Calpinator: A Configuration Tool for Building Facades

Andrés F. Barco and **Elise Vareilles** and **Michel Aldanondo** and **Paul Gaborit**[1]

Abstract. Reducing energy consumption of residential and commercial buildings is a major challenge nowadays. One of the strategies to achieve a significant reduction lies on building renovation. On this regard, a project targeting the industrialization of high performance thermal renovation for apartment buildings is been executed. The renovation is based on an external envelop composed by rectangular wood-made panels that cover the whole building. Two concurrent configuration tasks within the project have been identified: The configuration of each one of the panels w.r.t. to the facade and the configuration of the entire facade using a set of these panels. We focus our efforts on the development of a decision support system for the configuration of panels and facades. In this paper we introduce Calpinator, a Java-based configuration tool which is the heart of the decision support system for the project. The tool uses the notion of Constraint Satisfaction Problems as underlying model and implements a smart greedy-recursive algorithm to find a feasible configuration. In this communication we present the tool's design, its features and its implemented algorithm. We use a real-world scenario to illustrate the kind of facades the system can deal with.

1 INTRODUCTION

Energy consumption of residential and commercial buildings is constantly growing and currently it exceeds industry and transport sectors. It represents more than a third of the energy consumption in developed countries: 44% in France[2], 37% in Europe [19], 36% in North America [7] and 31% in Japan [5]. The increase in population, the enthusiasm for new technologies and the improvement of living comfort combined with the domestic habits creates an energy demand of buildings that will continue to increase in the coming years. Therefore, reducing energy consumption of buildings is now a priority in national and international levels.

According to Falcon et al. [8] one of the strategies to achieve a significant reduction lies on thermal building renovation. However, old methods involving by hand configuration, human scheduling and craft assembly, are expensive both in time and costs (bill of materials). It is therefore essential to assist this massive renovation of buildings with decision support systems [13].

Our work is part of project called CRIBA (for its acronym in French of Construction and Renovation in Industrialized Wood Steel) [8]. This project focuses on the industrialization of energetic renovation for residential buildings. The challenge, very ambitious, is to have a building energetic performance under $25kWh/m^2/year$ after the renovation. To do this, the building is completely covered with a new envelope composed of rectangular panels that are prefabricated in factories. The core of our work lies on the two concurrent configuration tasks that have been identified: To configure each one of the panels w.r.t. to the facade and to configure the entire facade using a set of these panels [23, 24]. We focus our efforts on the development of a decision support system for the configuration of panels and facades.

In this paper we introduce Calpinator, a Java-based configuration tool which is the heart of the decision support system for the CRIBA project. The tool uses the notion of Constraint Satisfaction Problems as underlying model and implements a smart greedy-recursive algorithm to find *one* feasible configuration of panels and facades. In this communication we present the tool's design, its features and briefly describe the implemented algorithm. It is worth noting that the algorithm, whose details can be fond in [2], is not part of the contribution of the present work. Instead, we focus our efforts on the implementation of the algorithm.

1.1 Related work

Layout synthesis, also known as space planning, techniques have been used within different contexts and scenarios. For instance, finding solutions for room configurations [25], apartment layouts [15] and activities within a business office [12]. Also, some tools have been implemented using different approaches, here we name a few of them. For example, in [22] Shikder et al. present a prototype for the interactive layout synthesis of apartment buildings including design information and an iterative design process. In [4] is introduced WRIGHT, a constraint-based layout generation system that exploits disjunctions of constraints to manage the possibilities on positioning two-dimensional objects in a two-dimensional space. Another system, LOOS [9], is able to configure spaces using rectangles that can not be overlaped but that may have holes. It uses test rules applied by steps to the rectangles in order to reach a good configuration based on its orientation and relation with other rectangles. The same authors have developed SEED [11]: A system based on LOOS used for early stages on architectural design. A comparison between WRIGHT and LOOS can be found in [10]. The system HeGel [1] (for Heuristic Generation of Layouts) is yet another space planning tool that simulates human design based on experimental cases. Finally, Medjdoub et al. presents in [17] the system ARCHiPLAN which integrates geometrical and topological constraints to apartment layout planning.

2 PROBLEM CONTEXT

In order to achieve the CRIBA project goals and ensure the sealing of the building, each facade of the renovated building must be

[1] Université de Toulouse, Mines d'Albi, Route de Teillet Campus Jarlard, 81013 Albi Cedex 09, France, email: abarcosa@mines-albi.fr

[2] http://www.developpement-durable.gouv.fr/IMG/pdf/Rep_-_chiffres_energie.pdf

completely covered by rectangular configurable panels, i.e., it is necessary a *configuration* of panels to cover the facade. Configuration is the task of designing a given product (here facades) from predefined generic components (here panels) [14, 21]. Components, which are described in terms of its functions, characteristic and prices, are usually arranged in a catalog. Customized solutions, are built from the combination of this catalog components and users requirements and preferences.

In our context, a configuration solution for a facade layout is therefore finding a spatial positioning of panels that covers the whole facade front, without overlapping nor holes. Keep in mind that, whereas components (i.e., panels) in our catalog have well-defined geometric shapes, dimensions and relations, their number is *not known in advance*.

2.1 Layout elements

The following elements are part of the renovation. We include the description of facades because its composing elements are important in the accurate configuration of panels.

- Facades: A facade is represented by a 2D coordinate plane, with origin of coordinates (0,0) as the bottom-left corner of the facade, containing rectangular zones defining:
  - Perimeter of facade to renovate with its dimensions (height and width).
  - Frames (windows and doors) with their dimensions (height and width) positioned in the reference plane.
  - Supporting areas (place to fix panels), with their permissible load, positioned in the reference plane.
  - Zones labeled as "out of configuration" which are areas that can not be covered by configured panels and therefore require specific panels design.
- Rectangular panels (shown in Figure 1): Panels are rectangular, of varying dimensions (from 1 to $45.5m^2$) and may include different equipment (joinery, solar modules, etc.). These panels are designed one at a time, when the definition of the layout configuration has been done, and manufactured in the factory prior to shipment and installation on the building site.

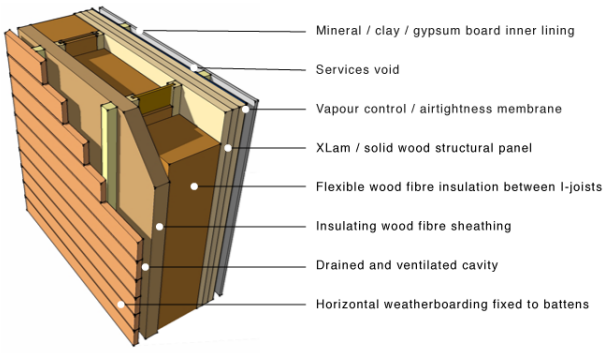

Figure 1. Prefabricated rectangular panels

2.2 Configuration process

The renovation process follows a series of steps going form the building site through the elaboration of panels and ending in its assembly [23]. At each level, a series of descriptive questions are asked to the user. Each answer has a potential impact on the permissible dimensions of panels. For example, the inaccessibility of a given facade may limit the dimensions of panels and therefore the surface covered by each one of them.

Once the descriptions of the site, building and facade are completed, the layout configuration of each facade can begin. Facades must wear a set of panels that must be the greatest as possible while respecting the architectural constraints, supporting areas, manufacturing and accessibility limitations. A rectangular panel is well configured if it meets the following conditions:

C1 It should cover the greatest possible area given the accessibility and the geometric position of frames.
C2 It can be installed in facade and supported by one or more supporting areas.
C3 It does not overlap with any other panel.
C4 It does not block the definition and configuration of the rest of the facade.

2.3 Configuration example

Consider the facade to renovate in literal (a) of Figure 2. The horizontal and vertical lines represent the places in which we are allowed to attach panels, i.e., the supporting areas. They correspond to various possible locations for the fasteners supporting the weight of panels. In this article, we assume that these places are capable of supporting a large enough weight to not constrain the surface of the panels.

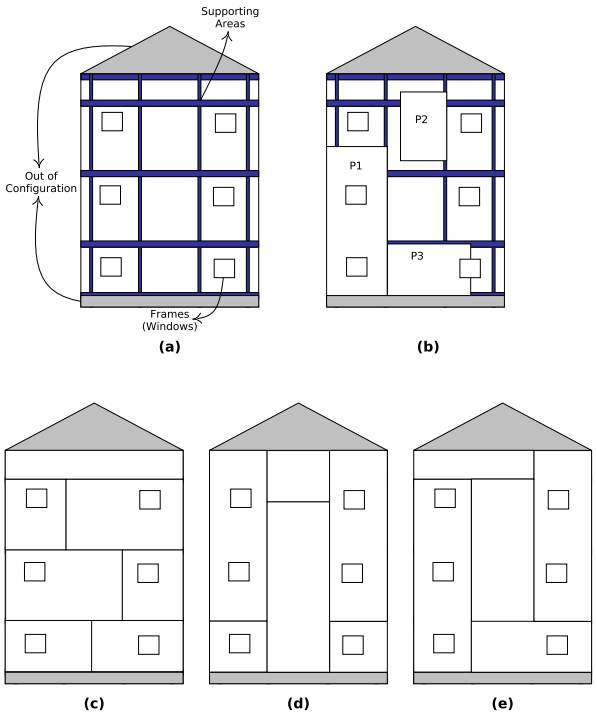

Figure 2. Well and ill-configured facades

Fasteners consist of two parts: One fixed directly onto the facade (wall bracket) and one installed on the panel at the factory. On the facades, the fasteners are positioned in the center of the supporting areas. At the level of the panels, brackets are fixed to the lower edge of the panels at equidistant (from 0.9 to 4 meters) from each other: These minimum and maximum distances allow to better distribute the weight of supported panels. A wall bracket can support a single panel (if it is on the perimeter of the panel) or two panels (if it is at the junction between two consecutive panels).

Small rectangles present on the facade to renovate in Figure 2 literal (a), correspond to the locations of frames (doors and windows).

Two areas of the facade are considered "out of configuration": The gable and the bottom part before the first horizontal supporting area. Two specific panels will be designed, one triangular for the gable and a square one for the specific building foot.

Figure 2 literal (b) presents a facade with three ill-configured panels: Due to the impossibility to place another panel north to the already placed panel $P1$, because there are no supporting areas at the corners of panel $P2$ and because panel $P3$ partially overlaps a frame. None of these configurations are valid. Facades in literals (c), (d) and (e) of Figure 2 present layout configurations where all panels meet the four conditions. From these, the facade (e) is preferred over the other two because it uses less panels.

3 UNDERLYING MODEL

Following the CSP model, we have identified 6 constraint variables, presented in Table 1, that allow us to represent the core of the layout configuration for a given facade: The spatial positioning of panels. Recall that a CSP problem is described in terms of a tuple $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{V}$ is a set of variables, $\mathcal{D}$ is a collection of potential values associated for each variable, also known as domains, and $\mathcal{C}$ is a set of relations over those variables, referred to as constraints [18].

Table 1. 6 variables used in the Calpinator implementation.

Variable	Description	Domain
(p_{x0},p_{y0})	Origin (bottom-left) of panel p	$x0 \in [0, w_{fac}]$, $y0 \in [0, h_{fac}]$
(p_{x1},p_{y1})	End (top-right) of panel p	$x1 \in [0, w_{fac}]$, $y1 \in [0, h_{fac}]$
w_p	Width of panel p	$[0.9, 13.5]$
h_p	Height of panel p	$[0.9, 13.5]$

The algorithm implemented in the tool uses the following parameters to set domains and to link variables: Width of facade (w_{fac}), height of facade (h_{fac}), environmental property (e_{fac}), for each frame f its origin point (f_{x0},f_{y0}) and its end point (f_{x1},f_{y1}) and, a collection of horizontal and vertical supporting areas each one of them with its origin point (sa_{x0},sa_{y0}) and its dimensions (sa_w,sa_h).

In what follows we briefly describe five of the six constraints that are part of the model and that are constraints in the Calpinator tool, more details about the model can be found in [2]. The sixth constraint, dealing with weight restrictions, is not presented because it is not yet included in the implementation.

Environmental The width w_p and height h_p of panels may be constrained because accessibility difficulties to the facade (e.g. trees, water sources, high voltage lines, etc), transportation issues (e.g. only small trucks available) or even climatological aspects (e.g. wind speed more than a given threshold).

Dimension Considering the panels suppliers and panel fabrication specifications, the width w_p and height h_p of each panel is in the range $[0.9, 13.5]$. However, this is actually a combination of values. In other words, it is possible to configure a panel with dimensions 0.9×13.5, 3×8.4 or 13.5×0.9, but it is not possible to configure one with dimensions 13.5×13.5, this is due to fabrication and transportation constraints.

Area A correct facade configuration is one in which the whole facade area is covered by prefabricated panels. Thus, a constraint forcing the sum of panel areas ($\mathrm{w}_p \times \mathrm{h}_p$) to be equal to the facade area ($\mathrm{w}_{fac} \times \mathrm{h}_{fac}$) is needed.

Non-Overlap In addition, we must ensure that the panels do not overlap so we can have a valid configuration. Thus, for each pair of panels p and q we apply the non-overlap constraint (also known as ndiff in different CSP tools).

Panel vs. Frames We adjust the width or height of a given panel if there exists a frame near to it. Either the panel overlaps the frame or the panel is right, left, up or down of the frame. In any case, due to the internal structure of the panel, borders of frames and borders of panels must be separated by a minimum distance given as input.

4 CALPINATOR: A FACADE CONFIGURATOR

Using the aforementioned model, we have developed two algorithms for solving the problem of facades configuration. The first algorithm is an attempt to find one layout configuration in a greedy fashion (more information can be found in [2]). The second algorithm uses global constraints and a constraint engine to find all possible panel configurations for covering the facades (more information can be found in [3]). In the current state of development of our tool, however, only the greedy-recursive algorithm has been implemented (Section 4.2). The constraint-based solution is planned for forthcoming releases of the tool and will, probably, use the constraint solver Choco [20] version 3 as underlying engine.

The result of our work is a Java-based tool that we call Calpinator[3]. It allows the user to input a building specification with an undefined number of facades and throws a solution for each of the facades if there is any. An intuitive view of the process is available by means of a friendly graphical user interface. In this Section we present the internal design of Calpinator, its implemented algorithm, the input and output formats, and the current options for customization.

It is worth mentioning that currently the user is suppose to be an architect, the building owner or a third-party contractor that is in charge of mapping the building data into the appropriated input format. Nevertheless, the goal, in a different stage of the project, is to automate the renovation process in every possible way. Thus, one of the partners in the CRIBA project is working on the automatic generation of the input for the configurator. In essence, they will use drones with pattern and image recognition to obtain most [4] of the facade related information.

4.1 Design

Calpinator has a very basic and modular design. The main characteristic of Calpinator is the implementation of a greedy algorithm for

[3] The name Calpinator is the combination of the French word *calpinage*, which means layout, and the word *configurator*. `https://bitbucket.org/anfelbar/calpinageprototype/wiki/Home`

[4] Some aspects can not be managed by drones. This is the case of the supporting areas maximum load, which is data that is recorded by the building constructors.

finding panels and facades configuration. Besides, we have enhanced the tool with an intuitive graphical user interface and provide a standard storage format (JSON) to allow a transparent communication with other software. Figure 3 presents the internal design of calpinator at first glance.

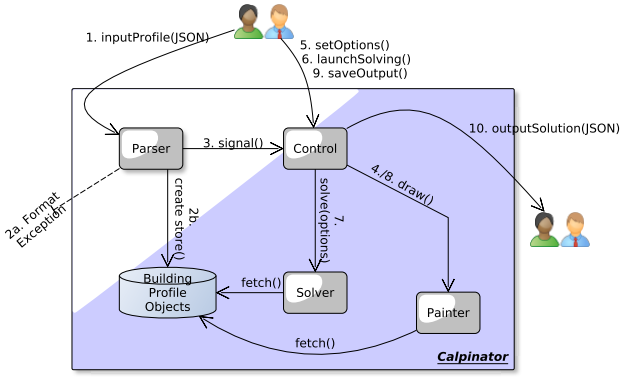

Figure 3. Calpinator internal design.

Let us explain further the execution and interaction between objects in the figure. Initially, the user inputs its building profile specification as a JSON file (Step 1). As expected, if the input file is not well formed, an exception is thrown (Step 2a). Alternatively, the system creates a data base (Step 2b) that stores all objects of the building, i.e., facades, frames, etc. Once the parsing is done, it informs the control it can enable the solving process (Step 3). The first task of the Control (Step 4) is to send the Painter object to draw the facades and its elements. Afterwards, (Step 5) the user may customize the solving process as explained in Section 4.4. If no user-parameters are given, Calpinator uses the default options (see Section 4.4). Next, when the user asks for the solution (Step 6), the Control calls the Solver (Step 7) which executes the greedy-recursive algorithm presented in Section 4.2. If a solution is found, the Control tells the Painter (Step 8), by user's demand, to draw one panel of the solution at a time. Finally, the user may save the solution to another JSON file (Steps 9-10).

Take into account that each time the user opens a new building profile, the data base with the profile objects is re-instantiated. This is done in order to avoid conflicts between elements of different building profiles.

4.2 Algorithm internals

Using the elements description in Section 3, we have developed an algorithm that solves the layout configuration in a greedy fashion [2]. This means that the algorithm makes local decisions for positioning panels following a well-known approach in layout synthesis field called *constructive* [12, 16]. Such decision making process is opposite to previous works where a search space is explored using backtracking search (see [6, 25] for instance). The implemented algorithm exploits recursion, simulating backtracking, when positioning a panel is not possible due to constraint conflicts. In what follows, we present the algorithm which an adaptation of the original one presented by the authors in [2]. The difference between this algorithm and the original one resides in the non-implementation of the weight constraint (postponed for further releases of the tool).

Step 1-: It begins by retrieving an available origin point and finding an end point given the heuristic for panel orientation. At this point, consistent with dimensions upper bounds, the panel is as big as possible.

Step 2-: It proceeds by generating a new valid point by means of solving conflicts between panels and frames. If dimensions of the panel violate dimensions constraints then it fails at positioning the panel.

Step 3-: It checks whether it is possible to install it using an horizontal or vertical supporting areas.

Step 4-: To install the panel, either in an horizontal or a vertical supporting area, it checks if the corners of the panel match supporting areas. This ensures that the panel can be installed as well as panels above it and at its right.

Step 5-: In the case it is not possible given the absence of supporting areas, it reduces the dimensions of the panel until the corners are matched with supporting areas.

Step 6-: Finally, if the panel is well positioned, it proceeds by computing new origin points and adding the next panel recursively.

Step 7-: If the next panel can not be placed, dimensions for current panel are reduced and another check is run. Otherwise we have found a solution so add it to the solution list and return.

4.3 Profiles and solutions

In order to use Calpinator, the user must know how to input the information and how to retrieve solutions. In this section we present the formats used by the tool.

4.3.1 Input

At the current state of development, Calpinator tool receives as input a building description that we call a *profile*. A building profile is, in essence, a table with alphanumeric values describing each of the facades in the building. In order to input this data into the tool, we have adopted a well-known format called JSON which is a composition of entries in the form *key:value*. This decision is attractive given that many formats (such as excel sheets and XML files) can be mapped to JSON files and vice versa. For instance, a simple excel sheet can be easily mapped into a JSON file using the open source program *Mr. Data Converter*[5]. Support for other formats, such as excel sheets and XML files, will be provided in forthcoming versions of the tool.

In order to avoid ambiguity, Calpinator is able to read only a particular set of values stored in a JSON file. The JSON input file for Calpinator is described in what follows.

- **type:** This key represents the type of element described by the entry. Allowed values are: *'facade'* which informs that there is a new facade in the building: *'floor end'* which is an horizontal supporting area: *'cross wall'* which is a vertical supporting area: *'crossing'* which describes the place in which an horizontal and vertical supporting areas meet: *'window'* a new window in the facade: *'door'* a new door in the facade and: *'out'* a zone out of configuration. There can be any number of elements in the building profile. Furthermore, elements do not follow any particular order inside the JSON file.

[5] The program is available online at `http://shancarter.github.io/mr-data-converter/`

- **id:** Each element is associated with an unique alphanumeric value that distinguishes the element from any other.
- **ref:** Each element, except from facades, belongs to another element. The key *'ref'* is an alphanumeric value referring to the element that the current element belongs to.
- **x:** Origin coordinate in x-axis.
- **z:** Origin coordinate in z-axis.
- **width:** Width of the element (in meters).
- **height:** Height of the element (in meters).

It is worth mentioning that Calpinator makes a distinction of all elements in a building profile. To do so, it uses the element identifier and the reference the element belongs to. Simply stated, all elements in a given facade must have different identifiers. However, elements of different facades may have the same identifiers provided they have different references. A given element will be part of the facade referenced by the field *'ref'* regardless the *'id'* value of the element.

Given that most users are used to excel sheets, we present an input example using an excel table and show its corresponding JSON translation. Table 2 presents a building with one facade, one window, one door, one zone out of configuration and three different supporting areas. Table 3 shows the corresponding translation into JSON.

Table 2. Building profile example using excel sheet.

type	id	ref	x	z	width	height
facade	fac1		0	0	18,95	10,64
floor end	1	fac1	0,16	0	18,79	0,16
cross wall	1	fac1	0	0	0,16	10,64
crossing	1	fac1	0	0	0,16	0,16
window	1	fac1	0,92	1,11	1,4	1,3
door	1	fac1	9,69	0,16	0,8	2,25
out	1	fac1	5,88	0	2	2

Table 3. Building profile example using JSON format.

```
[
{'type':'facade', 'id':fac1,  'ref':'',  'x':0,  'z':0,
                        'width':18.95,'height':10.64},
{'type':'floor end','id':1,'ref':'fac1','x':0.16,'z':0,
                        'width':18.79,'height':0.16},
{'type':'cross wall','id':1,'ref':'fac1', 'x':0, 'z':0,
                        'width':0.16,'height':10.64},
{'type':'crossing','id':1,'ref':'fac1',  'x':0,  'z':0,
                        'width':0.16,'height':0.16},
{'type':'window','id':1,'ref':'fac1','x':0.92,'z':1.11,
                        'width':1.4,'height':1.3},
{'type':'door','id':1,'ref':'fac1', 'x':9.69, 'z':0.16,
                        'width':0.8,'height':2.25},
{'type':'out','id':1,  'ref':'fac1',  'x':5.88,  'z':0,
                        'width':2,'height':2}
]
```

Recall that this is the first version of the Calpinator tool and thus the input data is limited to that used by the greedy-recursive algorithm. In consequence, important data as the y-coordinate (for a 3D model), facade adjacency and facade inclination have been currently left out of the configurator's input. Forthcoming developments will take into account these values but will have, necessarily, to be implemented with other versions or algorithms of that presented in Section 4.2.

4.3.2 *Output*

The output of a configuration is another JSON file containing the information of each one of the panels. Additionally, the output contains all information concerning frames inside panels. In short, each frame (e.g., window or door) covered by a panel has a relative position w.r.t. the origin of the panel. This is necessary for the fabrication of the panel. i.e., each panel must be fabricated with the corresponding holes for frames. Thus, for each panel or frame the output specify: **type:** Type of element(*'panel'* or *'frame'*), **id:** Panel or frame identifier, **ref:** Facade id or panel id that the element belongs to, **x:** Origin x-coordinate (relative to facade origin or the panel origin), **z:** Origin z-coordinate (relative to facade origin or the panel origin), **width:** Width of the element, **height:** Height of the element.

4.3.3 *Facades with no solution*

Calpinator tool allows for any kind of facade to be used as input. Nonetheless, it is not the case that any facade has a valid configuration given the constraints in our model or given the user preferences. For instance, literal (a) in Figure 4 does no have supporting areas in necessary places (no supporting areas at meter 15). Or perhaps, a given facade has no possible configuration because there is not enough distance between frames and supporting areas which is the case of literal (b) in Figure 4. Lastly, a facade may not be configured with Calpinator because an ill definition of zones out of configuration, as presented in literal (c) of Figure 4: No supporting areas at the top of the zone. As a workaround, the user should extend the zone out of configuration until the next horizontal supporting area. In the figure, the doted square shows the result of extending the zone.

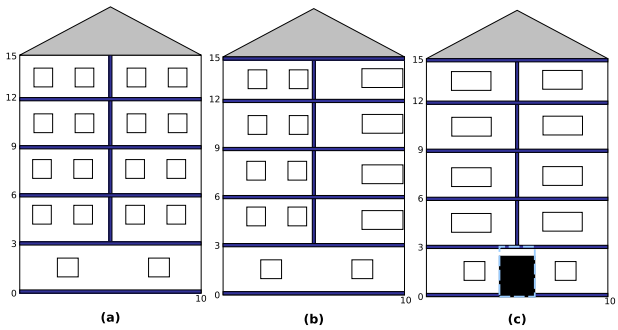

Figure 4. Three facades with no solution.

4.4 Parameterization

In its current state, our configurator is customizable in two ways. On the first hand, the user may choose an heuristic that defines a preference in the orientation of panels. On the other hand, the user may change the lower and/or upper bound for panel dimensions. As a consequence of such parameterization, the tool finds different solutions for the same facade. Nevertheless, as the implemented algorithm is deterministic, any given customization will result in the same configuration for a given input.

4.4.1 *Orientation heuristic*

When we talk about orientation we refer to relation between width and height which have an impact on the internal structure of the panel. In essence, if the width of the panel is bigger than its height, we consider the panel as *horizontally oriented*. Conversely, if the

panel height is bigger than its width, we consider it as *vertically oriented*. The user, for instance, may prefer to use horizontal panels in its facade. Calpinator will try then to put each panel horizontally, i.e., $w_p \in [0.9, 13.5] \wedge h_p \in [0.9, 3.5]$ (see the constraint *Dimensions* in Section 3). If a given panel can not be placed in the preferred orientation due to constraints conflicts, calpinator tries to place it using the other orientation. At the model level we consider the heuristic as a soft constraint, i.e., it can be violated without causing failure. This is why we do not include soft constraints in the core of our model.

4.4.2 *Dimensions range*

Recall that given the environmental aspects of the facades, the dimensions for panels may be reduced to a given interval. In addition, the user may, optionally, further constrain the dimensions for all panels in the facade according to its preferences. This is done by changing the lower and upper bound of the panel dimensions. As expected, the tool will respect the consistency between environmental constraints and the user preference. For instance, if the environmental properties constrain the width of a panel to be in the interval $[0.9, 8]$ and the user preferred upper bound is 9.5, the tool will set the upper bound in 8. This is due to the monotonic properties of CSPs. For this customization the tool presents three options:

- **Manually:** The user may change either the lower bound, the upper bound or both values.
- **Random:** The system chooses a random value for the upper bound. This constraints only one dimensions, the width for horizontal orientation and the height for vertical orientation. Note that the random strategy is applied for each panel in the facade. Thus, it is likely that most of the panels have different dimensions. This is interesting because, on the one hand, each time the user runs the algorithm it will find a different configuration of panels. On the other hand, it is more likely that the algorithm finds a valid configuration because it will try new values until exhaustion.
- **Square:** Try square panels only, i.e., constraints the upper both of vertical and horizontal orientation to be in the range of $[0.9, 3.5]$

Keep in mind that a given facade may have no configuration solution given its properties. Thus, constraining dimensions may reduce the number of chances to find one feasible facade configuration.

5 USING CALPINATOR

In this section we present a brief description of how Calpinator works in practice using some examples in real-world scenarios. As Calpinator is implemented in Java, the user needs to count with an updated version of the Java Virtual Machine. In addition, several dependencies are necessary in order to run the application. The libraries[6] used by the tool are *Oracle Commons* libraries (beanutils, collections, io, lang and logging) and *Maven* libraries (ezmorph and json-lib).

After launching the application, the user opens a JSON file specifying a building profile with any number of facades and elements (see Section 4.3.1). Then, all facades inside the building profiles are shown in the application, each facade in one tab. For instance, a building with two facades will be visualized as presented in the Initial State of Figure 5.

[6] For simplicity, these libraries are included in the distribution of Calpinator. Recall that these libraries are free software but each may have its own License agreement. Calpinator is distributed under General Public License version 3 and can be fount at `https://bitbucket.org/anfelbar/calpinageprototype/wiki/Home`

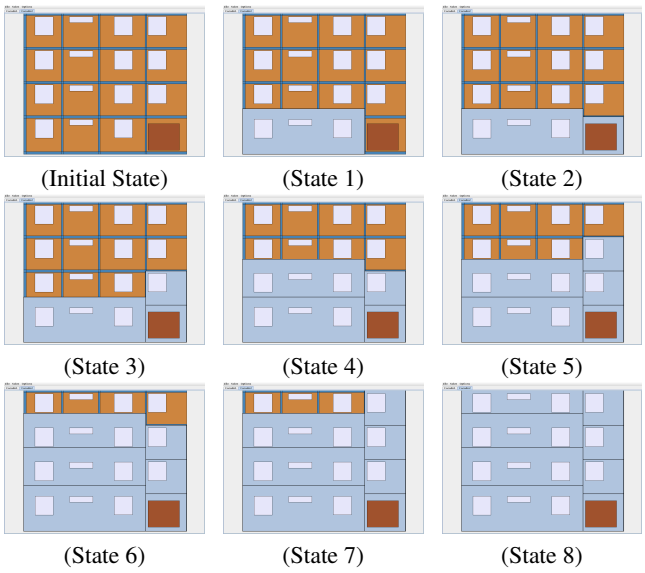

Figure 5. View of the configuration evolution.

Next, a customization may be done by changing the panels dimensions and choosing an heuristic as explained in Section 4.4. Afterwards, selecting the `solve` entry in the menu bar, the tool will try to find one feasible configuration for the facade in the current selected tab. For instance, Figure 5 presents a configuration solution for a facade with w_{fac} equals 12.59 meters and height equals 10.907. The customization for this facade is horizontal panels with maximum width of 13.5 meters for each panel. Each of the states in the figure presents different views reached by making left click on the canvas of Calpinator. Additionally, if the user wants to go back and see a partial configuration he may do so by using the right click on the canvas. Ultimately, the tool allows to save the configuration solutions by choosing `save` in the menu bar. Note that only those solved facades will be saved in the output. Given that this is work in progress and that the greedy algorithm is a deterministic one, the tool will only find one solution (if there exist) that satisfies the four conditions presented in Section 2.2. In consequence, the potentially many solutions for the facade layout are not found by Calpinator and thus no heuristic or criteria for choosing the best one is necessary. Ongoing investigation is looking into the possibility of finding different solutions by combining the greedy approach and search trees.

5.1 Examples

In this section we present some examples with different panel orientation and panel dimensions. The illustrated facades are part of the working site La Pince in the commune Saint Paul-lès-Dax in the department of Landes, France. Each of the columns of Figure 6 presents one facade of La Pince. The original facades, i.e., its frames, doors and supporting areas, are presented in literals (1a) and (2a).

Literals (1b) and (1c) in Figure 6, for the facade on the left, show configurations thrown by Calpinator using horizontal panels, with 3 meters as width upper bound for literal (1b) and 9.5 meters for literal (1c). Next, in literal (1d) and (1e) we present the configurations of the same facade using vertical orientation, with 6 meters as height upper bound for literal (1d) and 13.5 meters for literal (1e).

Conversely, the right column of Figure 6 presents some configuration configurations for the facade in literal (2a). The first two config-

Figure 6. La Pince facade 1 (right) and facade 2 (left).

urations present an horizontal orientation of panels and width upper bound of 8 and 13.5 meters for literals (2b) and (2c), respectively. Finally, in literals (2d) and (2e) of Figure 6 we present the configurations with vertical panels and height upper bound of 8 meters and 13.5 meters, respectively.

6 CONCLUDING REMARKS

Controlling energy consumption in buildings is one of the major challenges of the 21^th century. Reducing energy consumption in buildings is now focused on the renovation of existing buildings. To achieve renovation goals set by the French Government in 2009 and 2013, it is essential to assist massive renovation with technological tools and industrial methods rather than artisanal ones.

We presented in this paper a tool dedicated to the definition of layout configuration for building facades. The novelty of the tool lies on the implementation of a greedy-recursive algorithm that takes into account the many constraints inherited by facades in order to find a feasible configuration of panels. This work falls under the project CRIBA which aims to industrialize the renovation from the outside of buildings of residential housing in order to achieve an energy performance close to $25kWh/m^2/year$.

We have presented our first problem of layout configuration describing the specifics details related to the insulation of facades outside. In a second step, we have brefly described the knowledge model supporting this configuration problem based on constraints. The set of constraints was formalized by CSP in [2]. These formalize both manufacturing constraints and transportation, but also constraints relating to the geometry and structure of building and the internal structure of rectangular panels. The first version of the layout configuration tool incorporating all of these constraints is then presented and illustrated on an example from the pilot project site. The solutions proposed by our algorithm are all consistent with the constraints of the layout problem.

However, not the algorithm nor the tool take into account aesthetics preferences of users (e.g. architects' preferences). To avoid the generation of non-compliant solutions, additional "business" knowledge should be added to the (constraint) knowledge model. They are mainly related to the building after aesthetic renovation, such as an alignment constraint of connection joints between panels.

6.1 Future work

We acknowledge that our work is still in its infancy. Different efforts in crucial aspects will improve results in the model, algorithms and the tool. On this regard, the following objectives are strategic directions within the project.

a. Implement the constraint-based algorithm introduced in [3] is a priority. The algorithm is conceived to throw all possible panel configurations for the facade. This goal includes finding a constraint solver with appropriated filtering and search capabilities.
b. Improve greedy-algorithm with pre-processing and post-processing capabilities. Intuitively, a human configuration takes advantages of the facade dimensions and positions of frames to find a solution. Thus, it is adequated to add new constraints consequence of previous structural analysis of the facade.
c. Add more variables, hence constraints, to the model and improve or create new algorithms. For instance, there exists a constraint for fasteners and panel's edges distances which is important for the panel's stability. Also, there are some constraints over inclination of the facade, or the building itself, and panels positions. These and other relations will increase both the detail and the complexity of the problem, but are mandatory steps for the industrialization of the renovation.
d. Implement in Calpinator tool the weight constraint. The weight constraint to be implemented involves a new constraint variable, fai_{load}: Maximum weight load of fastener which is in the range of $[0, 500]$ kilograms. The constraint is is defined as follows.

   Weight Constraint A given fastener in a supporting area is defined by its coordinates and its maximum weight load. Let ATP_i be the panels attached to the fastener fa_i and let

$computeWeight(p)$ be a function[7] that returns the weight of panel p. Constraint over panels weight is defined by

$$\sum_{j=1}^{|ATP_i|} computeWeight(ATP_i[j]) \leq fai_{load}$$

This constraint is not implemented yet because we have not extracted and validated knowledge on how to distribute the panel's weight in the supporting areas. Up-to-now, we know that half of the panel's weight have an impact on a supporting area if there is only one fastener interacting between the panel and the supporting area. Otherwise all the panel's weight will be supported in area. Figure 7 shows some examples of this knowledge.

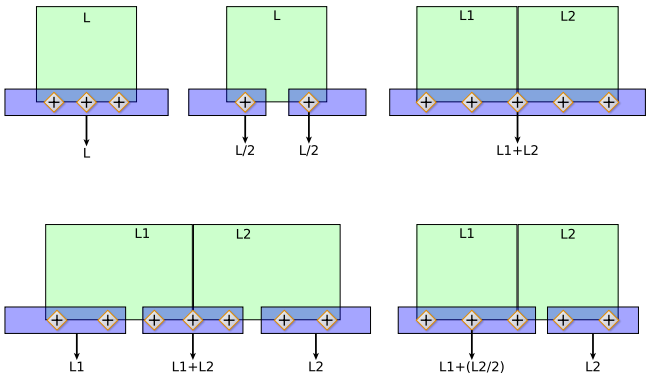

Figure 7. Distribution of weight in supporting areas.

e. Finally, a big challenge is to model and implement the concurrent renovation of multiple-adjacent facades. This particular scenario introduce different problems. Consider, for instance, a vertical supporting area at the right edge of a facade which is, in fact, the first vertical supporting area in the next facade. A given configuration has to take into account the weight in both facades over the same supporting area. Another issue is the angle between two adjacent facades and its implications for the width of panels.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the TBC Générateur d'Innovation company, the Millet and SyBois companies and all partners in the CRIBA project, for their contributions to the CSP model. Special thanks to the referees for their comments and to Philippe Chantry from École des Mines d'Albi for his contribution to the tool's GUI and some graphics in the paper.

REFERENCES

[1] Ö. Akin, B. Dave, and S. Pithavadian, 'Heuristic generation of layouts (hegel): based on a paradigm for problem structuring', *Environment and Planning B: Planning and Design*, **19**(1), pp. 33 – 59, (1992).

[2] A. F. Barco, E. Vareilles, M. Aldanondo, and P. Gaborit, 'A recursive algorithm for building renovation in smart cities', in *21st International Symposium on Methodologies for Intelligent Systems. To appear.* Springer-Verlag, (June 2014).

[3] A. F. Barco, E. Vareilles, M. Aldanondo, P. Gaborit, and M. Falcon, 'Constraint-based decision support system: Designing and manufacturing building facades', in *Join Conference on Mechanical, Design Engineering and Advanced Manufacturing. To appear.* Springer-Verlag, (June 2014).

[4] Can A. Baykan and Mark S. Fox, 'Artificial intelligence in engineering design (volume i)', chapter WRIGHT: A Constraint Based Spatial Layout System, 395–432, Academic Press Professional, Inc., San Diego, CA, USA, (1992).

[5] The Energy Conservation Center, *Energy Conservation Handbook*, The Energy Conservation Center, Japan, 2011.

[6] P. Charman. Solving space planning problems using constraint technology, 1993.

[7] U.S. Green Building Council, *New Construction Reference Guide*, 2013.

[8] M. Falcon and F. Fontanili, 'Process modelling of industrialized thermal renovation of apartment buildings', *eWork and eBusiness in Architecture, Engineering and Construction*, 363–368, (2010).

[9] U. Flemming, 'Knowledge representation and acquisition in the LOOS system', *Building and Environment*, **25**(3), 209 – 219, (1990).

[10] U. Flemming, C.A. Baykan, R.F. Coyne, and M.S. Fox, 'Hierarchical generate-and-test vs constraint-directed search', in *Artificial Intelligence in Design '92*, eds., J.S. Gero and Fay Sudweeks, 817–838, Springer Netherlands, (1992).

[11] U. Flemming and R. Woodbury, 'Software environment to support early phases in building design (seed): Overview', *Journal of Architectural Engineering*, **1**(4), 147–152, (1995).

[12] M. M. D. Hassan, G. L. Hogg, and D. R. Smith, 'Shape: A construction algorithm for area placement evaluation', *International Journal of Production Research*, **24**(5), pp. 1283–1295, (1986).

[13] Y. Juan, P. Gao, and J. Wang, 'A hybrid decision support system for sustainable office building renovation and energy performance improvement', *Energy and Buildings*, **42**(3), 290 – 297, (2010).

[14] U. Junker, *Configuration.*, Chapter 24 of Handbook of Constraint Programming (Foundations of Artificial Intelligence). Elsevier Science Inc., New York, NY, USA, 2006.

[15] K.J. Lee, H.W. Kim, J.K. Lee, and T.H. Kim, 'Case-and constraint-based project planning for apartment construction.', *AI Magazine*, **19**(1), pp. 13–24, (1998).

[16] R. S. Liggett, 'Automated facilities layout: past, present and future', *Automation in Construction*, **9**(2), pp. 197 – 215, (2000).

[17] B. Medjdoub and B. Yannou, 'Separating topology and geometry in space planning', *Computer-Aided Design*, **32**(1), 39 – 61, (2000).

[18] U. Montanari, 'Networks of constraints: Fundamental properties and applications to picture processing', *Information Sciences*, **7**(0), 95 – 132, (1974).

[19] L. Pérez-Lombard, J. Ortiz, and C. Pout, 'A review on buildings energy consumption information', *Energy and Buildings*, **40**(3), 394 – 398, (2008).

[20] C. Prud'homme and JG. Fages, 'An introduction to choco 3.0 an open source java constraint programming library', in *CP Solvers: Modeling, Applications, Integration, and Standardization. International workshop.*, Uppsala Sweden, (2013).

[21] D. Sabin and R. Weigel, 'Product configuration frameworks-a survey', *IEEE Intelligent Systems*, **13**(4), 42–49, (July 1998).

[22] S Shikder, A Price, and M Mourshed, 'Interactive constraint-based space layout planning', *W070-Special Track 18th CIB World Building Congress May 2010 Salford, United Kingdom*, 112, (2010).

[23] E. Vareilles, A. F. Barco, M. Falcon, M. Aldanondo, and P. Gaborit, 'Configuration of high performance apartment buildings renovation: a constraint based approach', in *Conference of Industrial Engineering and Engineering Management (IEEM)*. IEEE., (2013).

[24] E. Vareilles, C. Thuesen, M. Falcon, and M. Aldanondo, 'Interactive configuration of high performance renovation of apartment buildings by the use of csp', in *15th International Configuration Workshop*, pp. pp. 29 – 34. CEUR Workshop Proceedings, (aug 2013).

[25] M. Zawidzki, K. Tateyama, and I. Nishikawa, 'The constraints satisfaction problem approach in the design of an architectural functional layout', *Engineering Optimization*, **43**(9), pp. 943–966, (2011).

[7] This function uses the next values to calculate the weight of a panel: dimensions of the panel, insulation type of the panel, weight of the frames within the panel (if any) and weight of any other component (e.g. solar modules).