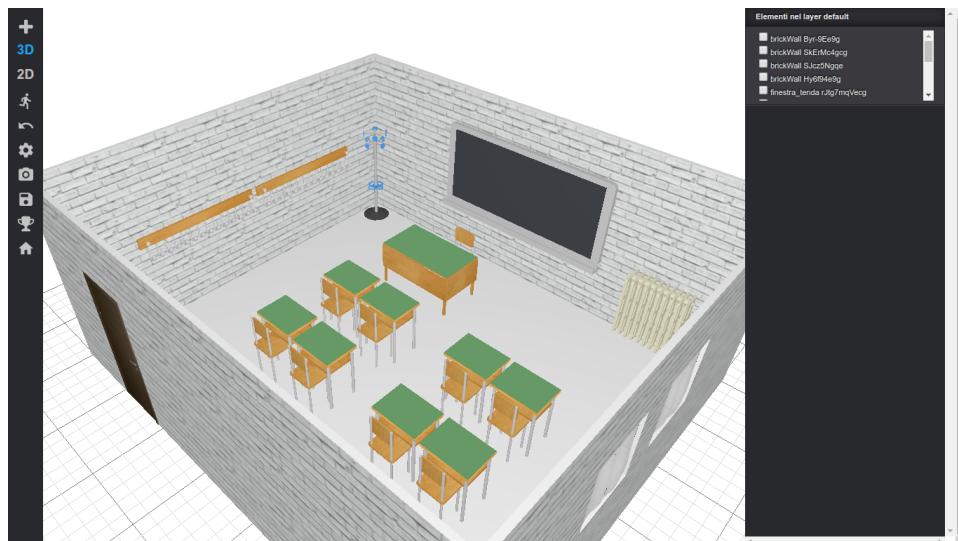


Figura 2.1: Schermata con visuale 3D

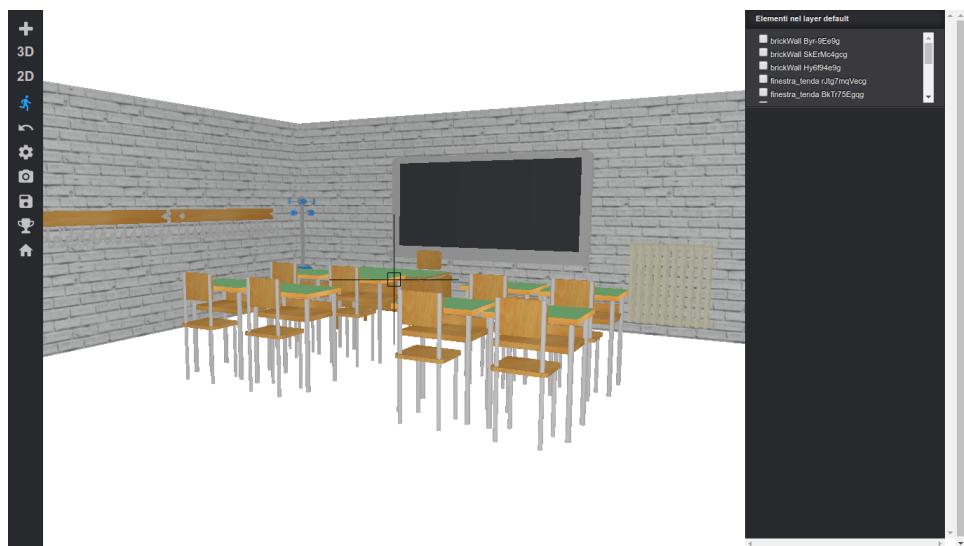


Figura 2.2: Schermata con visuale in prima persona

Ogni *building element* infatti comprende sia un *funzione generatrice 2D* (*2Dgf*) e un *funzione generatrice 3D* (*3Dgf*), utilizzate per ottenere modelli nella planimetria 2D e in 3D.

2.2 User Interface

La *User Interface* ha un ruolo importante perchè tramite essa vengono azionate tutte le funzionalità implementate dal framework. L'area di lavoro (Figura 2.3) è suddivisa in tre sottoparti:

- toolbar
- content-area
- sidebar

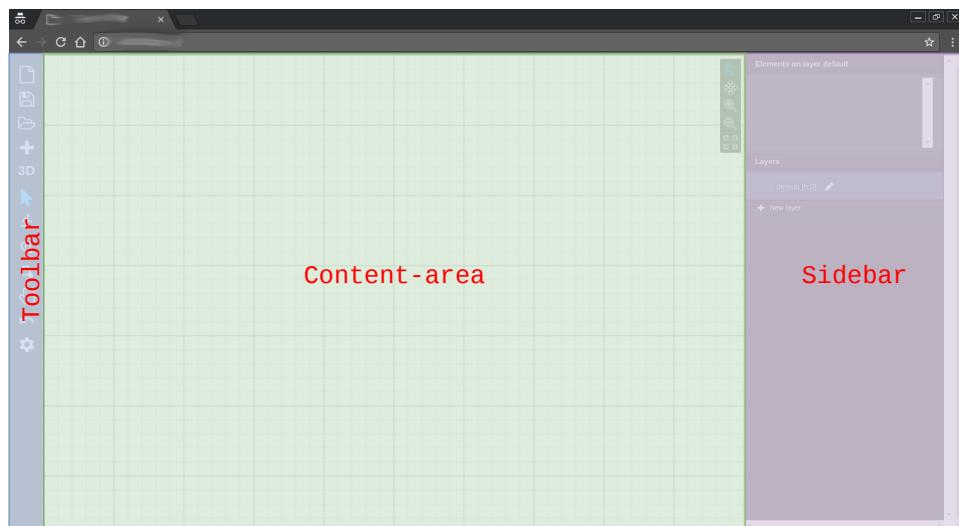


Figura 2.3: Schermata interfaccia utente

Dalla *toolbar* l'utente può accedere alle funzionalità relative a: ciclo di vita del progetto (new, save, load); switching modalità vista/interazione (2D, 3D); cambio di modalità di interazione (selecting, pan, zoom).

La *content-area* è un area nella quale l’utente può interagire con il modello attuale. Nella modalità 2D il modello è visualizzato come una proiezione 2D dall’alto e l’interazione consiste nell’inserimento, selezione e modifica dell’elemento (in accordo con le specifiche interattive del prototipo). Nella modalità 3D un modello 3D può essere inspezionato e navigato attraverso due modalità differenti: in prima persona usando il mouse e la tastiera ponendo la vista ad altezza uomo, o dall’alto cambiando la posizione della telecamera con il solo mouse. Anche in modalità 3D è possibile selezionare e modificare le proprietà degli oggetti.

La *sidebar* visualizza le proprietà dell’elemento correntemente selezionato. Nel pannello delle proprietà è possibile vedere la descrizione dell’elemento, aggiungere/rimuovere metadati, e modificare qualsiasi proprietà. Quest’ultima modalità di interazione consente al utente di associare annotazioni semantiche su ogni parte del modello.

2.2.1 Viewer 2D

Il *2D-viewer* invoca la funzione *2Dgf* dei *building elements* aggiunti al modello e genera in output il modello renderizzato usando gli elementi SVG. Per far fronte ai frequenti aggiornamenti provenienti dall’interazione con il disegno da parte dell’utente, sfrutta la *Virtual DOM* [27], che permette di aggiornare solo la parte modificata evitando così il completo rendering della scena. Per eseguire le operazioni di pan e zoom, tipicamente necessaria in questo tipo di strumento, è stato sviluppato un componente ad-hoc di React denominato *ReactSVGPanZoom*¹.

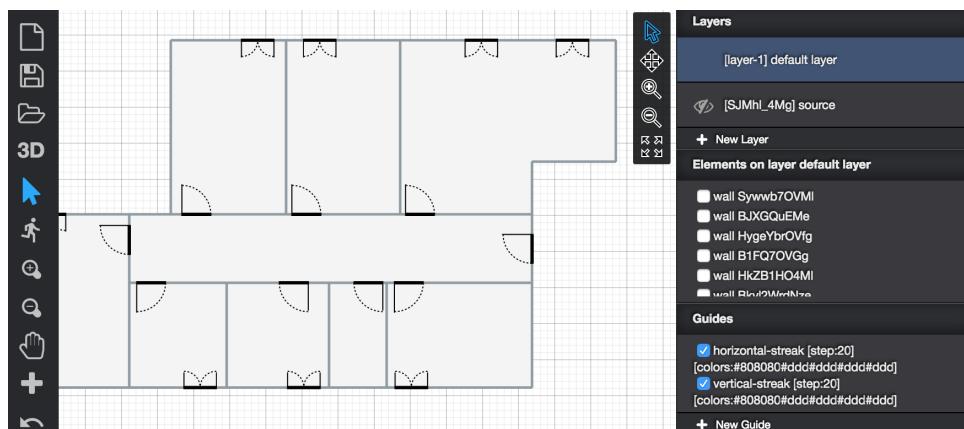


Figura 2.4: Schermata viewer 2D

¹<https://github.com/chrvadala/react-svg-pan-zoom>

2.2.2 Viewer 3D

Il *3D-viewer* invoca la funzione *3Dgf* dei *building elements* e aggiunge al modello una vista 3D usando le primitive WebGL *Three.js*². In particolare durante l’interazione con la scena si dispone delle seguenti *operazioni* sui *Plugin*:

- *add*;
- *replace*;
- *remove*;

Per eseguire l’update è stato implementato un *diff* e *patch* di sistema, definite nel Jsonpatch [3]: gli oggetti Three.js sono associati con elementi costruttivi all’interno dello stato dell’applicazione, in modo che ogni volta che l’utente attiva un’azione che si traduce in una modifica dello stato, l’applicazione calcola la differenza tra il vecchio stato e quello nuovo e cambia solo l’oggetto interessato.

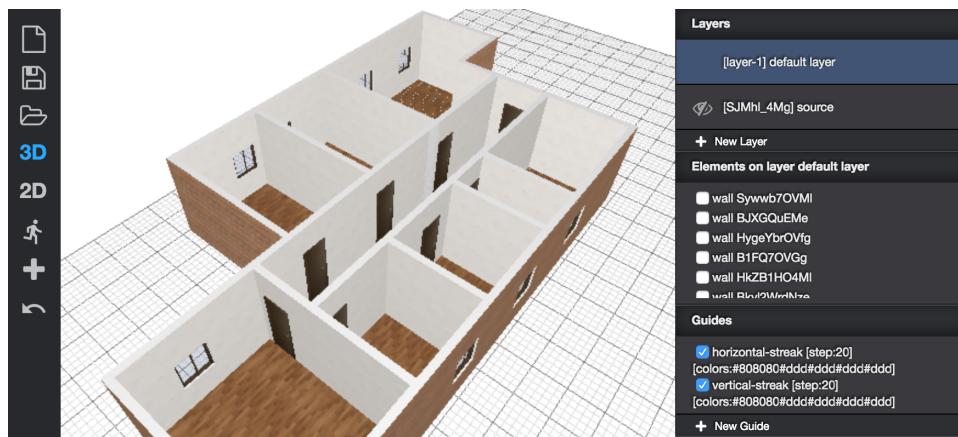


Figura 2.5: Schermata viewer 3D

²<https://threejs.org/>

2.2.3 Plugin Catalog

Il plugin catalog è l'elemento centrale che fornisce all'utente un sistema con un ricco catalogo di *Plugin*, in cui ogni elemento presente è definito da un nome, una descrizione ed una immagine di anteprima del modello 3D (Figura 4.2 (a)). Quando l'utente è al suo interno sceglie il *Plugin* che vuole inserire con un click. Dopo la selezione di un elemento del catalogo, il framework cambierà il suo stato portandosi nella modalità 2D, dove l'utente potrà decidere il posizionato del *Plugin* all'interno della scena (Figura 4.2 (b)).

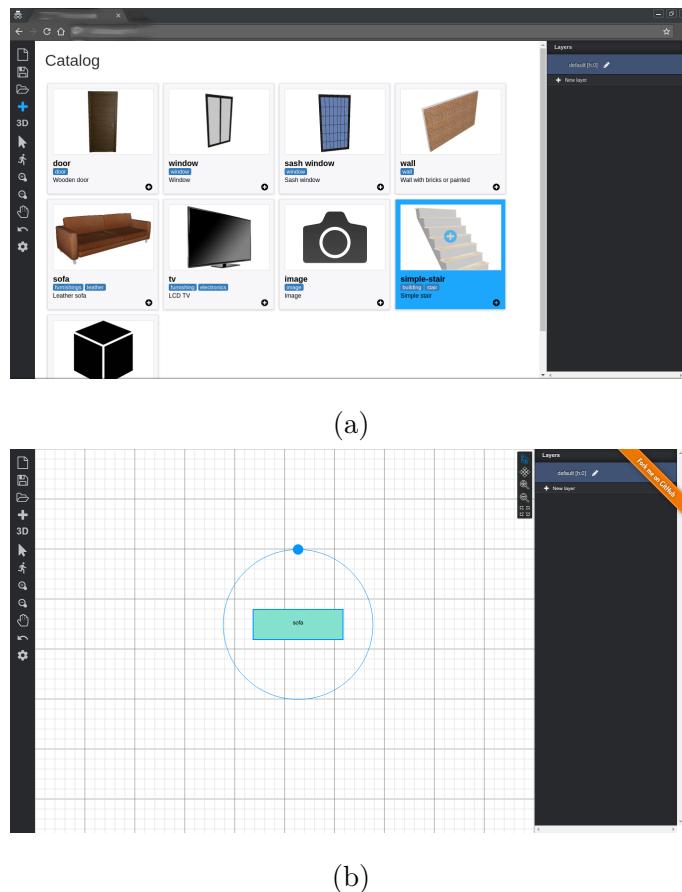


Figura 2.6: Dettaglio Plugin: (a) Vista dei plugin nel catalogo, (b) inserimento oggetto dopo la selezione nel catalogo

2.3 Stack Tecnologico

La User Interface è stata sviluppata seguendo i *Web Components pattern* [28], supportati dal framework *React*. L'idea principale è definire un'applicazione di frontend come una collezione di componenti indipendenti, ognuno dei quali si riferisce ad uno specifico sottoinsieme di stati centralizzati ed in grado di fare il render in accordo con i valori effettivi di quella porzione dello stato. I Web Components sono la base per contenitori generici di alto livello, come la barra degli strumenti o il catalogo, a di quelli a grana molto fine, come ad esempio i pulsanti. I più interessanti sono i visualizzatori del modello di costruzione: il *2D-viewer* e *3D-viewer*.

2.3.1 React

React (definito React.js or ReactJS) è una libreria open-source JavaScript per la costruzione di user interfaces. È sviluppato da Facebook, e sostenuto da Instagram e una comunità di sviluppatori tramite la sua disponibilità su repository GitHub [17] [10] [5]. Secondo il servizio di analisi JavaScript Libscore [18], React è attualmente utilizzato sui siti web di Netflix, Imgur, Bleacher Report, Feedly, Airbnb, SeatGeek, HelloSign, Walmart, e altri.

Storia

React è stato creato da Jordan Walke, un ingegnere del software di Facebook. È stato influenzato da XHP, un HTML framework di componenti per PHP. È stato distribuito sulle newsfeed di Facebook nel 2011 e in seguito Instagram.com nel 2012. È stato reso open-source durante il JSConf Stati Uniti nel maggio 2013. React Native, che consente lo sviluppo di applicazioni native su iOS, Android e Windows Mobile attraverso React, è stato annunciato alla Facebook's React.js Conf in febbraio 2015 e reso open-source nel marzo 2015.

Flusso dati unidirezionale

Le proprietà ed un insieme di valori immutabili, sono passate al componente di rendering come proprietà nel suo tag HTML. Un componente non può modificare direttamente le proprietà che gli sono state passate, ma può usare funzioni di callback le quali vanno a modificare i valori.

Virtual DOM

Un'altra caratteristica degna di nota è l'utilizzo di un virtual Document Object Model, o virtual DOM. React crea una struttura cache dei dati in memoria, calcola le differenze risultanti, e poi aggiorna la DOM [7] del browser visualizzandola in maniera efficiente. Questo permette al programmatore di scrivere codice come se l'intera pagina venisse cambiata, mentre le librerie React fanno il render delle sole sottocomponenti che in realtà cambiano.

JSX

I componenti React sono tipicamente scritti in JSX, una estensione della sintassi JavaScript che permette di citare HTML e utilizzano la sintassi dei tag HTML per fare il render delle sottocomponenti. [6] La sintassi HTML è trasformata in chiamate JavaScript della libreria React. Gli sviluppatori possono anche scrivere in JavaScript puro.

2.3.2 Threejs

Three.js è una libreria cross-browser JavaScript utilizzata per creare e visualizzare grafica animata in 3D in un browser web. Three.js utilizza WebGL. Il codice sorgente è ospitato in un repository su GitHub³.

Overview

Three.js permette la creazione di animazioni 3D accelerate dalla GPU utilizzando il linguaggio JavaScript come parte di un sito web senza fare affidamento su plugin proprietari dei browser. [32] [33] Questo è possibile grazie all'avvento delle specifiche WebGL. [16]

Storia

Three.js è stato pubblicato da Ricardo Cabello su GitHub nel mese di aprile 2010. [20] Le origini della libreria risalgono al coinvolgimento di Cabello con il demoscene nei primi anni 2000. Il codice è stato sviluppato originariamente in ActionScript, poi nel 2009 portato su JavaScript. Nella mente di Cabello, i due punti di forza per la riscrittura in JavaScript sono stati non essere costretti a compilare il codice prima ogni run e l' indipendenza dalla piattaforma. Con l'avvento di WebGL, Paul Brunt è stato in grado di integrare il renderer WebGL all'interno di Three.js creando un modulo di rendering appropriato. [4] I contributi di Cabello comprendono la progettazione API, CanvasRenderer, SVGRenderer ed è il responsabile per la gestione dei commit da parte dei vari collaboratori nel progetto.

Il secondo contributo, in termini di commit, è di Branislav Ulicny che ha iniziato a lavorare su Three.js nel 2010 dopo aver pubblicato una serie di demo WebGL sul proprio sito. Voleva che la capacità di rendering di WebGL in Three.js fossero superiori a quelle di CanvasRenderer o SVGRenderer. [4]

³<https://github.com/>

I suoi maggiori contributi comportano migliorie nei moduli dei materiali, degli shaders e di post-processing.

Caratteristiche

Three.js include le seguenti caratteristiche: [22]

- Effetti: Anaglyph, cross-eyed e parallax barrier;
- Scena: aggiunta e rimozione oggetti a run-time;
- Camera: prospettica e ortogonale; controllers: trackball, FPS, path e molte altre;
- Animazioni: armatures, forward kinematics, kinematics inversa, morph e keyframe;
- Luci: ambientale, direzionale, in un punto e spot lights; shadows: generata e ricevuta;
- Materiali: Lambert, Phong, smooth shading, textures e molte altre;
- Shaders: l'accesso alle capacità OpenGL Shading Language(GLSL) : lens flare, passaggio in profondità e un vasta libreria di post-elaborazione;
- Oggetti: meshes, particles, sprites, lines, ribbons, bones and more - tutti con un livello di dettaglio;
- Geometrie: piani, cubi, sfere, tori, testo 3D e molte altre; modificatori: lathe, extrude e tube;
- Data loaders: binary, image, JSON e scene;
- Utilities: set completo di funzioni matematiche in 3D tra cui troncoconiche, matrici, quaternion, UVs e molte altre
- Export and import: utilities per creare file JSON-Three.js compatibili con : Blender, openCTM, FBX, Max, e OBJ

- Supporto: documentazione delle API è in costruzione, forum pubblico e wikipedia;
- Esempi: Oltre 150 file di codice d'esempio con diversi tipi di carattere, modelli, texture, suoni e altri file di supporto;
- Debugging: Stats.js, [21] WebGL Inspector, [31] Three.js Inspector [30].

Three.js è eseguibile in tutti i browsers che supportano WebGL.

2.4 Stato dell'applicazione

Lo stato dell'applicazione è modellato utilizzando la struttura dei dati mostrata nel listato 2.1. Si tratta essenzialmente di un insieme di *layers*, ciascuno contenente un insieme di vertici, linee, aree ed oggetti, ognuno dei quali è racchiuso in una struttura composta da:

- informazioni necessarie dal prototipo dell'oggetto;
- riferimenti per la mappatura e la relazione con gli altri oggetti;
- metadati, tra cui ad esempio il punto in cui posizionare l'oggetto.

La ridondanza delle informazioni viene sfruttata per ridurre i tempi di accesso. Collezioni di oggetti sono indicizzate da un identificativo univoco `id` che permette la ricerca in tempo costante e del campo `selected` in ogni layer che consente l'accesso diretto ad elementi selezionati senza cercare. Lo stato dell'applicazione può caricare i layer indipendentemente, anche uno alla volta, consentendo quindi la visualizzazione dei soli layer selezionati, come può avvenire in un framework di modellazione Desktop.

```

1  {
2      "width": 3000, // canvas width
3      "height": 2000, // canvas height
4      "unit": "cm", // unit of measurement
5      "selectedLayer": "layer-1", // current layer
6      "layers": {
7          "layer-1": {
8              "name": "default",
9              "id": "layer-1",
10             "altitude": 0,
11             "opacity": 1,
12             "visible": true,
13             "vertices": {
14                 "HJAe59YF8Ux": {...}
15                 // ...
16             },
17             "lines": {
18                 "Hype99FK88x": {...}
19                 // ...
20             },
21             "openings": {
22                 "rljaKYUIg": {...}
23                 // ...
24             },
25             "areas": {
26                 "BygloFKUIe": {...}
27                 // ...
28             },
29             "objects": {
30                 "rkKU89U8e": {...}
31                 // ...
32             },
33             // selected element
34             "selected": {
35                 "vertices": [],
36                 "lines": [],
37                 "openings": [],
38                 "areas": [],
39                 "objects": ["rkKU89U8e"]
40             }
41         }
42     }
43 }
```

Listing 2.1: stato serializzato in JSON, struttura complessiva

Flusso dati unidirezionale

La struttura dei dati descritta rappresenta lo stato centralizzato richiesto dal *Unidirectional data flow pattern* [11] sfruttato dall'applicazione tramite la libreria *Redux.js*⁴, la quale è un contenitore di stato per le applicazioni JavaScript. Il modello prevede che lo stato possa essere modificato solo da attori specifici, chiamati *reducers*, le cui attività sono eseguite da specifiche *actions* che contengono tutte le informazioni necessarie per ogni reducer per realizzare il cambiamento nello stato. Ogni caratteristica applicazione deve essere attuata di conseguenza come una porzione di codice a sé stanti(*action/reducer*). Degli esperimenti preliminari⁵ su strumenti di disegno 2D hanno evidenziato la complessità di sviluppo di un'applicazione di questo tipo, in quanto lo stato interno dell'applicazione viene modificato da diverse interazioni dell'utente. È presente un'associazione tra l'evento scatenato dall'utente sull'interfaccia e la logica dell'applicazione. Nell'architettura del framework si definisce uno *state engine*, che rappresenta la logica dell'applicazione, che comprende actions e reducers e ne incapsula lo stato in modo centralizzato.

⁴<http://redux.js.org/>

⁵<https://github.com/cvdlab/walle>

Immutability pattern

L'*Immutability pattern* [19] si applica anche per evitare effetti collaterali sui cambiamenti di stato eseguiti dai reducers. Lo stato può essere visto come una struttura immutabile ad albero le cui modifiche vengono applicate come segue:

- clonazione dello stato precedente s per ottenere un nuovo stato s' ;
- applicazione delle modifiche sullo stato clonato s' ;
- aggiornamento del riferimento da s a s' .

Vale la pena notare che questo approccio fornisce il supporto per operazioni out-of-the-box come undo/redo: un stato vecchio/recente può essere ripristinato mediante la sostituzione dello stato attuale con il precedente/successivo. Nonostante la sua semplicità, questo modello può tuttavia comportare sprechi di memoria, a causa delle diverse copie dello stato che è necessario mantenere in memoria. Questo problema è stato affrontato utilizzando *Immutable.js*⁶, una libreria che sfrutta la condivisione strutturale tramite processi con mappe hash e vettoriali, riducendo al minimo la necessità di copiare o mettere i dati in una cache appositamente creata.

⁶<https://facebook.github.io/immutable-js/>

2.5 Architettura

Secondo Mike Roberts [26], un’architettura serverless fa largo uso di servizi di terzi, o componenti *as-a-Service*, che sostituiscono software e hardware ad-hoc per soddisfare gli stessi compiti. In particolare si è scelto di delegare i seguenti aspetti di servizi esterni e specializzati: la distribuzione delle applicazioni, la gestione degli utenti, generazione di modelli (computazione pesante), la collaborazione degli utenti e lo stato dell’applicazione.

In questo capitolo è stato descritto il framework *Metior*, vedendo in dettaglio l’architettura e lo stack tecnologico utilizzato. Si è introdotto il framework descrivendo l’interfaccia con cui interagisce l’utente. Nel prossimo capitolo verrà descritta in dettaglio l’implementazione dei *Plugin*.

Capitolo 3

Metior Plugin

In questo capitolo si descrive come vengono implementati i *Plugin* utilizzati all'interno del framework *Metior*, dando una descrizione completa delle proprietà caratteristiche ad alto livello, fino a descrivere in modo formalizzato il processo implementativo.

3.1 Definizione

Plugin è un componente software che può essere perfettamente integrato nel sistema, al fine di estenderne le sue capacità. In *Metior*, un plugin rappresenta un elemento architettonico che estende le Building Information Model progettate. Tecnicamente, un plugin rappresenta un *prototype* di un elemento di costruzione che può essere inserito (“istanziato”) nel *canvas*, definendo così un nuovo *elemento*, in altre parole un nuovo componente del modello.

Proprietà

Un plugin è descritto dalle seguenti otto proprietà:

- un nome univoco;
- una descrizione;
- l'*occupation type* (uno tra *linear*, *area* or *volume*);
- il *placement type* (*inside* or *over*);
- un insieme di proprietà specifiche che mappano la semantica da associare al plugin;
- una *generating function* che restituisce la rappresentazione 2D dell'elemento in formato SVG, da usare nel *2D-mode*;
- una *generating function* che restituisce la rappresentazione 3D dell'elemento in formato OBJ, da usare nel *3D-mode*;
- un insieme di metadati che consente l'inserimento di informazioni generiche;

3.2 Tassonomia

I Plugin possono essere organizzati in accordo con *occupation type* e *placement type*. L'*occupation type* può essere identificato da tre differenti tipi di Plugin:

- *linear*;
- *area*;
- *volume*;

I Plugin di tipo *linear* si estende in una dimensione (a meno di uno spessore radiale) (e.g. linee idrauliche, cavi elettrici). Il Plugin *area* si estende in due dimensioni (a meno di uno spessore lineare) (e.g. elementi di separazione). Si possono dividere in *horizontal area* (e.g. pavimento e celle), e *vertical area*, (e.g. muri). Il Plugin *volume* si estende in tre dimensioni. Si possono avere *fixed volume*, (e.g. un pezzo di arredo) e un *scalable volume*, che può essere scalato proporzionalmente o no, (e.g. pilastri, scale).

L'*occupation type* determina un modo differente di instanziare e inserire i Plugin nel canvas. In particolare, nel *2D-mode*, i Plugin *linear* sono inseriti disegnando linee attraverso l'interazione drag&drop; Il Plugin *area* sono inseriti disegnando una bounding-box dell'elemento attraverso l'interazione drag&drop; Il Plugin *volume* sono inseriti scegliendo la posizione dell'elemento attraverso l'interazione point&click, e sistemando la loro dimensione modificando la bounding-box attraverso il drag&drop.

Il *placement type* determina se l'elemento può essere inserito all'interno del canvas in un specifico punto occupato o meno da altri elementi. In altre parole, esso determina la relazione tra una nuova istanza del Plugin e l'istanza di altri Plugin precedentemente aggiunti al modello. La relazione può essere di due tipi: *inside* o *over*. I Plugin appartenenti alla categoria *inside* possono essere aggiunti solo all'interno di altri elementi (che possono essere *linear*, *area* o *volume*); e.g., una “finestra” è un elemento “volume inside vertical area”, mentre un “linea idraulica” è un elemento “linear inside horizontal area”. I Plugin della categoria *over* possono essere aggiunti solo sopra ad altri elementi (di qualsiasi tipo) e.g., un “pilastro” è un elemento “volume over horizontal area”, mentre un “pannello elettrico” è un elemento “volume over vertical area”. In fase di progettazione, un elemento che non soddisfa i vincoli di posizionamento definiti dal *placement type* è notificato dal sistema come un warning, visualizzando la sua bounding-box in semitrasparenza di colore rosso.

3.3 Proprietà Specifiche

Ogni plugin ha una serie di proprietà specifiche degli elementi costruttivi che rappresenta. Ogni proprietà è definita da:

- un *name*;
- un *type* (come “number”, “text”, “boolean”, o “custom”);
- un *value*.

In accordo con il proprio tipo, ciascun valore di ogni proprietà presenta un interfaccia dedicata per l’inserimento dei valori. Ad esempio, un valore della proprietà booleana è impostato tramite una casella di controllo (checkbox), mentre una proprietà di testo è inserita attraverso una casella di testo.

Il sistema è progettato per accettare tipi di proprietà custom. Un proprietà custom è richiesta per definire il componente della UI che permette all’utente di inserire il suo valore. Per esempio, una proprietà “colore” può essere introdotta definendo un componente della UI composto da tre box di testo (ad esempio per ogni componente RGB), mentre una proprietà “length” può essere introdotta definendo un componente UI includendo una box di testo per il valore e menu drop-down per le unità di misura.

Le proprietà specifiche di un elemento possono essere modificate nel relativo pannello nella sidebar, una volta che l’elemento è stato selezionato nel content-area.

3.4 Generazione modelli server-side

La generazione dei modelli 3D e 2D è realizzata in modalità *asynchronous*. Il ritorno dell’invocazione di una funzione generatrice non è il modello stesso, ma una *promise*¹ che sarà risolta con l’output del rendering. Tale scelta progettuale è importante poiché il calcolo per la generazione di modello può richiedere una notevole quantità di tempo. Nel frattempo l’utente deve essere in grado di interagire con l’interfaccia, che a sua volta deve rimanere reattiva. Basandosi su questa architettura, la generazione dei modelli può essere facilmente delegata a un server, sollevando così il browser dall’onere di calcoli onerosi. Il server espone un API JSON REST-like al browser.

¹<https://www.promisejs.org/>

3.5 Processo implementativo

Con processo implementativo di un *Plugin* si intende la formalizzazione delle fasi progettuali. La formalizzazione del procedimento traccia metodologicamente una sequenza di passi da seguire: (a) studio della geometria, (b) definizione della funzione di render 2D, (c) definizione della funzione di render 3D, (d) instanziazione nel catalogo, (e) testing nella scena.

3.5.1 Studio della geometria

Nella prima fase implementativa dopo aver scelto quale oggetto andare a modellare all'interno del framework, si passa ad uno studio della geometria dell'oggetto reale (Figura 3.1 - a). In presenza di un modello complesso bisogna pensare a come realizzarlo cercando di scinderlo in parti geometriche semplici. Ad esempio analizzando l'oggetto estintore, esso è composto da una geometria complessa che è scindibile in parti geometriche semplici, come un cilindro più una semisfera per il corpo centrale (Figura 3.1 - b). Si segue questa procedura per tutte le componenti fino alla realizzazione dell'intero modello virtuale (Figura 3.1 - c).

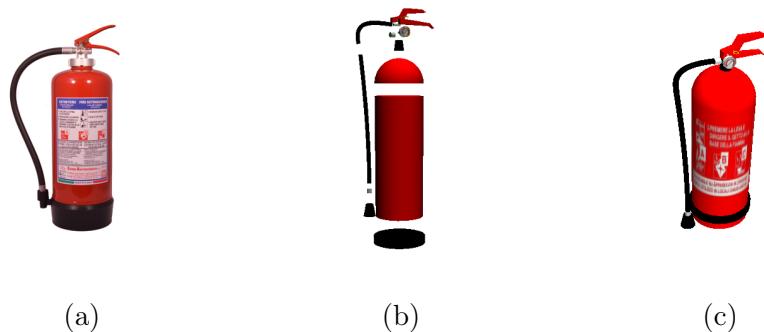


Figura 3.1: (a) Foto modello reale, (b) Modello 3D esploso in elementi geometri semplici, (c) Modello 3D virtuale

Dopo aver scomposto l'oggetto in elementi geometrici semplici, è necessario reperire le informazioni come larghezza, profondità, altezza e altri dati utili alla modellazione.

3.5.2 Definizione funzione di render 2D

La funzione di render 2D, attraverso il linguaggio *SVG* (Scalable Vector Graphics), consente di visualizzare oggetti di grafica vettoriale e, pertanto, di gestire immagini scalabili dimensionalmente. Questa definizione consente di inserire all'interno del Virtual DOM della content-area la rappresentazione 2D dell'oggetto aggiornando l'area interessata del canvas. In questo modo il Plugin viene visualizzato all'interno di Metior durante la visualizzazione in 2D della scena.

```

1  render2D: function (element, layer, scene) {
2    return (
3      <g transform={'translate(${-RADIUS/(RADIUS/2)}, ${-(RADIUS+5)/(RADIUS/2)})'}>
4        <ellipse key="1" cx="0" cy="0" rx={RADIUS+5} ry={RADIUS}
5          style={{stroke: element.selected ? '#0096fd' : '#000',
6                  strokeWidth: "2px", fill: "#ff0000"}}/>
7        <line key="2" x1={0} x2={0} y1={RADIUS} y2={2*RADIUS}/>
8        <line key="3" x1={-RADIUS/2+.15*RADIUS} x2={-RADIUS/2+RADIUS/2}
9          y1={1.2*RADIUS} y2={2* RADIUS}/>
10       <line key="4" x1={0} x2={-RADIUS/2+.85*RADIUS}
11         y1={2*RADIUS} y2={1.2*RADIUS}/>
12       <text key="5" cx={RADIUS} cy={RADIUS}
13         transform={'translate(${RADIUS/8}, ${0})',
14                     scale(1,-1) rotate(${angle/2})'}>
15         {element.type}
16       </text>
17     </g>
18   )
19 },
20 },
```

Listing 3.1: Definizione della funzione di render 2D

3.5.3 Definizione funzione di render 3D

La funzione di render 3D definisce attraverso una porzione di codice scritto in Javascript ed utilizzando la libreria open-source *Threejs*, un modello 3D del Plugin; questa definizione ne consente pertanto la rappresentazione all'interno di Metior durante la visualizzazione in 3D della scena.

```

1  render3D: function (element, layer, scene) {
2
3      var red = new Three.MeshLambertMaterial({ color: 0xff0000 });
4
5      var fireExtinguisher = new Three.Object3D();
6
7      var bodyGeometry = new Three.CylinderGeometry(0.1, 0.1, 0.5, 32);
8      var body = new Three.Mesh(bodyGeometry, red);
9      body.position.set(0, 1, 0);
10     fireExtinguisher.add(body);
11
12     var geometrySphereUp = new Three.SphereGeometry(0.1, 32, 32);
13     var sphereUp = new Three.Mesh(geometrySphereUp, red);
14     sphereUp.position.set(0, 0.25, 0);
15     body.add(sphereUp);
16
17     ...
18
19     let value = new Three.Box3().setFromObject(fireExtinguisher);
20
21     let deltaX = Math.abs(value.max.x - value.min.x);
22     let deltaY = Math.abs(value.max.y - value.min.y);
23     let deltaZ = Math.abs(value.max.z - value.min.z);
24
25     if (element.selected) {
26         let bbox = new Three.BoxHelper(fireExtinguisher, 0x99c3fb);
27         bbox.material.linewidth = 5;
28         bbox.renderOrder = 1000;
29         bbox.material.depthTest = false;
30         fireExtinguisher.add(bbox);
31     }
32
33     fireExtinguisher.rotation.y += -Math.PI / 2;
34     fireExtinguisher.position.y += -HEIGHT / 1.15 + newAltitude;
35     fireExtinguisher.position.z += 10;
36
37     fireExtinguisher.scale.set(RADIUS/deltaX,RADIUS/deltaX,HEIGHT/deltaY);
38
39     return Promise.resolve(fireExtinguisher);
40 }
```

Listing 3.2: Definizione della funzione render 3D

3.5.4 Instanziazione nel catalogo

Dopo aver implementato un Plugin è necessario, al fine di poterlo inserire all'interno di una scena, instanziarlo all'interno del *Plugin Catalog*.

```
1 import {Catalog} from 'react-planner';
2 ...
3 import estintore from './items/estintore/estintore';
4 ...
5 catalog.registerElement(estintore);
6 ...
7 export default catalog;
```

Listing 3.3: Instanziazione del Plugin nel Catalogo

3.5.5 Testing nella scena

La fine del processo richiede il testing del nuovo *Plugin* implementato, testandolo all'interno di una scena per verificare che le caratteristiche, geometriche e dimensionali corrispondano con quelle del modello reale dal quale si è preso spunto per la realizzazione del modello virtuale.

In questo capitolo si è fornita una descrizione completa delle proprietà caratteristiche e la loro tassonomia ad alto livello, fino a descrivere in modo formalizzato la procedura implementativa di un *Plugin* all'interno del framework *Metior*. Nel prossimo capitolo si farà un overview sui contesti di applicazione affrontati, vendendo come sono stati personalizzati i *Plugin* nei casi d'uso reali.

Capitolo 4

Custom Plugin

In questo capitolo si descrivono i contesti di applicazione del framework web *Metior*. La prima sezione presenta un overview sul progetto *BaM* 4.1 realizzato in collaborazione con il Comitato Nazionale Geometri. La seconda sezione descrive la modellazione di un *Virtual CED* 4.2 in collaborazione con SOGEI. Infine nell’ultima sezione si descrive il progetto *Deconstruction* 4.3 per la previsione dei costi di demolizione e smaltimento dei materiali che compongono gli edifici.

4.1 BaM

Il Progetto BaM (Building and Modelling), sviluppato in collaborazione con il CNG (Comitato Nazionale Geometri), facente parte del progetto nazionale *Georientiamoci*, per l’orientamento nella scelta del percorso di studi degli alunni, per il passaggio tra le scuole medie e le superiori. Questo progetto nasce dall’idea di far realizzare, agli studendi frequentanti le scuole medie in Italia, “*l’aula che vorrei*”. Questa iniziativa si prefigge di sensibilizzare gli studenti nell’utilizzo di materiali ecosostenibili e al rispetto dell’ambiente. Gli alunni avranno la possibilità di modellare un aula scegliendo gli elementi con cui personalizzarla. La lista degli *Bulding Element* presenti in libreria

ha l'obiettivo di far sperimentare l'uso di uno strumento di progettazione e di rappresentazione della realtà e di dare spunti su alcune tematiche come:

- impatto Ecologico;
- impatto Energetico;
- sicurezza;
- rispetto della diversità/disabilità;

Per questo motivo il framework *Metior* è stato strutturato in modo che a ogni elemento sia associato un punteggio seguendo la categorizzazione precedentemente definita. Gli alunni, suddivisi in gruppi, un volta terminata la modellazione dell'aula, salveranno il loro modello che sarà valutato dal framework. Vincerà il gruppo che avrà realizzato la classe con il punteggio più alto, risultante dalla somma dei materiali scelti tra quelli presenti in libreria. Per ottenere un punteggio alto bisogna rispettare i principi delle “3E” (Edilizia – Energia - Economia) e delle “3R” (Ridurre, Riutilizzare, Riciclare). In Figura 4.1 un modello di aula realizzato con il framework *Metior*.

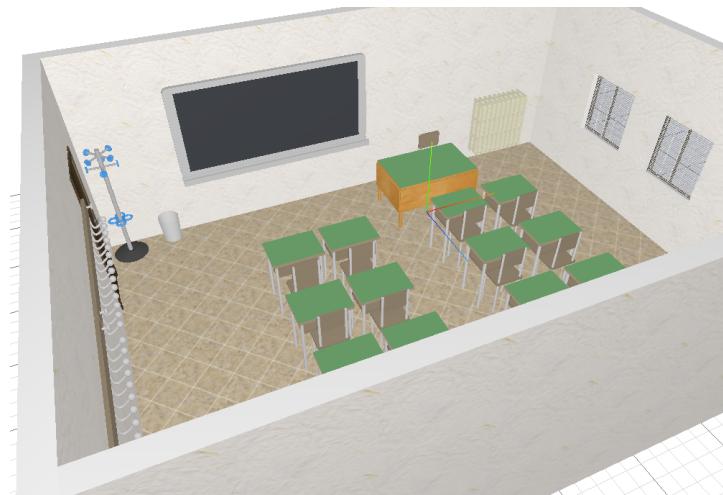


Figura 4.1: Vista 3D del modello di un'aula

Come descritto in precedenza i *Plugin* presenti all'interno del catalogo sono stati suddivisi in categorie per consentire una valutazione del modello di un'aula al termine dell'esperienza da parte degli studenti. Il primo gruppo di *Plugin* riportato sono (Figura 4.2): un appendiabiti, un armadietto, un attaccapanni ed un portaombrelli.

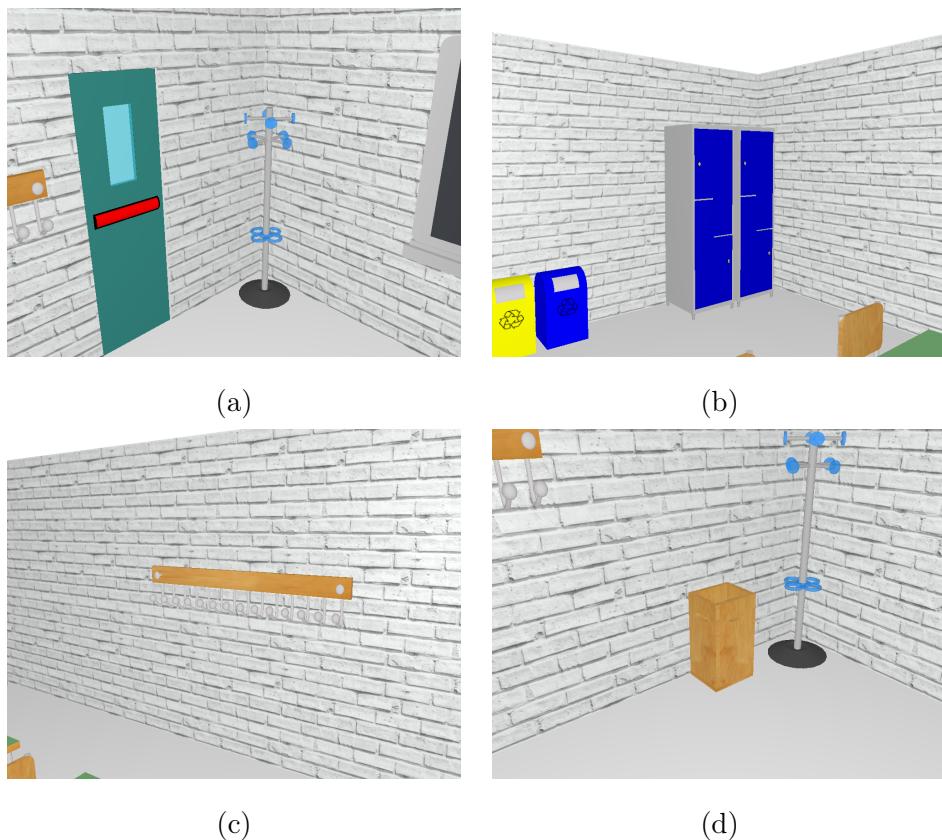


Figura 4.2: Dettaglio Plugin: (a) appendiabiti, (b) armadietto, (c) attaccapanni, (d) portaombrelli

Il secondo gruppo di *Plugin* (Figura 4.3) è composto dal mobilio classico prenseste in aula scolastica: un banco, una cattedra, una lavagna ed una lavagna interattiva multimediale (lim).

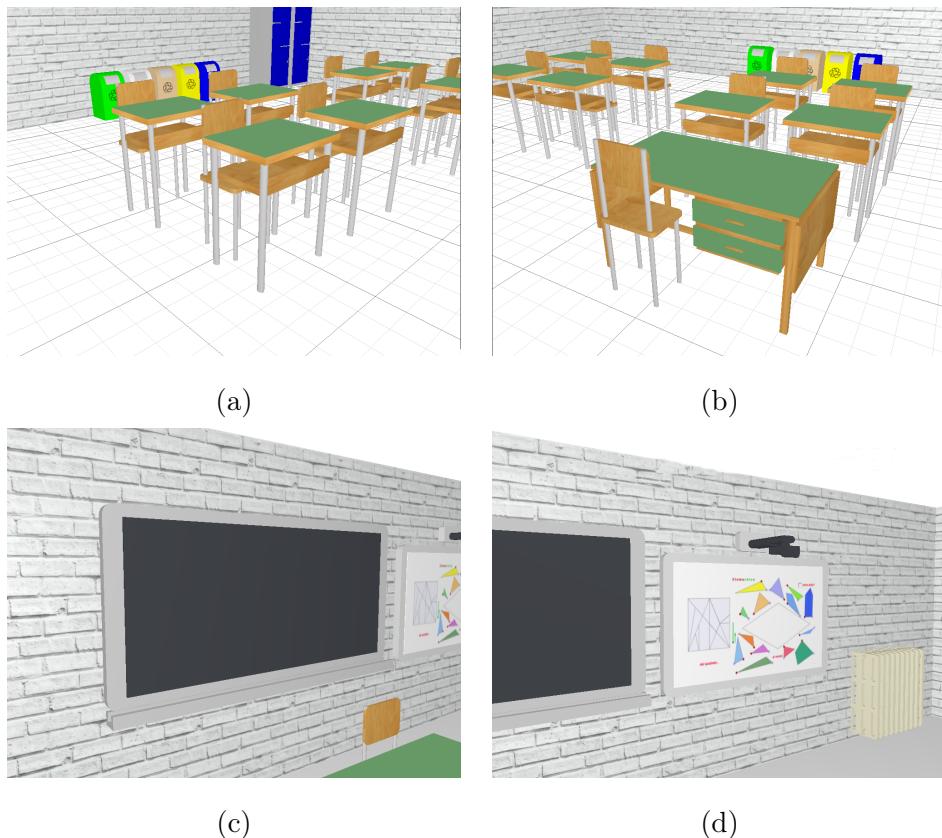


Figura 4.3: Dettaglio Plugin: (a) banco, (b) cattedra, (c) lavagna, (d) lim

Il terzo gruppo di *Plugin* è importante per quanto riguarda la tematica del rispetto dell'ambiente ed dell'ecologia. Come si vede (Figura 4.4) sono presenti: un gruppo di cestini per la raccolta differenziata, un condizionatore d'aria ed un termosifone.

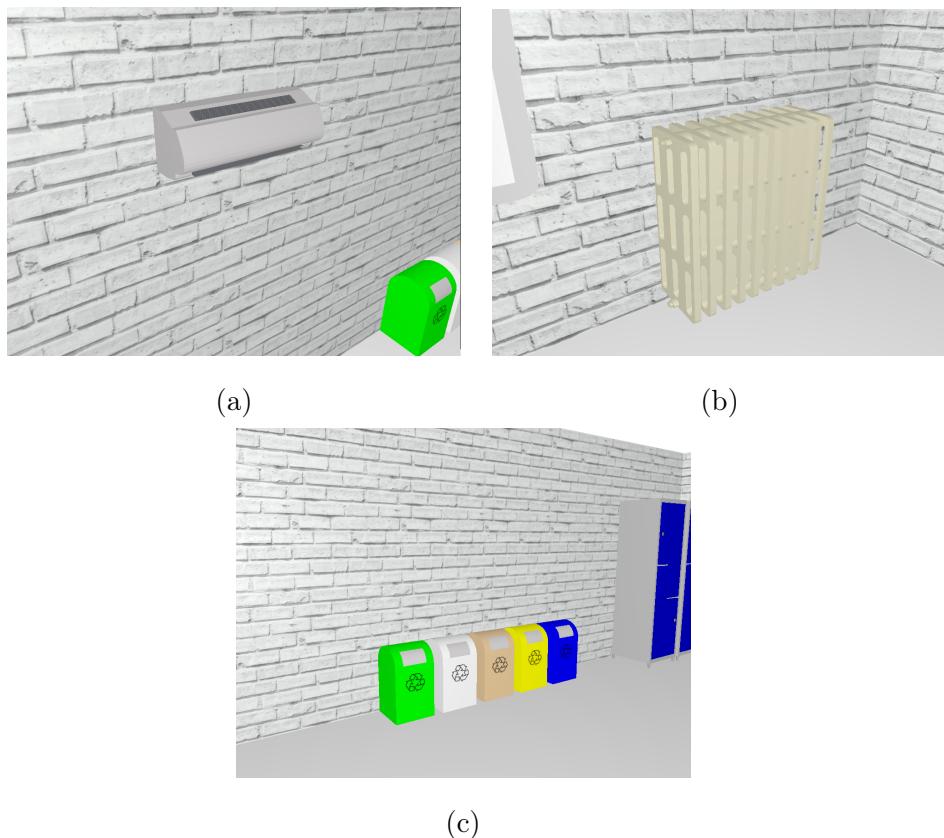


Figura 4.4: Dettaglio Plugin: (a) condizionatore, (b) termosifone, (c) cestini differenziata

Il quarto gruppo di *Plugin* riportato sono (Figura 4.5): una libreria, un computer, una finestra con tenda ed una con veneziana.



(a)



(b)



(c)



(d)

Figura 4.5: Dettaglio Plugin: (a) libreria, (b) computer, (c) finestra con tenda, (d) finestra con veneziana

4.2 Virtual CED

Il progetto di modellazione di un Virtual CED (Centro Elaborazione Dati) sviluppato presso SOGEI, è stato realizzato con l'intento di avere un modello 3D navigabile per consentire di sviluppare un sistema di Indoor Mapping e Indoor Navigation. Sono stati sviluppati dei *Plugin* coerenti con il contesto, che hanno portato alla realizzazione in un Virtual CED mostrato nella Figura 4.6 nella sua vista 3D.

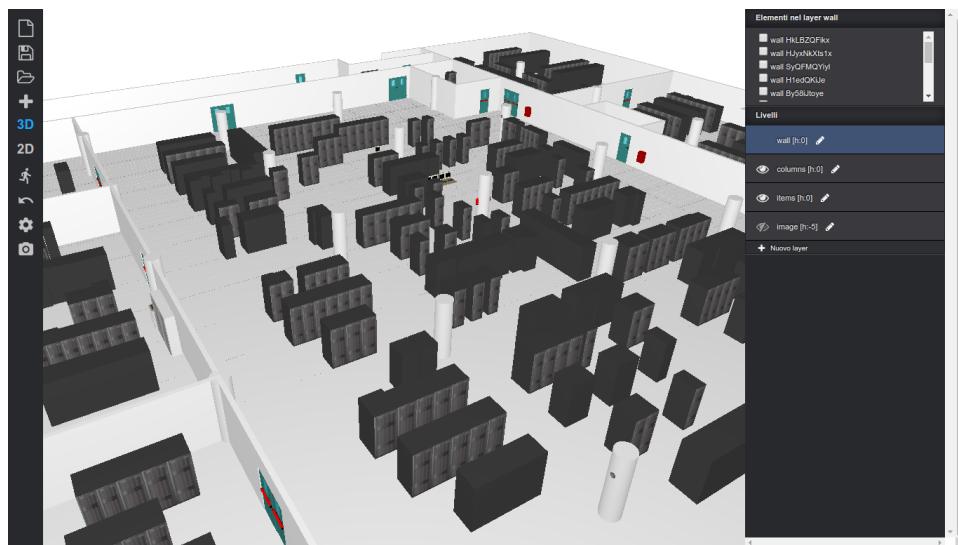


Figura 4.6: Vista 3D di un Virtual CED

Il primo gruppo di *Plugin* riguardano il contesto della sicurezza degli edifici (Figura 4.7), essi sono: un estintore, un rilevatore di fumo, una cassetta naspo ed una porta antipanico.

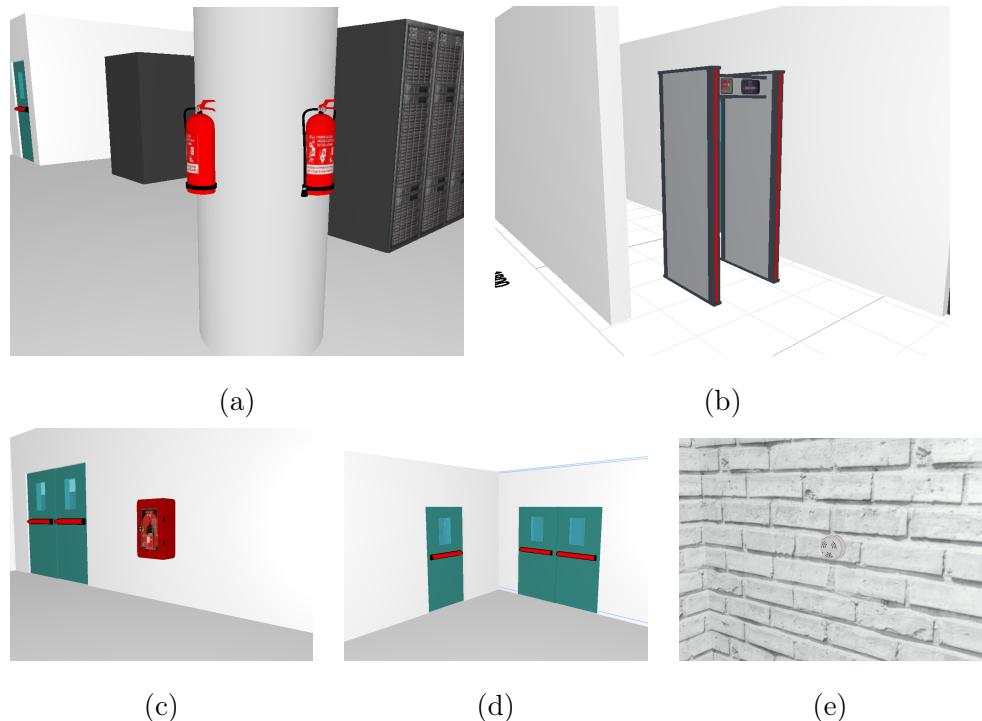


Figura 4.7: Dettaglio Plugins: (a) estintore, (b) metal detecton, (c) rilevatore fumo, (d) cassetta naspo, (e) porta antipanico, (f) rilevatore di fumo

Il secondo gruppo di *Plugin* riguardano il contesto tecnologico (Figura 4.8), essi sono un quadro elettrico, un rack server, un router wifi ed una telecamera. Questo gruppo ha un importanza rilevante in quanto questi oggetti possono consentire un accesso remoto ed essere categorizzati come dispositivi *IoT*.

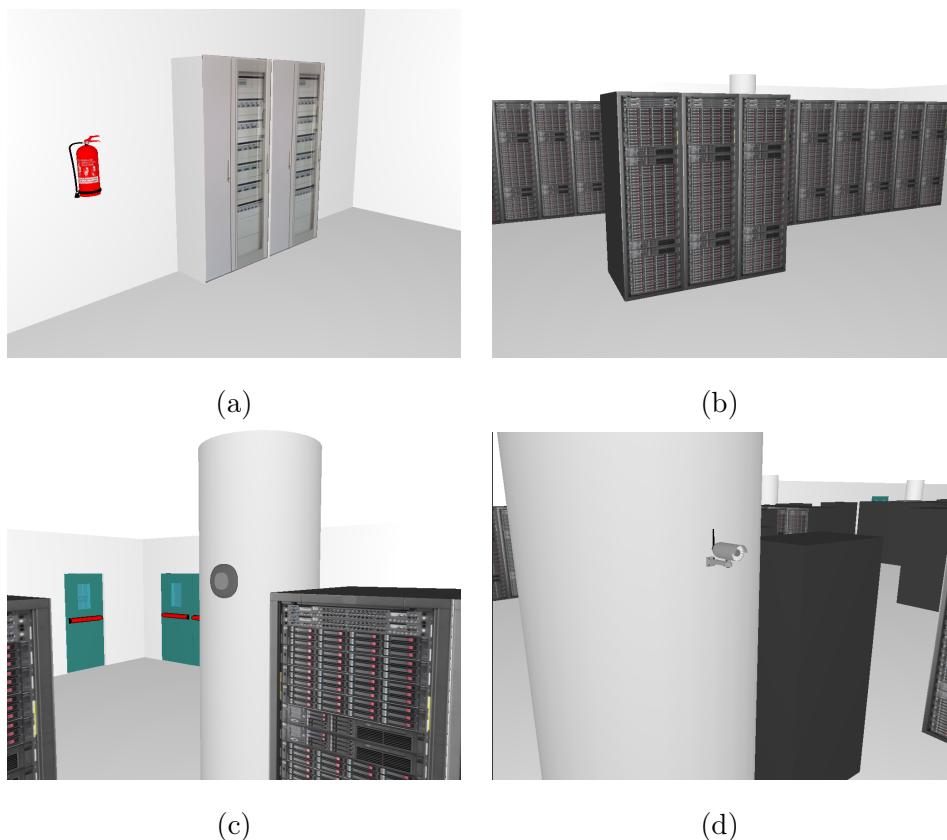


Figura 4.8: Dettaglio Plugin: (a) quadro elettrico, (b) rack server, (c) router wifi, (d) telecamera

4.3 Deconstruction

Il progetto *Deconstruction* nasce con l’idea di fare delle previsioni sui costi di demolizione e smaltimento dei materiali di scarto degli edifici (Figura 4.9). Il progetto ha lo scopo di promuovere l’uso di strumenti informativi semplificati per sostenere la decostruzione. In particolare, fornisce un modellazione geometrica semplificata dell’edificio permettendo l’integrazione di una descrizione semantica delle componenti e dei loro materiali. La realtà Virtuale/Aumentata aiuta a superare le difficoltà amministrative, a condizione di avere una corretta identificazione dei rifiuti prodotti. Questo approccio aumenta l’adozione di comportamenti virtuosi, come il recupero e riutilizzo. In particolare, una modellazione geometrica dell’edificio permette di individuare:

- costi/entrate derivanti da alternative di riciclo/riutilizzo, invece di smaltimento;
- composizione ed integrazione di informazioni utili per la pianificazione delle attività di costruzione;
- realizzazione delle soglie di riutilizzo/recupero previsti dalla normativa;
- capacità di confrontare economicamente diverse opzioni.

Il progetto ha avuto inizio prendendo in considerazione il sistema SMART-Waste [12]. Questo approccio permette di ricavare stime delle quantità di materiali, fornendo una descrizione del tipo di edificio e la zona in cui è stato costruito. Con queste informazioni si fornisce una rappresentazione aggregata dei dati di interesse riempiendo automaticamente delle form. Il framework *Metior* nel contesto della decostruzione al contrario fornisce sia una modellazione geometrica di sottosistemi e componenti edili e un’annotazione semantica con materiali da costruzione, come una sorta di *BIM semplificato*. È un dato di fatto che il settore edile nazionale sia fortemente

eterogeneo, e che necessiti di una modellazione dettagliata per ottenere delle informazioni sufficientemente accurate. Una specifica di questo approccio è il carattere iterativo incrementale, che consente in ogni fase della modellazione una validazione e stima dei costi parziali.

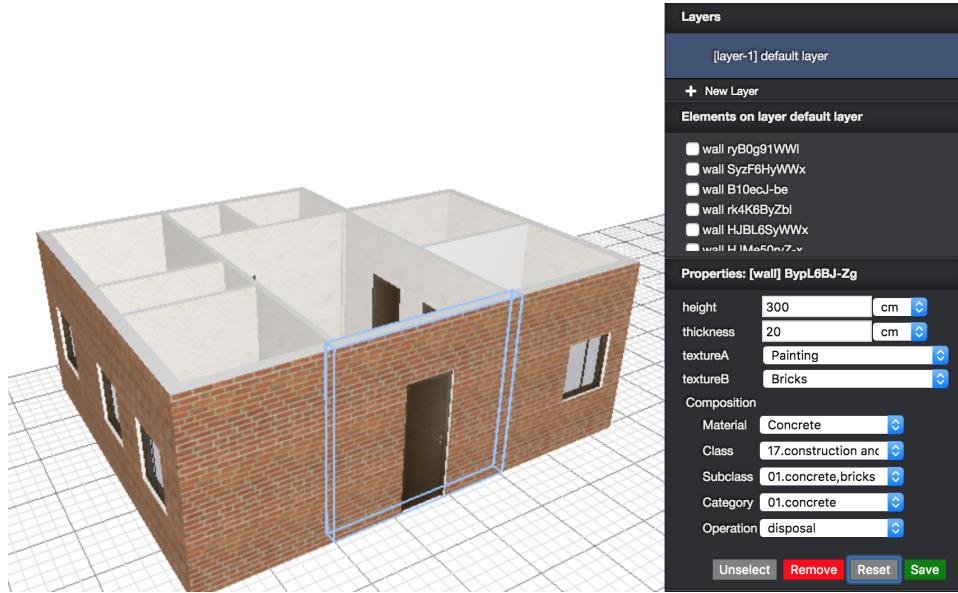


Figura 4.9: Vista 3D di un modello per il Deconstruction

Un rapporto completo sull'utilizzo BIM su come affrontare una decostruzione di un edificio è fornita da M Galic [9]. Uno studio sull'uso del BIM come supporto per la progettazione per Deconstruction è svolta da Olugbenga O Akinade [1]. In questa configurazione, al contrario di quella usata nel framework *Metior*, la decostruzione deve essere presa in considerazione a partire dall'inizio della progettazione degli edifici.

Il framework *Metior* (dal latino: a *misura* o *stima*), ha come obiettivo superare tali difficoltà, tramite:

- progettazione e realizzazione di un *web service* fornendo un’interfaccia utente semplificata;
- memorizzazione un database crescente di *Plugin* che rappresentano un modello per le parti di edificio geometricamente più complesse;
- utilizzo un motore geometrico estensibile e affidabile;
- l’offerta di integrazioni semantiche flessibili attraverso la specializzazione di (Industry Foundation Classes [13]) IFC classi associate ai sottosistemi di costruzione e le parti;

Considerando ad esempio un progetto che coinvolge la deconstuzione di grandi edifici, il processo è gestito da imprenditori, i quali hanno a disposizione e utilizzano strumenti specifici e hanno delle competenze in questo campo. La maggiorparte dei materiali di scarto prodotti sono gestiti da geometri provenienti da aziende di piccole o medie dimensioni o anche da singoli professionisti. Questo tipo di società necessitano di un sostegno dato da strumenti in cui le complessità, sia burocratiche che tecniche, anche se nascoste, devono essere correttamente gestite.

In questo capitolo sono stati descritti i contesti di applicazione del framework *Metior*, facendo un overview sul progetto *BaM* realizzato in collaborazione con il CNG, la modellazione di un *Virtual CED* in collaborazione con SOGEI, ed infine il progetto *Deconstruction* per la previsione dei costi di demolizione e smaltimento degli edifici. Nel prossimo capitolo si trarranno le conclusioni sul progetto sviluppato durante il percorso di tesi, esponendo i vantaggi introdotti dall’utilizzo del framework *Metior* nella modellazione su piattaforme web, proponendo dei possibili spunti di riflessione per degli sviluppi futuri.

Capitolo 5

Conclusioni e sviluppi futuri

In questo capitolo si fa un resoconto sul progetto sviluppato durante il percorso di tesi, esponendo i vantaggi introdotti dall'utilizzo del framework *Metior* nella modellazione su piattaforme web, proponendo dei possibili spunti di riflessione per degli sviluppi futuri.

5.1 Conclusioni

In questa tesi si è contribuito allo sviluppo del framework *Metior* per supportare la modellazione di edifici in ambiente Web, ed in particolar modo l'implementazione dei *Plugin* personalizzandoli a seconda del caso d'uso reale. Nel framework l'architettura serverless ha dato dei benefici in termini di disponibilità, affidabilità, scalabilità, facilità di implementazione, manutenzione e aggiornabilità ottenutasi implementando un applicazione logica come client-side con un solo stato centralizzato (nella forma di un documento JSON) che può essere messo in un document oriented database anche di terze parti e caricato nell'architettura frontend, la quale trasparentemente ricarica lo stato nella sua versione serializzata che gli è stato passato. L'applicazione stessa può essere servita da un CDN (Content Delivery Network) evitando così la necessità di un server web.

5.2 Sviluppi futuri

I possibili sviluppi e campi di utilizzo del framework *Metior* implementato possono essere tanti. Dopo aver utilizzato l'applicativo in scenari reali, si è ritenuto di focalizzare gli sviluppi successivi nei seguenti ambiti: (a) integrazione con dispositivi IoT, (b) progettazione collaborativa, (c) fotorealismo.

5.2.1 IoT

Gli oggetti presenti nella vita di tutti i giorni contengono hardware e software che consentono di interagire con essi da remoto. L'evoluzione tecnologica ha permesso a questi dispositivi di comunicare con il resto del mondo in tempo reale. Questi dispositivi sono denominati *IoT* (Internet of Things) e consentono l'estensione su Internet degli oggetti e dei luoghi reali. Questo consente di pensare all'inserimento di metadati all'interno dei *Plugin* implementati nel framework *Metior*, consentendo all'utente un'interazione realtime con i dispositivi e le loro informazioni.



Figura 5.1: Plugin con informazioni IoT visualizzate in real-time

5.2.2 Progettazione Collaborativa

Un altro possibile sviluppo del framework è l'aggiunta della modalità di *Progettazione Collaborativa*, consentendo a più utenti di collaborare contemporaneamente su uno stesso progetto, ottimizzando i tempi di lavoro. I software proposti per implementare questa funzionalità sono: *Firebase* e *jsondiff*.

Firebase [8] è una piattaforma mobile e per applicazioni web con strumenti e infrastrutture progettati per aiutare gli sviluppatori a creare applicazioni di alta qualità. Si è scelto di utilizzare il Database Realtime di Firebase il quale è un database cloud-hosted. I dati vengono memorizzati come JSON e sincronizzati in tempo reale ad ogni client connesso. Quando si crea applicazioni cross-platform con tecnologie Android, iOS SDK, e JavaScript, tutti i client condividono una istanza in tempo reale del database e ricevono automaticamente gli aggiornamenti con i dati più recenti. Esso è stato scelto per implementare un possibile sistema di autenticazione degli utenti, fare un controllo degli accessi e dei permessi sul progetto.

Si è deciso inoltre di utilizzare il gestore di pacchetti e moduli *npm* [24] il quale rende facile agli sviluppatori JavaScript condividere e riutilizzare il codice e rendendolo facile da aggiornare. Un *pacchetto* è un file o una directory con uno o più file in essa contenuti, che viene descritto tramite un file chiamato “*package.json*” contenente i metadati del pacchetto stesso. Una tipica applicazione, ad esempio un sito web, può dipendere da decine o centinaia di pacchetti. Questi pacchetti sono spesso piccoli. L’idea generale è che si crea un piccolo blocco di istruzioni che risolve un problema. In questo modo è possibile risolvere un problema più grande, attraverso una composizione personalizzata di questi piccoli blocchi condivisi.

Si è scelto il modulo *jsondiff* fornito da *npm*, il quale consente di confrontare nel contesto collaborativo i file JSON su cui lavorono gli utenti per trovare le modifiche apportate.

```

1      ...
2
3      "items": {
4          "BkrJVNl9e": {
5              "id": "BkrJVNl9e",
6              "prototype": "items",
7              "type": "cattedra",
8              "properties": {
9                  "altitude": {
10                     "length": 0,
11                     "unit": "cm"
12                 }
13             },
14             "selected": false,
15             "x": 452.23424450039795,
16             "y": 1750.8919038286344,
17             "rotation": 0
18         },
19         "Hk1xEVg5g": {
20             "id": "Hk1xEVg5g",
21             "prototype": "items",
22             "type": "banco",
23             "properties": {
24                 "altitude": {
25                     "length": 0,
26                     "unit": "cm"
27                 }
28             },
29             "selected": false,
30             "x": 444.0737738367833,
31             "y": 1555.0406079018835,
32             "rotation": 0
33         }
34     },
35
36     ...

```

Listing 5.1: JSON originario
prima della modifica di un utente

```

1      ...
2
3      "items": {
4          "BkrJVNl9e": {
5              "id": "BkrJVNl9e",
6              "prototype": "items",
7              "type": "cattedra",
8              "properties": {
9                  "altitude": {
10                     "length": 0,
11                     "unit": "cm"
12                 }
13             },
14             "selected": false,
15             "x": 452.23424450039795,
16             "y": 1750.8919038286344,
17             "rotation": 0
18         },
19         "rkiM4Ngcl": {
20             "id": "rkiM4Ngcl",
21             "prototype": "items",
22             "type": "estintore",
23             "properties": {
24                 "altitude": {
25                     "length": 100,
26                     "unit": "cm"
27                 }
28             },
29             "selected": false,
30             "x": 831.6984550413513,
31             "y": 1620.1209426149287,
32             "rotation": 0
33         }
34     },
35
36     ...

```

Listing 5.2: JSON dopo la
modifica di un utente

```

1   layers:
2     layer - 1:
3       items:
4         Hk1xEVg5g:
5           -
6             id:          Hk1xEVg5g
7             prototype:  items
8             type:        banco
9             properties:
10            altitude:
11              length: 0
12              unit:   cm
13            selected: false
14            x:        444.0737738367833
15            y:        1555.0406079018835
16            rotation: 0
17            - 0
18            - 0
19         rkiM4Ngcl:
20           -
21           id:          rkiM4Ngcl
22           prototype:  items
23           type:        estintore
24           properties:
25           altitude:
26             length: 100
27             unit:   cm
28           selected: false
29           x:        831.6984550413513
30           y:        1620.1209426149287
31           rotation: 0

```

Listing 5.3: Output dopo l'esecuzione del modulo `jsondiff` con le differenze tra i due file JSON precedenti

Sfruttando le tecnologie sopra descritte, si può pensare di sviluppare il framework *Metior* per la Progettazione Collaborativa gestendo le modifiche apportate dai diversi utenti che lavorano su uno stesso progetto seguendo un workflow simile a quello che usa GitHub nella gestione dei repository, consentendo all'utente di effettuare operazioni come merge, push e pull. Per gestire il problema della conflittualità sulle modifiche apportate dagli utenti, si potrebbe inserire una gestione intelligente dei layer consentendo all'utente di fare un “lock” sul layer sul quale sta lavorando non consentendo così agli altri utenti di apportare modifiche fino al suo rilascio.

5.2.3 Fotorealismo

Con *Fotorealismo* si intende semplicemente che una scena simulata è indistinguibile da una fotografia, o per estensione dalla vita di tutti i giorni. Questo è possibile attraverso un processo di *rasterizzazione*, che consiste in un algoritmo che permette di convertire un'immagine a due dimensioni in una formata da pixel per avere fotogrammi proiettabili sugli schermi.

Il servizio *Baking* [25] è un servizio remoto che prende una rappresentazione della scena in JSON come input, calcola le texture lightmapped, le Mappe Ambientali per riflessione e rifrazione e memorizza le informazioni grafiche in un formato che è compatibile con quello d'ingresso.



Figura 5.2: Esempio di un ambiente realizzato con il fotorealismo

Lo scopo principale è semplificare il workflow durante la visualizzazione 3D per fornire una *User Experience* ad alto livello sul browser (desktop, tablet e mobile) o su *wearables* come Google Carboard.

Lo scopo di questo sviluppo è compattare le strutture dati 3D prodotte dall'editor sul browser, trasmetterle ad un servizio di baking web remoto, e restituire una piacevole esperienza VR in real-time con alto realismo e frame rate.

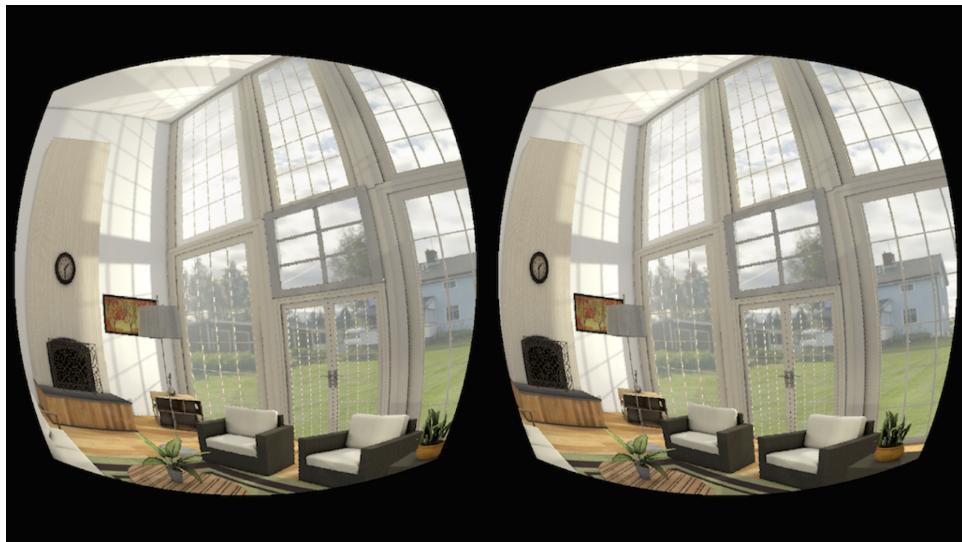


Figura 5.3: Esempio di visione di un ambiente attraverso VR

Il fotorealismo può essere esteso in un contesto che fornisce Indoor mapping e indoor/outdoor 3D di modelli realistici partendo da (a) documenti catastali e/o disegni di costruzione, e (b) scansioni outdoor/indoor realizzate tramite droni che restituiscono un set di fotografie e di nuvole di punti.

In conclusione si può dire che il possibile range di applicazioni in cui utilizzare il framework *Metior* spazia dalla sicurezza all'interno di piccole aree o edifici pubblici, a e-commerce, accesso virtuale al patrimonio culturale, ai videogames, e molti altri ancora.

Bibliografia

- [1] Olugbenga O Akinade et al. «Waste minimisation through deconstruction: A BIM based Deconstructability Assessment Score (BIM-DAS)». In: *Resources, Conservation and Recycling* 105 (2015), pp. 167–176.
- [2] Paola Altamura. «Gestione eco-efficace dei materiali da costruzione nel ciclo di vita del fabbricato». (in Italian). Tesi di dott. Sapienza Università di Roma, 2012.
- [3] P. Bryan e M. Nottingham. *JavaScript Object Notation (JSON) Patch*. Rapp. tecn. 6902. RFC Editor, 2013, pp. 1–18.
- [4] Rob Crossley. *Study: Average dev costs as high as \$28m*. <http://www.develop-online.net/news/study-average-dev-costs-as-high-as-28m/0106030>. 2010.
- [5] Chris Dawson. *JavaScript's History and How it Led To ReactJS*. <https://thenewstack.io/javascripts-history-and-how-it-led-to-reactjs/>. 2014.
- [6] Facebook. *JSX In Depth*. <https://facebook.github.io/react/docs/jsx-in-depth.html>.
- [7] Facebook. *Refs and the DOM*. <https://facebook.github.io/react/docs/refs-and-the-dom.html>.
- [8] Firebase. <https://firebase.google.com/>.

- [9] M Galic et al. «BIM in planning deconstruction projects». In: *eWork and eBusiness in Architecture, Engineering and Construction: ECPPM 2014* (2014), p. 81.
- [10] Zef Hemel. *Facebook's React JavaScript User Interfaces Library Receives Mixed Reviews*. <https://www.infoq.com/news/2013/06/facebook-react>. 2013.
- [11] Thom Hos. *Reactivity, state and a unidirectional data flow*. URL: <https://blog.deptagency.com/reactivity-state-and-a-unidirectional-data-flow-340e793ebf89>.
- [12] James W. Hurley. «How to SMARTWaste the Construction Industry». In: *10th Symposium Construction Innovation and Global Competitiveness*. Conference Proceedings for the 10th Syposium Construction Innovation and Global Competitiveness. 2002.
- [13] *Industry Foundation Classes, ISO 16739:2013*. http://www.iso.org/iso/catalogue_detail.htm?csnumber=51622. Accessed: 2016-12-29. 2013.
- [14] D. Jackson. *WebGL Specification*. Khronos Recommendation. "<https://www.khronos.org/registry/webgl/specs/1.0.3/>". Khronos, 2014.
- [15] Dean Jackson et al. *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. W3C Recommendation. "<http://www.w3.org/TR/2011/REC-SVG11-20110816/>". W3C, ago. 2011.
- [16] *Khronos Releases Final WebGL 1.0 Specification to Bring Accelerated 3D Graphics to the Web without Plug-ins*. <https://www.khronos.org/news/press/khronos-releases-final-webgl-1.0-specification>.

- [17] Paul Krill. *React: Making faster, smoother UIs for data-driven Web apps.* <http://www.infoworld.com/article/2608181/javascript/react--making-faster--smoother-uics-for-data-driven-web-apps.html>. 2014.
- [18] *Libscope.* <http://libscore.com>.
- [19] James Long. *Immutable Data Structures and JavaScript.* URL: <http://jlongster.com/Using-Immutable-Data-Structures-in-JavaScript>.
- [20] Mrdoob. *First public version. Still a lot to do.* <https://github.com/mrdoob/three.js/commit/a90c4e107ff6e3b148458c96965e876f9441b147>. 2010.
- [21] Mrdoob. *JavaScript Performance Monitor.* <https://github.com/mrdoob/stats.js>.
- [22] Mrdoob. *White Paper for Three.js? #1960.* <https://github.com/mrdoob/three.js/issues/1960>.
- [23] Jay Munro et al. *HTML Canvas 2D Context.* W3C Recommendation. <http://www.w3.org/TR/2015/REC-2dcontext-20151119/>. W3C, nov. 2015.
- [24] *npm.* <https://www.npmjs.com/>.
- [25] Alberto Paoluzzi. *Web 3D Indoor Authoring and VR Exploration via Texture Baking Service.* <http://paoluzzi.dia.uniroma3.it/web/pao/doc/web3d2016.pdf>. 2016.
- [26] Mike Roberts. *Serverless Architectures.* URL: <http://martinfowler.com/articles/serverless.html>.
- [27] Jarrod Rotolo. *The Virtual DOM vs The DOM.* URL: <http://revelry.co/the-virtual-dom>.

- [28] Alex Russell. *Web Components and Model Driven Views*. URL: <https://fronteers.nl/congres/2011/sessions/web-components-and-model-driven-views-alex-russell>.
- [29] Paolo Emilio Serra. *Cominciare a editare le famiglie* html. <http://puntorevit.blogspot.it/2007/11/cominciare-editare-le-famiglie.html>. 2007.
- [30] *Threejs Inspector*. <http://zz85.github.io/zz85-bookmarklets/threelabs.html>.
- [31] *WebGL Inspector*. <http://benvanik.github.io/WebGL-Inspector/>.
- [32] Wikipedia. *O3D*. <https://en.wikipedia.org/wiki/O3D>.
- [33] Wikipedia. *Unity*. [https://en.wikipedia.org/wiki/Unity_\(game_engine\)](https://en.wikipedia.org/wiki/Unity_(game_engine)).