Computational tools and file format for virtual interactive indoor mapping

Anonymous cvm submission

Paper ID ****

Abstract

This paper introduces FIVE (Framework for Indoor Virtual Environments) and HIJSON (Hierarchical Interactive JSON), respectively a web toolkit for indoor mapping applications and a novel cartographic document format. Client-side FIVE applications, entirely based on web technologies, rely on HIJSON documents produced server-side using LAR, a novel representation scheme for topology and geometry.

An *interactive indoor mapping* environment is a virtual reconstruction of a physical indoor space, where the user may interact with virtual objects, experienced in the actual position they occupy in the real world. Our approach outlines a specialized and evoluted 3D *user interface* giving a glimpse of a section of the real world, that the user can handle intuitively. Furthermore, the virtual indoor environment API provides a platform where many different applications can rely upon. Accessible via web browsers from any kind of device, several applications may coexist on this platform. IoT monitoring, realtime multi-person tracking, and cross-storey user navigation, are already implemented using an automatic search for all valid walkable routes, and taking into account both architectural obstacles and furniture.

The HIJSON format is used to represent any geometry of the indoor space of complex buildings, capturing their hierarchical structure, a complete representation of their topology, and all the objects (either smart or not) contained inside. Such textual representation allows the FIVE framework to offer a web environment in which the user is presented with 2D or 3D models to navigate. With respect to current cartographic formats, HIJSON suggests four major enhancements: (a) exposes a hierarchical structure; (b) uses local metric coordinate systems; (c) may import external geometric models; (d) accepts semantic extensions. The semantic extensions supported by the FIVE architecture encapsulate the details about communication protocols, rendering style, and exchanged and displayed information, allowing the HIJSON format to be extended with any sort of models of objects, sensors or behaviors.

1. Introduction

An *interactive indoor mapping* environment consists of a virtual reconstruction of a physical indoor space, in which the user can move around and interact with virtual objects, that are found in the same position they actually occupy in the real world. Such an interactive indoor mapping can be thought as a specialized and very evoluted *user interface* capable of giving a glimpse of a section of the real world that the user can handle in a natural and intuitive way. Such a reconstructed virtual indoor environment can be considered a general platform where many different applications can rely upon. Both promising and already well explored ICT applications may find in *virtual indoor mapping* the perfect context to be integrated into.

This paper quickly outlines the generation of geometric data of a complex building, to provide both an explicit semantic and a hierarchical model of indoor spaces. LAR, a general representation for geometric and solid modeling is used for this purpose. The generated LAR structures are exported to HIJSON format, extending GEOJSON for indoor mapping and the Internet-of-Things. A convenient way to extend the representation capabilities of IoT *smart objects* is also mentioned as semantic extensions, that affects both document format and the web framework, and can be easily collected in a public repository.

In particular, for environments with massive presence of sensor-equipped (or "smart") objects, which realize the so-called *IoT* (Internet of Things), the interactive indoor mapping represents an ideal integrated interface for IoT monitoring systems. To be specific, it can be the container of indoor navigation systems, giving the user, to be routed across an indoor environment, the opportunity to interact with objects along the suggested paths. Furthermore, in conjunction with the advancements in the field of user indoor location, whose efforts are nowadays focused to realize an integration of positioning systems like GNSS (Global Navigation Satellite system), Wi-Fi, Bluetooth and LTE (Long Term Evolution), to support continuos outdoor/indoor navigation by means of integration of technologies, it represents the most natural interface to perform realtime access monitoring and multi-person tracking.

To enable such an interactive mapping platform it is of

the utmost importance to set up a descriptive representation of the indoor environment. This description belongs to the field of indoor cartography, which as digital evolution of plain floor plans, has arrived to arouse the interest of big players like Google, that has integrated indoor plans of specific locations of interest [11] into Google Maps. In general, it can be considered "of interest" — such to justify and motivate indoor cartographic applications — both public or commercial places of vast dimensions, as for example airports, train stations, shopping malls, and also private buildings subject to strict access protocols, like warehouses, logistic centers, data centers, etc.

Despite of the growing attention regarding indoor cartography, efforts to specify open formats for indoor representation are few and partial, and certainly not intended to support the interactive indoor mapping, which is conversely the main purpose of this paper.

This work, jointly developed by Sogei S.p.A., an ICT company fully owned by Italian Ministry of Economy and Finance, and the CVDLAB (Computational Visual Design Laboratory) of the "Roma Tre" University, is inspired by the necessities of Sogei itself, which runs one of the largest data center of Europe, so requiring very strict access control policies, which include the recording and the real-time interaction with man/machine maintenance scenarios. Support for this interactive framework, where realtime awareness of the maintainer position inside the data center helps to reduce intervention times and to increase safety and security, has been chosen as case study of interactive indoor mapping, based on the proposed indoor cartographic format.

The remainder of this document is organized as follows. In Section 2 we provide an overview of the state of the art in the field of indoor document standards and related applications. Section 3 is devoted to describe the advances introduced by the novel cartographic document proposed, while section 4 presents the document syntax. Section 5 reports about the toolkit specifically developed to handle the new document format. In Section 6 it is depicted the overall architecture and the implementation of the web based application framework, which is in turn used to achieve the objectives stated above. Section 7 presents a case-study application of the document format discussed in this paper. Finally, Section 8 proposes some conclusive remarks and future developments.

2. Related work

Research on the cartographic representation of indoor environments is extensive and heterogeneous with respect to the strategies applied. Different information sources are used, and accuracy of the produced solution depends on the adopted approach. In some cases the information is obtained with automatic or semi-automatic processing of files that describe the architectural structure of a building, such as BIM (Building Information Modeling) [8] and/or IFC (Industry Foundation Classes) that describe a building project [4]. Image processing is also used to extract topological information from floor plan images [9]. In other works, building information and descriptive parameters are redefined from scratch [12]. Such approaches suffer from the non appropriateness of their representative formats: images contain poor information and CAD files are not designed for this kind of use.

A recurring theme among the use of cartographic information is *indoor navigation* [9, 12, 4]. The proposed approaches are very different in this case too, and based on several strategies with some basic elements in common. An often adopted solution is based on the representation of the routing information as a graph, having a node for each room and an edge for each pair of connected rooms. In some cases the edges are weighted in function of Euclidean distance. The detail level of the graph, and hence the effective usefulness of the calculated paths, can vary depending on the technique and the design choices applied, but in general most of the proposed solutions retrieve information only from architectural structure.

A subject related to navigation is the *location of users*. To locate the exact position of a user inside a building, the currently most applied techniques are based on fingerprinting and triangulation of radio signals (Wi-Fi, Bluetooth, LTE, etc.) flanked by more original solutions based, for example, on image recognition [1]. User tracking issue is faced with solutions that range from the clever utilization of inertial tracking sensors embedded in many smartphones [1] to the adoption of ad hoc devices [9].

The actual "de-facto" standard in terms of geospatial data representation is the *GeoJSON* format [10], which can be easily used for any type of geographical annotation. In some cases it has been slightly adapted to be used in indoor environments: it is the case of the *IndoorJSON* format [13].

2.1. The GeoJSON format

GeoJSON is a geospatial data interchange format based on JSON, suitable for a geometrical encoding of various geographic data structures. As opposed to GIS formats, GeoJSON is an open standard. Positions need to be expressed in geographical coordinates (usually WGS84).

GeoJSON supports some geometric primitives, including: `Point`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString`, and `MultiPolygon`. Lists of geometries are represented by a `GeometryCollection`. Geometries with additional properties are `Feature` objects. Lists of `Feature` are represented by a `FeatureCollection`.

A single `Feature` is composed essentially by two mandatory fields: `geometry`, which describes the object

geometry accordingly to the previously recalled primitives, and `properties` which contains additional information about the `Feature`.

In GeoJSON it is possible to define complex shapes through the composition of simpler objects. Mainly due to its simplicity, GeoJSON is widely used and deeply integrated into several applications and services.

2.2. The IndoorJSON format

IndoorJSON is a GeoJSON variant defined and used by *indoor.io*, a Finnish company devoted to indoor environment mapping. IndoorJSON is compliant with GeoJSON syntax, and it may consist of any number of `Features` and/or `FeatureCollections`. All `Features` are interpreted similarly regardless of their grouping into nested `FeatureCollections`. IndoorJSON supports all GeoJSON geometry types.

Some particular properties are used to correctly define the indoor elements:

- `level`: describes which storey contains the feature;
- `geomType`: identifies the category of the object, useful during the visualization process.

There are some additional but not mandatories properties, useful for the indoor representation:

- `accessible`: describes if an element is walkable or not;
- `connector`: defines if the element is a connection between two storeys;
- `direction`: describes the direction of the connection (both ways, only up, only down);

A syntax validator is provided by *indoor.io*, but the commercial nature of this project limits the number of tools available to deal with this format.

3. Advances on cartographic document formats

The focus of this work is the definition of a novel format of cartographic documents along with the software ecosystem rooted on it. A simple but effective algorithm to find indoor valid routes is also provided. HIJSON (Hierarchical Indoor JSON) is the name chosen for the new document format; it the accompanying software framework, aim to realize a mapping of real indoor spaces with a virtual interactive web environment. HIJSON is based upon ideas and design principles collected from previous formats and identifies four critical improvements with respect to them: it exposes a *hierarchical structure*, uses *metric local coordinate system*, may import *external geometric models* and accepts *semantic extensions*.

3.1. Hierarchical structure

The HIJSON format allows for hierarchical description of indoor spaces. The introduction of a hierarchical structure establishes a parent-child relation between entities of the model, reflecting a container-contained relationship. This directly implies a neater representation then the plain linear structure adopted by GeoJSON, being a perfect analogy of objects contained (i.e. placed) into spaces.

Therefore, an organized arrangement of spaces is allowed by HIJSON, via logical (or even physical) grouping: concepts like building wings, sections, storeys, departments, etc. can be directly introduced, in order to reflect into the document structure the actual logical or physical divisions, categories or relationships among the modelled spaces.

Hierarchical structures are common in computer graphics, where they are used as scene graphs. This accordance of underlying structures really simplifies 3D rendering algorithms of HIJSON documented environments. Furthermore, the container-contained relation enables a recurring use of local reference frames.

3.2. Metric local coordinate system

Supported by the hierarchical underlying structure, the HIJSON document format allows the use of local coordinate systems. Hence the shape of all elements can be conveniently modelled using local coordinates, and then placed in the right position with respect to the position of the parent (or container) element applying a rotation, followed by a translation transformation.

Another substantial advantage is represented by the adoption of a metric reference frame, consequently simplifying the compilation of the document, either manually generated or produced by software tools. Just remember that the GeoJSON coordinates are geographical, a pairs of (absolute) latitude and longitude angles, like the ones provided by GNSS systems. This kind of coordinates are certainly not particularly user friendly, when positioning a smart device or a furniture element within a specific building room.

The HIJSON document format is specially designed to guarantee the user to be routed seamlessly from outdoor to indoor and vice versa. Even if indoor geometries are entered in a local metric coordinate system, continuos outdoor/indoor navigation is ensured through the processing pipeline detailed below.

3.3. Hyperlinked geometric models

The HIJSON document may further import external geometric models — either of the buildings themselves or the interior furniture or devices — that are topologically complete (in the sense of solid modeling [15]) and very compact. Such models, coming from a source outside the document, are acquired by hyperlinking JSON files that contain

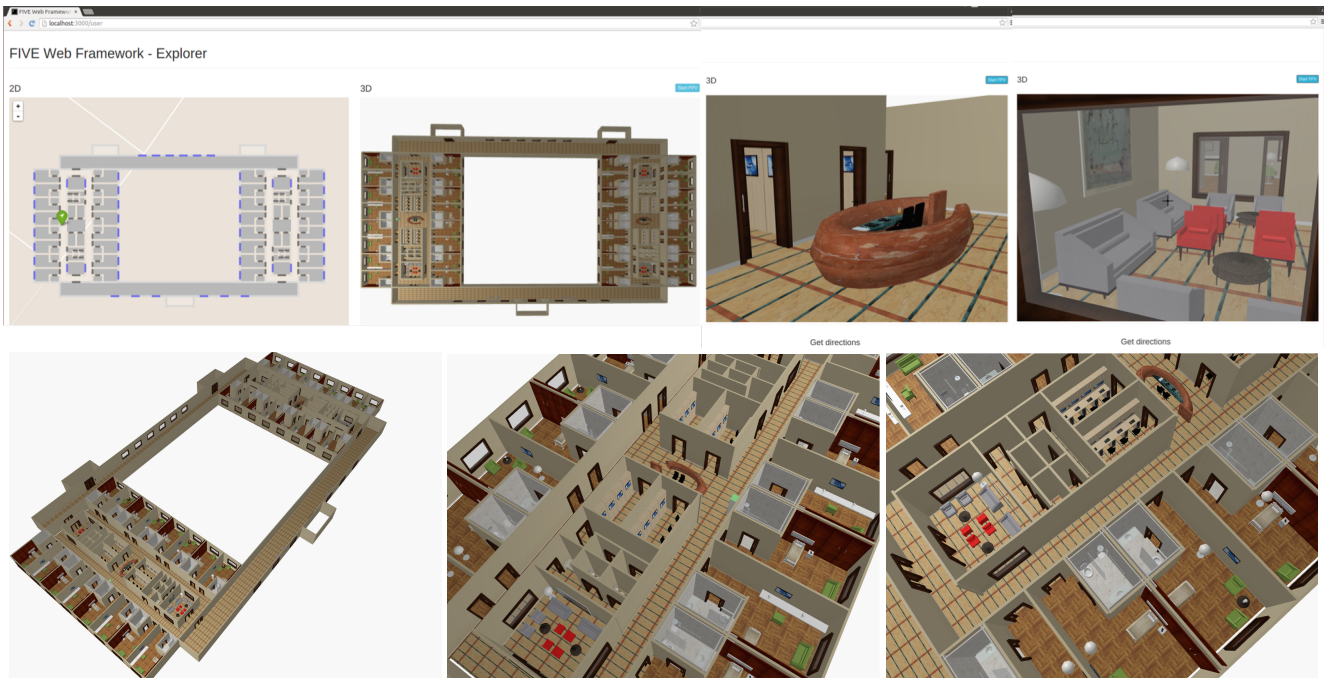


Figure 1: example caption

a Linear Algebraic Representation (LAR) of topology and geometry, to be expanded for visualization or interaction at any useful level of detail.

The LAR scheme [7] is characterised by a very large domain, including architecture, building and construction [14], 2D and 3D engineering meshes, non-manifold geometric and solid models and meshes, and high-resolution 3D images [6]. This scheme uses the set of *Combinatorial Cellular Complexes* (CCC) as mathematical domain [2], and various compressed representations of *sparse matrices* [5] as codomain.

Since LAR provides a complete representation of the topology of the represented space, the matrix $[\partial_d]$ of the boundary operator shall be used to compute the coordinate representation $[c]$ of the *boundary* chain of *any subset* c of cells, though *a single* operation of SpMV multiplication [5] between the `CSR` (Compressed Sparse Row) representation of $[\partial]$ and the `CSC` (Compressed Sparse Column) representation of the $[c]$ chain, resulting in very efficient computations on modern hardware, even mobile.

The expansion of a LAR model, to be considered as a general-purpose graphic primitive, may be executed on either the server or the supervisor client of the HIJSON Web Toolkit architecture (see Section 6.2.1), or even on the *Explorer* client, depending on the size and the locality of the model to be expanded.

3.4. Semantic extensions

Semantic extensions make the HIJSON format extendible and customizable, that is able to adequately respond to any need of objects representation. To define a semantic extension means to allow the HIJSON document to model an object previously not covered, or even to modify the behavior of a comprised one. Semantic extensions are to be defined both as HIJSON format syntax and as HIJSON Toolkit source code. In particular it is necessary to define respectively a new HIJSON Element and a new HIJSON Class, as specified below.

4. HIJSON structure and syntax

The HIJSON document is composed by a configuration section, followed by one or more `FeatureCollections`, containing the actual data.

Listing 1 shows a simplified HIJSON document, devoid of punctual details, to make clear to the reader the overall document structure.

```
{
  "config": {
    // ...
  },
  "data": [
    // ...
    {
      "id": "architecture",
      "type": "FeatureCollection",
```

Figure 2: example caption

```
      "features": [
        // ...
      ]
    },
    {
      "id": "furniture_1",
      "type": "FeatureCollection",
      "features": [
        // ...
      ]
    },
    // ...
  ]
}
```

Listing 1: Example of HIJSON document.

The configuration includes parameters and settings needed for building representation in the form of a JSON Object. One of the core information in this section is defined by the correspondence between three points of the local coordinate system and three points of the real world, expressed in geographical coordinates. This is needed to ensure a seamlessly passage from local to geographical coordinate system and vice versa.

After the configuration part, the document includes a list of `FeatureCollection`. An example of `FeatureCollection` is given in listing 2.

```
{
  "id": "architecture",
  "type": "FeatureCollection",
  "features": [
    // ...
    {
      "type": "Feature",
      "id": "room_0.1",
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [ [0, 0], [11, 0], [11, 19], [0,
19] ]
        ]
      },
      "properties": {
        "class": "room",
        "parent": "level_0",
        "description": "Office of Mr. Smith",
        "tVector": [10, 20, 0],
        "rVector": [0, 0, 90]
      }
    },
    // ...
  ]
}
```

Listing 2: Example of `FeatureCollection`.

Each element of the list is given in the form of a GeoJSON `FeatureCollection`, that contains an arbitrary number of HIJSON Elements. Each `FeatureCollection` imposes a logical relationship that can be used to group together related HIJSON Elements. Since HIJSON Elements adhere to the GeoJ-

SON format, each `FeatureCollection`n results compliant with GeoJSON syntax and then accepted by any GeoJSON validator. As detailed below, the HIJSON format introduces some additional rules that allow the adoption of this format for indoor representation.

4.1. HIJSON Element

Dealing with indoor environments, there are essentially two classes of objects that is necessary to represent. They are (a) architectural elements, like a room, a corridor, a wall, etc. and (b) furnishings, intended in a broad sense, such as to contain both furniture, like a desk or a chair, and/or "smart objects", like an IP-cam or a connected thermostat.

A HIJSON Element defines a GeoJSON compliant syntax to describe both geometry and properties of an object. It represents the atomic component of a HIJSON document. It would be a best practice to group together related JSON Element using `FeatureCollections`: several classification strategies can be applied, for example by grouping the elements by storey or even by room. Alternatively, since the furnishings are more likely to change than the architectural components of a building, these two different kinds of elements can be isolated in different `FeatureCollections`.

The hierarchical structure of the document gives visible form to the capability of HIJSON Elements to have children elements. A unique ID is mandatory for every HIJSON Element.

Three Geometry types can be used here: `Point`, `LineString` and `Polygon`. The choice of the Geometry type to be associated to a HIJSON Element implicitly defines the category of the element: `Point` is used for furnishings, `LineString` for walls and doors, while `Polygon` may describe levels and rooms.

The Geometry coordinates are expressed in metres, by convention starting at the bottom-left corner of the element, whose position is used to set-up the origin of a local coordinate frame. Unlike GeoJSON, where all properties are optional, in HIJSON some strict requirements are imposed, and some attributes are mandatories:

- `class`: represents the element category, used to instantiate the appropriate *HIJSON Class*;
- `parent`: contains the ID of the parent of the element;
- `tVector`: represents the translation relative to the parent element, expressed in metres;
- `rVector`: represents the rotation relative to the parent element, expressed in nonagesimal degrees.

Specific classes may require the mandatory presence of other properties. For example, the classes `internal_wall` and `external_wall` that define the internal partitions and the external envelope, respectively, require a `connections` array, containing the IDs of the adjacent elements. This information is used by the connector children of the element (e.g. doors) to identify the areas linked together.

Given the nature of the GeoJSON format from which HIJSON derives, the elements are represented by their 2D shape, like on a planimetry. The property `height` was introduced to assign a value to the height of the object, intended as a third dimension.

A `description` property can provide further information about the element. Arbitrary optional fields can be added without restrictions, in order to enrich and extend the expressivity of the representation, or simply for the sake of documentation.

5. HIJSON Toolkit

The HIJSON Toolkit is a software module that implements common operations and transformations on HIJSON documents. Written in *JavaScript* language, this software module has been built to be deployed in the web environment. It is *modular* and entirely *isomorphic*, i.e. can run on the server as well as on every client. Working in the web environment, the Toolkit benefits of the "fertility" commonly concerning the software development in this field: for example, it takes advantage of libraries and frameworks such as *React*, "the JavaScript library for building user interfaces" by Facebook, and as *Three.js*, the current de-facto standard to deal with *WebGL* technologies.

The Toolkit executes the instantiation and extension logic of a HIJSON document, and provides a multistage transformation pipeline that, according to the requirements, can be used either entirely or only in part.

5.1. Processing pipeline

The HIJSON processing pipeline implements the sequence of preliminary transformations that have to be applied to a HIJSON document before any further operation. It is not strictly required to complete each stage of the pipeline: the exit stage depends on the specific use case.

The application of the transformation pipeline has a double aim. The first one consists in generating the graph of valid paths among all the interesting elements. The second objective is the generation of one *GeoJSON* document for each storey of the building described by the HIJSON document. In this way a bidimensional layout can be provided for every level of the building, and visualized through any compliant GeoJSON viewer.

The HIJSON processing pipeline is composed by six elaboration stages, denoted as *validation*, *georeferencing*, *parsing*, *graph paths generation*, *2D layers generation*, *marshalling*. The pipeline of transformations and the output of each stage are shown in Figure 3.

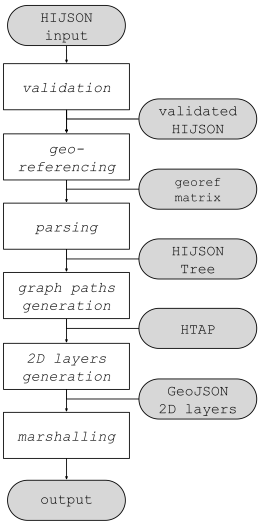


Figure 3: HIJSON processing pipeline

1. `validation` - The first one is the validation stage. In order to begin with the effective transformations the input HIJSON document must be compliant with both the syntax format and the structural requirements. In the case the validation stage fails, processing aborts and does not continue to following stages; intead if this stage successes, then the output for the next stage is a validated HIJSON.
2. `georeferencing` - In the second stage, in order to allow for continuous outdoor/indoor navigation, the system needs to compute the georeferencing matrix, a linear operator able to transform local coordinates into global coordinates (world coordinate system — latitude and longitude angles) and vice versa. This task is accomplished by solving a linear system obtained from information contained in HIJSON configuration part and precisely from the correspondence of three real world points to three points included into the HIJSON document.
3. `parsing` - The parsing stage takes the validated and georeferenced HIJSON as its input, that as illustrated before can be thought of as a list of HIJSON Elements, parses them and produces an instance of the HIJSON Tree. The HIJSON Tree is an object in memory representing the hierarchical structure of the building described by the HIJSON document.
4. `graph of paths generation` - The fourth stage is in charge of the generation of the graph of paths. The algorithm to achieve such a goal is introduced in Section 5.1.1. The graph of paths can be used to compute valid directions between pairs of points of interest inside the building model. Once the graph of paths has been computed, the input HIJSON Tree is augmented with paths information, becoming what has been called an HTAP (HIJSON Tree Augmented with Paths). Augmentation always takes place in the form of an addition of leaf nodes as children of a specific element (e.g. “room”).
5. `2D layers generation` - The fifth stage concerns the generation of GeoJSON *layers*. For each storey of the building, the Toolkit generates a GeoJSON layer that can be used for the creation of a 2D map. Each layer contains only the children of a ‘level’ node of the HIJSON Tree. The presence of a specific element inside the layer can be finely tuned by means of a Boolean value. The geographical coordinates of every elements are calculated by a series of multiplications between transformation matrices obtained during the tree traversal to the local coordinates.
6. `marshalling` - The last stage is responsible for executing a serialization of the the transformed data. This stage, in which are performed tasks like breaking dependency-loops and stringification, is mainly useful server-side, as the output is there stored ready to be served to any requiring client.

5.1.1 Automatic generation of valid paths

The fourth stage of the processing pipeline is responsible for the generation of a graph of valid paths through the entire model represented by the input HIJSON document. The graph generated according to the algorithm described in the following, although non optimal, ensures a complete coverage of the surface while limiting the number of generated nodes. The resulting graph is weighted on the edges with nodes distances. Each graph node may represent either:

a. a *standard path node*, i.e. a junction node or possibly an endpoint of a path;
b. a *connection node*, used as subproblem composing element in the divide et impera approach adopted;
c. an *element node* i.e. HIJSON Element (whose HIJSON Class explicitly grants his presence in the graph), typically an endpoint of a path.

The graph of paths allows for calculations of directions between any two given nodes. Although different approaches have been explored [3], a very classical solution has been selected in this case, so directions are actually computed client-side by applying the Dijkstra algorithm to the graph.

Taking advantage of the hierarchical structure of the HIJSON document, and according to the divide et impera approach, the problem of paths generation is split in several sub-problems, which consist in the computation of the sub-graphs relative to each individual space, more generally a single room. The sub-graphs are then linked together

through the connection nodes (which in most cases represent doors). The resolution of each sub-problem (as depicted in Figure 4), is composed by four steps, as detailed below.

1. Computation of the walkable area of the space: this task is accomplished by subtracting the shape of the obstacles from the area of the space; the result is typically a surface with holes.
2. Triangulation of the walkable area: the computed surface is triangulated taking into account the presence of holes.
3. Identification of graph nodes: for each triangle side completely internal to the area, its midpoint is selected as standard path node.
4. Junction of nodes: nodes relative to the same triangle are then linked pairwise; both element nodes and connection nodes (i.e. doors) are linked to the nearest node of the space (i.e. room).

5.2. HIJSON Class definition

To make better use of the possibilities offered by the HIJSON Toolkit and by the HIJSON document format, some custom dynamic behaviors can be described. These behaviors encapsulate the specificities relative to communication protocols with the sensors, as well as to features of user interaction. The interface for such behavior is the HIJSON Class.

Figure 5: HIJSON Element/Class/Node relashionship

Every HIJSON Element of the input HIJSON document has a dynamic counterpart, a running instance called *HIJSON Node*, instantiated according to the corresponding HIJSON Class via reflection methods (see Figure 5).

To specify a new *HIJSON Class* means to extend the Toolkit to deal with a new class of HIJSON Element. To extend the toolkit in order to deal with a new class of HIJSON Element is required to specify a new HIJSON Class, by defining the following properties and methods:

- `in_graph`: a Boolean value to express if the element is an approachable point in the graph of paths;
- `in_2D_map`: a Boolean value to express if the element must be shown in the 2D map;
- `get2DStyle()`: a method that returns the 2D map appearance of the element, essentially HTML and CSS code;
- `get3DModel()`: a method that returns the 3D model appearance of the element, i.e. an instance of `Object3D` of the *THREE.js* framework;
- `getWidget()`: a method that returns the information widget, a *React* component;
- `getProxy()`: a method that returns the server-side proxy which encapsulate the IoT sensor communication protocol, i.e. a *Node.js* module.

User's needs for new indoor elements, greatly different sensor equipments, alternative representations of 2D or 3D viewports are accepted by the definition of new HIJSON Classes, that so provide single-point custom extensions of the Toolkit capabilities.

6. HJSON Web Framework

The HIJSON Web Framework responds to the needs of an extendable, customizable, and scalable web framework which provides at the same time IoT monitoring, realtime multi-person tracking and cross-storey user navigation.

Expandability and customizability derive from both design choices and HIJSON inherent characteristics, i.e. the possibility of semantic extensions. Scalability is directly borrowed from technologies used for software development: *JavaScript* language, using *Node.js*, in particular *Express.js* as backend framework, exploiting the power of WebSocket protocol through the *Socket.io* library.

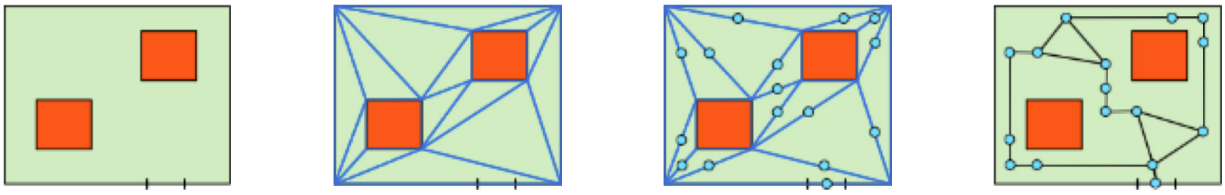
Being supported by the *web-as-a-platform*, the framework exposes also an high availability: it is so simple to use as to visit a website, both from desktop or mobile devices, without explicit requirements to install any software package from proprietary stores—access to which is often denied from business devices.

The HIJSON Web Framework deeply relies on HIJSON Toolkit and offers an all-inclusive client/server architecture with a convenient and highly interactive user interface, leaving aside the specific indoor positioning system and the IoT sensors to deal with. A robust application interface is provided and described in the following section.

6.1. Applications

The Framework has been designed with focus on two different kind of users: the *Explorer* and the *Supervisor*. They have different requirements and are likely equipped with different devices: while the *Supervisor* monitors the indoor environment through a desktop workstation, the *Explorer* has a smartphone available and needs to be routed across the building.

In both cases, the web platform ensures a perfect alignment with the BYOD (Bring Your Own Device) approach, nowadays often supported by companies that encourage employees to use personal devices.



(a) (b) (c) (d)

Figure 4: graph of paths generation: (a) detection of obstacles and computation of walkable area; (b) triangulation of walkable area; (c) identification of graph nodes; (d) junction of nodes.

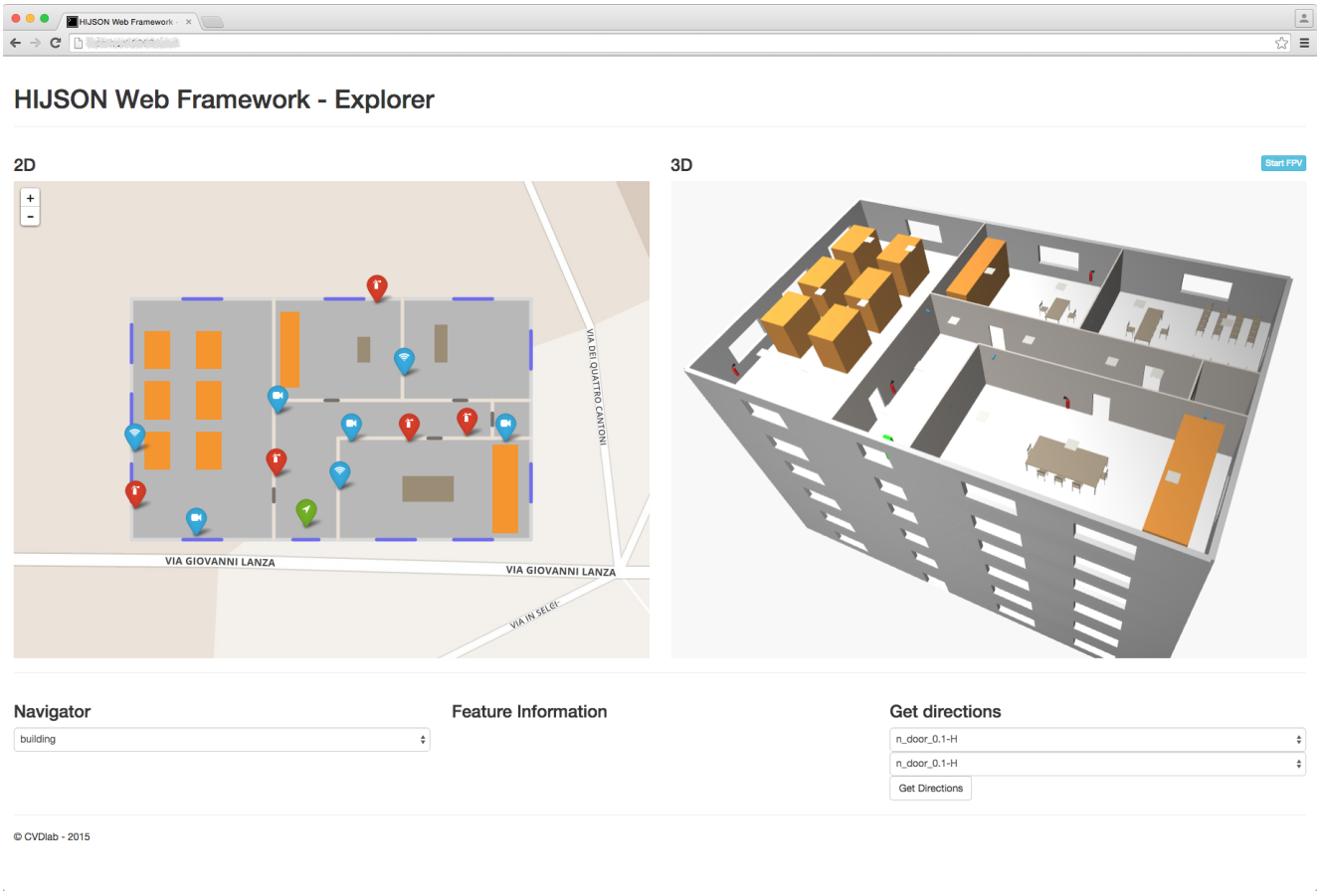


Figure 6: HIJSON Web Framework UI

6.1.1 IoT monitoring

An *IoT monitoring application* consists of an interface showing to the user, in a single, integrated and centralized way, the information collected from all the smart objects modelled in the HIJSON document. IoT monitoring application provides bidirectional communication, since the interface let the user receive information coming from smart objects while allowing him to send commands to them.

As the name itself may suggest, it is an activity specifically performed by a *Supervisor* user, but it can be also suitable to be deployed for the *Explorer* user, since she can take advantage of the interactive information coming from the surroundings objects while she moves across the indoor environment.

Monitoring different smart objects may require different ways to visualize and/or send data and commands. Modularity and extendibility of the application respond superbly to these requirements, by providing for each class of ob-

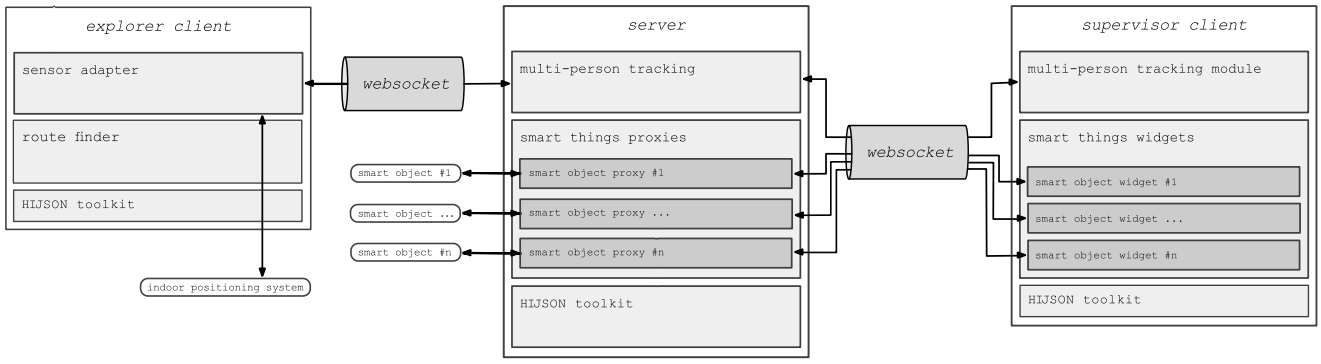


Figure 7: HIJSON Web Toolkit architecture

jects a different interface of visualization and interaction, as a result of the polymorphism principles introduced by the HIJSON Class. In particular, the user interface is characterized by a dual-display mode, that allows the user to see at the same time a 2D map that gives an overall glance in a simplified plan, and a 3D virtual environment to navigate into, as shown in Figure 6.

Alongside with typical smart objects, suitable to deal with like thermostats, where the user can read the room temperature and turn the heating on/off, other kinds of objects, that are not properly considered "smart", can be integrated into the HIJSON environment. It is the case, for example, of fire extinguishers, that are able to show the date of their last check, stored in a database.

6.1.2 Realtime multi-person tracking

Realtime multi-person tracking allows a *Supervisor* to monitor the current real position of people inside the building. This kind of task can be useful for several reasons, including security, logistics or to supervise composite operative workflows. Each device equipped with the *Explorer* application is in charge of locating itself, interacting with the indoor positioning system, and notifying the current position in continuos mode. Evidence of the people position is given to the *Supervisor* both into a 2D map and an immersive 3D virtual environment (see Figure 6).

6.1.3 Cross-storey user navigation

The HIJSON Framework also provides the capability to give directions to *Explorer* users that must move across the indoor environment. The user specifies a starting and an ending point and the system provides him with a valid connection path. This feature strongly rely on the graph of paths generated by the Toolkit, so starting and ending points must be nodes of the graph. *Connection nodes* are introduced to represent stairs or elevators, enabling cross-storey paths to be computed. Since paths can span more than one storey, the most effective way to display them to the user is to show the connection nodes visualized in one or more 2D maps.

6.2. Architecture

Like the vast majority of the web based applications, the Framework exposes an overall architecture that is inherently *client/server*. In particular, two different types of possible clients are identifiable, one for each different kind of users: the *Supervisor* client and the *Explorer* client. Both of them connect to the same server.

The indoor space described by the input HIJSON document is processed by the server via the processing pipeline. After that, any connecting *Explorer* client, presumably via a mobile device, will be provided with the information to perform cross-storey navigation of the building, while reporting the user position to the server. The server will feed any connecting *Supervisor* client with users positions, along with data from sensor-equipped objects present in the environment, achieving both IoT monitoring and realtime multi-person tracking.

6.2.1 Server Architecture

An architectural scheme of the framework is provided in Figure 7. A web server module is responsible for listening to connecting clients. Each client connection is handled by the web server module providing all the required resources and then by opening a WebSocket channel, in order to have both *Explorer* and/or *Supervisor* communication protocol data flow within. In particular, the `multi-person tracking` module receives position data from *Explorer* clients. It aggregates and sends these information to connected *Supervisor* clients through the WebSocket channel, using a simple but reliable protocol described later. Independence from particular IoT sensor equipment communication protocols is achieved introducing a `smart object proxy` module. This one is defined in the HI-

JSON Class and is obtained via the `getProxy()` method for each smart object modelled as previously described.

6.2.2 Explorer client architecture

The *Explorer* client architecture is generally deployed on a mobile device, which is usually supplied to a user who needs to be routed across the environment described by the HIJSON document. The `sensor adapter` module encapsulates the communication logic with the indoor positioning system. The presence of this module ensures independence from particular technologies, so allowing the *Explorer* client to rely on different indoor positioning systems (Wi-Fi, Bluetooth, LTE, etc.).

Every time a `sensor adapter` observes a perceptible modification in user position, it sends the new position information to the server through the single opened WebSocket, spawning a simple message with the syntax described in listing 3.

```
currentPosition = {
  coordinates: [x, y],
  levelId: level-ID
}
```

Listing 3: Example of message sent by the *Explorer* client to the server.

Relevant information includes, beside current coordinates, the indication of the storey of the possibly multilevel building the user is in.

It is to remark that, when not outflanked by the introduction of an external server, the problem of communication between positioning system and the low level APIs of the browser is left to positioning system itself or to whom is in charge of specific deployments of the HIJSON Web Framework. The `smart object widget` module, being in common with the *Supervisor* client, will be discussed in the next section.

6.2.3 Supervisor client architecture

The architecture of *Supervisor* client includes two modules. The first one, named `multi-person tracking` module, is responsible to receive through the WebSocket, from the server information about *explorers* of the environment, showing them in the user interface. The second module, named `smart object widget`, communicates with the server to propose the user realtime information about sensor-equipped objects in the environment. Data passes through the single WebSocket opened between the server and every *Supervisor* client. Relying on a naive but effective communication protocol, each `smart object widget` exchanges data only with its corresponding `smart object proxy` on the server. To ensure the data are posted only when the user requires the information relative to a specific smart object, a *widget lifecycle protocol* is implemented. This one is based on 4 event triggers `on_before_show`, `on_show`, `on_before_hide`, `on_hide`, as suggested by their names. When the user requires information about a smart object, its widget has to be rendered, and `on_before_show` the server is notified to connect via relative proxy to the sensor. Once connected, the server begin to send data via WebSocket. Received data are shown through the widget to the user. When done, the `on_before_hide` event of the widget is triggered, a notification is sent to the server announcing to stop sending data, and the proxy closes the connection to the sensor. Widget lifecycle protocol ensures that only required data are sent from the server to the client.

7. Case study

As case study of the discussed approach, we have taken into account the need of Sogei S.p.A. to support its maintenance service workflow, since its data center, one of biggest data centers in Europe, is subject to very strict access control policies.

The overall state of the data center can be monitored through the *Supervisor* client. In this case the considered smart objects belong to a range of different devices, going from webcams, that provide on request the captured video streams, by way of alarm and antifire systems, till to individual servers, that can be monitored along several dimensions, including operating temperature, workload, etc.

The most common maintenance scenario consists of an intervention by a technician that have to move across the environment, and locate within a huge data center the machine on which to operate, a not trivial task due to the presence of thousands of similar-looking machine racks. Thus the operator will be equipped with an *Explorer* client, which will drive him to the target machine on which operate, while continuously notifying to a security *Supervisor* his position, obtained by interacting with the indoor positioning system. The maintenance workflow supervisor, using the *Supervisor* client, is able to monitor the operator position within the data center, verifying that he does not deviate on unauthorized paths, triggering some console alarm if this happens.

The purpose is to support the process of those in charge of carrying out the “ticket-maintenance” activities, as quickly as possible and without error. The “maintenance man” will be guided to the right sub-system among thousands of racks. The real-time awareness of the relative positions between the “maintenance man” and the rack — containing the sub-system — will help to reduce intervention times and to increase safety. By knowing when the maintenance process starts, the system can automatically move, in real time, services, which are hosted on virtual machines, to other systems, thus maintaining the continuity of services

and, at the same time, reducing the global risk factor. When the "ticket-maintenance" is over, and the technician goes away, it is possible to immediately restore the pre-existing conditions of services after a complete test has been performed.

8. Conclusions

In this paper a novel document format, named HIJSON, for indoor cartographical descriptions has been introduced. Utilization of local metric coordinate system, avoiding the manipulation of global geographical coordinates, really inconvenient when dealing with indoor spaces and objects, greatly enhances the modeling and rendering of the document content. Currently, we produce the HIJSON document from a python script using two libraries for geometric computing (`pyplasm` and `larcc` [7, 14, 6]). The modeling process can be further improved by implementing a LAR-based graphical editor to assist the user during the description of the indoor space. The realization of such an editor is already in our plans.

The HIJSON format focuses on a hierarchical representation of the indoor spaces that allows for completely capturing their topology. On the basis of this representation a virtual web environment can be rebuilt working as a unifying platform to run a bunch of different applications. The reference architecture of such a platform has been also implemented and described in this work.

The architecture supports a whole range of applications: IoT monitoring, realtime multi-person tracking and user cross-storey navigation are already implemented and described. A very convenient way to extend the representation capabilities of smart objects is also mentioned as semantic extensions. These extensions, which affects both document format and its web framework, might be easily collected in a public repository. Community could both use public available extensions or contribute by mapping new (smart) objects inside the HIJSON document format.

References

[1] B. Al Delail, L. Weruaga, M. Zemerly, and J. Ng. Indoor localization and navigation using smartphones augmented reality and inertial tracking. In *Electronics, Circuits, and Systems (ICECS), 2013 IEEE 20th International Conference on*, pages 929–932, Dec 2013. 2

[2] T. Basak. Combinatorial cell complexes and Poincaré duality. *Geometriae Dedicata*, 147(1):357–387, 2010. 4

[3] W. Bian, Y. Guo, and Q. Qiu. Research on personalized indoor routing algorithm. In *Distributed Computing and Applications to Business, Engineering and Science (DCABES), 2014 13th International Symposium on*, pages 275–277, Nov 2014. 7

[4] M. Boysen, C. De Haas, H. Lu, X. Xie, and A. Pilvinyte. Constructing indoor navigation systems from digital building information. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 1194–1197, March 2014. 2

[5] A. Buluç and J. R. Gilbert. Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments. *SIAM Journal of Scientific Computing (SISC)*, 34(4):170 – 191, 2012. 4

[6] A. DiCarlo, M. Jirik, and A. Paoluzzi. Cad models from medical images using the linear algebraic representation. In *CAD'15*, London, UK, June 22-25 2015. Accepted for publication in Computer-Aided Design and Applications, Taylor & Francis. 4, 12

[7] A. Dicarlo, A. Paoluzzi, and V. Shapiro. 4, 12

[8] C. Eastman, P. Teicholz, R. Sacks, and K. Liston. *BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors*. Wiley Publishing, 2008. 2

[9] L. Faramondi, F. Inderst, S. Panzieri, and F. Pascucci. Hybrid map building for personal indoor navigation systems. In *Advanced Intelligent Mechatronics (AIM), 2014 IEEE/ASME International Conference on*, pages 646–651, July 2014. 2

[10] GeoJSON contributors. Geojson. `http://geojson.org/`, 2015. Accessed: 2015-03-23. 2

[11] Google, Inc. Go inside with Indoor Maps. `https://www.google.com/maps/about/partners/indoormaps`, 2014. Accessed: 2015-03-23. 2

[12] D. Gotlib, M. Gnat, and J. Marciniak. The research on cartographical indoor presentation and indoor route modeling for navigation applications. In *Indoor Positioning and Indoor Navigation (IPIN), 2012 International Conference on*, pages 1–7, Nov 2012. 2

[13] indoor.io. Indoorjson specification and validator. `https://github.com/asaarinen/indoor-json`, 2013. Accessed: 2015-03-23. 2

[14] A. Paoluzzi, E. Marino, and F. Spini. LAR-ABC, a representation of architectural geometry: From concept of spaces, to design of building fabric, to construction simulation. In Ph.Block, J.Knippers, W.Wang, and N.Mitra, editors, *Advances in Architectural Geometry*, LNCS (Lecture Notes in Computer Science). Springer, 2014. To appear. 4, 12

[15] A. G. Requicha. Representations for rigid solids: Theory, methods, and systems. *ACM Comput. Surv.*, 12(4):437–464, Dec. 1980. 3

A. Appendix

A.1. Short summary of the LAR scheme

```
V = [[5.,0.],[7.,1.],[9.,0.],[13.,2.],[15.,4.],[17.,8.],[14.,9.],[13.,10.],[11.,11.],[9.,10.],[5.,9.],[7.,
9.],[3.,8.],[0.,6.],[2.,3.],[2.,1.],[8.,3.],[10.,2.],[13.,4.],[14.,6.],[13.,7.],[12.,10.],[11.,9.],[9.,7.],
[7.,7.],[4.,7.],[2.,6.],[3.,5.],[4.,2.],[6.,3.],[11.,4.],[12.,6.],[12.,7.],[10.,5.],[8.,5.],[7.,6.],[5.,5.]]

FV = [[0,1,16,28,29],[0,15,28],[1,2,17],[1,16,17,33],[2,3,17],[3,4,18,19],[3,17,18,30],[4,5,19],[5,6,19],
[6,7,20,21,22,32],[6,19,20],[7,8,21],[8,9,21,22],[9,11,23,24],[9,22,23],[10,11,24,25],[10,12,25],[12,13,25,
26],[13,14,27],[13,26,27],[14,15,28],[14,27,28,29,36],[16,29,34],[16,33,34],[17,30,33],[18,19,31],[18,30,
31],[19,20,31,32],[22,23,32,33],[23,24,34,35],[23,33,34],[24,25,27,36],[24,35,36],[25,26,27],[29,34,35],[29,
35,36],[30,31,32,33]]
```

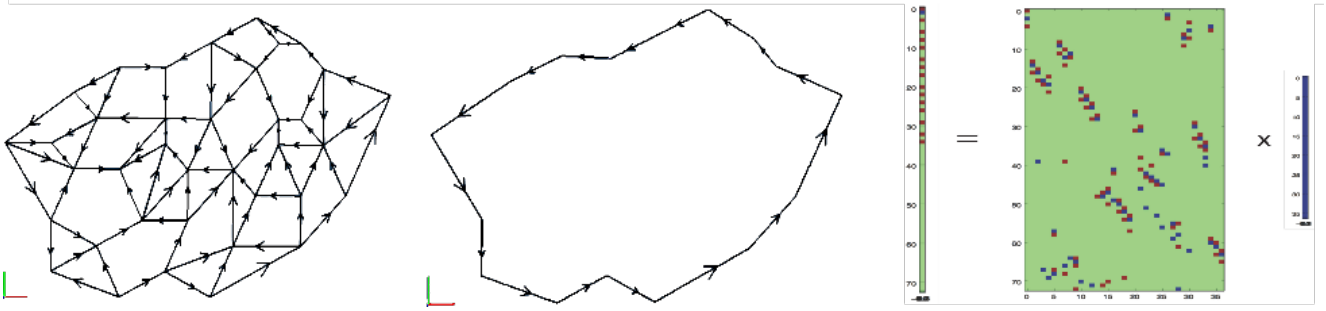


Figure 8: A toy example of the LAR scheme: (a) the bare minimum of data with *complete* information about topology; (b) the extracted boundary; (c) the extraction method $[e] = [\partial][f]$ giving the coordinate representation (in the discrete basis of the 1-cells) of the boundary edges $[e]$ by product of the sparse boundary operator matrix $[\partial]$ times the coordinate representation $[f]$ of the 2-cells (faces), in the discrete basis of the 2-cells.