Visual Programming meets Tangible Interfaces

Generating city simulations for decision support in early design stages

Gerhard Schubert[1], *Ivan Bratoev*[2], *Frank Petzold*[3]
[1,2,3] *Technische Universität München / Chair for Architectural Informatics*
[1,2,3] *{schubert|ivan.bratoev|petzold}@tum.de*

The utilization of visual programming languages (VPL) as tools for generating complex simulations has seen a constant increase in application in architect planning phases. The major advantage of such languages is, that they enable the user to create programs without needing traditional software development skills. In the last few years the CDP // Collaborative Design Platform was developed that seamlessly connects physical models with analyses and simulations in real-time. To facilitate an easier creation, modification and user interaction with the individual simulations, a VPL and an accompanying IDE were conceptualized and developed. In the context of this paper the core requirements, the concept and prototypical implementation of these new components are described in detail.

Keywords: *visual programming language, tangible interface, simulation, urban planning*

INTRODUCTION

Increasing requirements for architectural tasks, combined with a rapidly growing project size, lead to more complex conditions and connections in the planning and decision-making process. This applies to both, the conceptual urban planning phases, as well as to subsequent decision-making processes at the building scale. This rapidly changing situation requires totally new approaches and answers with regard to the design, planning and communication process. These changes do not only have an impact on the processes that are taking place, but also directly on the design and communication tools used.

The focus here is on the question how the effects of architectural and planning decisions can be made clear during the conceptual design phases, while at the same time an individual adaptation of the tools to the different design- problems is possible.

In addition to the consideration of established working methods and the tools used for design-thinking and design-working, the focus is primarily on the investigation of the digital methods that can be transferred to the conceptual planning context - the keyword is: Design Decision Support. A solution approach can therefore be found in the direct integration of decision-support tools like analyses and simulations into the early, creative planning- and design-process. In this way it is possible to visualize the effects of planning decisions directly and in early phases - both for decision-makers, as well as affected citizens or other stakeholders.

In order to meet the individual requirements of architectural design questions as well as solution approaches, in addition to the direct integration into the work process, a simple, flexible creation and adaptation of the analytical methods to the particular application is a prerequisite for such a system.

CONCEPT

Taking into account the situation just described, the proposed method / prototype provides an intuitive, collaborative design platform coupled with advanced, real time computer analyses and simulations - easy to create and adjust. Starting from the requirements in the architectural planning process and in the context of decision-support tools, two relevant requirements for decision support systems within the scope of architectural design tools for the exploration of ideas in a creative context can be identified:

- Direct embedding in the creative thinking process: The use of simulations and analyzes nowadays occurs almost exclusively in later planning phases - usually only for the verification of one or several already made decisions. This creates a linear process of "generating ideas", "detailing ideas" and final "evaluation" of these, without a direct feedback into the design process and thus without impact on the planning decision. In order to provide a direct feedback of analyzes and simulations into the thinking process, a seamless embedding of these in the design process is required, in order to bridge the existing gap between established design tools and digitally supporting tools (simulations + analyzes). In that way, an integrated decision support can be implemented directly in the planning process and the existing linear sequence can be replaced by a circular, decision-supported thinking process.
- Easily creation and customization of the tools: Each design task, design approach or concept is characterized by different requirements and premises. This means that the parameters to be checked by the analyses and simulations can not be determined absolutely universally. Rather, an individual system has to be used, which allows both: the simple creation and the easy adaptation of the necessary analysis tools. In this way, a flexible response to the most diverse requirements in the architectural design processes is made possible. One approach can be seen here in the application of a VPL (Visual Programming Language) as a simple, code-free method of implementing different user-specific analyzes and simulations.

RELATED WORK

The concept of utilizing visual languages for programming purposes is not a new phenomenon. A few VPLs and their environments where selected, based on their already commercial use in the field of architecture or their theoretical potential. They were observed and analyzed how they solve issues during the different types of validation.

Grasshopper (1) is one such tool, which is widely utilized for parametric modelling, lighting performance analysis and other similar evaluations that architects traditionally utilize during the design stages. Grasshopper offers a very intuitive way to utilize even the most complex features that it offers. By the integration of individual plugins the functionality is easily extendable. (see Figure 1)

A similar tool is Revit's Dynamo (2). It focuses on utilizing not just geometry objects, but applying the building information modeling (BIM) on top of the shapes. With the rise in popularity of the tool it started integrating cross-platform features to work directly with Grasshopper or to directly use the same third party tools its counterpart does (Kensek 2015). (see Figure 1)

VCCL (Preidel and Bormann 2015) is a VPL used to define building codes and guidelines for construction projects. It then uses it's development environment to continuously validate the current construction project if it fulfills these requirements in real time. The language offers also the possibility to visualize intermediate results of the validation process, which enables the user to highlight the parts of the project that directly violate said rules.

The USP (Seifert and Mühlhaus 2014) focuses on building regulations as well. Its primary goals is to observer how changes to those regulation would affect potential existing buildings. This is achieved through a two mode environment, the first part consisting of a VPL that allows users to define any regulations that they want to observe and the ways they want to modify them. The second mode interprets the context of the programmed regulations and visualizes them. These rule sets can be then applied to any concrete model that the environment supports.

One issue that all of the above mentioned VPLs and their development environment share, is that they utilize only the traditional ways of receiving input from the user. All construction projects and buildings are defined digitally either through preexisting files or through the tools themselves. This is in stark contrast to the way architects work during the early design stages, where physical objects are used to represent buildings and concepts. The proposed VPL attempts to bridge this gap between the physical models and digital real-time interactive simulations.

SYSTEM SETUP

Based on the presented requirements a two-part system architecture was conceptualised and developed as a part of a research project.

CDP / Design Platform

The basis of the project is an interactive design and communication platform for early urban development phases (e.g. scale 1:500 / 1:1000) that was conceived and implemented in the past few years (Schubert 2014). The focus is on the direct coupling of physical work models with interactive simulations

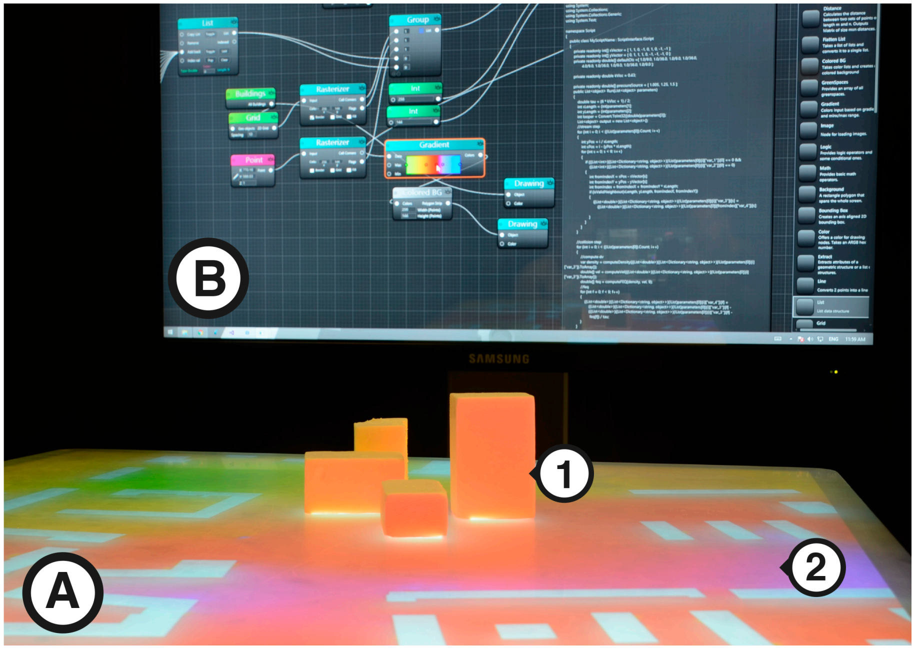


Figure 1
System setup: CDP // Collaborative Design Platform (A) - Physical models (1) on a multi touch table (2) are digitally reconstructed in 3D and realtime and serve as the simulation-base. Visual Programming Interface (B) - Easy setup and adjustment of analyses and simulations (e.g. noise) via nodes. The output of the calculations is displayed on the table-surface (2) and thus within the physical model (1).

based on a tangible interface. The physical model becomes the basis for the simulation: changes of the model (repositioning, cutting, rotating) have a direct influence on the analyses and simulations in real time. The calculation results are displayed in real-time in the physical model, adding additional layers of information to the model. Details of the physical - digital coupling and the framework of the CDP can be read in the following publications: Schubert et al. 2011b, Schubert et al. 2013, Schubert 2014.

VPL / Visual Programming Language

In order to meet the changing requirements of different building tasks and, to enable the different user groups to make individual adaptations, a visual programming language (VPL) and an accompanying integrated development environment (IDE) have been conceptualised and implemented. The choice to create a new VPL is based on the traditional user-friendly interface and the minimal technical knowledge required to program with such languages (Hils 1992). To fulfill the requirement of real-time interactability with the simulations the IDE uses the VPL as an interpreted language, which trades the compilation and the associated performance gain for a fluid, uninterrupted user interaction. This also offers an easier reusability of simulations and higher productivity from the user (Ousterhout 1998). This solution serves as an add-on to the main framework (see Figure 2). The language and IDE have as a primary goal to circumvent the existing development process of simulations for the CDP (Schubert et al. 2011a), which requires not only in-depth knowledge of the implementation of the framework but also has to be recompiled every time a modification is made to the code.

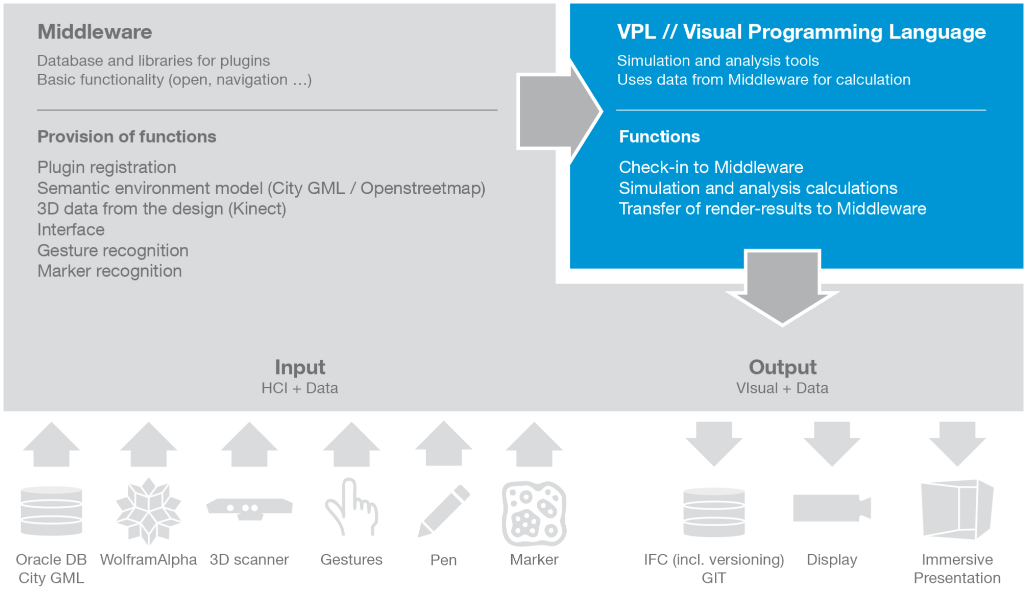

Middleware
Database and libraries for plugins
Basic functionality (open, navigation ...)
Provision of functions
Plugin registration
Semantic environment model (City GML / Openstreetmap)
3D data from the design (Kinect)
Interface
Gesture recognition
Marker recognition
VPL // Visual Programming Language
Simulation and analysis tools
Uses data from Middleware for calculation
Functions
Check-in to Middleware
Simulation and analysis calculations
Transfer of render-results to Middleware
Input
HCI + Data
Output
VIsual + Data
Oracle DB
City GML
WolframAlpha
3D scanner
Gestures
Pen
Marker
IFC (incl. versioning)
GIT
Display
Immersive
Presentation

Figure 2
Software Framework: Build up on a Plugin Framework, the VPL is directly included in the main-application

To fulfil the preconditions of the creative design phases and the limitations of the framework, the following requirements were determined:

- programming operations are represented by nodes (see Figure 3), which have different inputs, outputs and control values.
- every change in the input of nodes, or the provided context of the CDP, is processed and visualised in real-time, without having to interrupt the design process.
- the interface is designed to offer architects, familiar with Grasshopper or similar applications, a recognisable layout
- the VPL and IDE are extensible, the development of further nodes or features requires minimal extra effort.

TECHNICAL REALISATION

The IDE implements an Interface of the CDP. This offers an easier integration of the IDE as a pseudo simulation plugin for the framework. Through the interface, the IDE has direct access to all events, information and interactions on the main framework. For the visualization of the environment the Windows Presentation Foundation (WPF) is used.

Each syntax component of the VPL utilizes a main Interface that enables the easier interpretation and interaction between each other. These syntax components are then visualised by so-called Node visual elements in the IDE. Each Node contains a Grid Element in which all further syntax specification are placed. These specifications can be easily represented through traditional User Interface (UI) Elements like buttons, sliders, check boxes, etc. To exchange information between each Node we use a set of input/output connectors that exchange information. How that information is interpreted, utilized and stored is left to the implementation of each Node.

If we want a syntax component to react to a change or interaction in the framework or have access to some information, e.g. building information, then they implement one or more of the six available sub-interfaces. Each node implementing them will be called from the IDE when a specific event is sent through the Framework Pipeline.

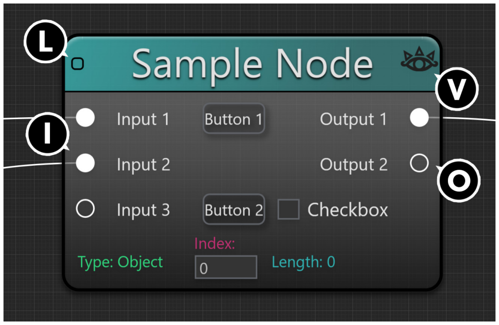


Figure 3
Sample node layout: (I) Input connection, (O) Output connection, (L) Loop Trigger, (V) visibility. The node itself can be used for static input e.g. via Buttons, Checkboxes or Input-fields.

Although the environment mimics functionalities of similar programs, it also offers a variety of features that are unique for the needs of the project. Five of the currently 35 implemented components are described in detail. (see Figure 4)

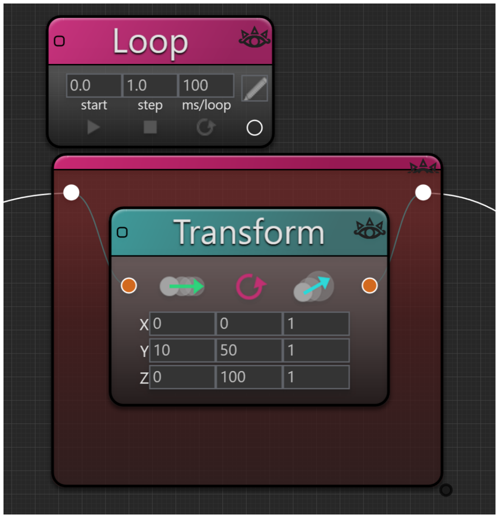
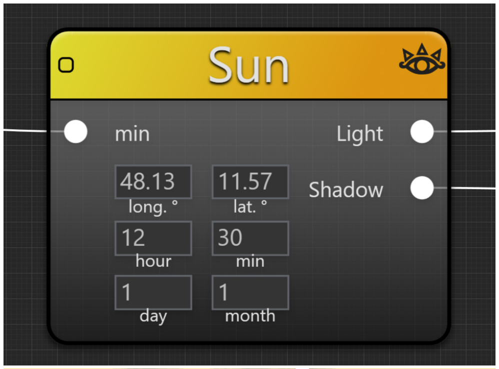


Figure 4
Loop Node: Increments a counter that has a start and step value each time cycle. Alternative executes nodes in the Loop box each time cycle.



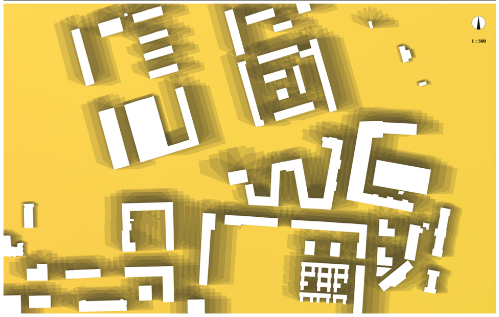
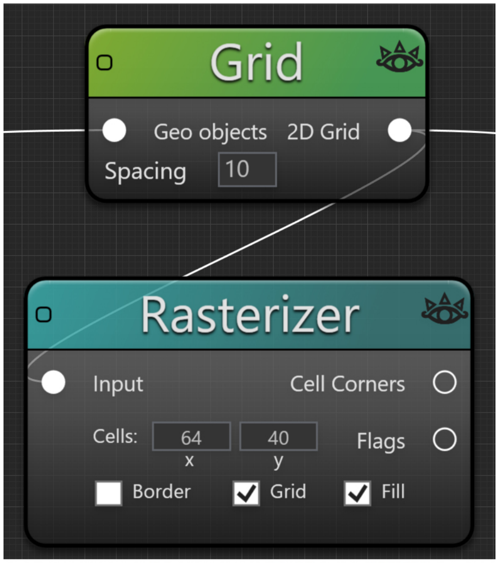


Figure 5
Sun Node:
Computes the shadows and the sun light for a given latitude and longitude position and the day of the year.

Additionally if the user wants to see how the shadows change throughout the day in real-time, they can add an input that represents how many minutes have passed since the specified point. To visualise the simulation the node provides two outputs, one for the shadows and one for light colour. (see Figure 6)

Rasteriser and Grid Node
The grid node takes any geo-object and generates 2D points that are inside of the surface of the object. The spacing value defines the exact distance between two points.



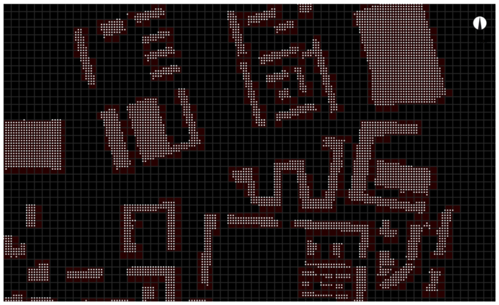


Figure 6
Grid Node:
Represents each object through points with defined spacing between them. Rasterizer Node: Discretizes space in X*Y Cells.

Loop Node
The node starts a timer that increments a counter each loop cycle. By defining the step value, a user can define the increments with which the counter will increase. The ms/loop value specifies how often this will happen in milliseconds. The current value of the counter is then given as an output. Alternatively, it is possible to bind other nodes, that will be executed every cycle of the loop.

Sun Node
This node is an example of how a whole simulation can be represented as a single node. It uses all geo-objects sent to this project from the core framework. The user can specify the geographical position of the objects and the time and date for which it should compute the shadows. (see Figure 5)

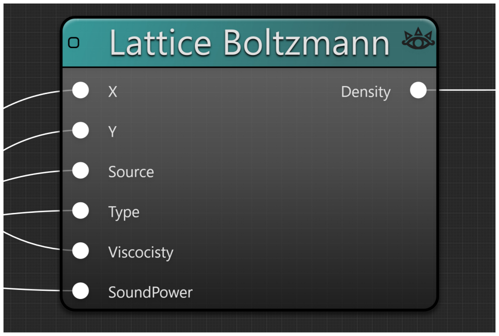
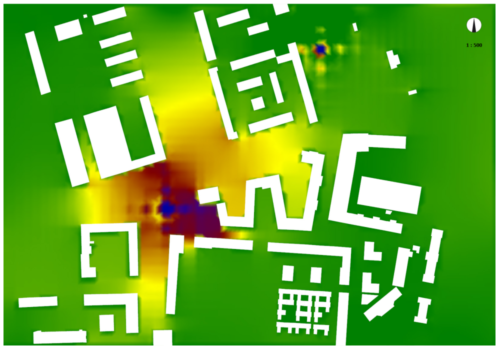


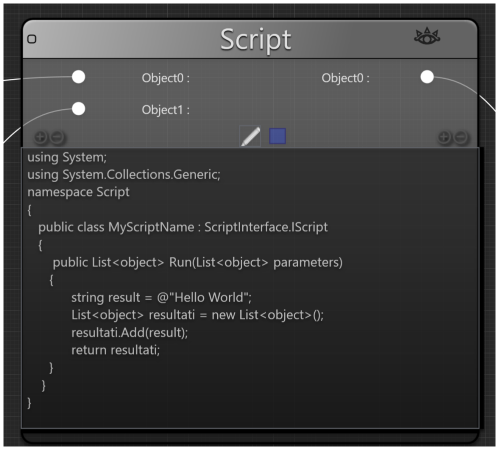
Figure 7
Lattice Boltzmann Node: Simulates one step of the LBM using a D2Q9 model with BGK for collision operation. A full simulation can be ran with extra utilization of the Loop Node.



The rasteriser node performs two sets of operations. First it discretises the screen space based on the x and y value. Then it takes the provided set of 2D points and marks each cell that contains at least one point as full. Because most types of simulation that use a discretisation of space have boundary conditions, the user can request, that all cells lying on the border of the screen domain are marked as such and override the previous check. It provides an output of two equally sized sets that contain the screen coordinates for the top left corner of each cell and what type of cell it is. The rasteriser node also provides a set of visualisation options. The grid option shows the size of each cell, while the fill function shows which cells would be marked as empty/full. (see Figure 7)

Lattice Boltzmann Node

The Lattice Boltzmann Node implements the identically titled method from the computational fluid dynamics field. It uses a D2Q9 model for the discretization of the velocity and space (He and Zou 1997) and the Bhantnagar-Gross-Krook model (Bhatnagar et al. 1954) for its collision operator. It assumes that all sources of disturbance are based on increased pressure, and can be defined by the SoundPower input value. The Viscocisty input can control the type of substance that are simulated. The Source and Type inputs are bitmasks that define for each cell if it is a solid, fluid or border and independent of that if the cell is a source of disturbance to the simulation. The two bitmasks should have the same size as the product of the X and Y inputs, that define the discretization in 2D. The output is the result of one iteration of the simulation represented through the density of each cell.



```
using System;
using System.Collections.Generic;
namespace Script
{
  public class MyScriptName : ScriptInterface.IScript
  {
     public List<object> Run(List<object> parameters)
     {
          string result = @"Hello World";
          List<object> resultati = new List<object>();
          resultati.Add(result);
          return resultati;
     }
   }
}
```

Figure 8
Script Node: Compiles and executes C# code with user defined amount of inputs and outputs

Script Node

The script node provides the user with an option to create their own "mini-programs" that they can execute as part of their simulation. The node takes the input, output and source code, written in C#, and compiles a small library that it then executes with the parameters provided by the user. With this node, the user can define complex mathematical operations and simulations without having to extend the core framework or the project. Continues real-time use can be achieved by linking it with a Loop Node. (see Figure 8)

SUMMARY

Within the framework of the project shown here, an interactive, visual programming interface was designed and implemented on top of the existing CDP // Collaborative Design Platform. The implemented prototype clearly demonstrates the potential of an iconic, visual programming interface for the creation and also for the adaptation of digital analyses and simulations. The implementation as a reactive computing software is of particular importance here and allows direct feedback from the visual programming interface. The direct coupling to a tangible interface expands the use of the CDP // Collaborative Design Platform enormously and enables laymen (such as stakeholders, authorities, etc.) to be directly involved in the planning process, taking into account objective criteria. First applications e.g. noise simulations, and shading analyses show the potentials as well as new fields of application.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the development support by Alejandro Segura Rueda and Fabian Frank Petter for the co-development of the VPL and IDE as part of their master course.

REFERENCES

Bhatnagar, PL, Gross, EP and Krook, M 1954, 'A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems', *Physical Review*, 94(3), pp. 511-525

He, X and Zou, Q 1997, 'Theory of the lattice Boltzmann method: From the Boltzmann equation to the lattice Boltzmann equation', *Physical Review E*, 56(6), p. 6811

Hils, DD 1992, 'Visual Languages and Computing Survey: Data Flow Visual Programming Languages', *Journal of Visual Languages and Computing*, 3, pp. 69-101

Kensek, K 2015, 'Visual Programming for building information modeling: energy and shading analysis case studies', *Journal of Green Building*, 10(4), pp. 28-43

Ousterhout, JK 1998, 'Scripting: higher level programming for the 21st Century', *Computer*, 31(3), pp. 23-30

Preidel, C and Borrmann, A 2015 'Automated Code Compliance Checking Based on a Visual Language and Building Information Modeling', *Proceedings of the 32nd ISARC*, Oulu, Finland

Schubert, G 2014, *Interaction forms for digital design: a concept and prototype for a computer-aided design platform for urban architectural design scenarios*, Ph.D. Thesis, Technical University of Munich

Schubert, G, Artinger, E, Petzold, F and Klinker, G 2011a 'Tangible tools for architectural design - seamless integration into the architectural workflow', *Proceedings of the 31st Annual Conference of the Association for Computer Aided Design in Architecture*, Banff, pp. 252-259

Schubert, G, Artinger, E, Petzold, F and Klinker, G 2011b 'Bridging the Gap. A (Collaborative) Design Platform for early design stages', *Proceedings of eCAADe 2011*, Ljubljana, Slovenien, pp. 187-193

Schubert, G, Riedel, S and Petzold, F 2013 'Seamfully connected | Real working models as tangible interfaces for architectural design', *Proceedings of CAAD Futures 2013*, Shanghai, pp. 210-221

Seifert, N and Mühlhaus, M 2014 'Decision support for inner-city development – An interactive customizable environment for decision-making processes in urban planning', *Proceedings of eCAADe 2014*, Newcastle, UK, pp. 43-52

[1] http://www.grasshopper3d.com/

[2] http://dynamobim.org/