# Modeling Semantics for Building Deconstruction

X. Xxxxxx[2], X. Xxxxx[2], X. Xxxxxxxx[1], X. Xxxxxxx[1], X. Xxxxxx[1], X. Xxxxxxx[3], X. Xxxxxxxxx[3]
[1]*Department of Mathematics and Physics, Xxxx Xxx University, Xxxx, Xxxxx*
[2]*Department of Engineering, Xxxx Xxx University, Xxxx, Xxxxx*
[3]*XXXXXX X.x.X., Xxxx, Xxxxx*
*{xxxxxx, xxxxxxxx, xxxxxx, xxxxx, xxxxxx}@xxx.xxxxxxxx.xx, {xxxxxxxx, xxxxxxxxxx}@xxxxxx.xx*

Keywords: Building modeling, BIM, Deconstruction Semantics

Abstract: In this paper we discuss the motivation, the technology, the design and the use-model of a novel web service for quantity surveyors, aiming to exploit virtual and augmented reality methods to implement a “zero waste” model, i.e. a new design paradigm where the waste materials from demolition become resources for reconstruction. The goal of this project is to provide virtual/augmented reality tools through quick modeling of buildings and their fast augmentation with semantic content.

## 1 INTRODUCTION

A tendency to minimize the humanization of new territories and to push for reusing already built accommodation or accommodation which has fallen into disuse has become a pressing need in advanced societies. We have to integrate the “zero energy” model (each building has to produce the same amount of energy that it consumes) with the “zero waste” model , i.e. a new design paradigm where the waste materials from demolition become resources for reconstruction (Altamura, 2012). Building, contract, and design processes need to be renewed to take account of environmental concerns.

To reduce the impact of construction projects on the environment, the design needs to take the issue of building materials into consideration. Public administrations need suitable tools for the calculation and the control of reused or disposed materials. The new tools should handle the digital processing of materials throughout the project life cycle, supporting new project requirements such as: Design for Deconstruction, Design for Recycling and Design for Waste.

In particular, a building life cycle, underpinned by a construction process which envisages cycles aligned to natural phenomena is the focus of this paper.

In this work[1] we propose solutions that serve to close the circle of the building life-cycle, moving away from a traditional linear response with excessively high consumption energy rates (cradle to grave) and towards the reuse of materials in deconstruction/reconstruction (cradle to cradle), supported by computer aided selective demolition process.

All restructuring cycles of buildings should envisage de-construction and re-construction steps, targeted towards the replacement of materials in order to achieve greater efficiency. The handling of these materials requires appropriate encoding both for the disposal (ECW – European Coding of Waste) and for the planning and design of new buildings (BIM – Building Information Modeling). For this purpose we need geo-referenced scenes of augmented reality based on fast, easily navigable and measurable 3D models.

We already have excellent knowledge about construction costs (from scratch) but little is known about replacement rates (complete selective demolition). A modern selective demolition process requires human intervention, with high insurance costs due to the danger involved for those working in these activities. This latter point demands an alternative to human effort in these process. We suggest that automated robots could replace human effort; drones could operate in a semantically familiar context and give real-time updates as the reality contextually changes.

We believe, therefore, that there is a big need for modern and easy-to-use modeling frameworks for building deconstruction in the AEC industry, to enable an augmented reality through semantic recognition by computer vision and by photogrammet-

[1]Partially supported by grant from XXXXXX X.x.X., a web service company owned by Xxxxx X.x.X. and XXXxXX — the ICT company of the Xxxxxx Xxxxxxx of Xxxxxxx and Xxxxxxx and the Xxxxxxx Xxxxxxxx Xxxxx of Xxxxxxxx Xxxxxxxxx, respectively.

ric precision up to centimetric definition. Such virtual/augmented reality tools require both fast 3D building modeling and augmentation with semantic content, in order to be controlled in almost real time: this real challenge is also required by the future development of IoT.

In this section we have discussed the motivation of the project described in this paper. The remaining sections are organized as follows. Section 2 introduces a more technical viewpoint about the state of deconstruction topics in Europe and in Italy. Section 3 describes the client application and the proposed workflow for quantity surveyors. Section 4 illustrates the framework architecture. Section 5 shortly recalls the methodology, programming style and computational environment of our geometric programming approach to solid modeling. In the conclusion section we outline the work to be done and provide our forecast about possible developments.

## 2 DESIGN TO DECONSTRUCT

Waste management is an issue that in recent decades has become increasingly important, considering its economic, environmental and energy impact.

### 2.1 Regulatory framework

The European Community has defined standards (EU 98/2008, 1357/2014) and goals (qualitative and quantitative) that Member States must comply with through the enactment of national regulations and the definition of economic instruments.

Waste disposal, with direct and indirect costs involved, is enforced by Construction and Demolition activities. In European Waste Code/Classification (ECW) waste from the demolition of buildings [955/2014 EU] are identified as class 17 – Construction And Demolition Wastes. This classification allows a correct identification of both the waste generated from adoptable demolition modes, and the waste produced by actions of re-use, recycling or landfilling.

The regulatory framework for waste management is far from clear, in particular in the Italian national context. Despite its direct and indirect cost, the landfill is often preferred rather than risking administrative or criminal penalties for failure to comply with unclear rules.

### 2.2 Proposed approach

Given this cumbersome regulatory framework, our project is promoting the use of simplified IT tools to support the deconstruction. In particular, a quick simplified geometric modeling of the building allows for integration of a semantic description of component parts and their materials. Virtual/Augmented Reality strongly helps overcome the administrative difficulties, provided the correct identification of the waste produced. This approach will increase the adoption of virtuous actions, namely the recovery and reuse.

In particular, a geometric modeling of the building allows to identify: (a) cost / income resulting from alternatives of recycling / re-using instead of disposal; (b) the composition and integration of information useful to the planning of construction activities; (c) achievement of the thresholds of reuse / recovery required by the regulations; (d) ability to economically compare different options.

We started by considering the SMARTWaste system (Hurley, 2002). Their approach allows to derive estimates of the quantities of materials by providing a description of the type of building and the area where it was built. With this information, the forms that provide an aggregated representation of the data of interest are automatically filled.

Our approach to deconstruction conversely provides both a geometric modeling of building subsystems and components and a semantic annotation with construction materials, like a sort of *simplified* BIM. As a matter of fact, our national construction industry is strongly heterogeneous, so that we need a pretty detailed modeling to obtain enough accurate information. One vantage point of this approach is an incremental iterative character, where each modeling stage may be followed by validation of partial costs.

A large corpus of specialized literature exists about BIM for existing buildings. A very interesting review paper, discussing hundreds of references is (Volk et al., 2014). Its abstract states that: “While BIM processes are established for new buildings, the majority of existing buildings is not maintained, refurbished or deconstructed with BIM yet. Promising benefits of efficient resource management motivate research to overcome uncertainties of building condition and deficient documentation prevalent in existing buildings.”

The *Metior* (from Latin: to *measure* or *estimate*) project, introduced in this paper, is exactly aiming to overcome such difficulties, via (a) the design and implementation of a *web service* providing a strongly simplified user-interface, designed for quantity surveyors (b) storing a growing database of template plugins for more geometrically complex building parts; (c) using an extensible geometry engine and server based on decades of research; (d) offering flexible semantic additions via specialization of EFC classes associated to building subsystems and parts.

Metior specifically targets quantity surveyors. In fact, although large sites to be deconstructed are operated by main contractors, where skills and specific tools might be widely available and already used, most of deconstruction activities, and hence the largest amount of waste material produced, are managed by quantity surveyors from small or medium companies — or even by single professionals. Such kind of firms may need to be supported with tools where the complexity, both bureaucratic and technical, have to be hidden although correctly managed.

## 3 DECONSTRUCTION APP

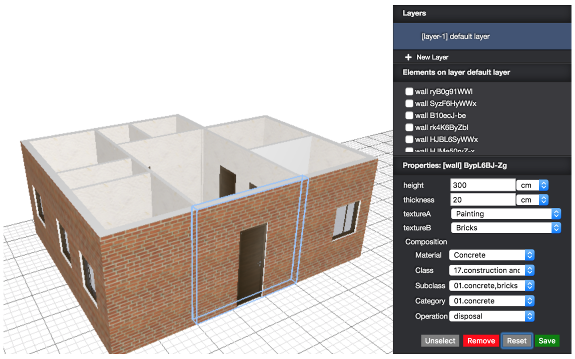
A deconstruction oriented to the maximum reuse of materials must be supported by a *workflow* that guides the user towards estimating the costs of the process. The aim is the determination of *building demolition costs*, depending also on transportation and disposal of materials.

### 3.1 Workflow

**Project definition** The first phase of the modeling process is a gross description of the building, in order to provide basic clues for a correct attribution of the semantic attributes. In particular, we ask for: the apparent age, the style of construction, the historic use register, and the geolocation. The estimated age and the style of construction are used to determine the needed data about the used materials; the record of use destinations allows you to to reconstruct the hazard notes of the items to be disposed of; geolocation finally allows to find out the recycling facilities closest to the site.

**Building modeling** The user describes the construction using some predefined classes of elements, either instancing some predefined parametric plugins or through wire-frame input of 2D layouts. First is defined the building skeleton (backbone), i.e. the set of beams and columns or load-bearing walls, providing the structural grid. On its horizontal sections, the user specifies the exterior and interior walls (vertical enclosure and partitions), on which the fixtures are positioned (horizontal communications). Ceilings and floors (horizontal partitions) are instead automatically generated from the topology (1-boundary) of the 2-skeleton of the floors (subsets of 2-cells). Finally, various elements such as stairs, elevators (vertical communications), foundations and roofs (horizontal enclosures) are placed, using ad-hoc templates interactively provided by Metior's plugin server, for appropriate sizing and part dimensioning.

**Semantic annotations** At this stage (see Figure 1) the previously inserted elements are annotated semantically by means of references to database of materials, including densities. The annotation may consists of one or more materials, including percentages. For disposal control imposed by regulations, the user should assign one / more ECW codes and degrees of dangerousness (see Section 2.1). In this stage it is also assigned a pair of links that refer to the time schedule for disposal of building components.



wall ryB0g91WWl
wall SyzF6HyWWx
wall B10ecJ-be
wall rk4K6ByZbl
wall HJBL6SyWWx

Properties: [wall] BypL6BJ-Zg

| | | |
|---|---|---|
| height | 300 | cm |
| thickness | 20 | cm |
| textureA | Painting | |
| textureB | Bricks | |

Composition

| | |
|---|---|
| Material | Concrete |
| Class | 17.construction anc |
| Subclass | 01.concrete,bricks |
| Category | 01.concrete |
| Operation | disposal |

Figure 1: Graphical interface for semantics annotation of a modeled object

**Augmented reality visualization** After modeling and attribution of semantics to components, the quantity surveyer can validate the entire model by spatial merging into a 3D point cloud (see Figure 2) previously obtained by using flying drones for the exterior, or 3D laser scanners for the interior. In this way one can assess the adhesion of the modeled building structure to reality, possibly retracing to some previous step if the result is still not satisfactory.

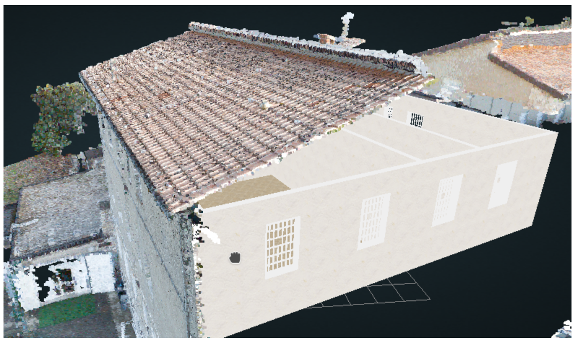


Figure 2: A model inside a point cloud

## 3.2 Process output

Once the work is done and the model geometry is validated, the application provides a final report. In particular, the final report will allows the quantity surveyer and/or the other professionals involved in the deconstruction design team, to determine whether the decision taken is convenient both economically and/or environmentally.

The final report consists of four documents. (i) *An estimate of volumes and weights of materials*, computed by appropriate integration calculations, based on the geometry of components and annotations defined on them. (ii) *An estimate of demolition costs*, including disposal and recovery. Starting from volumes and densities of materials, and from ECW codes and hence the mode of disposal, it is estimated the cost of the contribution to landfills. (iii) *The transportation costs* to move the materials to the closest landfills, taking into account the geographical position of the building, and by calculating the most convenient road routes. (iv) *An estimate of the expected time* for the complete demolition of the building, located on a Gantt chart. A PERT program, automatically generated while annotating semantically the components of the building, is used for this purpose.

## 4 DESIGN AND ARCHITECTURE

Workflow and requirements described in the previous section have been received in a prototypal application serving as proof of concept. With the aim of maximize accessibility for quantity surveyors, it is strongly web based and runs in all modern browsers. It is built using React by Facebook and an MVC design pattern with *unidirectional data flow* (Abramov, 2016): it ensures the best code maintainability and debuggability by centralizing access to the application state in a single controller.

### 4.1 UI and user experience

The web application presents itself as a simplified CAD software, where the user interface comprises three main areas of interaction: *toolbar*, *canvas* and *sidebar*, as shown by Figures 3 and 4.

From the *toolbar* the user can access functionalities related to: project life cycle (`new`, `save`, `load`); project editing (`show-catalog`); view/interaction mode switching (`2D`, `3D`); interaction mode changing (`selecting`, `pan`, `zoom`).

The *canvas* is the area in which the user can interact with actual model data. It supports two different

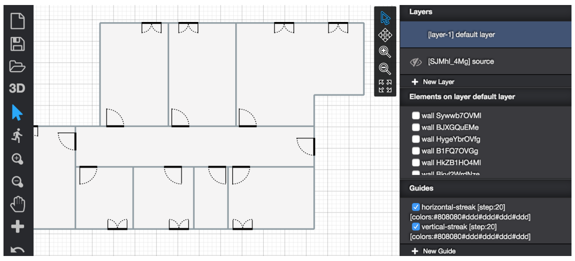

3D
Layers
[layer-1] default layer
[SJMhl_4Mg] source
New Layer
Elements on layer default layer
wall Sywwb7OVMl
wall BJXGQuEMe
wall HygeYbrOVfg
wall B1FQ7OVGg
wall HkZB1HO4Ml
Guides
horizontal-streak [step:20] [colors:#808080#ddd#ddd#ddd#ddd]
vertical-streak [step:20] [colors:#808080#ddd#ddd#ddd#ddd]
New Guide

Figure 3: *Metior* user interface: 2D canvas.

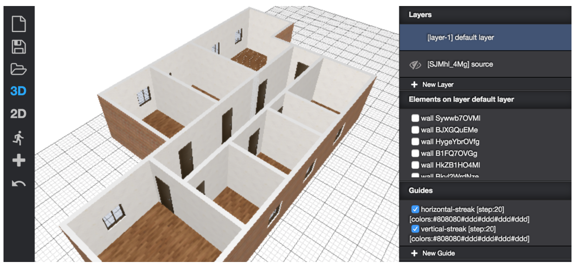

3D
2D
Layers
[layer-1] default layer
[SJMhl_4Mg] source
New Layer
Elements on layer default layer
wall Sywwb7OVMl
wall BJXGQuEMe
wall HygeYbrOVfg
wall B1FQ7OVGg
wall HkZB1HO4Ml
Guides
horizontal-streak [step:20] [colors:#808080#ddd#ddd#ddd#ddd]
vertical-streak [step:20] [colors:#808080#ddd#ddd#ddd#ddd]
New Guide

Figure 4: *Metior* user interface: 3D canvas.

view and interaction modes. In *2D-mode* the model is displayed as a 2D projection from the top, and the interaction consists of element insertion, selection and editing (according to specific plugin interaction prototype, see 4.2). In *3D-mode* a 3D model can be inspected and navigated, respectively via trackball or first-person interaction style, while object picking allows for element selection.

The *sidebar* shows the properties of the currently selected element. In the properties panel it is possible to view the description of the element, to add/remove metadata, and to modify any property. The latter is the interaction mode that allows the user to associate semantics annotations to every part of the model.

### 4.2 Plugin-architecture

The application has been designed to provide a small set of core interaction functionalities and to encapsulate the generation logic for architectural components (from the very basic to the most articulated) into specific plugins.

A *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 OOP) 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.

Table 1: Plugin examples according to taxonomy

| | **inside** | **over / free** |
|---|---|---|
| **linear** | `pipe` | `electrical-conduit` |
| **ver. area** | `window, door` | `wall` |
| **hor. area** | `light-panel` | `ground, ceil` |
| **volume** | `pillar` | `staircase` |

**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*.

**Plugins taxonomy** 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.

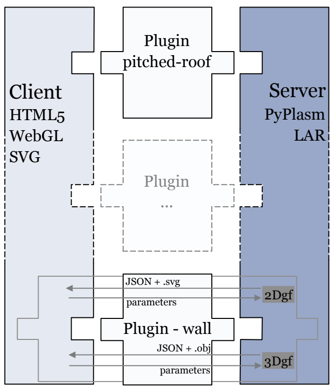


Figure 5: Client/Server architecture for server-side model generation.

**Plugin specific properties** 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 permit 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.

## 4.3 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 1 reports examples of plugins arranged according to the taxonomy introduced in Section 4.2.

## 4.4 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 5), 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 defined by the plugin, with superclasses *2Dgf* and *3Dgf*, are actually executed on the server, as shown in Figure 5 .

## 4.5 Plugin examples

```
def spiralStair(thickness=0.2,R=1.,r=0.5,riser=0.1,pitch=2.,
                nturns=2.,steps=18):
    V,CV = larSolidHelicoid(thickness,R,r,pitch,nturns,steps)()
    W = CAT([[V[k],V[k+1],V[k+2],V[k+3]]+
        [SUM([V[k+1],[0,0,-riser]]),SUM([V[k+3],[0,0,-riser]])]
        for k,v in enumerate(V[:-4]) if k%4==0])
    for k,w in enumerate(W[:-12]):
        if k%6==0: W[k+1][2]=W[k+10][2]; W[k+3][2]=W[k+11][2]
    nsteps = len(W)/12
    CW =[SUM([[0,1,2,3,6,8,10,11],[6*k]*8])
                for k in range(nsteps)]
    return W,CW
```

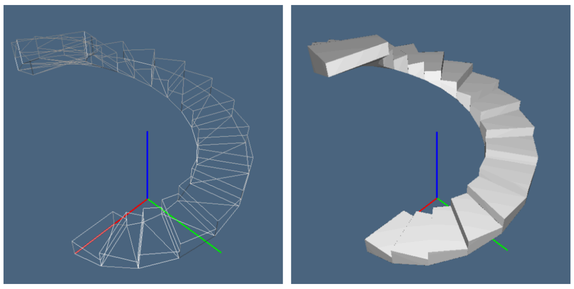


Figure 6: (a) The generating function of a strongly parameterized `spiralStair`; (b) default `spiralStair`.

Figure 6 shows the plugin used to generate concrete spiral stairs, with steps from coded update of a polyhedral approximation of a `larSolidHelicoid()`, minor and major radiuses (`r` and `R`) and `int` number of `steps`. The `nturns` float is in radians.

The model generated by a plugin for automatic generation of spatial concrete frames is shown in Figure 7. In this case the user interface produces a small `.csv` file specifying, for each component *planar frame*, the grid-like pattern of members, the 2D sections of columns and beams, and the proper roto-translation, i.e. the four numbers $\alpha, tx, ty, tz$ that instantiate each frame in the reference frame of the previous one. The connecting beams are automatically generated, and the various classes of members (`foots,timbers,columns,linkBeams`) are made available for interactive picking and possible local modification of geometry and/or materials by the system user, and of course, for quantity surveying.

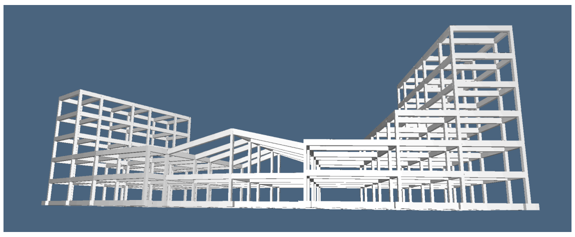


Figure 7: A spatial concrete frame generated, from a small `.csv` file, by the `pyplasm` plugin `spatialFrame`.

## 5 SOLID MODELING

The server-based geometric core of the modeling architecture is based on a set of python libraries, including `pyopengl`, `scipy`, `pyplasm` and `larlib`, that provide the current implementation of the LAR (Linear Algebraic Representation) scheme for solid modeling, 3D imaging and mesh representation, and which is briefly described in the following.

### 5.1 Linear Algebraic Representation

LAR is a general-purpose representation scheme (DiCarlo et al., 2014) for geometric and solid modeling introduced recently. The domain of the scheme is provided by dimension-independent *cellular complexes*, while its codomain is that of *sparse matrices*, stored using either the CSR (Compressed Sparse Row) or the CSC (Compressed Sparse Column) `scipy`'s memory format. The LAR polyhedral domain coincides with complexes of connected *d*-cells, even non-convex and/or including any number of holes.

The very general shape allowed for cells makes the LAR scheme notably appropriate for solid modeling of buildings and their components. E.g., the whole frontage of a construction can be described as a single 3-cell of its solid model. Also, the algebraic foundation of LAR allows not only for fast queries about incidence and adjacency of cells, but also to resolve—via fast SpMV computational kernels—the boundary extraction of any 3D subset of the building model.

It is worth noting that LAR provides a direct management of all subsets of cells and their physical properties trough the linear spaces of *chains* induced by the model partitioning, and their dual spaces of *cochains*. The linear operators of *boundary* and *coboundary* between such linear spaces, suitably implemented by sparse matrices, directly provide, depending on the dimension of the mapped spaces, the discrete differential operators of *gradient*, *curl* and *divergence*, while their product gives the *Laplacian* (DiCarlo et al., 2009).

## 5.2 Geometric computing of shape

Our computational environment is strongly oriented towards the most general parametric modeling of component shapes of buildings. This attitude is produced by two Python libraries, that provide a dimension-independent algebraic calculus with shapes (`pyplasm`) and their representation in the LAR scheme (`larlib`).

PLaSM (Paoluzzi et al., 1995; Paoluzzi et al., 2003), which stands for Programming Language for Solid Modeling, is a geometry-oriented extension of Backus' FL language (Backus, 1978; Backus et al., 1989). PLaSM is a project developed in the nineties in the framework of *Building Technologies Project* (*"PF Edilizia"*) of the Italian National Research Council. The `pyplasm` module (2006-) is the C++ porting of PLaSM to Python via SWIG wrapping.

On top of the Scipy/Pyplasm stack we started (2012), using literate programming methods, to build a set of software modules, named `larlib`, and using the LAR scheme. This library supports topological queries and physical properties of meshes and complexes, including integration of polynomials over the boundary of any chain of cells. For interactive visualization it relies on the `pyplasm` viewer, based on OpenGL. A porting of the most engaging parts of `larlib` to *Julia*, the last-generation programming language for scientific computing (Bezanson et al., 2014) started very recently, with the purpose of taking advantage of the great computational efficiency and parallelism of Julia in more demanding applications.

## 5.3 Plugin server framework

Our building deconstruction framework has a web-based client-server architecture, discussed in Section 4. *Metior*, the web client application, is illustrated in Section 3. 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 6) 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 (Solin, 2016) (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.

# 6 CONCLUSIONS

In this paper we have introduced a software architecture and framework to be used in design for building deconstruction. The immediate goal is to provide an advanced web-service for quantity surveyors, called to make reliable and accurate estimates of economic and environmental returns from unused buildings, according to new waste material regulations and to policies for land use control.

The reader may have easily seen that the Metior framework can be used as a simplified BIM instrument for renovation projects in existing buildings, and even for new design projects. Metior, currently in prototype development, is based on advanced technologies for web services and applications, and on recent developments in geometric computing. It is worthnoting that Metior might provide a basis for more demanding undertakings, in particular in a national scene in strong need for new large programs of renovation and seismic adaptation of its built heritage.

## REFERENCES

Abramov, D. (2016). Redux: predictable state container for javascript apps. `http://redux.js.org/`. Accessed: 2016-11-09.

Altamura, P. (2012). *Gestione eco-efficace dei materiali da costruzione nel ciclo di vita del fabbricato*. PhD thesis, Sapienza Università di Roma. (in Italian).

Backus, J. (1978). Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs. *Commun. ACM*, 21(8):613–641.

Backus, J., Williams, J., Wimmers, E., Lucas, P., and Aiken, A. (1989). FL language manual, parts 1 and 2. Technical report, IBM Research Report.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2014). Julia: A fresh approach to numeric computing.

DiCarlo, A., Milicchio, F., Paoluzzi, A., and Shapiro, V. (2009). Chain-based representations for solid and physical modeling. *Automation Science and Engineering, IEEE Transactions on*, 6(3):454 –467.

DiCarlo, A., Paoluzzi, A., and Shapiro, V. (2014). Linear algebraic representation for topological structures. *Comput. Aided Des.*, 46:269–274.

Hurley, J. W. (2002). How to smartwaste the construction industry. In *10th Symposium Construction Innovation and Global Competitiveness*, Conference Proceedings for the 10th Syposium Construction Innovation and Global Competitiveness.

Paoluzzi, A., Pascucci, V., and Vicentino, M. (1995). Geometric programming: a programming approach to geometric design. *ACM Trans. Graph.*, 14(3):266–306.

Paoluzzi, A., Pascucci, V., Vicentino, M., Baldazzi, C., and Portuesi, S. (2003). *Geometric Programming for Computer Aided Design*. John Wiley & Sons, Inc., New York, NY, USA. 815 pages.

Solin, P. (2016). Creative Computing Platform: Learn Coding and 3D Modeling! Accessed: 2016-11-12.

Volk, R., Stengel, J., and Schultmann, F. (2014). Building information modeling (bim) for existing buildings — literature review and future needs. *Automation in Construction*, 38:109 – 127.