Università degli Studi *"Roma Tre"*

Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di laurea magistrale

Metior una piattaforma Web per la modellazione e visualizzazione di edifici

Laureando

Stefano Perrone

Relatore

Prof. Alberto Paoluzzi

Correlatori

Dott. Enrico Marino, Dott. Federico Spini

Anno Accademico

2015/2016

Dedicato alla mia famiglia

Indice

Elenco delle figure

Ringraziamenti

Grazie a...

Introduzione

Il lavoro presentato in questa tesi, svolto presso il CVDLAB, ed in collaborazione con SOGEI ed il CNG, è consistito nello studio delle tecnologie e nello sviluppo del framework Metior. L' obiettivo del progetto è stato portare il concetto di BIM sul Web, al fine di modellare i diversi ambienti a seconda del contesto.

Nel primo capitolo si descrive il concetto di BIM, nell'ambito della modellazione 3D su piattaforme Web. È stato fatto uno studio sullo stato dell'arte per fornire un overview sugli applicativi Desktop disponibili oggiorno. Il passo successivo è stato descrive come è possibile portare la modellazione sulle piattaforme Web. Nel secondo capitolo si descrive la scelta fatta per portare il BIM sul Web, proponendo una nostra soluzione Metior. Il framework implementa le funzionalità del BIM attraverso l'utilizzo di librerie software, in particolar modo Reactjs e Threejs, dando all'utente un applicativo paragonabile a quelli Desktop, che consente 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. Nel quarto capitolo si descrivono i contesti di applicazione del progetto sviluppato.

Capitolo 1

BIM e Web

In questo capitolo si descrive il concetto di BIM, nell'ambito della modellazione 3D su piattaforme Web. La prima sezione fornisce un overview sugli applicativi Desktop disponibili oggiorno. La seconda sezione descrive come é possibile portare la modellazione sulle piattaforme Web.

1.1 Building Information Modeling

Una tendenza a minimizzare l'umanizzazione di nuovi territori e di spingere per il riutilizzo di alloggi giá costruiti, caduti in disuso è diventata una necessitá pressante nelle societá avanzate. Noi abbiamo intrato il modello "zero energy" (ogni costruzione é stata prodotta con il consumo della stessa energia) con il modello "zero waste", ad esempio é un nuovo paradigma di progettazione dove i materiali di riufito della demolizione diventano risorse per ricostruire[1]. I processi di costruizione, contrato, 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 stru-

menti dovrebbero gestire l'elaborazione digitale dei materiali in tutto il ciclo di vita del progetto, supportando i requisiti del nuovo processo come: progettazione per Demolizione, Progettazione per il Riciclo e per i Rifiuti. In particolare, un ciclo di vita dell'edificio, sostenuto da un processo di costruzione che prevede cicli allineati ai fenomeni naturali è al centro di questo documento. 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 (dalla culla alla tomba) e verso il riutilizzo dei materiali in decostruzione / ricostruzione (dalla culla alla culla), supportato da computer aided processo di demolizione selettiva. Tutti i cicli di ristrutturazione deglii edifici dovrebbero prevedere passi di decostruzione 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 il CAE (Catalogo europeo dei rifiuti) codici, e per la pianificazione e la progettazione di nuovi edifici, seguendo la metodologia BIM (Building Information Modeling). A questo scopo abbiamo bisogno di scene georeferenziati di realtà aumentata sulla base di modelli 3D veloci, facilmente navigabile 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. Suggeriamo che i robot automatici potrebbero sostituire sforzo umano; droni potrebbero operare in un semanticamente contesto familiare e dare aggiornamenti in tempo reale come la realtà Contestualmente modifiche. Riteniamo, quindi, che c'è un grande bisogno di un moderno e facile framework di modellazione da usare per la costruzione/decostruzione nel settore AEC (Architecture, Engineering and Construction), per consenti-

re una realtà aumentata attraverso il riconoscimento semantico da computer vision e fotogrammetrico di precisione fino a definizione centimetrica. Tali strumenti di realtà virtuale / aumentata richiedono sia veloce modellazione costruzione 3D e aumento di contenuto semantico, per poter essere controllata vicino realtime: questa è la vera sfida richiesta anche dal futuro sviluppo di sistemi IoT.

1.1.1 Applicazioni Desktop

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[1], 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[2]. 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 quello 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.

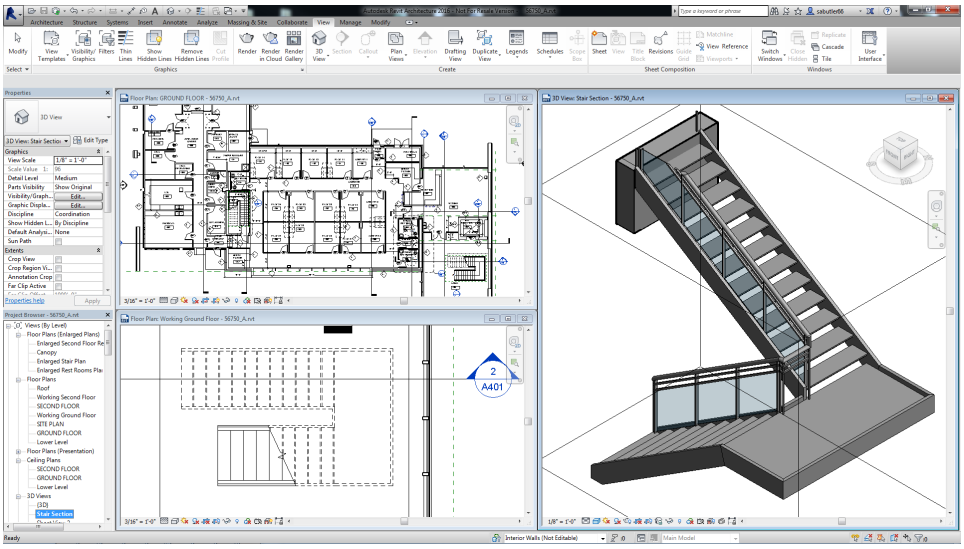

Figura 1.1: Schermata Revit

1.2 Applicazioni BIM nel Web

Oggigiorno stiamo assistendo ad una migrazione inarrestabile di prodotti software verso i servizi accessibili attraverso il mezzo web. Ciò è dovuto principalmente ai benefici innegabili in termini di accessibilità, usabilità, manutenibilità e spalmabilità concesso 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 porta desktop applicazione nel regno Web, un aspetto da prendere in considerazione anche per le differenze rilevanti tra hardware tutti i dispositivi dotati di un browser Web. Può essere ancora più arduo di affrontare un' architettura di software distribuito(un client / server almeno uno) indotta dalla piattaforma Web. Tuttavia sono sempre più ricche e complesse le applicazioni Web che sono apparite, sostenute dalle API arricchite HTML5, che grazie al WebGL [4] (Che consente l'accesso diretto alla GPU), Canvas [7] (API raster 2D) e SVG [5] (API di disegno vettoriale), ha aperto la strada per l'ingresso di applicazioni Web Graphic. In questo lavoro riportiamo il nostro impegno verso la definizione di uno strumento di modellazione 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 rispettivamente. Un'architettura serverless, al contrario di ciò che il nome può suggerire, in realtà si avvale di molti server specifici diversi, funzionamento e la manutenzione di cui non le fanno onere per lo sviluppatore del progetto. Questi diversi server possono essere visti come servizi di terze parti (tipicamente cloud-based) o funzioni eseguite in contenitori effimeri (può 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 ottenuto tramite un terzo API parti per la collaborazione gli utenti remoti.

L'interfaccia utente strumento, interamente basata sul modello componenti Web, è stato mantenuto il più semplice possibile: è richiesto all'utente interagire principalmente con bidimensionali segnaposto simbolici rappresentano parti dell'edificio, evitando così complesso 3D interazioni. La complessità del modello è quindi spostato dal modellatore per lo sviluppatore, che compila un estensibile *catalog* di *bulding elements* personalizzabili. Il modellatore deve solo selezionare l'elemento richiesto, il luogo e parametrizzare esso a seconda delle esigenze. È ovvio che un gran numero di elementi costruttivi deve essere fornito per garantire l'adempimento delle maggior parte delle esigenze di modellazione.

In questo capitolo é stato trattato 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 il BIM sul Web, Metior. Il framework implementa le funzionalità del BIM attraverso l'utilizzo di librerie software, in particolar modo React e Threejs, dando all'utente un applicativo paragonabile a quelli Desktop, che consente l'interazione del modello creato tramite una visualizzazione 2D e 3D.

2.1 Architettura

Secondo [8], l'architettura serverless schierate fanno largo uso di servizi di terzi, o *as-a-Service* componenti, che sostituiscono software e hardware ad-hoc per soddisfare gli stessi compiti. In particolare siamo stati in grado 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 stato di memoria.

2.2 Stato

Lo stato dell'applicazione è modellato utilizzando la struttura dei dati mostrato nel listato Si tratta essenzialmente di un insieme di layers, ciascuno contenente un insieme di vertici, linee, aree ed oggetti, ciascuno dei quali è racchiuso in una struttura composta da: (i) le informazioni necessarie dal prototipo dell'oggetto; (ii) i riferimenti mappatura della relazione con altri oggetti; (iii) i metadati, vale a dire il punto in cui personalizzare l'oggetto. fornire esempi di strutture dati adottati per modellare un vertice e una linea rispettivamente. licenziamenti di informazione vengono sfruttati per ridurre i tempi di accesso. Collezioni di oggetti sono indicizzati da `id` permettendo così di ricerca in tempo costante, del campo `selected` di ogni layer, consente l'accesso diretto ad elementi selezionati senza cercare. Lo stato può essere caricato uno layer alla volta per sostenendo la frammentazione dello stato così, come puø' avvenire in un progetto di modellazione di un edificio molto grande.

Unidirectional data flow

La struttura dei dati descritta rappresenta lo stato centralizzato richiesto dal *Unidirectional data flow pattern* [3] sfruttato con l'applicazione tramite la libreria *Redux.js*[1]. Il modello prevede che lo stato può essere modificato solo da attori specifici, chiamati *reducers*, le cui attività sono attivate da specifiche *actions* che contengono tutte le informazioni necessarie per ogni reducer per realizzare il cambiamento di stato. Ogni caratteristica applicazione deve essere attuata di conseguenza come un paio di pezzi ben isolate di codice (*action*/*reducer*). Esperimenti preliminari [2] su strumenti di disegno 2D, infatti, evidenziato la complessità di sviluppo di un'applicazione di questo tipo in termini di grande stato interno modificato da diversi interazioni

[1] `http://redux.js.org/`

[2] `https://github.com/cvdlab/walle`

con l'utente, che doveva essere ascoltato e applicata, che comporti un livello elevato accoppiamento tra logica dell'applicazione e l'interfaccia utente. Nella nostra messa a punto, invece abbiamo definito un *state engine*, che rappresenta la logica dell'applicazione, che comprende actions e reducers e ne incapsula lo stato centralizzato. Su questo strato può contare in modo trasparente interfacce differenti.

Immutability pattern

Immutability pattern [6] 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: (i) clonare lo stato precedente `s` per ottenere un nuovo stato `s '`; (ii) applicare le modifiche sullo stato clonato `s '`; (iii) di riferimento Aggiornamento 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* [3], una libreria che sfrutta la condivisione strutturale tramite mappe hash tentativi e tentativi vettoriali, riducendo al minimo la necessità di copiare o mettere i dati nella cache.

2.3 Stack Tecnologico

La User Interface é stata sviluppata seguendo i *Web Components pattern* [10], supportati dal framework *React.js*[4]. 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 ingrado 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 quelli a grana molto fine, i pulsanti per esempio. I più interessanti sono gli visualizzatori del modello di costruzione: il *2D-viewer* e *3D-viewer*.

[3] `https://facebook.github.io/immutable-js/`

[4] https://github.com/facebook/react

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 sostenuto da Facebook, Instagram e una comunità di singoli sviluppatori e aziende. [2] [3] [4] Secondo il servizio di analisi JavaScript Libscore, React è attualmente utilizzato sui siti web di Netflix, Imgur, Bleacher Report, Feedly, Airbnb, SeatGeek, HelloSign, Walmart, e altri. [5]

history

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. [6] É stato distribuito sulle newsfeed di Facebook nel 2011 e in seguito Instagram.com nel 2012. [7] È stato aperto-sourced a JSConf Stati Uniti nel maggio 2013. React Native, che consente iOS native, lo sviluppo Android e UWP con React, è stato annunciato alla Facebook's React.js Conf in Febbraio 2015 e open-source da marzo 2015.

Notable features

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 fanno gli modificare i valori. Il meccanismo della promise è espresso come proprietà di scorrimento verso il basso; azioni portate fino.

Virtual DOM

Un'altra caratteristica degna di nota è l'utilizzo di un virtual Document Object Model, o virtual DOM. React crea una cache struttura di dati in memoria, calcola le differenze risultanti, e poi gli aggiornamenti del browser

visualizzati efficientemente nel DOM [8] 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.[9] La sintassi HTML è trasformata in chiamate JavaScript della libreria React. Gli sviluppatori possono anche scrivere in JavaScript puro. JSX è simile ad un altro estensione di sintassi creata da Facebook per PHP, XHP.

Architecture beyond HTML

L'architettura di base di React si applica al di là del rendering HTML nel browser. Ad esempio, Facebook da grafici dinamici che fanno il rendering di un <canvas> tags, [10] e Netflix e PayPal utilizzano carico isomorfo per il rendering HTML in modo identico sia sul server che sul client. [11] [12]

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.

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 del browser di proprietà.[3][4] Questo è possibile grazie all'avvento di WebGL.[5]

History

Three.js per la prima volta viene pubblicato da Ricardo Cabello su GitHub nel mese di aprile 2010.[2] Le origini della libreria può risalire 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 il trasferimento di 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 aggiungere il renderer per questo abbastanza facilmente come Three.js creata con il codice di rendering come modulo anziché nel nucleo stesso.[7] 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 commits, Branislav Ulicny iniziato con Three.js nel 2010 dopo aver pubblicato una serie di demo WebGL sul proprio sito. Voleva la capacità di rendering di WebGL in Three.js fossero superiori a quelli di CanvasRenderer o SVGRenderer.[7] I suoi maggiori contributi comportano generalmente materiali, shaders e post-processing.

Subito dopo l'introduzione di WebGL su Firefox 4 nel marzo 2011, ha contribuito Joshua Koo. Ha costruito la sua primo demo Three.js per il testo 3D nel settembre 2011.[7] I suoi contributi spesso si riferiscono alla generazione geometrica. Ci sono oltre 650 collaboratori in totale.[7]

Features

Three.js include le seguenti caratteristiche:[8]

- 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,[9] WebGL Inspector,[10] Three.js Inspector[11]

Three.js è eseguibile in tutti i browsers supportati da WebGL.

2.4 User Interface

Qui si fa una descrizione della User Interface...

2.4.1 Viewer 2D

Il *2D-viewer* invoca il *2Dgf* degli elementi costruiti e aggiunti al modello e genera un suo output usando gli elementi SVG Per far fronte ai frequenti aggiornamenti provenienti dall'interazione con il disegno da parte dell'utente, sfrutta la *Virtual DOM* [9], 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, sviluppiamo un componente ad-hoc di React denominato *ReactSVGPanZoom*[5].

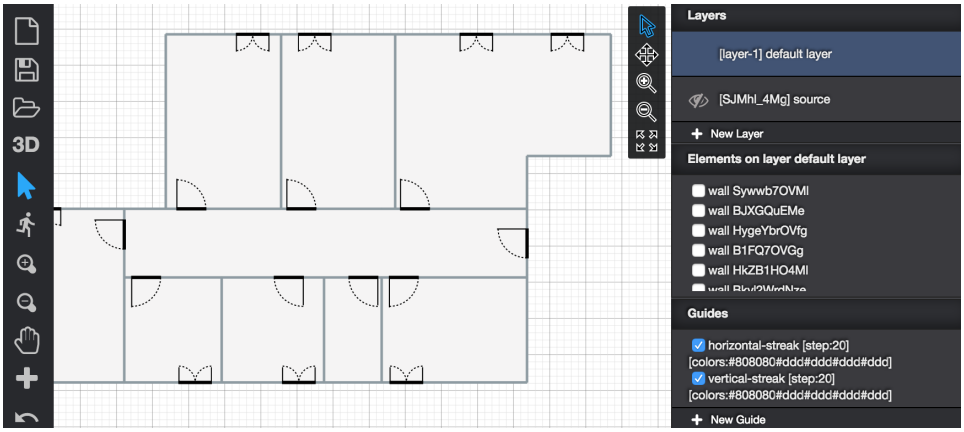

Figura 2.1: Schermata viewer 2D

[5]https://github.com/chrvadala/react-svg-pan-zoom

2.4.2 Viewer 3D

Il *3D-viewer* invoca il *3Dgf* dagli elementi di costruzione aggiungi al model e lo renderizza in output usando le primitive WebGL *Three.js* [6]. È stato implementato un *diff* e *patch* di sistema, standardizzate in [2]: gli oggetti Three.js sono associati con elementi costruttivi all'interno dello Stato, 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. In particolare possiamo avere le seguenti *operazioni*: (i) *add*, (ii) *replace* and (iii) *remove*.

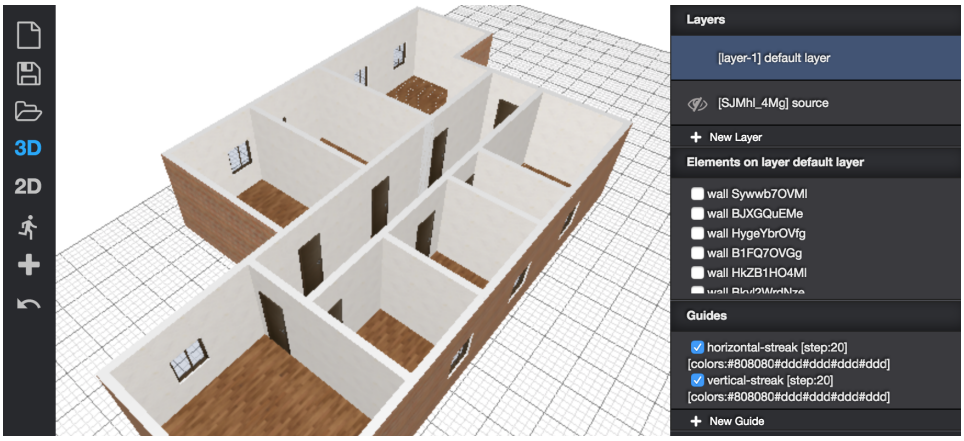

Figura 2.2: Schermata viewer 3D

[6] https://threejs.org/

2.5 User Experience

L'application experience si concentra sul sostegno che l'utente in un compito di modellazione edificio. L'approccio modellistico sfruttato l'utente deve affrontare il più possibile un'interfaccia bidimensionale che permette di definire il piano e per posizionare elementi architettonici complessi (chiamati *building elements*) su di esso. Tali *building elements* può essere trovato in un catalogo pre-riempito, e quando richiesto può essere configurato ulteriormente e personalizzato attraverso un pannello laterale. Questa mossa parte approccio di modellazione della complessità verso lo sviluppatore degli elementi costruttivi personalizzabili, lasciando all'utente finale il compito di posizionare e configurare gli elementi impiegati. Un ricco catalogo di elementi è quindi fondamentale per rispondere alle esigenze di modellazione degli utenti.

Una volta che il flor-plan è stato definito in base al 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 **??**. 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.

In questo capitolo é stato descrito il framework Metior. Nel prossimo capitolo vedremo l'implentazione dei plugins al suo interno.

Capitolo 3

Metior Plugins

In questo capitolo si descrive come vengono implementati i plugins utilizzati all'interno di Metior, dando una descrizione completa delle caratteristiche intrinseche dei modelli.

3.1 Definizione

plugin is a software component that can be seamlessly integrated into the system in order to extends its capabilities. In *Metior*, a plugin represents an architectural element that extends the Building Information Model design. Technically, a plugin represents a *prototype* (namely a "class" in Object Oriented Programming) of a construction element that can be inserted ("instantiated") into the *canvas*, thus defining a new *element*, i.e. a new component of the model.

Plugin definition A plugin is described by the following eight properties: (1) a unique name; (2) a description; (3) a set of metadata; (4) the *occupation type* (one among *linear*, *area* or *volume*); (5) the *placement type* (*inside* or *over*); (6) a set of specific properties mapping the semantic to associate to the plugin; (7) a *generating function* that returns the 2D representation of

the element in SVG format, to be used in the *2D-mode*; (8) a *generating function* that returns the 3D representation of the element in OBJ format, to be used in the *3D-mode*.

3.2 Tassonomia

The plugins can be organized according to *occupation type* and *placement type*.

In the *occupation type* three different kind of plugins can be identified: *linear*, *area* or *volume* plugins. The *linear* ones extend in one dimension (unless a radial thickness) (e.g. hydraulic lines, electrical cables). The *area* plugins extend in two dimensions (unless a linear thickness), (e.g. separation elements). They can be divided into *horizontal area* (e.g. floor and ceil), and *vertical area*, (e.g. walls). The *volume* plugins extend in three dimensions. They can be *fixed volume*, (e.g. a piece of furniture) and *scalable volume*, that can be scaled (proportionally or not), (eg. pillars, staircases).

The *occupation type* determines a different way to instantiate and to insert the plugins into the canvas. In particular, in *2D-mode*, *linear* plugins are inserted drawing lines by mean of a drag&drop interaction; the *area* plugins are inserted drawing the bounding-box of the element by mean of a drag&drop interaction; the *volume* plugins are inserted picking the position of the element by mean of a point&click interaction, and adjusting their dimensions modifying the bounding-box by drag&drop.

The *placement type* determines if the element can be inserted into the canvas in a specific point occupied or not by other elements. In other words, the placement type determines the relationship between a new instance of a plugin and instances of other plugins previously added to the model. The relationship can be of two kind: *inside* or *over*. Plugins belonging to the *inside* category can be added only inside other element (that can be *linear*, *area* or *volume*); e.g., a “window” is a “volume inside vertical area” element,

while an “hydraulic line” is a “linear inside horizontal area” element. Plugins of the *over* category can be added only over other elements (of any type); e.g., a “pillar” is a “volume over horizontal area” element, while an “electric panel” is a “volume over vertical area” element.

In the design phase, an element that doesn't meet the placement constraints defined by the *placement type* is notified by the system as a visual warning, showing its bounding-box in semi-transparent blinked red color.

3.3 Proprietá Specifiche

Each plugin has a set of specific properties of the building elements it represents. Each property is defined by (1) a *name*, (2) a *type*, such as “number”, “text”, “boolean”, or “custom”, and by (3) a *value*. According to its type, each property value can be inserted in different ways. For example, a boolean property value is set through a checkbox, while a textual property is set through a text box.

The system is designed to accept custom kinds of property. A custom property is required to define the component of the UI that permits the user to insert its value. For example, a “color” property can be introduced by defining a UI component composed by three text boxes (one for each RGB components), while a “length” property can be introduced by defining a UI component including a text box for the value and a drop-down menu for the unit of measure.

The specific properties of an element can be edited in the relative panel in the sidebar, once the element is selected in the canvas.

3.4 Plugin Catalog

It is pivotal to provide the system users with a rich catalog of plugins, to cover all the basic as well as the most advanced modeling requirements.

Table **??** (see Figure 3.1) reports examples of plugins arranged according to the taxonomy introduced in Section **??**.

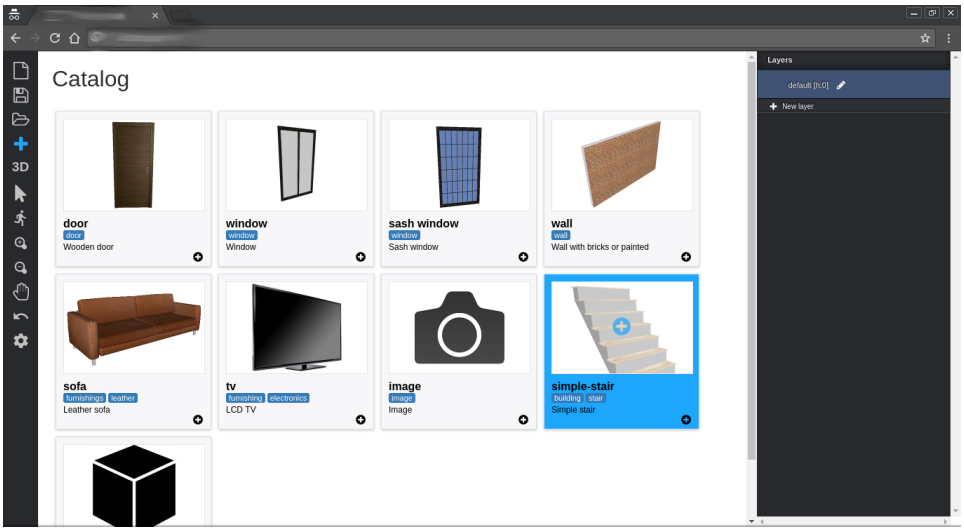

Figura 3.1: Catalogo dei Plugins

3.5 Server-side models generation

Both the 3D and 2D model generations have been designed as *asynchronous*. The actual result of the invocation of a generating function is not the generated model itself, but rather a *promise* of the expected result. Such a design choice is important since the computation for model generation may require some while. In the meantime the user must be able to interact with the interface, which in turn must remain responsive. Relying on this architecture, generation of the models can be easily delegated to a server (as shown in Figure **??**), thus relieving the client from the burden of onerous computations. The server exposes a REST-like HTTP-based JSON API to the client. The plugins span from the client to the server, since the 2D and 3D generating functions (*2Dgf* and *3Dgf*) defined by the plugin are actually executed on the server, as shown in Figure **??**.

3.6 Server Framework API

Our building deconstruction framework has a web-based client-server architecture, discussed in Section **??**. *Metior*, the web client application, is illustrated in Section **??**. The server-side of the framework, discussed in this section, is a plugin server written in Python, which capitalizes on the stack of geometric programming tools described above.

The Metior user quickly develops a 3D hierarchical assembly of different parts of the building envelope, as well as the horizontal and vertical partitions, using very simple 2D drawing tools. The more geometrically complex parts of the construction are conversely set up by user picking from context-based boards of predefined plugin templates, that are Python scripts (see Figure **??**) generating solids models which are interactively dimensioned, either using 2D drawing tools, or by user's numeric input from keyboard.

Of course, our list of *plugin templates* embraces most of building parts that are not manageable for quick shape input via 2D interaction. In particular, the picking boards include templates for planar concrete frames, spatial building frames, building foundations, roofs and stairs of different types, attics and dormers, fireplaces and fitted wardrobes, shover cabins and sanitary equipments, doors and windows, etc.

It is worth noting that, by virtue of the great expressiveness of the PLaSM operators and its functional style of programming and dimension-independent geometry, the development of a new plugin template is very easy even for non-experienced programmers, and usually requires a tiny amount of time and code, that may range between 4-8 hours, and between 10-100 lines of Python/pyplasm code.

Two important points we would like to remark are: (a) the great *expressive power* of the geometric language, strongly empowered by currying, i.e. by translating the evaluation of a function—that takes either multiple arguments or a tuple of arguments—into evaluating a sequence of functions,

each with a single argument; (b) the *ease of development*. Python/pyplasm is used even to teach geometric programming to K12 students [11] (see `https://nclab.com/3d-gallery/`). Several plugin templates used by Metior were developed in class by students, in the framework of the computer graphics course being taught by one of authors.

In questo capitolo sono stati implementati i plugins utilizzati all'interno di Metior. Nel prossimo capitolo si farà un overview sui contesti di applicazione.

Capitolo 4

Casi d'uso

In questo capitolo si descrive i contesti di applicazione del progetto sviluppato. Le prima sezione fanno un overview sul BAM e sul Deconstruction due progetti del Comitato Nazionale dei geomtri. La terza sezione descrive la modellazione del CED all'interno di Sogei.

4.1 BAM

Progetto sviluppato in collaborazione con il CNG

Appendiabiti

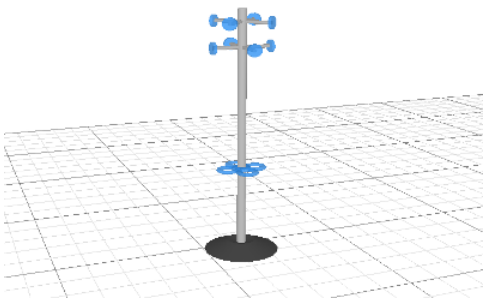

Figura 4.1: Modello 3D appendiabiti

(see Figure 4.1)

Armadietto

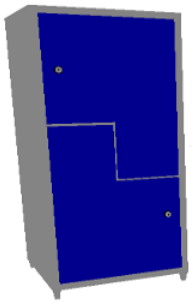

Figura 4.2: Modello 3D armadietto

(see Figure 4.2)

Attaccapanni

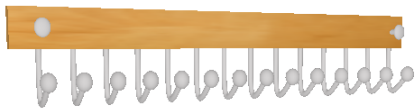

Figura 4.3: Modello 3D attaccapanni

(see Figure 4.3)

Banco

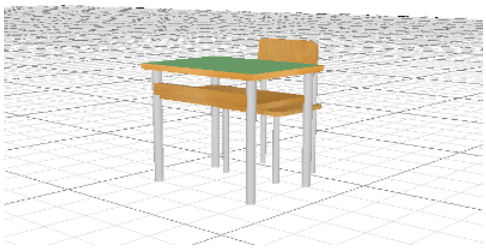

Figura 4.4: Modello 3D banco

(see Figure 4.4)

Cattedra

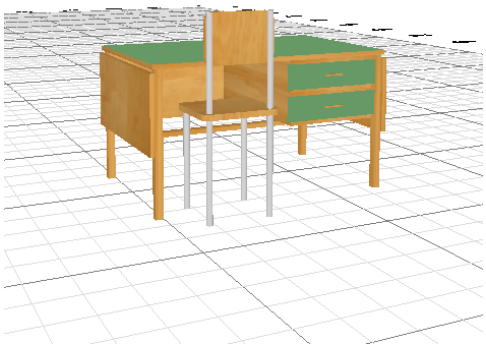

Figura 4.5: Modello 3D cattedra

(see Figure 4.5)

Cestino

Figura 4.6: Modello 3D cestino

(see Figure 4.6)

Condizionatore

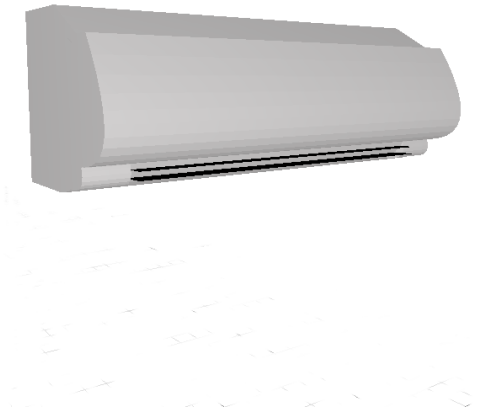

Figura 4.7: Modello 3D condizionatore

(see Figure 4.7)

Lavagna

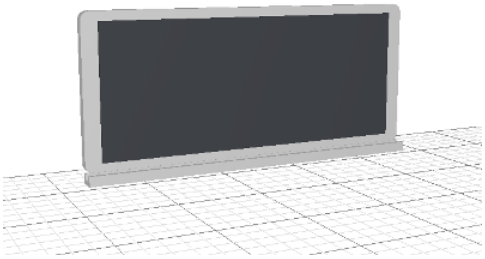

Figura 4.8: Modello 3D lavagna

(see Figure 4.8)

Libreria

Figura 4.9: Modello 3D libreria

(see Figure 4.9)

Lim

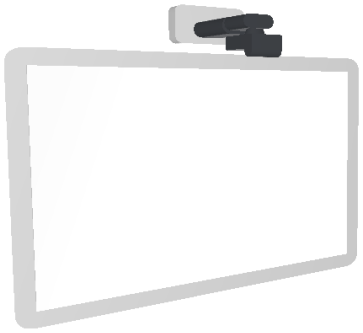

Figura 4.10: Modello 3D lim

(see Figure 4.10)

Porta Ombrelli

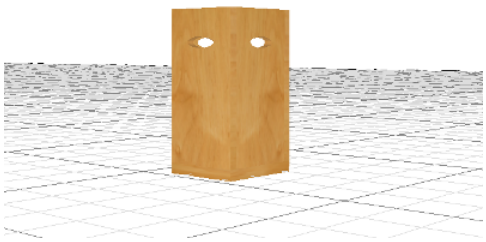

Figura 4.11: Modello 3D porta ombrelli

(see Figure 4.11)

Cestini Differenziata

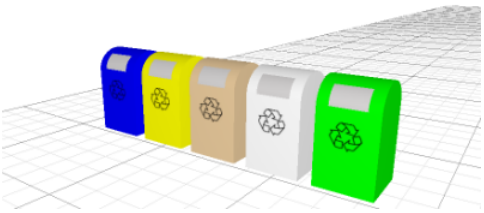

Figura 4.12: Modello 3D cestini differenziata

(see Figure 4.12)

Termosifone

Figura 4.13: Modello 3D termosifone

(see Figure 4.13)

Finestra con Veneziana

Figura 4.14: Modello 3D finestra con veneziana

(see Figure 4.14)

Finestra con Tenda

Figura 4.15: Modello 3D finestra con tenda

(see Figure 4.15)

4.2 Deconstuction

Progetto sviluppato in collaborazione con il CNG

4.3 CED

Progetto sviluppato in collaborazione con il SOGEI

Estintore

Figura 4.16: Modello 3D estintore

(see Figure 4.16)

Naspo

Figura 4.17: Modello 3D naspo

(see Figure 4.17)

Porte Antipanico

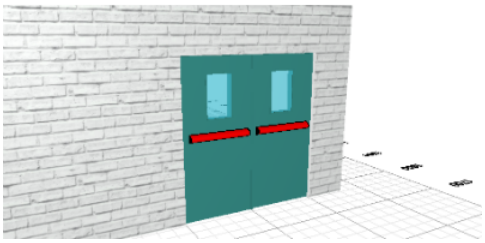

Figura 4.18: Modello 3D porte antipanico

(see Figure 4.18)

Rack Server

Figura 4.19: Modello 3D rack server

(see Figure 4.19)

Rilevatore fumo

Figura 4.20: Modello 3D rilevatore fumo

(see Figure 4.20)

Router Wifi

Figura 4.21: Modello 3D router Wifi

(see Figure 4.21)

Telecamera

Figura 4.22: Modello 3D telecamera

(see Figure 4.22)

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

Capitolo 5

Conclusioni e sviluppi futuri

Questo capitolo fa un resoconto sul progetto sviluppato ed i possibili sviluppi fututi.

5.1 Conclusioni

In this work we outlined a serverless architecture to support buildings modeling in a Web environment. The serverless architecture that gives benefits in terms of availability, reliability, scalability, easiness of deployment, maintainability and upgradability is obtained by implementing the application logic as a client-side only centralized state Web application exploiting the unidirectional data flow pattern. This approach allows for a easy-to-serialize state (in the form of a JSON document) that can be pushed on a third party document oriented DB-as-a-Service and loaded back in the frontend reactive architecture, which transparently reload the state once its serialized version is passed in. The application itself is served by a CDN (Content Delivery Network) thus avoiding any need for web server. Offline routines rely on Function-as-a-Service platform as well as users management and collaboration features.

5.2 Sviluppi futuri

I possibili sviluppi e campi di utilizzo del framework implementato possono essere tanti. Pensando al contesto Io-T si può pensare all'inserimento di meta dati all'interno dei plugin implementati, consentendo all'utente un interazione realtime con i plugin consentendo la fruizione di informazioni intriseche al modello...

Un altro possibile sviluppo del framework é renderlo collaborativo, consentendo a più utenti di collaborare su uno stesso progetto, ottimizando i tempi di lavoro. Si fa riferimento all'utilizzo all'interno di uno studio di geometri...

Un altro possibile sviluppo é nell'ambito mobile nella reazione di un sistema di Indoor Navigation all'interno di edifici...

Bibliografia

[1] 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.

[2] P. Bryan e M. Nottingham. *JavaScript Object Notation (JSON) Patch*. Rapp. tecn. 6902. RFC Editor, 2013, pp. 1–18.

[3] Thom Hos. *Reactivity, state and a unidirectional data flow*. URL: https://blog.deptagency.com/reactivity-state-and-a-unidirectional-data-flow-340e793ebf89.

[4] D. Jackson. *WebGL Specification*. Khronos Recommendation. https://www.khronos.org/registry/ Khronos, ott. 2014.

[5] 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.

[6] James Long. *Immutable Data Structures and JavaScript*. URL: http://jlongster.com/Using-Immutable-Data-Structures-in-JavaScript.

[7] Jay Munro et al. *HTML Canvas 2D Context*. W3C Recommendation. http://www.w3.org/TR/2015/REC-2dcontext-20151119/. W3C, nov. 2015.

[8] Mike Roberts. *Serverless Architectures*. URL: http://martinfowler.com/articles/serverless.html.

[9] Jarrod Rotolo. *The Virtual DOM vs The DOM*. URL: http://revelry.co/the-virtual-dom.

[10] Alex Russell. *Web Components and Model Driven Views*. URL: https://fronteers.nl/congres/2011/sessions/web-components-and-model-driven-views-alex-russell.

[11] Pavel Solin. *Creative Computing Platform: Learn Coding and 3D Modeling!* Accessed: 2016-11-12. 2016.