FROM FLOOR PLAN DRAFTING TO BUILDING SIMULATION - AN EFFICIENT SOFTWARE SUPPORTED PROCESS

Gerhard Zimmermann
Informatik, University of Kaiserslautern, Germany

Abstract

The paper describes a process that efficiently supports the early building design stage with prototype tools to capture topological floor plan sketches, calculate geometries, explore design alternatives, and generate data for performance simulations, for example for EnergyPlus. It is also shown how such data are used to generate object-model based simulators in the Virtual Building Laboratory environment. Examples and preliminary efficiency data are presented.

Introduction

The work presented here is the result of an interdisciplinary research cooperation of computer scientists and building architects. We hope, that this research will result in new and unusual approaches to the benefit of both disciplines.

The building design process can be modeled by three stages: conceptual design, design development, and construction design. Stages two and three are well supported by CAD-tools, whereas the early design stage, the conceptual design, is still mainly manual work.

During the conceptual design stage, decisions of great impact on the quality and the cost of the product are made. Therefore, tool support for the evaluation of the consequences of design decisions during early design stages is of great importance. One quality aspect is the visual appearance, supported by tools for capturing sketches. The quality aspect we are interested in is building performance, for example thermal properties.

Building performance is typically evaluated by simulation. One open problem is the integration of sketching and simulation such that the creativity of the designer is not restricted and the additional effort for simulation is in the right proportion to the quality gain. A large quantity of detailed data is required for performance simulations. The generation of these data is one of the reasons why existing performance simulators are rarely used in the conceptual design stage.

Here, we will show an alternative solution to this problem. In our approach, the conceptual design process starts with space program development and documentation, floor plan sketching, manual entry of the sketch topology with a simple graphical tool, automatic generation of resulting floor plan geometries, the refinement of geometries, automatic simulation data generation, and simulation. The geometry generation adds wall dimensions and features like openings to the floor plan. In our approach, the simulation phase consists of a semi-automatic simulator generation and a simulation execution phase. We call our process and tool environment the Virtual Building Laboratory (ViBuLab). Data for other object oriented simulator interfaces can be generated as well.

Because of the highly efficient tool support, design alternatives can be easily evaluated and compared. In contrast to automatic layout approaches, the designer maintains the full control over the floor plan topology while getting analysis results that are very close to the final geometry of the building under design. The tools also help to maintain data consistency to free the designer from tedious data checks.

In the next section, the related work is presented. In section Process Overview, the conceptual design support is explained in more detail. The section The Building Planner is a description of the topological floor plan generation and geometrical transformations. In section Simulator Interface the simulator generation process is introduced in so far as necessary to define the input data requirements and the interface to EnergyPlus is explained. Finally, results of design experiments are shown in section Results, followed by a conclusion.

Related Work

The general building design process and especially the early design stages are treated in many publications. Here, we will only refer to (Polladis, S.N.,Bakos, Y. 1987) and (Rozenburg, N., Eekels, J. 1994).

Several papers deal with capturing and 3D-visualizations of sketches as for example (Lee, M.-C. 2001). In principle, such tools could also be extended to feed data to simulators, but since this is not their purpose, important material data are not represented.

Another approach is the simplification of simulator data entry by rigorously using grids, abstractions, and defaults for dimensions and materials. A very efficient implementation of this type of tools is SEMPER (Mahdavi, A. 1996).

A different research area supports early design by automatic layout. Liggett (Liggett, R.S. 2000) gives an excellent overview over the main methods. Although a single criterion optimizations such as total path length minimization show good results, many other criteria are of equal importance during floor planning. Therefore, most designers still prefer the manual placement of spaces.

Tam's approach (Tam, K. 1991) is based on creating rectangular floor plan topologies by recursive dissection. This method does not require a grid, but is in principle limited to rectangular floor plans that can be represented by dissection trees. This excludes so called pin-wheels. In (Suter, G., Mahdavi, A. 2003) this method is extended to also create L- and U-shaped spaces in the topology (called a schematic model) and also to extent wall dimensions to create floor plan geometries (detailed models).

Dissection based floor plan geometries have also been used extensively to create floor plans for VLSI (Very Large Scale Integration) computer chips (Schürmann, B. et al. 1992). This hierarchical layout process has many similarities to building floor plan generation: logic and memory cells relate to activity floor areas, wiring spaces relate to circulation and wall areas, and pads at the chip perimeter or on the surface relate to openings for daylight. Therefore, the work of Fleming on the optimization of building floor plans (Fleming, U. 1986), that led to the SEED (Fleming, U. et al. 1994) project, is also one of the basic works in chip floor planning. The use of shape functions in chip floor planning (Zimmermann, G. 1988) could be transferred to building floor plan geometries. In (Zimmermann, G., Suter, G. 2004) we have demonstrated how dissection based topologies can be automatically transformed into valid building geometries by expressing the constraints in the form of shape functions. In section The Building Planner we will show this process in more detail. So far, this transformation is limited to 2 dimensions or single floors. Currently, we are extending it to 3 dimensions, the so called multi-floor problem. Here we show how the multi-floor problem can be solved interactively.

Another basis for our process are the advances in software engineering, especially in the field of reactive systems (Metzger, A., Queins, S. 2002). Building control and performance simulation are such systems. In the past, powerful performance simulation environments have been implemented with large efforts in manpower (EnergyPlus 2004), (Mahdavi, A. 1996). These legacy and research systems can be adapted to specific building instances, but this requires a large set of parameters and is therefore time consuming. Our re-

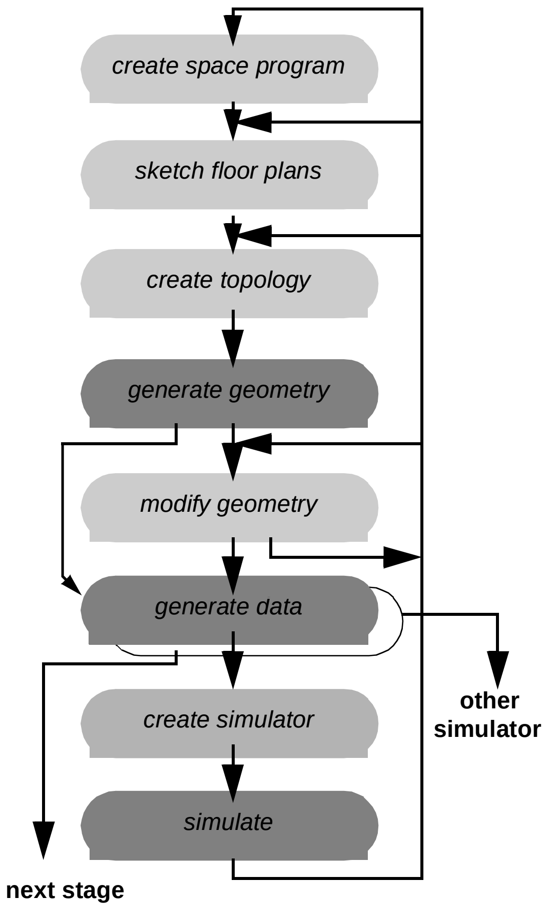

Figure 1 Conceptual design process. Light grey mea manual, dark gray automatic activities

search goal is the reduction of this time during conceptual design, based on the idea that during this early design phase a quick comparisons of alternatives is more important than the precision of results.

We have demonstrated that simulators for specific building instances and performance questions can be constructed on-the-fly and discarded afterwords (Zimmermann, G. 2002). Such a process can be made very efficient by reusing simple object-oriented physical models and by automatic prototype code generation. We do not claim that such simulators are better than the cited legacy systems. Their advantage lies in the extreme flexibility in integrating effects or functions as for example intelligent control systems, office user behaviors, or workplace features.

We did not find any published design effort measurements for the designs and simulations in the conceptual design stage. Therefore, our efficiency results can not be compared with others.

Process Overview

Figure 1 shows the basic activities in the proposed conceptual design process. The resulting documents are not shown, but are important parts of the product documentation. The feedback loops lead to design modifications or alternative design explorations as a results of the generated geometric floor plan drawings or of simulation results. In any case, document versions are created such that returning to former versions is always possible. Besides versioning, the process is also supported by an automatic time and effort recording environment for our design experiments.

create space program is the traditional manual activity performed by architect and customer and results in a textual document, listing the number of activity space types with their net floor areas. The extended space program also lists requirements for spaces like circulation areas, utility rooms etc. that are traditionally not part of the net area. A space requirement can take the form of a fixed dimension instead of an area, as in the case of hallways. In our environment this space program is a spread sheet and supports simple calculations for the *create topology* step. The space program is based on the requirements of the stake holders and can only be modified within defined flexibility ranges or with their consent.

The spaces in the space program of a building with several storeys are first assigned to these storeys, if this is not already defined in the program. Then *floor plan sketches* on paper are produced manually that roughly place all spaces in the different storeys. Several alternatives can be created. In principal, this step can be replaced by automatic floor plan optimization tools as reviewed in (Polladis, S.N.,Bakos, Y. 1987).

If floor plan sketches are used, *create topology* is a manual step that needs some experience and preparation. First, the floor plan of a storey can be dissected into larger sections as for example stair case shafts, fire sections, organizational units. Figure 2 shows a simple example with three floors and two shafts. Then, from the sum of the net floor areas of all spaces in each section, overall area requirements in each storey are calculated. With a spread sheet space program this can be done automatically. Typically, one horizontal dimension of the building is predefined. Together with the total storey area this defines the other horizontal dimension.

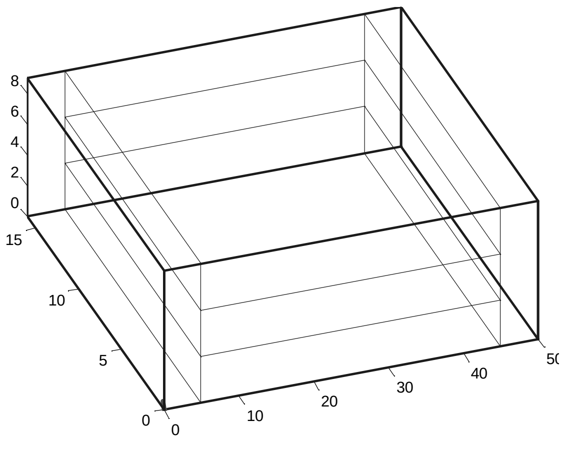

Figure 2 3D floor plan topology

create topology then proceeds by planning the order of the dissections, using the floor plan sketches. Here, we distinguish three approaches. The most flexible approach (1) is one that starts with a 3D-dissection of the building based on net areas. This results in a topology, the schematic model. Wall dimensions expand this topology in *generate geometry* to a geometry representing the gross area, the detailed model. We are currently working on this 3D-transformation. The second approach (2) starts with 2D-dissections of each storey separately and adding wall dimensions in *generate geometry.* This method is well established (Zimmermann, G., Suter, G. 2004), but has the disadvantage that the storey floor plans have to be adjusted interactively to align facades and some walls as for example load bearing and shaft walls. Both approaches allow for modifications of all dimensions and even in the topology in *modify geometry.* The third approach (3) is the least flexible. The 3D-dissection starts from the gross area and instead of a topology, the final geometry is fixed during *create topology.* In *generate geometry* walls protrude into the spaces without changing floor plan dimensions. *modify geometry* is not possible. In this paper we have used approach (2).

For efficiency purposes, wall dimensions and properties as for example thermal resistance or the glazing percentage are already defined during *create topology* and can be modified during the next two steps. Defaults exist for exterior and interior wall types and horizontal slab types. The advantage of early assignments of different types comes from the approach to propagate the types to all parts that are created during further dissections. If, for example, a new wall type is assigned to a facade, all segments of this facade inherit this type if no other type is assigned to a part of this facade. Properties of types can be changed at any time.

All three approaches use the same data structure and and can generate the same data for the simulation. It depends on the chosen simulation environment which file exchange format is chosen. If no further modifications are necessary, data for graphical CAD tools for the design development stage can also be produced.

The Building Planner

The Building Planner is a tool that was developed to aid designers in early stages of floor planning to quickly enter topologies into the computer and generate valid geometries. Valid in this context means that the spaces do not overlap, the total space is completely filled, and constraints are met. The topology is based on net areas, whereas the geometry includes additional wall and utility space areas and thus shows the total area. The algorithm for the transformation from topology to geometry relies on a limited shape flexibility of the spaces, expressed as shape functions. Therefore, floor plan geometries with different aspect ratios can be produced. The shape functions express constraints.

The Building Planner can be used for different purposes. In a very short time different floor plan topologies can be created and analyzed. Incremental changes are possible, especially changes in net areas of individual spaces or the width of circulation areas always lead to valid floor plans in the defined sense. With the resulting geometry data different analyses can be performed. For the purpose of performance analysis, simulator input data can be automatically produced.

For the latter purpose we have to look at the Building Planner's data structure. A prototype tool was implemented in Matlab, but here we will abstract from the implementation language. Also, details of the shape functions and the algorithms are of no importance for this purpose. More details can be found in (Zimmermann, G., Suter, G. 2004). We will first explain the general dissection based floor planning process and the resulting data structure.

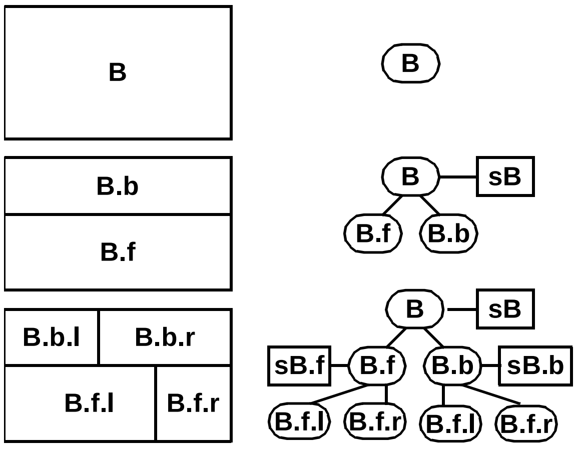

Figure 3 Dissected floor plan and dissection tree. Ovals denote spaces, rectangles denote dissection sheets. Orientations: f front, b back, l left, r right

To be more precise, we now define *sections* that are 3D spaces and *sheets* that are 2D spaces. The floor planning problem and the resulting data structures are 3D representations. Figure 3 shows a simple example. We start with a rectangular building section *B* with the size of the total net area. The aspect ratio is chosen close to the expected final floor shape, but this does not really matter. The first dissection sheet *sB* divides *B* into two sections *B.f* and *B.b*. The right side of Figure 3 shows the resulting dissection tree. After the third and forth dissection steps, a floor plan with four sections results.

The data structure in Figure 3 does not show the perimeter walls. In reality *B* is not the root of the dissection tree, but a *site* node with seven siblings: *B*, the *ground* and five surrounding sections (*topSur*, ...) as

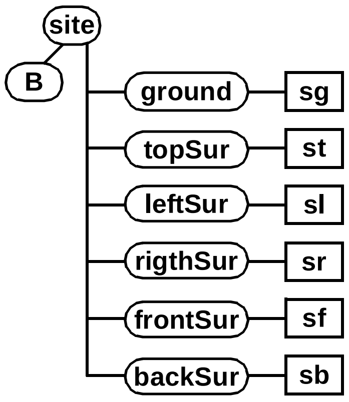

Figure 4 Generic dissection tree root structure

shown in Figure 4. The six sheets (*g.1*, ...) that represent the perimeter walls are not created by dissection and therefore are directly assigned to the six sections. Figure 3 and Figure 4 together define the dissection tree as it has been created so far.

For the purpose of performance simulation, the sheets have to be further refined. Especially for thermal calculations we need segments that uniquely define a heat transfer object between two sections. In Figure 5 this is shown for the first dissection sheet *sB*. The second dissection divides *sB* into *sB.l* and *sB.r*, the third dissection divides *sB.r* into *sB.r.l* and *sB.r.r*. Similarly, the six perimeter sheets have also been divided. This procedure guarantees that the leaves of all sheet subtrees have exactly two surfaces facing two sections. The leaves are called *Section Enclosure Segments* (*SES*).

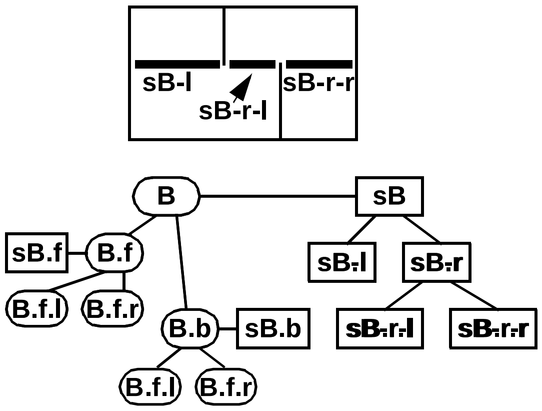

Figure 5 Sheet partitioning and resulting tree

It seems to be clear from Figure 5 which *SES* is adjacent to which sections. But this is only true for *sB-l* and *sB-r-r*. During the transformation from topology to geometry or due to manual or automatic changes of net areas or dimensions, the sheets *sB.f* and *sB.b* may change positions and instead of *sB-r-l* being adjacent to *B.b.r* and *B.f.l* in may have changed to *B.b.l* and *B.f.r*. Therefore, the adjacency relations are added after the topology-to-geometry transformation. Experienced designers will nevertheless try to start with a topology that correctly shows all final adjacencies.

generate geometry is the next step in the process. For this transformation the thicknesses of the walls are necessary. Since a building consists of a limited number of wall types, these are listed together with additional properties such as order, type, and thickness of the wall layers (also called construction), surface properties, glazing percentages or window types, number and type of doors and other openings etc. The properties depend on the type of simulation that is planned. As already mentioned, these data are propagated down the dissection tree during dissection if not otherwise determined by the designer. Defaults are defined for each building type. If the designer does not interrupt this process, all perimeter walls will be of one type, all internal wall of a second one. For connecting circulation areas and for creating non-rectangular shapes, also a virtual wall type is provided. In a first iteration, using the defaults is a very quick method to arrive at a valid geometry.

All properties can be interactively changed during *modify geometry* and the transformation is automatically repeated. This is a refinement process that will lead to final geometric floor plans.

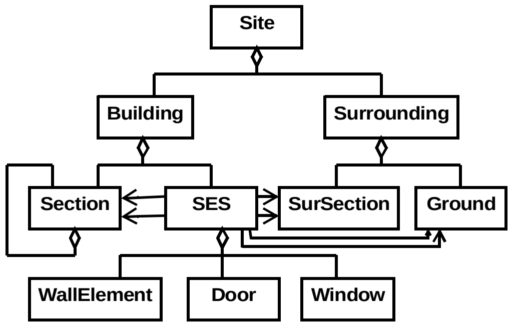

Figure 6 Object type diagram of data structure. Diamonds denote aggregation, arrows denote adjacency relations (front, back)

So far we have described instances of the data structure as a result of the simple example. Figure 6 shows its abstraction as object type diagram. It is a specialization of the Building System Model we proposed in (Zimmermann, G. 2003).

Figure 7 shows how the designer sees the example at this point in the process. It shows the main properties of the floor plan geometry. Now all adjacency relations are defined. For clarity we will distinguish red and blue dimensions. The blue dimensions describe the detailed model: the surfaces of all solid objects and thus also the dimensions of the net sections. The red dimensions relate to the schematic model. As Figure 7 shows, at the building envelope both models merge, inside of the building the red sheets basically mark the center planes of the solid objects. Thus, the red sheets

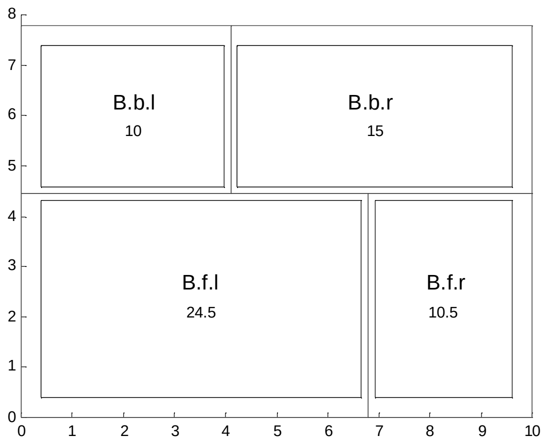

Figure 7 Floor plan geometry example

define sections that include all of the adjacent exterior walls and the adjacent slices of interior walls.

Objects of type *Section* are defined by the coordinates of the origin in the building coordinate system (x, y, z) and the dimensions of the red and of the blue section. Additional properties as for example thermal mass of additional objects in the section, heat sinks and sources, or light sources can be added if necessary for a simulation. They are stored as strings and can be copied directly into the input files of simulators.

Objects of type *SES* are defined by the coordinates of the origin and the dimensions of the two surfaces (blue) and the same set for the red sheet. Other properties include construction parameters that are necessary for the simulation. As in the case of sections, these properties are also stored as strings.

Adjacency relations between sections and SES instances are now completely defined. Some simulators generate these relations from the entered geometric data. In EnergyPlus they are part of the Surface records in the input file. In our simulation environment ViBuLab the objects and the aggregation relations are used to create the simulator object structure, the adjacency relations are used to create communication paths between objects.

Simulator Interface

We will demonstrate for the case of the simulator EnergyPlus and for our Virtual Building Laboratory which simulation data can be automatically generated.

EnergyPlus

EnergyPlus needs an input file (.idf) with all the information about the building that is necessary for the intended simulation. The format of this file is defined in (EnergyPlus 2004) as a list of records. All records consist of a keyword and a list of parameters. The file contains the following sections:

simulation parameters,

location, climate, etc.,
schedules,
surface construction elements,
zone description/geometry,
HVAC and lighting models and equipment

Example files show a considerable length. One example for a simple building with five rooms has about 5000 lines, each line a parameter entry. This size is mainly due to elaborate schedules and a large HVAC section. Both can only be the result of the design development and construction design stages. In the conceptual phase these have to be replaced by very simple defaults that can be used as a file template. Therefore, we have concentrated on the surface construction elements and the zone description/geometry sections.

Materials and Constructions of EnergyPlus are equivalent to *wallElement*s. Zones are equivalent to *sections* and can be easily generated. The geometry is described by surfaces that are directly generated from *SES* instances and adjacency relations. Since we found no distinction between the coordinates of the two surfaces of a wall in the EnergyPlus I/O description, we use the red sheet values although the blue coordinates would be more precise. Windows can be automatically generated with default z-coordinates and width depending on the width of the *SES* and the glazing percentage. The resulting input file for EnergyPlus can be easily edited with the IDFEditor that is provided with the tool set (EnergyPlus 2004).

At least for rectangular building geometries, this automatic process reduces the time from sketching to simulation considerably and provides the designer with a means to explore the design space with relatively precise performance analysis data.

The Virtual Building Laboratory

The long term goal of the ViBuLab (Mahdavi, A, Metzger, A., Zimmermann, G. 2002), (Zimmermann, G. 2003) is to model and simulate buildings as complete systems including user behavior. This is not possible with existing simulation environments. Using the advances in object oriented modeling, reuse, code generation, and object based simulation, simulators can be created with reasonable effort for specific buildings, tuned to specific analysis tasks. The special power of this approach is the ease with which continuous and discrete physical and logical effects and algorithms can be combined. We plan to also integrate human activities with control algorithms and building physics. So far we have shown that the approach works efficiently for performance simulations and control systems for buildings with up to 50 rooms and a time resolution in the range of seconds where necessary.

Here, we will limit the application to performance simulation during the conceptual design stage. Three tasks have to be solved in creating software for building simulation: Structure creation, behavior creation, and communication structure creation.

The structure of an object oriented software system is primarily defined by object type diagrams (class diagram) and secondarily by the instances (objects) that are created at run time. The building planner exactly contains this information. Therefore, the structure of Figure 6 is directly translated into an html table with a predefined syntax. The table also contains instance names that are either the automatically created names in Figure 7, names entered in the Building Planner manually, or generic names such as *rm1*, *rm2*, ... This table is read by a research tool called PROTAGOnIST (Metzger, A. 2004) that automatically creates structure diagrams in the modeling language SDL (Olsen, A. et al. 1994). Additional object types that are not modeled in the Building Planner such as light fixtures or heating equipment can be either added to the table in textual form or to the SDL diagrams in graphical form. In either case the other representation is automatically updated by PROTAGOnIST. More details can be found in (Zimmermann, G., Metzger, A. 2004).

All objects in SDL are software processes that are executed concurrently and can communicate asynchronously with each other. A typical object is a wall segment that is modeled by 5-10 layers. Each layer is modeled by an explicit linear equation. The behavior of the processes are defined as extended finite state machines in the form of state transition diagrams. This behavior is not part of the Building Planner and has to be entered manually for all object types. In Figure 6 we count 10 different types and for those a reuse library of behavior models exists for the typical physical effects. With every new effect that is needed for a building performance analysis this library is extended.

The next step is the definition of the communication paths. There are two possibilities in SDL. Channels with signals can be defined as part of the structure definition. We do this automatically following the aggregation hierarchy as shown in Figure 6. This results in a tree structure that is similar to the dissection tree. We mainly use this communication structure to automatically instantiate all objects by assigning concatenated instance names to all objects and by creating an index table in a top *simulator* object witch links these unique names with the unique process run-time identifiers (id).

Using this index table the *simulator* can read input files with data for named instances and send these data directly via remote procedure calls (RPC) to the object with the corresponding id. This is the second communication approach in SDL which is synchronous. With a similar procedure simulation results are sent from the objects to the simulator and supplied with time, object and parameter names written onto an output file. This file communication is used before the simulation time starts to send property values to all objects that are otherwise equal. During run-time this communication is used to enter weather data, or control interactions (automatic or manual).

Communication through channels can also be used to communicate for example between sections and SES instances. But because of our tree structure this could mean for example sending a signal from an *SES* first up to the site and the down to an external *SurSection*. To simplify this path, we also use the second communication approach with RPCs. In order to make this direct path from any object to any other object asynchronous, every object aggregates a send and a receive object.

The send objects have to know the ids of the receivers of signals that are sent by the parent. Since for example all instances of Sections are equal at the time of the simulator start, this information has to be entered from a file before the simulation time starts and is stored in tables in the individual send objects. The Building Planner provides this information using the adjacency relation.

As we have seen, most of the information for creating a specific simulator for a building instance is either provided automatically by the Building Planner or taken from a library of behavior models. Still, some manual interaction is necessary to complete the SDL model. This interaction is supported by the PROTAGOnIST tool set and by an SDL environment SDT (Telelogic). SDT also provides a code generator to translate SDL models directly into executable code and a run-time environment for the scheduling of the concurrent processes. The SDT run-time environment does not map processes and communication on the operating system, but provides a very efficient implementation as one total process.

Results

We claimed that the conceptual design process as shown in Figure 1 is so efficient that the creativity of designers is not reduced but increased. Since we found no data on the time it takes an experienced EnergyPlus user to create the input files for this tool, we can not do a comparison. Therefore, we can only support our claim with data from case studies. The case studies range from a simple 1-storey building with six rooms to a complex 3-storey building, as listed in Table 1. CS2 is a floor of an office building with two stair cases and a central hallway. CS3 is the second floor of Gaudi's Casa Calvet in Barcelona, taken from a floor plan without dimensions. Figure 8 shows the resultant floor plan geometry. CS4 is similar to CS2 in structure, but with three different storeys that were manually adapted by changing room areas until the outer dimensions matched. CS5 is not as regular as CS4.

To support our claim we measured the time needed for the manual steps *create topology* and *modify geometry.*

Table 1: Case studies

Study	Storeys	Activ. spaces	Total net area/m^2	Type
CS1	1	6	100	regular
CS2	1	24	400	regular
CS3	1	29	460	irregul.
CS4	3	61	1500	regular
CS5	3	60	1200	irregul.

Table 2: Efforts in minutes

Case Study	Space Prog., sketch	Topol. create	Geom. create	Object Struct. gener.
CS1	5	3	2	23
CS2	20	8	7	25
CS3	30	27	42	26
CS4	20	48	39	38
CS5	45	62	59	55

The steps *create space program* and *sketch floor plan* sketches are standard tasks in any design process and not part of our improvement. We have recorded the time for the mechanical parts of it, excluding the creative tasks. For the task *create simulator* we only have data for the object structure generation step.

If we assume that the creative task of creating space programs and floor plan topologies is in the order of several hours or even days, all reported efforts in Table 2 are within a reasonable range for extra time resulting from using the proposed tools. On the other hand, the advantage of getting valid floor plan geometries with the possibility to make changes for example to net areas, dimensions, aspect ratios, and material data with little extra effort more than compensates the spent time. The sum of the times in the three left columns also presents the time to create surface construction and zone geometry data for EnergyPlus simulations.

Conclusion

We presented as a result of interdisciplinary research a new idea for generating the input data from space programs and floor plan sketches for performance simulations during the conceptual design stage. We also demonstrated how prototype tools for this purpose can

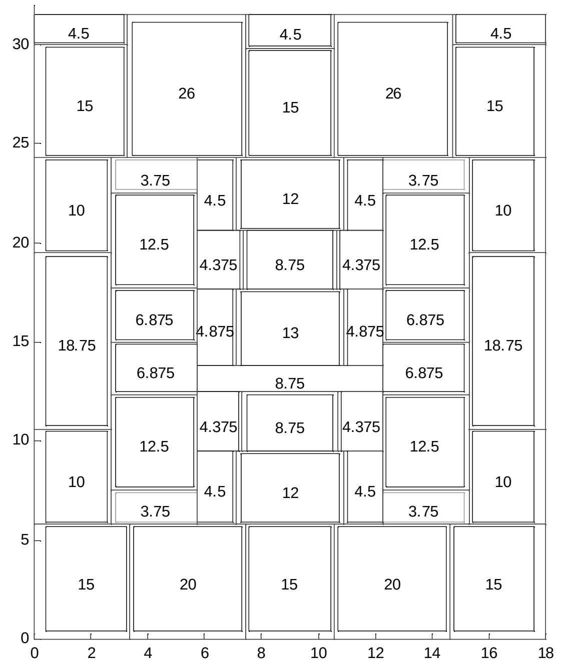

Figure 8 Floor plan geometry of CS3. Numbers are net area/m^2.

be interfaced with legacy tools like EnergyPlus or with another result of our research, an object-based simulator generation environment ViBuLab. We have shown preliminary performance data that indicate a sufficiently small overhead for using performance simulation during the conceptual design is possible. This allows architects to interactively modify early designs or explore design alternatives under the control of simulation. Further research and case studies are necessary to compare our results with traditional processes and to extent our simulation environment to additional effects and functions and to find the complexity limits.

References

EnergyPlus 2004. Input Output Reference, US Department of Energy, www.aenergyplus.gov

Fleming, U. 1986. On the representation and generation of loosely-packed arrangements of rectangles, Environment and Planning B (13) 189-205.

Fleming, U. et al. 1994. SEED-software environment to support early phases in building design, Proc. IKM94, Weimar, 5-10.

Lee, M.-C. 2001. SpaceMaker: A Symbol-based Three-dimensional Computer Modeling Tool for Early Schematic Development of the Architectural Design, http://depts.washington.edu/spmaker/

Liggett, R.S. 2000. Automated facility layout: past, present and future, Automation in Construction 9 (2000) 197-215.

Mahdavi, A. 1996. SEMPER: a new computational environment for simulation-based building design assistance, Proc. Int. Symp. of CIB W67, Vienna, Austria.

Mahdavi, A, Metzger, A., Zimmermann, G. 2002. Towards a Virtual Laboratory for Building Performance and Control, Cybernetics and Systems Vol. 1, Vienna, Austria, 281-286.

Metzger, A., Queins, S. 2002. Specifying building automation systems with PROBAnD, a method based on prototyping, reuse, and object-orientation, in OMER: Object-Oriented Modeling of Embedded Real-Time Systems, Koellen Verlag, Bonn, 135-140.

Metzger, A. 2004. Software-Qualitätssicherung durch Automatisierung – Ein modellbasierter Ansatz. Ph.D. Thesis. Department of Computer Science. University of Kaiserslautern.

Olsen, A. et al. 1994. System engineering using SDL-92, Elsevier, Amsterdam.

Polladis, S.N.,Bakos, Y. 1987. A framework for the design process, Havard Univ., Report LCT-87-2.

Rozenburg, N., Eekels, J. 1994. Product design: fundamentals and methods, John Wiley.

Suter, G., Mahdavi, A. 2003. Aspects of a representation framework for performance-based design. Proc. of the Eight International IBPSA Conference, Eindhoven, Netherlands, 1257-1264.

Schürmann, B. et al. 1992. Three-phase chip planning - an improved top-down chip planning strategy, Proc. 25th Design Automation Conference (DAC), Anaheim, USA, 60-65.

Tam, K. 1991. Simulated annealing algorithm for allocating space to manufacturing cells, Int. J. Prod. Res. 30 (1991) 63-87.

Telelogic. http://www.telelogic.com

Zimmermann, G. 1988. A new area shape function estimation technique for VLSI layouts, Proc. 25th Design Automation Conference, Anaheim, USA, 60-65

Zimmermann, G. 2002. Efficient creation of building performance simulators using automatic code generation, Energy and Building 34 (2002) 973-983.

Zimmermann, G. 2003. Modeling the building as a system, Proc. of the Eight International IBIPSA Conference, Einhoven, Netherlands

Zimmermann, G., Metzger, A. 2004. A software generation process for user-centered dynamic building system models, Proc. ECPPM 2004, 5th European Conference of Product and Process Modelling. Instanbul, Turkey.

Zimmermann, G., Suter, G. 2004. A Model for hierarchical floor plan generation based on shape functions, Proc. ECPPM 2004, 5th European Conference of Product and Process Modelling. Instanbul, Turkey.