



Università degli Studi “Roma Tre”

Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di laurea magistrale

*Formalizzazione della procedura di implementazione  
di Plugin per Metior  
e personalizzazione in scenari reali*

Laureando

Stefano Perrone

Relatore

Prof. Alberto Paoluzzi

Correlatori

Dott. Enrico Marino, Dott. Federico Spini

Anno Accademico

2015/2016



*Dedicato alla mia famiglia*

# Indice

<b>Ringraziamenti</b>	<b>v</b>
<b>Introduzione</b>	<b>vi</b>
<b>1 BIM e Web</b>	<b>1</b>
1.1 Building Information Modeling . . . . .	1
1.2 Applicazioni BIM su Desktop . . . . .	4
1.3 Applicazioni BIM nel Web . . . . .	6
<b>2 Metior</b>	<b>8</b>
2.1 User Experience . . . . .	9
2.2 User Interface . . . . .	10
2.2.1 Viewer 2D . . . . .	12
2.2.2 Viewer 3D . . . . .	13
2.2.3 Plugin Catalog . . . . .	14
2.3 Stack Tecnologico . . . . .	15
2.3.1 React . . . . .	15
2.3.2 Threejs . . . . .	17
2.4 Stato dell'applicazione . . . . .	19
2.5 Architettura . . . . .	22
<b>3 Metior Plugin</b>	<b>23</b>
3.1 Definizione . . . . .	23

3.2	Tassonomia . . . . .	24
3.3	Proprietà Specifiche . . . . .	26
3.4	Generazione modelli server-side . . . . .	27
3.5	Processo implementativo . . . . .	28
3.5.1	Studio della geometria . . . . .	28
3.5.2	Definizione funzione 2D . . . . .	29
3.5.3	Definizione funzione 3D . . . . .	30
3.5.4	Instanziazione nel catalogo . . . . .	31
<b>4</b>	<b>Custom Plugin</b>	<b>32</b>
4.1	BaM . . . . .	32
4.2	Virtual CED . . . . .	38
4.3	Deconstruction . . . . .	41
<b>5</b>	<b>Conclusioni e sviluppi futuri</b>	<b>43</b>
5.1	Conclusioni . . . . .	43
5.2	Sviluppi futuri . . . . .	44
5.2.1	IoT . . . . .	45
5.2.2	Collaboratività . . . . .	46
5.2.3	Fotorealismo . . . . .	47
	<b>Bibliografia</b>	<b>48</b>

# Elenco delle figure

1	Fasi coinvolte nel BIM . . . . .	vi
1.1	Schermata Revit . . . . .	5
2.1	Schermata con visuale in prima persona . . . . .	9
2.2	Schermata componenti interfaccia . . . . .	11
2.3	Schermata viewer 2D . . . . .	12
2.4	Schermata viewer 3D . . . . .	13
2.5	Dettaglio Plugin: (a) Vista dei plugin nel catalogo, (b) inserimento oggetto dopo la selezione nel catalogo . . . . .	14
3.1	(a) Foto modello reale, (b) Immagine modello scomposto in elementi geometri semplici . . . . .	28
4.1	Vista 3D modello aula . . . . .	33
4.2	Dettaglio Plugin: (a) appendiabiti, (b) armadietto, (c) attacca- panni, (d) portaombrelli . . . . .	34
4.3	Dettaglio Plugin: (a) banco, (b) cattedra, (c) lavagna, (d) lim	35
4.4	Dettaglio Plugin: (a) cestino, (b) cestini differenziata, (c) condizionatore, (d) termosifone . . . . .	36
4.5	Dettaglio Plugin: (a) libreria, (b) ???, (c) finestra con tenda, (d) finestra con veneziana . . . . .	37
4.6	PROVVISORIA INSERIRE VISTA 3D CED . . . . .	38

4.7 Dettaglio Plugins: (a) estintore, (b) rilevatore fumo, (c) nspo, (d) porta antipanico . . . . .	39
4.8 Dettaglio Plugin: (a) metal detector, (b) rack server, (c) router wifi, (d) telecamera . . . . .	40
4.9 Realtà aumentata di un modello reale . . . . .	41

# 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 telefonici e portatili. 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 *Plugins*, 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. Nello corso del 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. una private company ICT) nell’ambito della modellazione di un *Virtual CED* (Centro Elaborazione Dati), il quale è stato realizzato con l’intento di avere un modello 3D navigabile, per consentire lo sviluppo di un sistema per la generazione di Indoor Intercactive Virtual Mapping per il supporto ad applicazioni di Indoor-Location, Indoor-Navigation ed IoT-Mapping. Altra importante collaborazione quella con il *CNG* - (Comitato Nazionale Geometri) per il progetto *BaM* (Bulding and Modelling), facente parte del progetto nazionale *Georientiamoci*, per l’orientamento nella scelta del percorso di studi degli studenti, per il passaggio tra le scuole medie e le 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 dei materiali di demolizione degli edifici, al fine di ottimizzare i processi e far diventare i materiali di demolizione risorse per ricostruire.

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 Reactjs 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 descrive come vengono implementati i plugins utilizzati all’interno di Metior, dando una descrizione completa delle caratteristiche intrinseche dei modelli. L’implementazione ha l’obiettivo di dare all’utente un’esperienza soddisfacente attraverso la personalizzazione dei plugins a seconda del contesto. Nel quarto capitolo si descrivono i contesti di applicazione, facendo un overview sul progetto BaM 4.1 realizzato in collaborazione con il CNG e la modellazione di un Virtual CED 4.2 all’interno di SOGEI, ed infine il progetto Deconstruction 4.3.

In conclusione 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, e l’introduzione nel framework il concetto di *Collaboratività* 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 è stato fatto 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, 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[21]. 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 amministrazioni 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 lavoro proponiamo 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/ricostruzione, 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 ottenerne una maggiore efficienza. Il trattamento di questi materiali richiede la codifica appropriata sia per lo smaltimento, secondo i codici del CAE (Catalogo europeo dei rifiuti), sia per la pianificazione e la progettazione di nuovi edifici, seguendo la metodologia BIM (Building Information Modeling). A questo scopo abbiamo bisogno di scene georeferenziate di realtà aumentata sulla base di modelli 3D veloci, facilmente navigabili e misurabili. Abbiamo già un'ottima conoscenza sui costi di costruzione (da zero), ma poco si sa circa i tassi di sostituzione (completare 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 consentire una realtà aumentata attraverso il riconoscimento semantico attraverso computer vision e fotogrammetrico di precisione fino a definizione centimetrica.

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

Per questo motivo si è preso come parametro di confronto questo framework

### 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 appor tare alla progettazione [31]. Revit, come programma BIM, (come si vede in Figura 1.1) è da intendersi come un approccio più vicino alla realtà percepita dagli esseri umani. 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: Gialli e Rossi. I punti deboli del programma sono rappresentati, invece, dall'interfaccia talvolta poco intuitiva e dalla qualità dei rendering, che, pur utilizzando il motore Radiosity, non è paragonabile a quella ottenibile con software di rendering dedicati.

verificare coerenza motori utilizzati nel framework

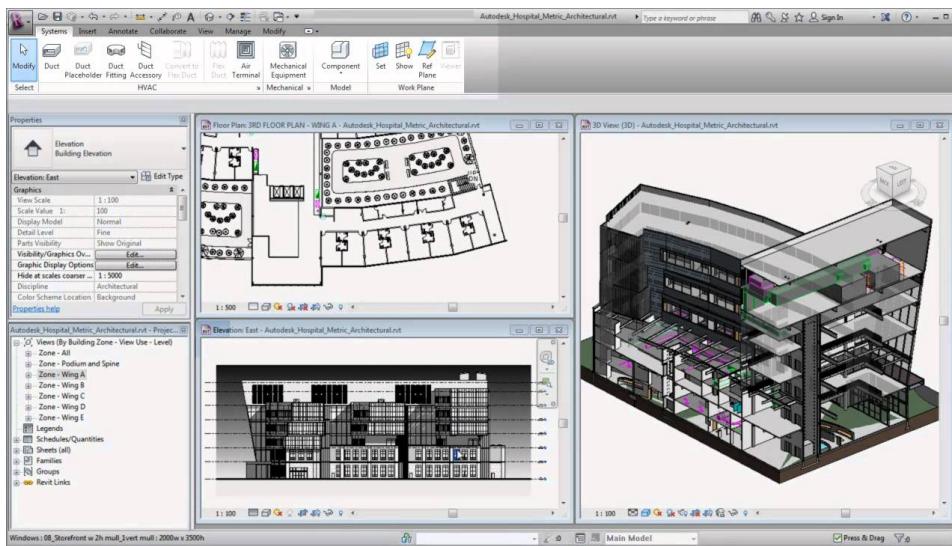


Figura 1.1: Schermata Revit

### 1.3 Applicazioni BIM nel Web

Il mondo di 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 lo sviluppo della complessità diventano maggiori preoccupazioni 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 di 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 [24] (che consente l’accesso diretto alla GPU), Canvas [27] (API raster 2D) e SVG [25] (API di disegno vettoriale), 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 parametrizzare esso 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 è stato trattato il concetto di BIM (Building Information Modelling), nell’ambito della modellazione 3D su piattaforme Web. Nel prossimo capitolo verrà descritta 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 con un elevata curva di apprendimento, ma che, 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 per conoscerne l’architettura del framework. Le funzionalità “semplificate” del BIM vengono implementate attraverso l’utilizzo di librerie software opensource, in particolar modo React e Threejs (aggiungere riferimento), 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 dell’interfaccia utente ci si è concentrati nel fornire all’utente una *User Experience* tale da semplificare il compito di modellazione di un edificio. L’interfaccia richiede l’utente utilizzi il più possibile l’interfaccia bidimensionale la quale permette di definire il piano e posizionare gli elementi architettonici complessi (chiamati *building elements*) su di essa. Tali *building elements* possono essere trovati in un catalogo pre-riempito, ed essere configurati ulteriormente e personalizzati attraverso il pannello laterale. Questo approccio di modellazione sposta la complessità verso lo sviluppatore dei *bulding 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 o in prima persona, (come mostrato in Figura 2.1).



Figura 2.1: 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 che ha generato il modello rispettivamente.

## 2.2 User Interface

Per definizione il software CAD è un acronimo inglese usato per indicare due concetti correlati, ma differenti:

- Computer-Aided Drafting;
- Computer-Aided Design;

L'applicazione web che si è implementata, si presenta come un software CAD semplificato, dove la user interface comprende:

- 2D canvas (Figura 2.3)
- 3D canvas (Figura 2.4).

L'area di lavoro (Figura 2.2) è suddivisa in aree ben specifiche, le quali adempiono a funzionalità che verranno approfondite in dettaglio nei paragrafi successivi.

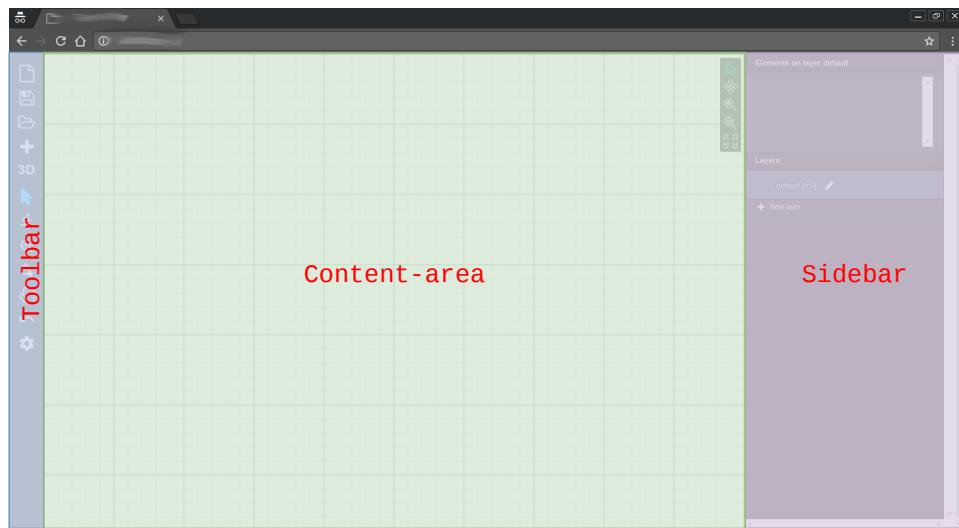


Figura 2.2: Schermata componenti interfaccia

Dalla toolbar l’utente può accedere alle funzionalità relative a: ciclo di vita del progetto (new, save, load); view/interaction mode switching (2D, 3D); interaction mode changing (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 posizionare gli oggetti. La sidebar visualizza le proprietà dell’elemento correntemente selezionato. Nel pannello delle proprietà è possibile vedere la descrizione dell’elemento, aggiungere/rimuovere metadata, 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* [29], 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*<sup>1</sup>.

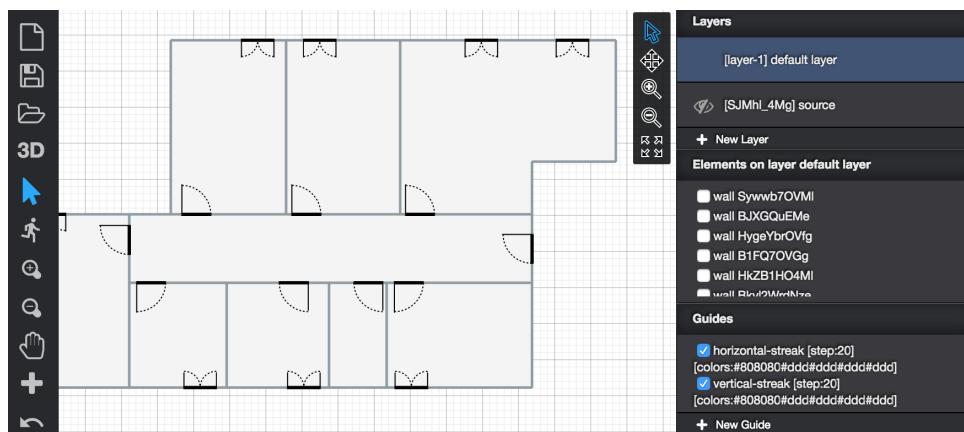


Figura 2.3: Schermata viewer 2D

---

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

### 2.2.2 Viewer 3D

Il *3D-viewer* invoca la funzione *3Dgf* dei *building elements* aggiunge al modello una vista 3D usando le primitive WebGL *Three.js*<sup>2</sup>. È stato implementato un *diff* e *patch* di sistema, definite nel Jsonpatch [22]: 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. (spostare prima causa effetto delle azioni) In particolare possiamo avere le seguenti *operazioni*:

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

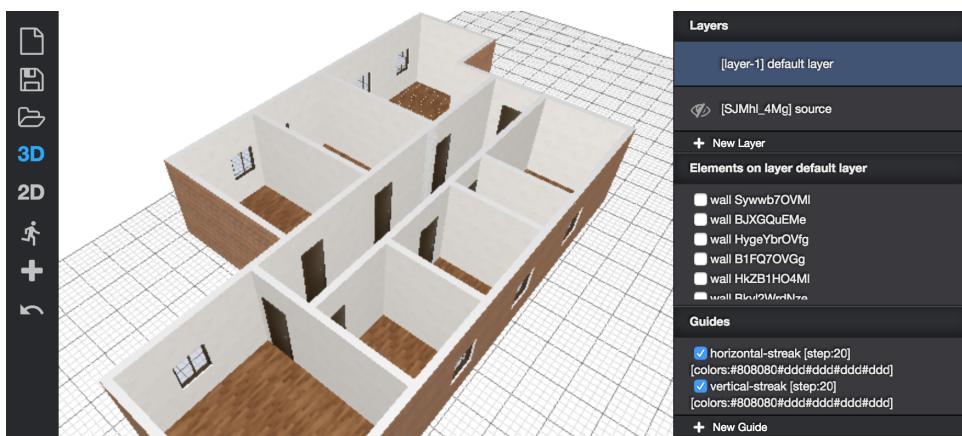


Figura 2.4: Schermata viewer 3D

---

<sup>2</sup><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 al suo interno 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 da inserire con un click dopo il quale si passa nella modalità 2D-view, dove verrà posizionato all'interno della scena (Figura 4.2 (b)).

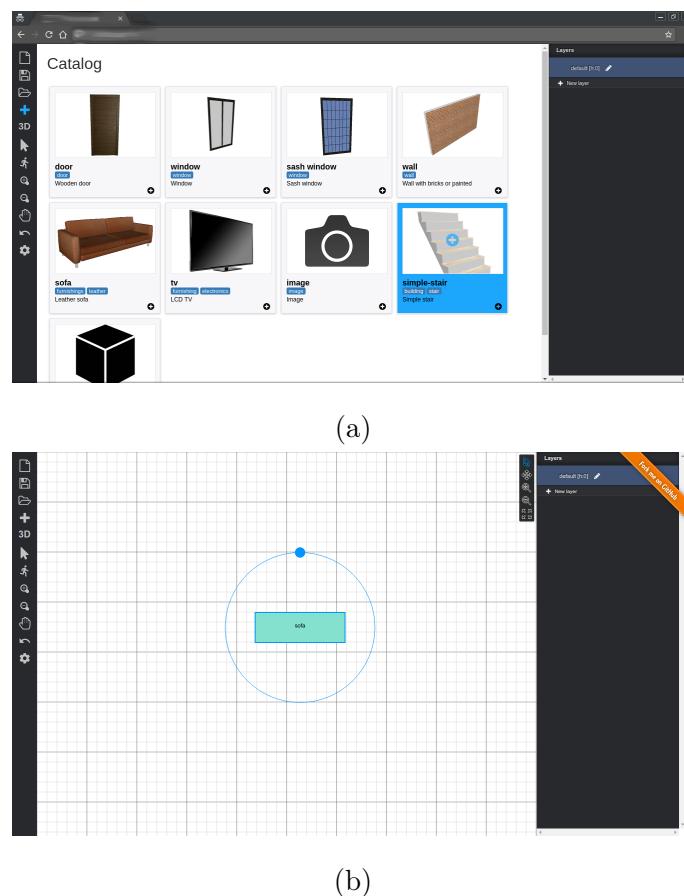


Figura 2.5: Dettaglio Plugin: (a) Vista dei plugin nel catalogo, (b) inserimento oggetto dopo la selezione nel catalogo

## 2.3 Stack Tecnologico

La User Interface è stata sviluppata seguendo i *Web Components pattern* [30], supportati dal framework *React.js*<sup>3</sup>. L'idea principale è definire un'applicazione 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 (a volte 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. [1] [2] [3]. Secondo il servizio di analisi JavaScript Libscore [4], 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 [5]. È stato distribuito sulle newsfeed di Facebook nel 2011 e in seguito Instagram.com nel 2012 [6]. 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

---

<sup>3</sup><https://github.com/facebook/react>

annunciato alla Facebook's React.js Conf in febbraio 2015 e reso open-source nel marzo 2015.

### Caratteristiche

In un flusso di dati unidirezionale le proprietà, ed un insieme di valori immutabili, sono passati al componente di rendering come proprietà nel suo tag HTML. Un componente non può modificare direttamente le proprietà passate ad esso, ma può usare funzioni di callback passategli che gli fanno 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 visualizzati in maniere efficiente. Questo permette al programmatore di scrivere codice come se l'intera pagina venisse cambia, 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 utilizzando la sintassi tag HTML per fare il render delle sottocomponenti. [8] 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/API 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 GitHub. (citazione)

#### 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. [12] [13] Questo è possibile grazie all'avvento delle specifiche WebGL. [14]

#### Storia

Three.js per è stato pubblicato da Ricardo Cabello su GitHub nel mese di aprile 2010. [20] Le origini della libreria risalgono al suo coinvolgimento con il demoscene nei primi anni 2000. Il codice è stato sviluppato 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. [15] I contributi di Cabello comprendono la progettazione API, CanvasRenderer, SVGRenderer e di essere responsabile per la fusione 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 quelli di CanvasRenderer o SVGRenderer. [15] I suoi maggiori contributi comportano migliorie nei moduli dei materiali, degli shaders e di post-processing.

## Caratteristiche

Three.js include le seguenti caratteristiche: [16]

- Effects: Anaglyph, cross-eyed and parallax barrier.
- Scenes: add and remove objects at run-time; fog
- Cameras: perspective and orthographic; controllers: trackball, FPS, path and more
- Animation: armatures, forward kinematics, inverse kinematics, morph and keyframe
- Lights: ambient, direction, point and spot lights; shadows: cast and receive
- Materials: Lambert, Phong, smooth shading, textures and more
- Shaders: access to full OpenGL Shading Language (GLSL) capabilities: lens flare, depth pass and extensive post-processing library
- Objects: meshes, particles, sprites, lines, ribbons, bones and more - all with Level of detail
- Geometry: plane, cube, sphere, torus, 3D text and more; modifiers: lathe, extrude and tube
- Data loaders: binary, image, JSON and scene
- Utilities: full set of time and 3D math functions including frustum, matrix, quaternion, UVs and more
- Export and import: utilities to create Three.js-compatible JSON files from within: Blender, openCTM, FBX, Max, and OBJ
- Support: API documentation is under construction, public forum and wiki in full operation

- Examples: Over 150 files of coding examples plus fonts, models, textures, sounds and other support files
- Debugging: Stats.js, [17] WebGL Inspector, [18] Three.js Inspector [19]

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:

- le informazioni necessarie dal prototipo dell'oggetto;
- i riferimenti per la mappatura e la relazione con gli altri oggetti;
- i 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 indicizzati da un identificativo univoco `id` permettendo la ricerca in tempo costante, del campo `selected` di ogni layer, consente l'accesso diretto ad elementi selezionati senza cercare. Lo stato può essere caricato un layer alla volta, consentendo la frammentazione dello stato così, come può avvenire in un progetto di modellazione di un edificio molto grande.

riscrivere tutto

```

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* [23] sfruttato dall'applicazione tramite la libreria *Redux.js*<sup>4</sup>. 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*).

sistemare Esperimenti preliminari<sup>5</sup> su strumenti di disegno 2D, infatti, hanno evidenziato la complessità di sviluppo di un'applicazione di questo tipo in termini di complessità di gestione dello stato interno dell'applicazione che viene modificato da diverse interazioni con l'utente, e che comporti un'associazione tra l'evento scatenato dall'utente sull'interfaccia e la logica dell'applicazione. Nell'architetture del framework, si definisce uno *state engine*, che rappresenta la logica dell'applicazione, che comprende actions e reducers e ne incapsula lo stato centralizzato. Di questo strato ne fanno uso in modo trasparente le differenti interfacce.

### Immutability pattern

L'*Immutability pattern* [26] 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  $s'$ ;

---

<sup>4</sup><http://redux.js.org/>

<sup>5</sup><https://github.com/cvdlab/walle>

- 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 possono essere ripristinati mediante 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 deve che si terrà in memoria. Abbiamo affrontato questo problema utilizzando *Immutable.js*<sup>6</sup>, 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.

## 2.5 Architettura

Secondo Mike Roberts [28], 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. Abbiamo visto come si presenta il framework descrivendo l’interfaccia con cui interagisce l’utente. Nel prossimo capitolo vedremo in dettaglio l’implementazione dei plugin.

---

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

# Capitolo 3

## Metior Plugin

In questo capitolo si descrive come vengono implementati i plugins 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 in dettaglio.

### 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* (RIVEDERE cioè una “class” in un Object Oriented Programming) 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*;

Quello *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 plugins *linear* sono inseriti disegnando linee attraverso l'interazione drag&drop; Il plugins *area* sono inseriti disegnando una bounding-box dell'elemento attraverso l'interazione drag&drop; Il plugins *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 plugins 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 proprietà presente un’interfaccia dedicata per l’inserimento dei valori. Ad esempio, un valore della proprietà booleana è impostato tramite una casella di controllo, 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* che sarà risolta con l’output del rendering. Tale scelta progettuale è importante poiché il calcolo per la generazione di modello può richiedere un certo 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. I plugin possono essere eseguiti sia da un client che da server, dal momento che le funzioni generatrici 2D e 3D (*2Dgf* e *3Dgf*) definite dai plugin possono essere eseguite dal server.(RIVEDERE ultima frase codice isomorfo)

## 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.

### 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 ), soprattutto in presenza di un modello complesso bisogna pensare a come realizzarlo cercando di scinderlo in parti geometriche semplici. Ad esempio prendiamo il caso di un estintore, esso è un oggetto composto geometria complessa, ed è scindibile in parti geometriche semplice come un cilindro più una sfera per il corpo centrale (Figura 3.1 - b ).



Figura 3.1: (a) Foto modello reale, (b) Immagine modello scomposto in elementi geometri semplici

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

### 3.5.2 Definizione funzione 2D

La definizione della funzione di render 2D definisce 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 rappresentare il Plugin all'interno di Metior durante la visualizzazione in 2D della scena.

```
1      render2D: (element, layer, scene) => {
2          let style = {
3              stroke: "#000",
4              strokeWidth: element.selected ? "2px" : "0px",
5              fill: element.properties.get('color')
6          };
7
8          return (
9              <g transform='translate(-50, -50)'>
10                 <rect x="0" y="0" width="100" height="100" style={style}/>
11             </g>
12         );
13     },
```

**Listing 3.1:** Definizione della funzione di render 2D

### 3.5.3 Definizione funzione 3D

La definizione della funzione di render 3D definisce attraverso codice scritto in Javascript e 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: (element, layer, scene) => {
2    let geometry = new BoxGeometry(100, 100, 100);
3    let material = new MeshBasicMaterial({
4      color: element.properties.get('color')
5    });
6
7    let mesh = new Mesh(geometry, material);
8
9    if (element.selected) {
10      let box = new BoxHelper(mesh, '#000000');
11      box.material.linewidth = 1;
12      box.material.depthTest = false;
13      box.renderOrder = 1000;
14      mesh.add(box);
15    }
16
17    mesh.position.y = +50;
18
19    return Promise.resolve(mesh);
20  }
```

**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 cube from './items/cubo/cubo';
4 ...
5 catalog.registerElement(cube);
6 ...
7 export default catalog;
```

**Listing 3.3: Instanziazione del Plugin nel Catalogo**

In questo capitolo è descritto come vengono implementati i plugin utilizzati all'interno di Metior. Nel prossimo capitolo si farà un overview sui contesti di applicazione affrontati.

## Capitolo 4

# Custom Plugin

In questo capitolo si descrivono i contesti di applicazione del progetto sviluppato. Le prima sezione presenta un overview sul progetto BaM realizzato in collaborazione con il Comitato Nazionale dei Geometri. La seconda sezione descrive la modellazione di un Virtual CED in collaborazione con SOGEI. Infine nell’ultima sezione si descrive il progetto Deconstruction per la previsione dei costi di demolizione e smaltimento degli edifici.

### 4.1 BaM

Il Progetto BaM (Building and Modelling), sviluppato in collaborazione con il CNG (Comitato Nazionale Geometri), nasce dall’idea di far realizzare, agli studendi frequentanti le scuole medie in Italia, “*l’aula che vorrei*”. Lo scopo principale di questa iniziativa è sensibilizzare gli studenti a utilizzare materiali ecosostenibili e rispettare l’ambiente. Gli alunni avranno poi la possibilità di scegliere degli elementi con cui personalizzare l’aula virtuale. La lista degli oggetti 3D 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 software è stato strutturato in modo che a ogni elemento sia associato un punteggio seguendo la categorizzazione precedentemente definita. (Spiegare in dettaglio competizione) Vincerà il team che realizzerà 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 è presente un modello di aula realizzato tramite il framework *Metior*.



Figura 4.1: Vista 3D modello aula

(specificare cosa si intende per valutazione) I plugin presenti all'interno del catalogo sono stati suddivisi in categorie per consentire una valutazione al termine dell'esperienza di modellazione dell'aula. Il primo gruppo di plugin riportato sono (Figura 4.2): un appendiabiti, un armadietto, un attaccapanni ed un porta ombrelli.

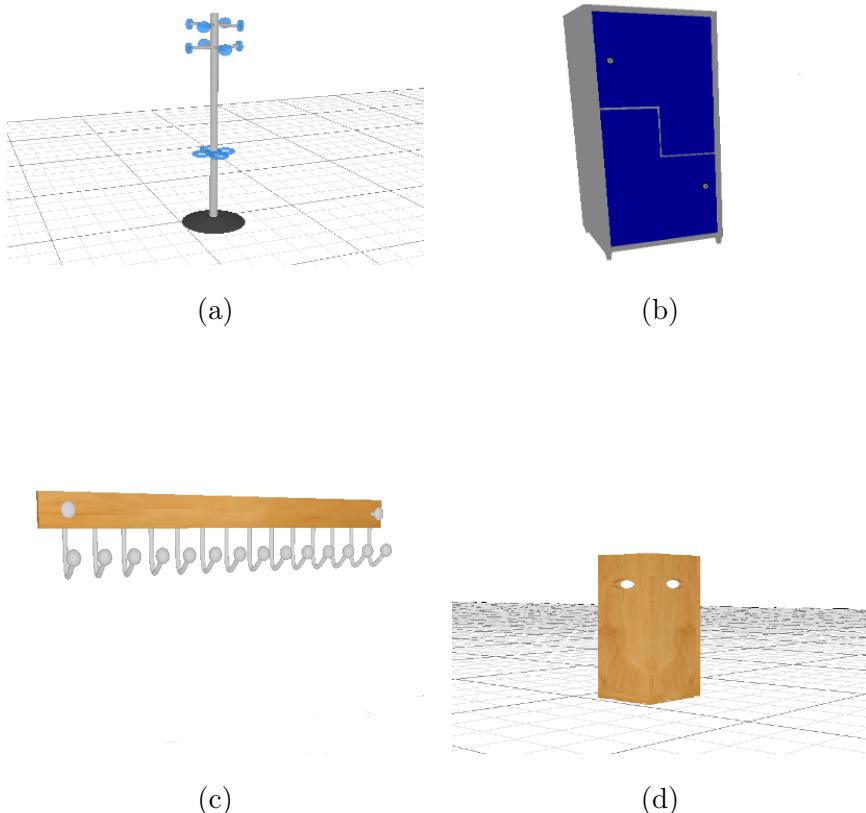


Figura 4.2: Dettaglio Plugin: (a) appendiabiti, (b) armadietto, (c) attaccapanni, (d) portaombrelli

Il secondo gruppo di plugin riportato sono (Figura 4.3): un banco, una cattedra, una lavagna ed una lavagna interattiva multimediale (lim).

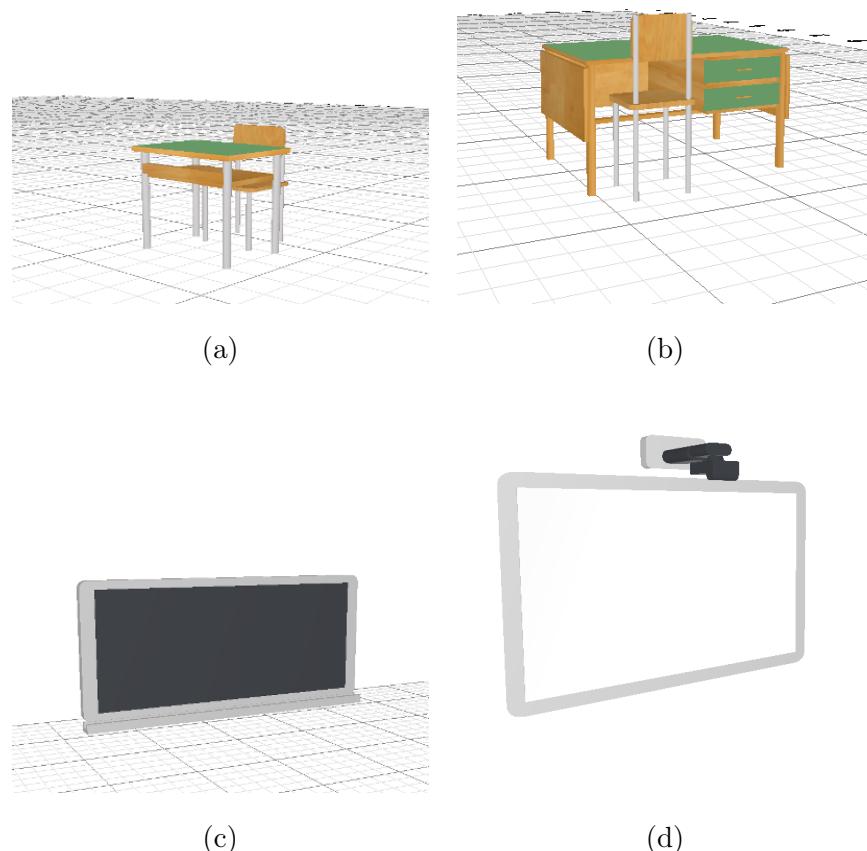


Figura 4.3: Dettaglio Plugin: (a) banco, (b) cattedra, (c) lavagna, (d) lim

Il terzo gruppo di plugin riportato sono (Figura 4.4): un cestino, un gruppo di cestini per la raccolta differenziata, un condizionatore d'aria ed un termosifone.

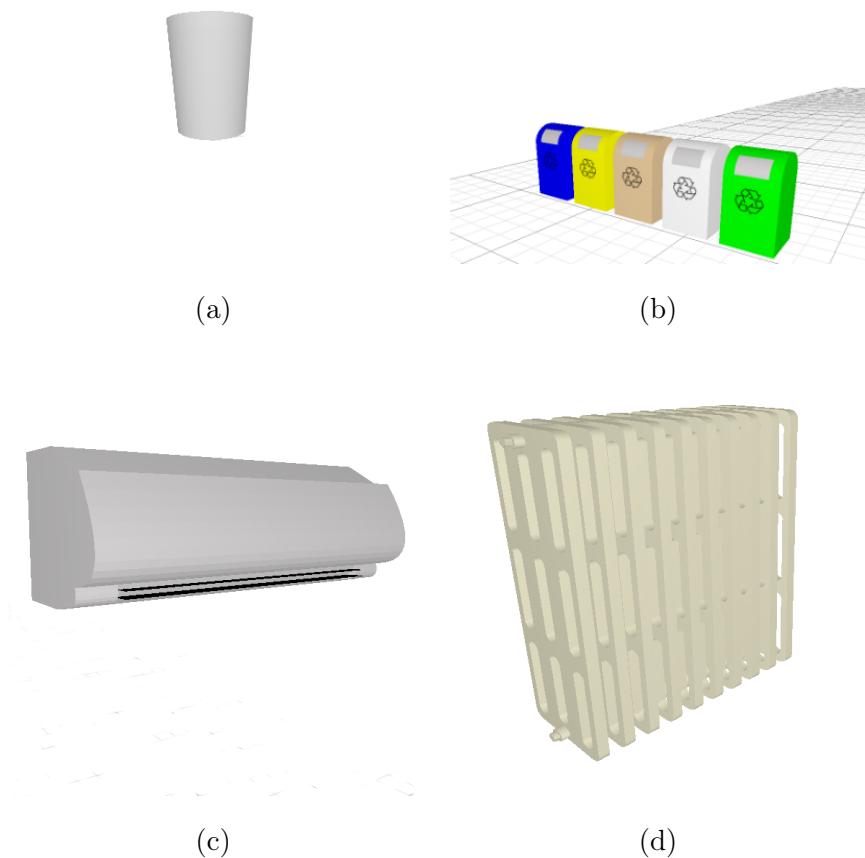


Figura 4.4: Dettaglio Plugin: (a) cestino, (b) cestini differenziata, (c) condizionatore, (d) termosifone

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

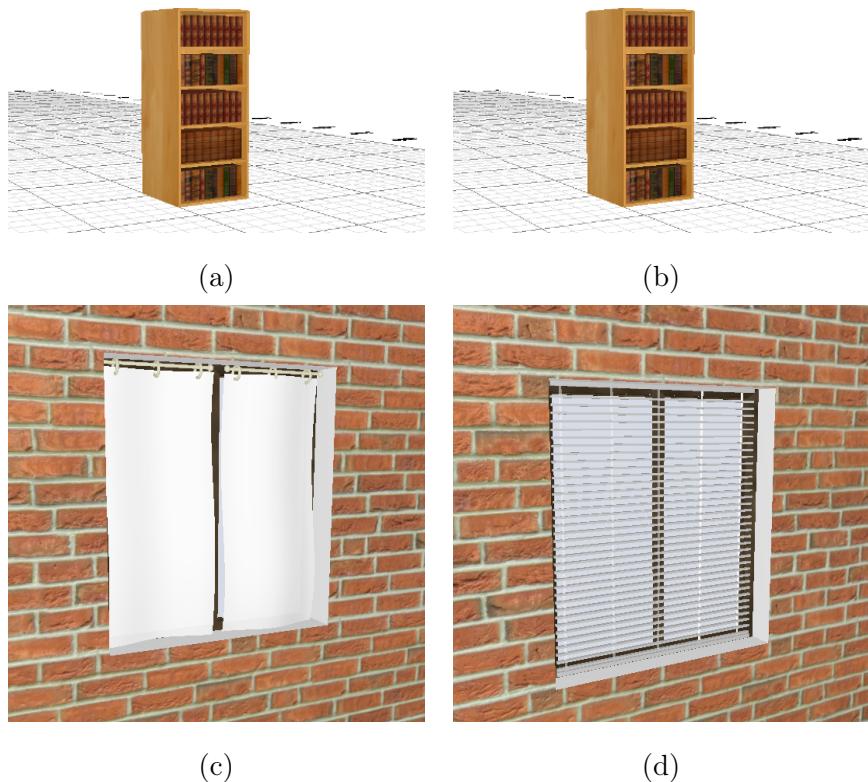


Figura 4.5: Dettaglio Plugin: (a) libreria, (b) ???, (c) finestra con tenda, (d) finestra con veneziana

## 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 plugins coerenti con il contesto, di seguito (come si evince dalla Figura 4.6) una vista 3D del modello dall'alto.

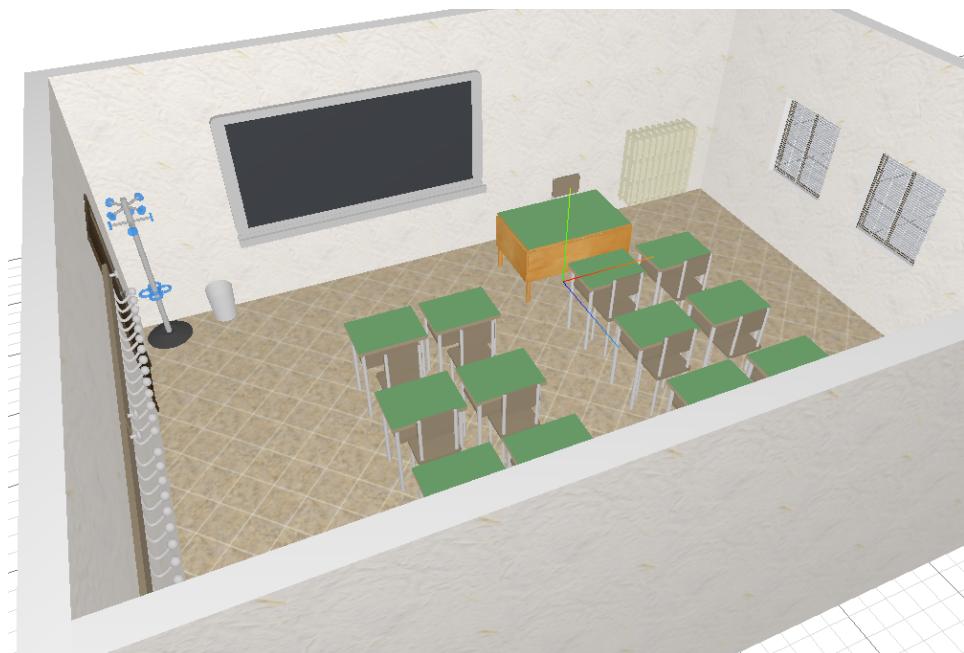


Figura 4.6: PROVVISORIA INSERIRE VISTA 3D CED

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



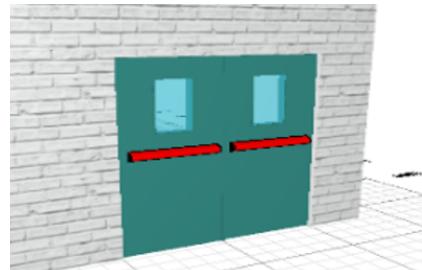
(a)



(b)



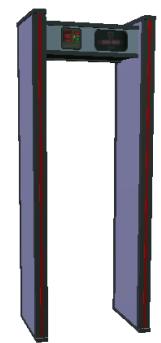
(c)



(d)

Figura 4.7: Dettaglio Plugins: (a) estintore, (b) rilevatore fumo, (c) naspo, (d) porta antipanico

Il secondo gruppo di plugin riguardano il contesto tecnologico (Figura 4.8), essi sono un metal detector, un rack server, un router wifi ed una telecamera.



(a)



(b)



(c)



(d)

Figura 4.8: Dettaglio Plugin: (a) metal detector, (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).



Figura 4.9: Realtà aumentata di un modello reale

In questo capitolo sono stati descritti i contesti di applicazione del framework Metior. Nel prossimo capitolo si trarranno le conclusioni, proponendo dei possibili 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 sviluppi futuri.

### 5.1 Conclusioni

In questa tesi si è contribuito ad implementare un'architettura serverless per supportare la modellazione di edifici in ambiente Web. 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. (le parti lato server rendering e gestione condivisa utenti può essere demanda su cloud)

## 5.2 Sviluppi futuri

I possibili sviluppi e campi di utilizzo del framework implementato possono essere tanti. Dopo aver utilizzato in scenari reali il framework, si è ritenuto di focalizzare gli sviluppi successivi nei seguenti ambiti:

- Integrazione IoT
- Collaboratività
- Fotorealismo

### 5.2.1 IoT

Nel mondo gli oggetti presenti nella vita di tutti i giorni hanno al proprio interno del hardware che consente di interagire con essi da remoto. Questi dispositivi sono denominati IoT (Internet of Things) e consentono l'estensione su Internet degli oggetti e dei luoghi reali. Questo ci consente di pensare all'inserimento di metadati all'interno dei plugin implementati, consentendo all'utente un'interazione realtime delle informazioni intriseche al modello.

### 5.2.2 Collaboratività

Un altro possibile sviluppo del framework è inserire la modalità *Collaboratività*, consentendo a più utenti di collaborare contemporaneamente su uno stesso progetto, ottimizzando i tempi di lavoro. Si fa riferimento all'utilizzo all'interno di uno studio di geometri. Lo stack tecnologico scelto per questo sviluppo sono *Firebase* e *jsondiff*. Il primo è stato scelto(perchè?), per consentire in un possibile sistema di autenticazione degli utenti, fare un controllo degli accessi e dei permessi al progetto. Il secondo modulo presente in npm(descrivere npm), consente di confrontare nel contesto collaborativo i file JSON su cui lavorano gli utenti per trovare le modifiche apportate.

### 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 la *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* (citazione) 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, in modo da permettere l'anello di retroazione di authoring(che significa?).

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. 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. (specificare cosa fa?) Si stanno sviluppando nuovi algoritmi che portino all'ottimizzazione, tra cui una soluzione per far fronte a scene di memoria basata su portali e frammentazione del ambienti caricando su richiesta solo una parte degli interi dati generati.(??????) 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. Il possibile range di applicazioni in cui utilizzare il framework spazia dalla sicurezza all'interno di piccole aree o edifici pubblici, a e-commerce, accesso virtuale al patrimonio culturale, ai videogames, e molto altro ancora.

# Bibliografia

- [1] <http://www.infoworld.com/article/2608181/javascript/react--making-faster--smoother-uis-for-data-driven-web-apps.html>.
- [2] <https://www.infoq.com/news/2013/06/facebook-react>.
- [3] <https://thenewstack.io/javascripts-history-and-how-it-led-to-reactjs/>.
- [4] <http://libscore.com>.
- [5] <https://www.quora.com/React-JS-Library/How-was-the-idea-to-develop-React-conceived-and-how-many-people-worked-on-developing-it-and-implementing-it-at-Facebook/answer/Bill-Fisher-17>.
- [6] <https://www.youtube.com/watch?v=A0Kj49z6WdM>.
- [7] <https://facebook.github.io/react/docs/refs-and-the-dom.html>.
- [8] <https://facebook.github.io/react/docs/jsx-in-depth.html>.
- [9] <https://code.facebook.com/posts/1014532261909640/react-native-bringing-modern-web-techniques-to-mobile/>.

- [10] [https://www.paypal-engineering.com/2015/04/27/isomorphic-react-apps-with-react-engine/.](https://www.paypal-engineering.com/2015/04/27/isomorphic-react-apps-with-react-engine/)
- [11] [http://techblog.netflix.com/2015/01/netflix-likes-react.html.](http://techblog.netflix.com/2015/01/netflix-likes-react.html)
- [12] [https://en.wikipedia.org/wiki/O3D.](https://en.wikipedia.org/wiki/O3D)
- [13] [https://en.wikipedia.org/wiki/Unity\\_\(game\\_engine\).](https://en.wikipedia.org/wiki/Unity_(game_engine))
- [14] [https://www.khronos.org/news/press/khronos-releases-final-webgl-1.0-specification.](https://www.khronos.org/news/press/khronos-releases-final-webgl-1.0-specification)
- [15] [http://www.develop-online.net/news/study-average-dev-costs-as-high-as-28m/0106030.](http://www.develop-online.net/news/study-average-dev-costs-as-high-as-28m/0106030)
- [16] [https://github.com/mrdoob/three.js/issues/1960.](https://github.com/mrdoob/three.js/issues/1960)
- [17] [https://github.com/mrdoob/stats.js.](https://github.com/mrdoob/stats.js)
- [18] [http://benvanik.github.io/WebGL-Inspector/.](http://benvanik.github.io/WebGL-Inspector/)
- [19] [http://zz85.github.io/zz85-bookmarklets/threelabs.html.](http://zz85.github.io/zz85-bookmarklets/threelabs.html)
- [20] [https://github.com/mrdoob/three.js/commit/a90c4e107ff6e3b148458c969652010.](https://github.com/mrdoob/three.js/commit/a90c4e107ff6e3b148458c969652010)
- [21] 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.
- [22] P. Bryan e M. Nottingham. *JavaScript Object Notation (JSON) Patch*. Rapp. tecn. 6902. RFC Editor, 2013, pp. 1–18.
- [23] Thom Hos. *Reactivity, state and a unidirectional data flow*. URL: <https://blog.deptagency.com/reactivity-state-and-a-unidirectional-data-flow-340e793ebf89>.

- [24] D. Jackson. *WebGL Specification*. Khronos Recommendation. "<https://www.khronos.org/registry/webgl/specs/1.0.3/>". Khronos, 2014.
- [25] 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.
- [26] James Long. *Immutable Data Structures and JavaScript*. URL: <http://jlongster.com/Using-Immutable-Data-Structures-in-JavaScript>.
- [27] Jay Munro et al. *HTML Canvas 2D Context*. W3C Recommendation. <http://www.w3.org/TR/2015/REC-2dcontext-20151119/>. W3C, nov. 2015.
- [28] Mike Roberts. *Serverless Architectures*. URL: <http://martinfowler.com/articles/serverless.html>.
- [29] Jarrod Rotolo. *The Virtual DOM vs The DOM*. URL: <http://revelry.co/the-virtual-dom>.
- [30] Alex Russell. *Web Components and Model Driven Views*. URL: <https://fronteers.nl/congres/2011/sessions/web-components-and-model-driven-views-alex-russell>.
- [31] Paolo Emilio Serra. *Cominciare a editare le famiglie*. <http://puntorevit.blogspot.it/2007/11/cominciare-editare-le-famiglie.html>. 2007.
- [32] Pavel Solin. *Creative Computing Platform: Learn Coding and 3D Modeling!* Accessed: 2016-11-12. 2016.