DATA INDEPENDENT MODEL STRUCTURE FOR SIMULATION WITHIN THE VIENNA UT MORESPACE PROJECT

Benjamin Rozsenich[a], Salah Alkilani[b], Martin Bruckner[c], Štefan Emrich[d], Gabriel Wurzer[e]

[a][b] Institute for Analysis and Scientific Computing, Vienna UT, Austria
[c] dwh GmbH - Simulation Services, Vienna, Austria
[d]Chair of Real Estate Development and Management, Vienna UT, Austria
[e]Digital Architecture and Planning, Institute of Architectural Sciences, Vienna UT, Austria

[a][b][d][e] {benjamin.rozsenich|salah.alkilani|stefan.emrich|gabriel.wurzer}@tuwien.ac.at
[c] martin.bruckner@drahtwarenhandlung.at

ABSTRACT

The Institute for Analysis and Scientific Computing at Vienna University of Technology (Vienna UT) has created models to simulate lecture room reservations, in order to test strategies for increasing the efficiency of utilisation, increasing booking fairness and perceived capacity without need to add more lecture rooms. Thus, the produced system had to satisfy some tough constraints: (1.) It had to be flexible and multifunctional, meaning that a dynamic code structure was needed so that program logic could depend on the imported data, but simulation algorithms would stay data-independent. (2.) The presentation logic had to be customized to fit the usability needs of the client, who would perform the simulation experiments himself. To report on both programming techniques and architectural decisions that enabled us to achieve these constraints is thus the main goal of this paper.

Keywords: customization, simulation, software architecture

1. INTRODUCTION AND OVERVIEW

For the past years, researchers over various institutions of Vienna UT have been working together on a project that that optimises utilisation of lecture rooms, by performing simulations on booking requests instead of having them reserved in a 'first-come-first-served' fashion. Together with dwh Simulation Services, this has led to the production of the novel simulation suite, on whose development this paper focuses.

During the initial steps of developing a new simulation, much effort has to be devoted to the correctness of algorithms. Two models had to be produced (see "Simulation Core", Section 2):

- a *scheduling model*, which performs static scheduling based on a set of heuristic rules (Section 2.1)
- a *pedestrian dynamics model*, which performs physical simulation of students moving within the built environment, according to their lecture timetable (Section 2.2).

Producing a working simulation core is necessary, but not sufficient for a piece of software to become a product (see Section 3, "Building MoreSpace"):

- *Data import and export* have to be tailored to the customer's needs, e.g. reading from and writing to databases that already exist. As a matter of fact, this interfacing aspect becomes a software in its own right, which needs to be written and supported specifically for the client in question (3.1, "Input/Output configuration").
- Orchestration of the simulation core's algorithms, given the often-changing demands and requirements of the client, can be seen as yet another required task. On the one hand, this means producing code that will call upon the pre-existing algorithms in an order that produces seemingly special-tailored simulations. On the other hand, such models need to have a user interface that meets the requirements of the (not necessarily technically skilled) simulation user. What is thus required is a rather non-technical step in which an analyst *customises* the software so that the inputs needed, simulations performed, and outputs generated clearly reflect what the customer has in mind (3.2, "Simulation customization").
- Once simulation results have been gained, *analysis* over what scenario gives the best performance has to begin (3.3, "Analysis"). There is, however, no absolute best - only alternative solutions that have to be weighted by the simulation user, according to practicability in implementation. We provide a set of visualisation views, intended for in-depth analysis and justification. The latter aspect is most important when a single result has to be selected and exported for use in a production system on which thousands of users rely (in our case: the reservation system of Vienna UT).

As a side-note for this paper, we wish to state that related work is given directly within the narrative. Additional material is to be found at the end of the paper (see Section 4), before the conclusion (Section 5).

2. SIMULATION CORE

The core of our simulation consists of two models that assign lectures to lecture rooms and let students simulate their would-be passages through the built environment, given the assigned lecture locations and per-term timetables. Both models are implemented in Java, and packed as Java libraries that ship with the application. Changes in the simulation core are thus only possible by using application updates, and only if they affect all clients (not individual customers). For the sake of completeness, we now describe the two models used in some detail:

2.1. Scheduling model

The scheduling model has the task of filling predefined slots, each one measuring half an hour and being within given lecture room of finite capacity, with lectures. In a pre-step, so-called booking requests have to be generated by Vienna UT's reservation system, each one stating *when*, *how long* a room *of what capacity* is needed in which building (*location*), given what *special equipment*. The scheduler then arranges the lectures so as to satisfy one of its implemented constraints, which we give here in simplified form:

- *Greedy scheduling*. Capacity, equipment and location of the booking request have to be satisfied exactly when finding a slot.
- *Tolerant scheduling*. The capacity of the booked room may be more (but not less) than the requested capacity, furthermore, the booker can be tolerant concerning the location.

An additional parameter controls whether the scheduler can *shift* lectures by half an hour plus and minus, in order to see whether the result becomes any better. The same goes for *splitting*, i.e. using multiple rooms for the same event, in case the booking request requires a capacity beyond available rooms.

In all cases, the result of the booking process is a number of successfully booked requests plus any leftover requests that require some (manual) work.

2.2. Pedestrian dynamics model

This model calculates the time that students need to change lecture rooms (see Figure 1). In detail, what is computed are the trails of each virtual student, given his/her timetable as input. This transition between different lecture rooms is governed by three layers of increasing complexity: The movement is calculated with reference to a physical model by (Blue and Adler 2001). Above that, route choice and movement along a circulative network is computed by using graph algorithms, quite similar to (Tabak 2009; Tabak, de Vries and Dijkstra 2010; Wurzer 2011). At the highest level, individual behaviour is governed by the timetable of lectures to visit. If there is more than one lecture that takes place in an instant, the student's probability of changing rooms is taken into account. This aspect could also be called a fourth layer, that of individual choice or preference. Extended work on this route choice, specifically for urban environments, has been done by (Dijkstra, Jessurun, Timmermans and de Vries 2011). More details on the used pedestrian model can also be found in (Bruckner, Tauböck, Popper, Emrich, Rozsenich and Alkilani 2012; Bruckner 2009).

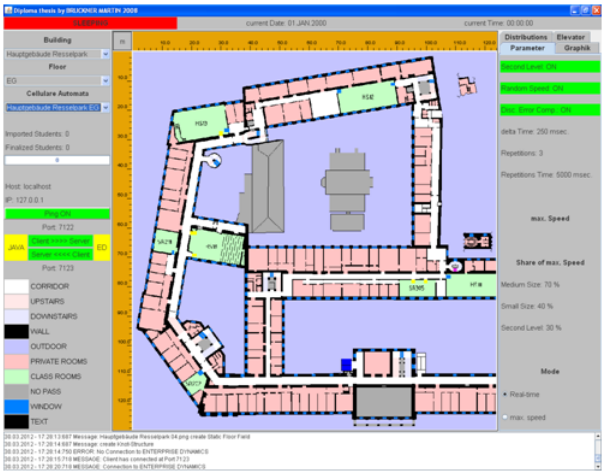

Figure 1: Pedestrian dynamics model

3. BUILDING MORESPACE

The product which we have designed, called MoreSpace, is a platform that glues different simulation algorithms together by making use of a common scripting platform (Rhino for Java). Conceptually, one might think of a common data structure (coined quite paradoxically as '*the data independent model*') existing as a database and a sequence of simulation algorithms acting on it. The following sections describe, in detail, the setup and workflow connecting the different components of our system, whose reusable and extensible architecture is the main contribution we would like to share. An overview of this workflow is also given in Figure 2:

1. External data stores hold the customer-specific data, which is converted to our data independent model upon import (see Section 3.1).
2. Simulation algorithms acting on our data structure are producing results within the same database (see Section 3.2).
3. These are visualised, in order to select one specific result that fits the end user by whatever criteria he/she sees fit (see Section 3.3).
4. Additional simulation types, such as dynamic simulation using pedestrian dynamics models, can also be incorporated into a post step, if they would take too long to compute to be practical for every simulation run.
5. The export process then records the chosen result as selected by the user, which it then

writes to a data store defined by the client. This might be the same as during import, or a different one (again see Section 3.1).

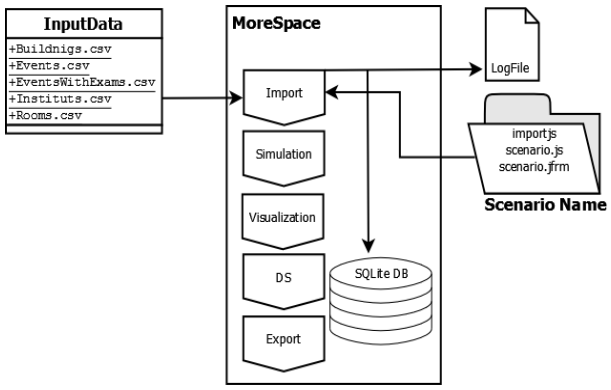

Figure 2: Workflow and outline of MoreSpace App

3.1. Input/Output configuration

Even though our application can have a different user interface and simulation setup for each customer, we rely on a data structure that is static and common to all installations. Once filled, the different simulation models perform their work on this structure, and save their results back into it. We employ the popular SQlite database together with an Object/Relational mapping tool (Apache Cayenne) for this purpose, giving each part of the application easy access to our entities (booking request, space, booked slot, etc.).

The complexity of the import/export process lies in the details of the (external) data stores, whose merging and querying has proved to be a consulting problem of its own. For us, the same application must be able to be used at any university, e.g. the Technical University of Vienna but also in other companies - each having an own data storage type available (e.g. Excel, database, XML). Before being able to simulate, we thus need to import booking requests from whatever source available into our common data structure. The same procedure will be required reverse upon export of the simulation results, either into the same data store or into a different one.

Usually, transformations of data structures are handled by special Extraction/Transformation/Loading (ETL) software packages. We have chosen a different approach, by defining the customer-specific import and export script in Rhino that does the necessary steps, tightly integrated into the application (File>Import and File>Export option). We also have a user interface intended at filtering and selecting data from this external data store so that it fits the scope of the simulation project.

3.2. Simulation customisation

A booking product such as ours must deliver a *project-specific* output, and thus requires also a project specific setup which we call *customisation*. Such a specially tailored approach is not new – it is used e.g. in accounting software (SAP ERP), Hospital Information Systems and further products that are not by definition finished 'off the shelf'. Simulation models have been and continue to be an example of such software, being highly specific to the customer and scenario in mind. We extend this notion by introducing a platform in which simulation can be used, i.e. glue between pre-existing models and the customer-specific setup, which is yet unprecedented to the best of our knowledge. In detail, we are using three essential concepts throughout our application:

- *Scenario*: a named sequence of invocations of a variety of simulation and algorithms which has a specially defined user interface where one can define inputs that are not hardcoded but rather given as parameters. Colloquially, each scenario stands for a given problem complex which is due to be analysed.
- *Experiment*: a specific choice of parameters for a given scenario. This scenario can contain several experiments, but at least one (the default experiment having default parameters).
- *Result*: the effect of an invocation of an experiment. Every result is reproducible by using the defined parameters and settings of the experiment again. In this case, the already existing data is overwritten. If two experiment runs with the same parameters are to be compared nevertheless (e.g. for validation), one must duplicate the experiment.

Each scenario is represented by a Rhino script, which invokes all the necessary simulation algorithms iteratively. Multiple simulation runs and aggregation are handled here, as well. As a general rule, scripts to not act directly on the database: They rather call upon algorithms, which load data, compute, and write back into the database, using the experiment ID as conceptual key for the result.

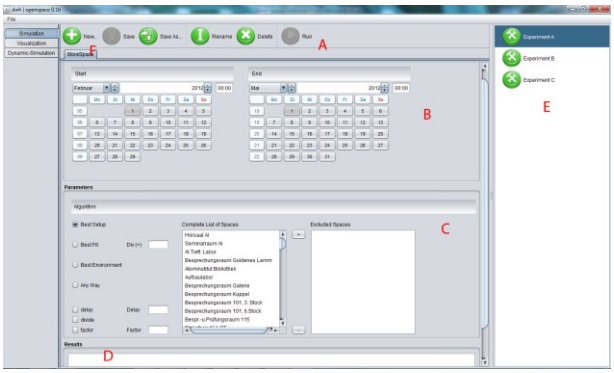

Figure 3: MoreSpace Graphical User Interface. (A) Experiment toolbar for creating and running experiments, (B) filter and (C) parameterisation user interface loaded for each scenario (D) experiment run output console (E) result panel (F) scenario selector.

Each scenario also has a graphical user interface that is dynamic and has been defined to suit the client's needs (also refer to Figure 3). We employ the Abeille Forms designer for that purpose, which outputs XML that can

be read back to construct Java Swing panels using the jGoodies forms library.

Semantically, a scenario user interface is always divided into two parts: the *filtering part* (Figure 3B) for selecting the data on which the simulation acts, and the *parameter part*, where one can set input data (Figure 3C). Using the experiment toolbar (Figure 3A), one can then create experiments and run them, producing results under the results panel (Figure 3E) and summary console (Figure 3D). This process can be repeated for every scenario that is defined (each being represented by a tab, as in Figure 3F).

3.3. Analysis

Without proper interpretation, the results produced during by the simulation remain useless. Therefore, our application has a visualization dashboard (see Figure 4) in which the produced results can be reviewed and compared by the client or analysts acting on his behalf.

The actual visualisation types used are currently bar charts, line charts, Gantt charts (see Figure 5) and histograms, in both single-experiment and multi-experiment layouts. Each visualisation window retains its position and state even if the program is closed. This enables analysts to show why a certain result was chosen without having to perform data analysis in front of the client.

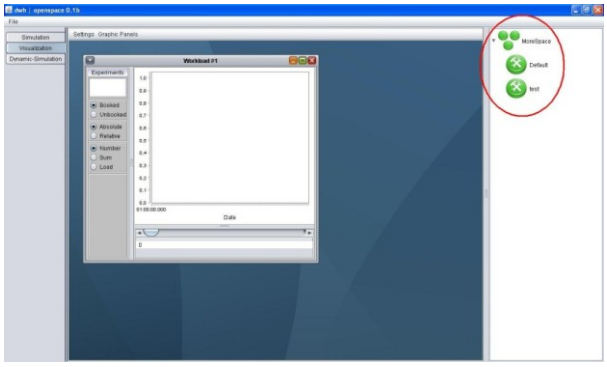

Figure 4: Visualisation dashboard for interpretation of results (circled).

3.4. Summary and big picture

Having presented the core as well as different components of the application, we may now begin to summarise and compact what has been said. One of the interesting aspects of the program, which applies well beyond the borders of our software, is the ability to initialise itself with a completely new user interface and simulation logic specially customised to the client. This means that the system is changing its nature without affecting a single line of java code of the core (which would require recompilation). Technically, this is done by bundling a set of scripts and user interface definitions together with the program, which we read that the programs start.

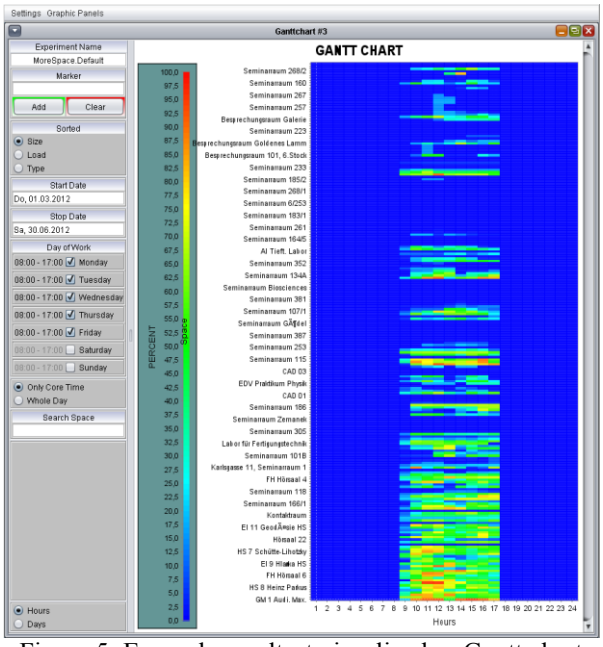

Figure 5: Example resultset visualized as Gantt chart.

In detail, we proceed as follows:

- For every scenario, the must exist a scenario script (‘scenario.js’) as well as filter (‘filter.jfrm’) and scenario user interface definition (‘gui.jfrm’) in a folder having the scenario name.
- Furthermore, we define to special folders containing the same files by the name of ‘Importer’ and ‘Exporter’, which contain the logic and user interface necessary for interfacing to the client's data infrastructure.

On start-up, the application knows only how to browse through a set of folders, instantiating scenarii (tabs) and importer/exporter (File>Import and Export) as it goes along. When it has finished doing so, the user is presented with a specially tailored application that is not only customised in its function, but also in language and terminology used. Updates to the functional core are kept at a minimum - what is changed lies mostly in the orchestration part (i.e. scripts), to be elaborated together with the client. The product is thus the executable artefact that incorporates the core library, while everything else belongs to the domain of a project.

3.4.1. Additional implementation notes

Some additional notes given here, although not important for the general picture, might be beneficial when developing a similar application:

- The data structures used for storing experiments and results sacrifice functionality for ease of implementation: One experiment may only have zero or one results. If an experiment has no result (i.e. it is new), its parameters may be updated. In all other cases, the experiment is said to be fixed - no alterations can be made, since that would corrupt the result data. One may, however,

duplicating an experiment (using a Save As.. option), in which case the parameter settings are copied to a new experiment which can be altered.

- The scenarios have a caveat as well: During import, we allocate a special scenario 'Scenario 0' containing a predefined 'Result 0', which is the result of a *manual assignment* of lectures to rooms. This Result 0 of Scenario 0 may not be overwritten; it may only be used for comparison with the actually simulated results.
- Another caveat concerns to input data; should these change, all experiment runs are obsolete. Therefore, a re-import is only possible in order to add data, not for update. Should this be desired, the current database has to be archived and reset (i.e. cleared). This also happens once data has been exported, in order to be ready for another simulation period.

4. BACKGROUND AND RELATED WORK

Our system operates according to the Model-View-Controller (MVC) concept (refer to Figure 6): *Model* objects are the parts of the application that implement the logic for the application's data domain. *Views* are the components that display the application's User Interface (UI). The UI is created using the model data as input. *Controllers* are the components that handle user interaction and update the model (Shelly 2005).

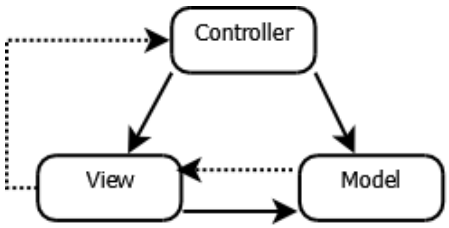

Figure 6: MVC

We take the notion of MVC one step further: Our model is a database with a static structure. The model itself has a delegate which we have called *Importer*, which maps from customer specific data sources to this format. The view and the controller are both put inside the scenario scripts, which are essentially client specific. The meaning of this reflects our project experience in dealing with customisation: a large part of the logic is special tailored and cannot be reused. Instead, we see it all customised called as being yet another delegate to the model, intended for rapid development and (if need arises) redevelopment.

Another topic of which we are well aware is that our base technology *scripting* has large issues with debugging. The used Rhino is an exception here, since there is the excellent Rhino debugger (Oliver and Boyd 2012) that can be utilised. Rhino itself is a Java implementation of JavaScript bundled with the Java Platform, therefore being widely available without additional installation.

Coming to the related work, we are not aware of any approach is similar to our 'glue code scripting' / rapid user interface development technique. It is certainly true that, for example, Anylogic (Borshchev and Filippov 2003) offers a similar experience when coding using the simulation's core library. However, Anylogic is using Java - a compiled language. Thus, customisation can only happen to very limited extent, off-site, where all the development tools are installed. Furthermore, the user interface tools given are very limited, non-scriptable.

As a second contrast point, the application area of Anylogic's core is currently concentrating on ABM, SD, DES. This is certainly an advantage: Functionality exists and can be readily used. At the same time, such closed and pre-existing Application Programmer's Interfaces (APIs) also have a drawback - extending them further might be very complicated, since the source is not available. We are thus taking pure Java as basis, aiming at optimal control of the employed algorithms and source availability rather than functional superiority. The latter may be achieved by incorporating code from a variety of open-source packages such as NetLogo (Wilensky 1999), or put differently: "*An open architecture allows developers to design systems that are made up of many small functional modules interconnected by a common software interface*" (Roschelle, DeLaura and Kaput 1996).

From a development perspective, combining Java and JavaScript also enables new forms of working together among distinct roles: A developer, can implement the core functions of software while a customizer or analyst oversees the design of the application by scripting (*Separation of Concerns*).

5. CONCLUSION

We have presented a simulation called MoreSpace, that is designed to be dynamic in its appearance and in relation to how simulation algorithms are employed. Upon bootstrapping itself, the application looks for specially-named script files together with user interface definitions, which it then goes on to interpret and display. Data is dynamically loaded from proprietary sources into its own comon data structure serving all simulation models, which are orchestrated within the script files. Such a high degree of customisation has previously only been found in Enterprise Resource Planning (ERP) and healthcare IT, which is why we see our efforts as being important not only for our application, but also in the light of the ongoing modularisation that modelling packages embrace nowadays.

REFERENCES

Blue, V.J., Adler, J.L., 2001. Cellular automata microsimulation for modeling bi-directional pedestrian walkways. Transporation Research Part B 35 (2001), 293-312.

Borshchev, A., Filippov, A., 2003. From System Dynamics and Discrete Event to Practical Agent

Based Modeling: Reasons, Techniques, Tools. *Proceedings of the 22nd International Conference of the System Dynamics Society*.

Bruckner, M., 2009. *Modeling of pedestrian dynamics in the university operation using cellular automata in the programming language JAVA*, Master Thesis, Vienna UT.

Bruckner, M., Tauböck, S., Popper, N., Emrich, Š., Rozsenich, B., Alkilani, S., 2012. A combined Cellular Automata and DEVS simulation, MathMod 2012 Full Paper Preprint Volume, Argesim Report S38, Available from: http://seth.asc.tuwien.ac.at/proc12/full_paper/Contribution356.pdf [accessed 1st June 2012]

Dijkstra, J., Jessurun, A.J., Timmermans, H.J.P., de Vries, B., 2011. A Framework for Processing Agent-Based Pedestrian Activity Simulations in Shopping Environments, *Cybernetics and Systems*, 42 (7), 526-545.

Oliver, C., Boyd, N., 2012, Rhino Debugger, Available from: http://www.mozilla.org/rhino/debugger.html [accessed 1st June 2012]

Roschelle, J., DeLaura, R., Kaput, J., 1996. Scriptable Applications: Implementing Open Architectures In Learning Technology, Proceedings of Ed-Media 96 - World Conference on Educational Multimedia and Hypermedia, 599-604.

Tabak, V., de Vries, B., Dijkstra, J., 2010. Simulation and validation of human movement in building spaces, *Environment and Planning B: Planning and Design*, 37 (4), 592-609.

Tabak, V., 2009, *User Simulation of Space Utilisation*, PhD Thesis, TU Eindhoven.

Wilensky, U., 1999. Netlogo, Available from: http://ccl.northwestern.edu/netlogo [accessed 1st June 2012]

Wurzer, G., 2011. *Prozessvisualisierung in der Krankenhausplanung*, PhD Thesis, Vienna UT.

AUTHORS BIOGRAPHY

Benjamin Rozsenich got his master degree in medical computer science from Vienna UT in 2010, finishing with his diploma thesis 'Statistical module of the tumor documentation system "HNOOncoNet" based on JBoss Seam and Hibernate concepts' for the Vienna General Hospital (AKH Wien). For the Morespace Project, he has been active as lead developer and technical architect, a job which he now continues in the private sector.

Salah Alkilani has a Bachelor in Software Engineering from Vienna UT. During the MoreSpace project, he has been working on the static scheduler, in which topic he has also published first papers. He is also involved in lecturing for the Afro-Asiatic Institute of Vienna University on the impact of the Arabic spring and the role of Islam in society.

Martin Bruckner studied mathematics at Vienna UT, receiving his master in 2009. His work on 'Modeling of pedestrian dynamics in the university operation using cellular automata in the programming language JAVA' has been used in the pedestrian simulation of the MoreSpace project, and beyond, for his PhD thesis that is currently under preparation. He currently works for drahtwarenhandlung simulation services, the company participating in the MoreSpace project.

Štefan Emrich has a master degree in mathematics (2007), where he was focusing on prediction of influenza epidemics using cellular automata and agent based systems in a comparative manner. After his degree, he went on to ETH Zurich, where he was involved in research on modelling workforce and workflow management. Returning to Vienna, he began his PhD on 'Simulation for Space Management' in the Real Estate Department of the Faculty of Architecture, in which function he also participated in the MoreSpace project. He is author of numerous articles on evaluation of usage in buildings, and was awarded the Austrian Construction Prize (Baupreis) in the category "best junior scientist" in the year 2011.

Gabriel Wurzer earned his Ph. D. degree in Process Visualization and Simulation for Hospital Planning from Vienna University of Technology in 2011. His research in architectural sciences focuses on tool support for early-stage planning of complex buildings, with regular contributions to both Pedestrian and Evacuation Dynamics conference (PED) and the Education and Research in Computer Aided Architectural Design in Europe conference (eCAADe), from which he was awarded the Ivan Petrovic Prize in 2009. He is also an active researcher in archaeological simulation, together with the Natural History Museum Vienna.