Refurb!

A tool allowing for iterative refurbishment

Gabriel Wurzer[1], Ugo Maria Coraglia[2]
[1]TU Wien [2]Sapienza University of Rome
[1]gabriel.wurzer@tuwien.ac.at [2]ugomaria.coraglia@uniroma1.it

Refurbishments and adaptations to existing building structures can be a challenging problem: Keeping track of all building measures (e.g. what walls, doors and installations to add or remove) is equally demanding as trying to keep an eye on the constraints (e.g. natural lighting) and functions that the changed structure will provide. It also demands a integrated view of the redesign (spatial aspect) and the refurbishment as a project (time aspect). To this end, we have been developing our planning tool ``Refurb!'', which lets a user plan refurbishments and adaptations to existing structures using a mixed metaphor of ``CAD-tool + project plan'', including a variety of analysis tools to compare the original state of a structure to the planned one (e.g. adjacency and circulation before and after adaptation). The tool is aimed at project planners and ranges from small scenarios (e.g. relocation/adaptation/refurbishment of a department) to big scenarios (e.g. relocation when bringing a hospital into service).

Keywords: *Refurbishment, Planning Tool, Cellular Automaton*

INTRODUCTION

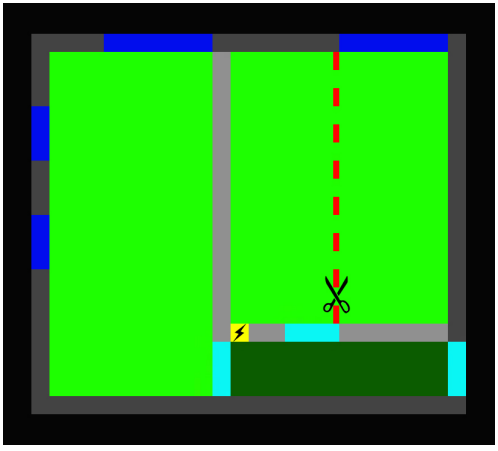
There are many buildings that are built using a regular beam structure and thus allow for easy reassignment of interior walls. In principle this would allow for rapid reallocation of rooms by means of merging, splitting, enlargement or decrease in area; however, this is not as simple, since rooms are constrained both in terms of adjacencies to other rooms as well as through their required installations (e.g. water, electricity) or cannot be relocated at all (e.g. stairs). Another wish is that relocated rooms have more or less the same amount of windows, natural lighting and other qualities that define the "feeling" of that space. Even for a moderately-sized building, there are so many combinations of these factors that the relocation problem is cannot be solved easily with a manual approach, but it is also hard to represent and work on digitally.

In our work, we have reformulated this problem into a simplified model in which we can describe refurbishments as a sequence of operations acting on interior walls (create/delete) and thereby also on rooms (merge/split/enlarge/shrink) which we are also able to relocate. The succession of operations mimics a project plan for a refurbishment project, where the order of activities is equally important to their implementation (Coraglia and Wurzer 2017). Apart from being able to enter operations in our tool 'Refurb', we also check whether all required installations and previous qualities still present, so that the

"feeling" is more or less the same in the new or modified space. The tool furthermore looks at changes in adjacency relations amongst the rooms, in order to inform over changed reachability and walking distances. The user then can go visually through the changes, thereby being able to see how the refurbishment project will evolve.

APPROACH

We use a grid representation (i.e. Cellular Automaton [CA], see figure 1) in which each room (defined by the properties editable [yes/no], functions [list], needed installations [list]) is composed of a set of cells having a certain type (door, window, [interior/exterior] wall, floor, circulation); additional properties within each cell signify the presence of installations (e.g. water line) and quantities (e.g. natural illumination). In that representation, we are now able to remove or add interior walls, which in turn creates, deletes, extends, shrinks, splits or merges rooms.



Topological queries on the grid representation before and after the change now allow us to build up an adjacency graph A and A', where each room is represented by a node, connections between rooms (shared door!) as edges, and node properties characterize the room (function, area, number of windows and doors, presence of installations, average quantities such as illumination, boolean flag signifying whether the room is relocatable). Comparing A with A' allows us to visualize the change in a room's properties and highlight problematic implications of that change (e.g. no entrance, no windows, needed installation not present, changed adjacency or reachability).

Apart from being able to edit, create or delete room definitions, we also want to be able to swap or duplicate rooms easily; we thus offer copy, cut and paste operations using a infinite clipboard; pasting can either (a.) be conducted in an "empty" room or (b.) using an already-occupied room as a target; the latter allows for merging of functions and required installations. Further editing options are for adding/deleting installations, windows, doors in the cell representation, as well as toggling the floor type between circulation and regular floor.

The sequence of operations forms is linear, however, the changes made are not: We are able to visualize the merging/splitting and so forth that result from these separately, in a fashion used by versioning systems (versioning tree). Evaluating that tree at the branches (i.e. final stage after all refurbishments were made) and tracing back to the original room can be used to give a change history - i.e. "where has this room evolve from".

Figure 1
Imported "floor plan" showing different cell types

CASE STUDY IMPLEMENTATION

The approach has been implemented as a NetLogo program in order to test it in practical terms. As case study, we have drawn one floor of our university department as color-coded bitmap (1 pixel = ☒ m^2, colors correspond to different cell types) which we imported into NetLogo's CA (i.e. grid). We did not attempt to include multiple floors at this time, facilitating this would mean using a different version of Netlogo (i.e. NetLogo 3D) and also having to extend our method beyond the concepts described herein (e.g. room must stay on same floor).

Querying the grid is an easy task in NetLogo, since its language (Logo) is functional and thus allows

Figure 2

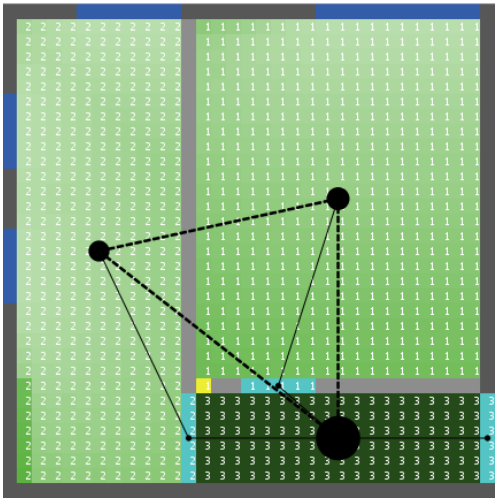
for easy selection of cells based on color (=type) and inscribed properties. A typical algorithm we use to determine the cells belonging to every room is the Flood Fill algorithm which is given in Algorithm 1.

```
Algorithm 1 - Flood Fill(cell, id)
if (cell is floor or circulation) and
 ↪ (cell.id is undefined):
    cell.id = {id}
    for each neighbor of cell:
      Flood Fill(neighbor, id)
```

Flood Fill determines the floor cells of a room, given that one starts at a certain cell inside this room. It is used by a labeling procedure (Algorithm 2) whose job is to identify all different rooms in a cellular floor plan. This is done by giving each cell an id property, which is a list containing the room number(s) it belongs to.

```
Algorithm 2 - Label Rooms
id = 1
while any cells without id of type
 ↪ floor or circulation:
    Flood Fill(one of these cells, id)
    floor = cells with cell.id being
     ↪ id
    if any? cells of type circulation
     ↪ in floor:
        set all floor cells type to
         ↪ circulation
    boundary cells = {}
    for each cell in floor:
        adjacent cells = neighbors of
         ↪ cell with neighbor.id not
         ↪ cell.id
        for each neighbor in adjacent
         ↪ cells:
            neighbor.id   {id}
        boundary cells   adjacent cells
    increase id
```

The results of our analysis are visualized in figure 2, which also gives a depiction of natural lighting (ray-tracing from the windows, lighter cells are more illuminated than darker ones).



After analyzing the CA for identifying the different rooms, we can build up a graph in which each node stands for a room (see again the circles in figure 2). The representation of this graph is visual - we use NetLogo nodes ("turtles") that can be positioned freely on top of the CA. Since every room knows its boundaries, we can check for shared doors with other rooms. Out of this analysis we can insert adjacencies (=edges ["links"] between two nodes; see again lines in figure 2) into the graph.

Operations on the cellular automaton (typically: changing walls, assigning a different cell type) are entered visually in the CA, other (non-visualizable) edits are done in a dialogue-driven fashion for each room. In the background we need to not know what constitutes a meaningful "operation" - for example cutting, pasting but not copying to the clipboard - so we can add it to the list of operations which behaves like an edit history. Figure 3 depicts this edit history where we attribute edits with "minor" and "major", and group these into different "edit phases" called changesets. In short, edits are not "major" if they do not lead to a split or merge in rooms, or the attribution of different functions or needed installations that are the basis for the programme checker that is given in figure 4.

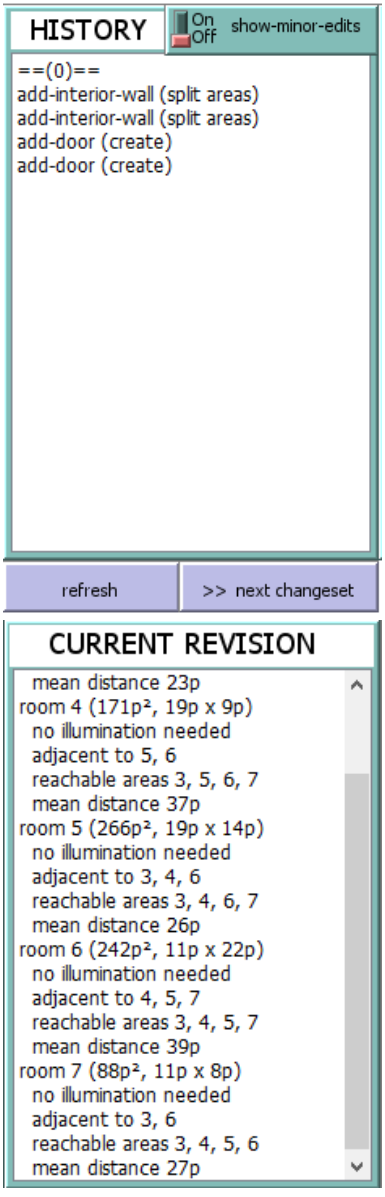


Figure 3
Version history: Depicts different edits (major or minor) done to the initial design; grouped into changesets.

Figure 4
Revision checker: Analyzes the current design according to the needed functions, illuminations, available adjacencies and reachability of other areas.

Through analysis of the list of operations, we can build the said adjacency graphs A and A' (see again 'Approach') out of each pair (successor, predecessor) in that list. The comparison between both is done with reference to the CA, so that we can find out that there was a split or merge in the room structure; we record this in the versioning tree which can be visualized using again NetLogo's nodes and edges. By tracing the leaves back to the tree root, one can obtain a version history for a certain room (see again circles and lines in figure 2 for the graph; figure 3 for the version history; figure 4 for the analysis of the current revision).

In further analysis, one can compare a changed design (figure 5) to baseline. This can either be the initial design or the design in the current changeset (think: design in the current project phase). We can get added, removed and unchanged {adjacencies, functions, pathways to rooms} as well as a network analysis relative as to what that means (e.g. the closeness centrality, depticted by the size of the room nodes in figure 5 and also logged as output of the analysis console in figure 6. All in all, the planner is informed over the implications of the changes at every step of the design.

DISCUSSION & FUTURE WORK

The main contribution and technique of our approach lies in the ability to enter refurbishments in a step-by-step manner, which helps to visualize the progression of the refurbishment work and can be used for drafting an initial project plan. Our approach is useful when being constrained to local changes rather than having the opportunity to relocate rooms globally; if this is possible we would rather recommend to use an automated floor planning method (e.g. using K-d trees and Evolutionary algorithms [Knecht and König 2010]). However, the goals of the latter are rather different than ours: Automated floor planning methods want to achieve a global optimum, we just want a method that allows for local refurbishment with the least amount of changes to the current floor plan, without considering optimization.

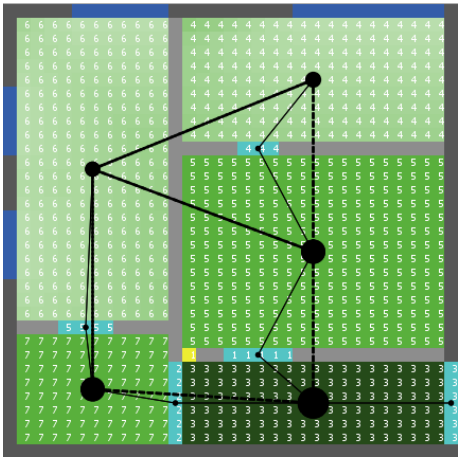


Figure 5
Changed design showing relative importance [closeness centrality] of rooms as size of room nodes, adjacencies [dashed] and reachability [solid] edges between these.

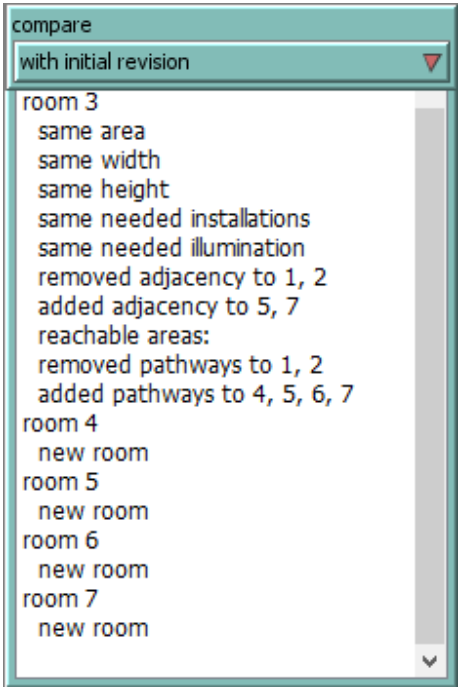


Figure 6
Comparison of current design based on either initial design or base design for current changeset

When it comes to future work, there many options that are on the table:

- The current graph measures used for analysis are based on added/removed or unchanged rooms. We investigate adajencies and the circulation graph (what room is reachable from another room, change in path lengths and so forth). However, as was shown in (Wurzer and Lorenz 2016), many more graph analysis measures are possible, such as clustering the resulting rooms and showing "public/private" spaces by means of centrality. Likewise, the "public/private" quality might also be a function of people crossing these spaces, which demands a process/occupancy model such as the one given by (Tabak 2009) or, more recently, by (Simeone et al. 2012).
- Clearly, the current implementation could benefit from not only editing the design, but outputting the project plan (a task which is easily done since we record every change that acts on the initial design, grouped into "changesets" = project phases and distinguished by "minor" or "major" edits = work packages or sub-packages). Such an approach has already been carried out in (Wurzer et al. 2019) and was thus not implemented in the showcase tool.

CONCLUSIONS

We have presented an approach for iterative refurbishment, using a grid representation of a floor plan as input, transforming that into a adjacency graph made out of rooms (nodes) and physical connections between these (edges). Refurbishments then become a sequence of operations which change the adjacency graph, creating, merging, splitting, enlarging, shrinking or deleting rooms or creating or deleting connections between them. A visualization/checking algorithm kicks in after each operation, informing the user of problems and visualizing changes. Because of this iterative nature of our ap-

proach, we can also visualize the progression of the refurbishment as a preliminary "project plan", which can help deal with the implications of refurbishments at different stages of a refurbishment project. The approach is best-suited for cases in which modifications to the existing building layout are to be made locally with minimal change to the existing layout rather than globally, in which case we would rather recommend to use an automated floor planning method which gives optimal results.

REFERENCES

Coraglia, UM and Wurzer, G 2017 'CONVIS: A tool enabling uninterrupted operation during refurbishments of complex buildings', *Proceedings of SIGraDi 2017*, Concepcion (Chile), pp. 376-380

Knecht, K and König, R 2010 'Generating Floor Plan Layouts with K-d Trees and Evolutionary Algorithms', *GA2010 - 13th Generative Art Conference*, Politecnico di Milano University, Italy, pp. 238-253

Simeone, D, Kalay, Y, Schaumann, D and Hong, SW 2012 'An Event-Based Model to simulate human behaviour in built environments', *Proceedings of eCAADe 2012*, pp. 525-532

Tabak, V 2009, *User simulation of space utilisation : system for officebuilding usage simulation*, Ph.D. Thesis, TU Eindhoven

Wurzer, G, Coraglia, UM, Pont, U, Weber, C, Lorenz, WE and Mahdavi, A 2019 'A Cell-Based Method to Support Hospital Refurbishment', *Proceedings of the 12th Envibuild – Buildings and Environment – From Research to Application*, pp. 553-560

Wurzer, G and Lorenz, WE 2016 'SpaceBook: A Case Study of Social Network Analysis in Adjacency Graphs', *Proceedings of eCAADe 2016*, Oulu, Finland, pp. 229 - 238