A Web Serverless Architecture for Buildings Modeling

Enrico Marino*, Danilo Salvati†, Federico Spini*, Christian Vadalà†
*Department of Engineering, Roma Tre University, Rome, Italy
Email:{marino,spini}@ing.uniroma3.it
†Department of Mathematics and Physics, Roma Tre University, Rome, Italy
Email:{salvati,vadala}@ing.uniroma3.it

***Abstract*—The motivations of relentless migration of software products toward services accessible via Web must be sought in the undeniable benefits in terms of accessibility, usability, maintainability and spreadability granted by the Web medium itself. It is the case of office suites, beforehand thought as resilient desktop applications, nowadays made available as Web applications, often equipped with real-time collaboration features and with no need for the user to explicitly install or upgrade them anymore. Although it could not be easy to envisage a Web-based graphic application due to its inherent complexity, after the recent and significant enrichment of the HTML5 APIs, a few first attempts appeared online in the form of vectorial drawing collaborative editors or VR oriented interior design environments. This paper introduces an effective Web architecture for buildings modeling that leverages the serverless pattern to dominate the developing complexity. The resulting front-end application, powered by Web Components and based on unidirectional data flow pattern, is extremely customizable and extendible by means the definition of plugins to augment the UI or the application functionalities. As regards the modeling approach, it offers (a) to model the building drawing the 2D plans and to navigate the building in a 3D first person point of view; (b) to collaborate in real-time, allowing to work simultaneously on different layers of the project; (c) to define and use new building elements, that are furnitures or architectural components (such as stairs, roofs, etc.), augmenting a ready to use catalog. This work suggests a path for the next-coming BIM online services, matching the professionals collaboration requirements typical of the BIM approach with the platform which supports them the most: the Web.**

1. Introduction

This work intend to show a web based tool for building design. About the architectural point has been introduced a serverless architecture (see [1]). This kind of architecture uses third party services (typically in the cloud) or functions executed into ephemeral containers (may only last for one invocation) to manage the internal state and for server-side logic.

This tool aims to build building model through a simplified UI (User Interface) that avoid complex 3D interactions that are replaced by interactions with two-dimensional symbolic part of the building.

The web character of the project permitted to test API for remote users collaboration, that allow to build together a building also if far each other.

The tool offer an high level of customization, thanks to: (i) **web components pattern** that allow to add new features; (ii) a customizable **catalog**, that allow to adapt building part of the model to the specific context in which the tool is using; (iii) a dynamic metadata handler the allow to set info on each part of the building.

Next sections describe this concepts, showing how they are realized and which choose were made: In details:

In the section 2 we will show some related works about web building modelling with a focus on strengths and weaknesses of our work.

In the section 3 we will show the core of the work introducing and commenting the most important architectural choose that we done

In the section 4 we will show the solution of the methodical choose that were made, plus a focus on some future works related to the project.

2. Literature review

As we have already seen in the previous section, this work introduce a platform for building design in the web. We will avoid comparisons with “desktop apps”, because they are usually complex (and with poor portability), anyway there are many related projects built as web platforms. Some offers only visualization, as the **Shapespark**[1] web viewer (it exists a desktop plugin for modeling), others also offer a modeling service like **bak3d** (see [2]). On the other hand, bak3d suffer of a complex user interaction. Another complex software is **Playcanvas**[2] that is a game engine which offers an integrated physical engine and other

1. https://www.shapespark.com/
2. https://playcanvas.com/

functionalities for the modeling results. In the architectural field, another interesting project is **Floorplan**[3], developed by Autodesk.
However, our objective was also to explore the web components paradigm and the uniflow pattern. The expected result, is an infrastructure for the building design and an extensible platform for users with the introduction of new components. This is the main difference with the above softwares, because our architecture is not only a modeling instrument but a builder for modeling softwares customized following the users needs.

3. Methodology

3.1. Application as Finite State Machine with centralized state

Development of the project focused on an individuation of a hierarchical Javascript object structure, defined state, that allow to fully represent all the parts that there are on a plan. The application state is described with an **immutable tree structure** of which is maintained a reference to the root node. Each atomic change require: (i) The build of a new state `s'` equal to the previous state `s`; (ii) Execution of changes on the new state `s'`; (iii) A replace of the root reference from `s` to `s'`. From the memory point this operations would require a lot of resource. To avoid this problem the implementation **Immutable.js** [4] made by Facebook, simulate immutability pattern, avoiding the deep clone of the memory.

The centralized state handling allow to develop features that have as unique goals the creation of a new state coerent with the requested change, without to manage on UI or any other component. The UI adjustment to the new state works thank to some mechanisms that we will illustrate in the section 3.4.

Thanks to the history of each state generated from the application and thanks to guarantees the offered by the immutability pattern we can, in each moment, perform a redo operation to an oldest state, by means of a replace of the current state with one from any of the past states.

To dominate the complexity of the application the system is modelled on a state machine. Thanks to a model based on actions and reducer offered by the pattern **unidirectional data flow** we have found a directed graph that show all available evolutions of the application state. Each node correspond to a **mode**, each edge correspond to an **action** that the user can execute starting from a node. Depending on the mode each event browser, for example `click` or `mousemove`, is mapped on an action. A part of the state machine is shown in figure 1.

Centralized state handling and the logic handling by mean of a state machine modelling has been possible thank to the library **Redux.js** [5].

3. http://www.homestyler.com/floorplan/
4. https://facebook.github.io/immutable-js/
5. http://redux.js.org/

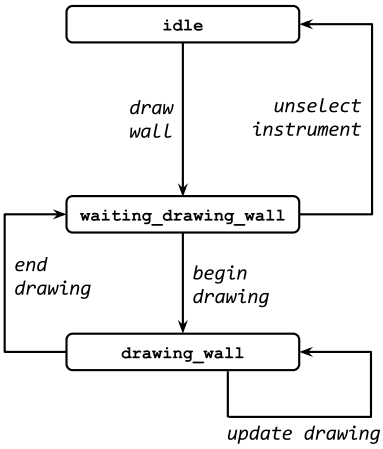

Figure 1. Subgraph of the state machine that show a wall creation. (i) Node `idle`: waiting mode of the application. No action has yet been taken. (ii) Node `waiting_drawing_wall`: user has selected wall design tool, but he hasn't already started the draw phase. (iii) Node `drawing_wall`: user has placed the starting point of the wall.

3.2. Collaboration

Now we will study the mechanisms chosen to provide collaboration for the same project, among users on different PCs. In literature there are many techniques for introducing collaboration between remote users. For example in [3] we can find the **Operational Transformation** technology, where a group of nodes exchanges messages without a *central control point*. It is based on two main properties: (i) you can transform changes relative to other changes and (ii) there is no matter in which order concurrent changes are applied, you end up with the same document.

However, the adoption of the uniflow pattern, with a global state, had suggested us to use another methodology for the collaboration feature. Following the principles of these patterns, the application is based upon a **centralized topology** for communication between nodes. With this approach the state is shared between several instances of the application with a communication infrastructure for the synchronization with the nodes. With this kind of topology, we avoided the peer-to-peer synchronization routines to remove implementation problems regarding the network infrastructure (for example ones caused by the *Network Address Translation*) common when using these kind of systems. Anyhow, this approach it does not influence the serverless nature of the architecture, as the communication systems are external (we adopted **Firebase**[6] for the data exchange between users)

3.3. Building elements

Each building element that the user can add on a plan is available through a **catalog**, structured in categories. Each

6. https://firebase.google.com/

category identify a group of use case whereby the user can interact.

Walls, each kind of wall (perimetral, interior, structural) is part of this category. The user can add a new wall selecting the begin and end of the element. The internal state of all the walls correspond to an undirected graph: each **node** match geometric points in which are intersected more walls and has a couple of coordinates that place it on a 2D space; each **edge** match a specific wall. This internal rappresentation offer a good interactive level to drag and drop walls operations: each wall drag is performed by means of a relocation of a node that cause, other that the displacement of the requested wall, a displacement of each adiacent wall.

Openings, each element that hole a wall (windows, doors, arches) is part of this category. The user can add a new opening thanks to a snap on an existent wall. The openings internal state link each element with its wall.

Areas, each floor belong to this category. They are automatically generated thanks to a wall graph analysis. The algorithm is composed of few phases: (i) Search of biconnected component by mean of Hopcroft-Tarjan algorithm (see [4]); (ii) Removal of edges that are not part of a biconnected component; (iii) Search of all cycles through an algorithm that do a double check of each edges sorted by angle; (iv) Search of maximal cycles correspondent to perimeter edges by an application of Gauss's area formula; (v) Removal of maximal cycles;

Objects, each element that can be arranged on an area is part of this category. The user can interact with the object and change its disposal and rotation.

3.4. UI components

The entire application has been developed using separated modules, following the **SOLID programming** principles, as it could be expanded. For the web part, we have used the **web components pattern**, with the implementation of **React.js**[7] written by Facebook. The main idea is to define the frontend application as a collection of components, each one rendered in a different way according with values assigned to the state. In Figure 2 we have an example of based on our application.

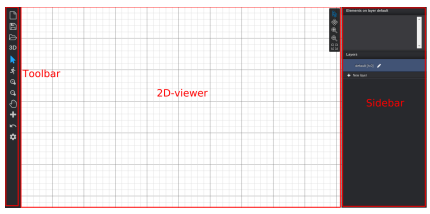

Figure 2. The user interface of the application with the most interesting components: the toolbar (where every button is another component), the sidebar and the 2D viewer

7. https://github.com/facebook/react

In particular, the most interesting components developed by us are the **viewers**, for example the 2D-viewer and the 3D-viewer. This component-based structure, allows us to define whatever state definition (for example in a tabular form) simply adding a new component.

The viewers structure is outlined in figure 3, where we can observe the great blocks. The first one is the *catalog* and as we have already seen in section 3.3 it contains all the building elements with their properties. We also the the application *core*, which manages the state and contains all drawing functionalities. It communicates with the catalog taking the building models properties. At least we have the *viewers*, that are chosen observing the current state mode (see section 3.1). At the moment we implemented the 2D and 3D viewers (see figure 4)

2D Viewer. This viewer creates a 2D view of the building model. Given the state, it exploit the **Virtual DOM** (see [5]) implemented by React.js, to update only changed parts avoiding continuously global updates.

3D Viewer. It exploits the WEBGL-based modeling library **Three.js** [8] to build a 3D view of our building. To avoid global refreshes every time we update a part of the state, it has been implemented a `diff` and *patch* system standardized in [6]. So it has been implemented a data structure which maps the Three.js objects with the building elements inside the state. Every time the user launches a Redux.js action, the application compute the difference between the old state and the new one a redraw only the changed objects. In particular we can have the following *operations*: (i) **add**, (ii) **replace** and (iii) **remove**. For every operation it is determined a behavior based on the changed building element, changing also related building elements.

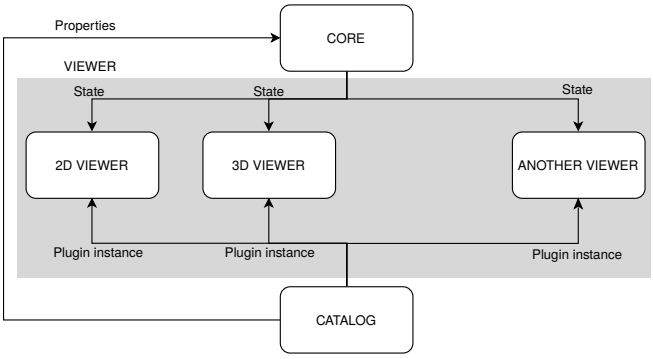

Figure 3. The architectural schema for the viewers. Here we can see that the core can instantiate several different viewers giving them the state for the representation

3.5. Serverless architecture

The project uses a serverless architecture. The application is most executed into the user browser, with follow

8. https://threejs.org/

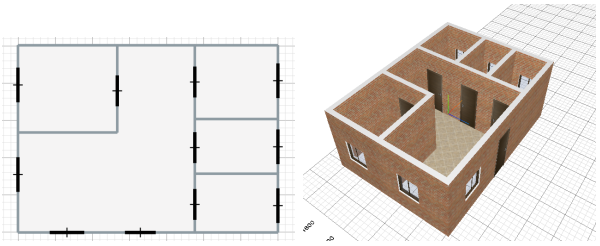

Figure 4. The 2D and 3D viewers for the state

advantages: (i) avoid user hard installation or updating typically performed by not technical users; (ii) offer a good abstraction level that makes the application platform and operation system independent; (iii) allows a centralized new version deploy through a CDN (Content Delivery Network) update; (iv) avoid a point of concentration of the calculation that doesn't leave the user computer.

To allow complex architectural building the system can operate with some FaaS (Function as a Service) worker delegating the CPU bound generation required for complex 3D geometric object. The figure 5 show a schema of the application.

To perform user collaboration the system uses Firebase for synchronization state messaging.

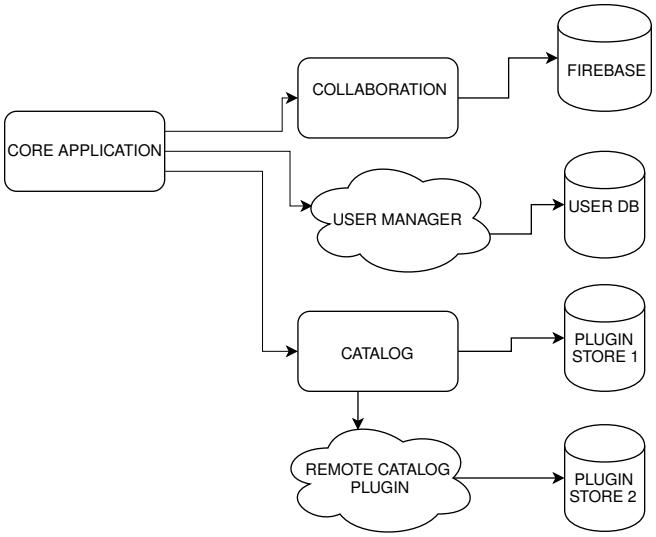

Figure 5. The serverless architecture for our application. We can recognize the collaboration component, based on Firebase. We can also see the FAAS container for the plugin catalog which is linked with a BAAS component for plugins generated on server with other modeling software. We can also manage an optional user login to the system, with another BAAS. Notice that the core application is entirely run on front end side

4. Results

Now we can see the results obtained from the adoption of the techniques we have studied in the previous sections. The uniflow pattern allowed us to identify a consistent and global state which describes the state of the project and the business logic. Moreover, in conjunction with the adoption of a serverless architecture, it led to a slim architecture for the collaboration. Using the uniflow pattern with the Virtual DOM, permitted us to automatically render only changes to the state without complex optimization algorithms, like the one based on differences and patches used for the 3D rendering of the building model. In addition the system is extensible with the addition of geometric elements, in fact the user can register external services to add new ones. In conclusion, use of the Web Components allow us to expose our software in a modular way, improving maintainability.

4.1. Future work

The system could be expanded thanks to the modularity given by the serverless paradigm. As we already have a good user experience, to help the adoption of this application in other modeling contexts, we should improve the catalog adding new building elements

Acknowledgments

The authors would like to thank Stefano..... and Prof....

References

[1] M. Roberts, "Serverless Architectures." [Online]. Available: http://martinfowler.com/articles/serverless.html

[2] F. Spini, E. Marino, M. D'Antimi, E. Carra, and A. Paoluzzi, "Web 3D Indoor Authoring and VR Exploration via Texture Baking Service," in *Proceedings of the 21st International Conference on Web3D Technology*, ser. Web3D '16. New York, NY, USA: ACM, 2016, pp. 151–154. [Online]. Available: http://doi.acm.org/10.1145/2945292.2945309

[3] C. A. Ellis and S. J. Gibbs, "Concurrency Control in Groupware Systems," *SIGMOD Rec.*, vol. 18, no. 2, pp. 399–407, Jun. 1989. [Online]. Available: http://doi.acm.org/10.1145/66926.66963

[4] J. Hopcroft and R. Tarjan, "Algorithm 447: Efficient Algorithms for Graph Manipulation," *Commun. ACM*, vol. 16, no. 6, pp. 372–378, Jun. 1973. [Online]. Available: http://doi.acm.org/10.1145/362248.362272

[5] J. Rotolo, "The Virtual DOM vs The DOM." [Online]. Available: http://revelry.co/the-virtual-dom

[6] P. Bryan and M. Nottingham, "JavaScript Object Notation (JSON) Patch," Internet Requests for Comments, RFC Editor, Tech. Rep. 6902, April 2013.