MODYPLAN: Early-Stage Hospital Simulation based on Treatment Chains

Gabriel Wurzer*. Wolfgang E. Lorenz.*
Matthias Rößler, Irene Hafner**, Barbara Glock**, Martin Bruckner**, Niki Popper****

**Vienna University of Technology, Architectural Sciences (E259), Vienna 1040, Austria (Tel: 01-58801-27201; e-mail: <firstname.surname>@tuwien.ac.at).*
***DWH Simulation Services, Vienna 1070, Austria (e-mail: <firstname.surname>@dwh.at) and Vienna University of Technology, Analysis and Scientific Computing (E101), Vienna 1040, Austria (Tel: 01-58801-10115; e-mail: <firstname.surname>@tuwien.ac.at).*

Abstract: Discrete-event simulation of hospitals typically specifies flow by means of a process graph through which patients are routed. While this is generally fine for models in which processes are clearly defined, e.g. smaller units such as emergency departments, it falls short of capturing the fact that a patient can in principle go from one unit to any other unit if medical procedure requires it. This problem becomes even more evident when modeling a whole hospital, at which level specifying each individual treatment through directed edges and nodes becomes unviably complex. In the past three years, we have been developing a combined hospital simulation/space design tool in which processes are defined individually by patient, as sequence of visited units imported from the Hospital Information System (treatment chains). The major advantage of this approach lies in the fact that we can now design a future space layout in which assumed capacities (staffing, equipment, required space) can be tested against the recorded patient volume, using an Agent-Based Simulation to re-enact each treatment chain. In contrast to other hospital simulations, this approach targets early stages of architectural conception, during which the actual building structure is elaborated. Using the method, we can compare and contrast different layouts during that stage, making simulation a driver for design rather than a tool for late optimization within the final floor plan.

Keywords: Hospital planning, design tool, treatment chain, Agent-Based Simulation, space layout planning.

1. INTRODUCTION

In this paper, we draw together the lessons learned during the development of our combined hospital simulation/space design suite called *Modular Dynamic Planning and Simulation for Health Care Facilities*, (MODYPLAN) which is based upon the idea of *schedule-based simulation* (see section 2) using the actual treatment data of a clinic, even though in anonymized form. One of the main realizations leading to the development of the tool was that the hospitals are not conceived from scratch - they are based on pre-existing spatial templates (e.g. Neufert 2000) which are adapted to fit the expected patient volume and specialization. There are, nevertheless, a variety of possible options to choose from, leading to the questions of "*How to compare two spatial layouts?*", which is where simulation comes into play: By letting agents simulate their individual schedule (coined as *treatment chain*, see section 3), they utilize spaces which have a finite capacity. By recording this utilization, space occupancy and agent histories among different layouts, the program can compare the "performance" of each of them.

From a project view, we have just finished the core implementation and are currently modeling a hospital as a test case. Section 4 draws together some insights from this process as well as on the visualization which we are currently completing.

2. RELATED WORK

Schedule-based simulation has been used for some time to simulate occupant behavior in buildings, mostly for obtaining the predicted energy demand. For example, Page (2007) models the presence of building users in each space and correlates that through a set of stochastic models to each individual type of energy-consuming activity (e.g. light switching, window opening). Tabak (2008) concentrates on surveying and classifying office tasks in order to be able to accurately model them. One of the advantages of his approach is that it takes interactions between agents into account (e.g. shared activities such as meetings). Goldstein et al. (2010) generate fictional schedules that reproduce recorded activities (the authors call this "schedule-calibrated" simulation). In more detail, the authors present a stateful approach where each activity can be based upon past behavior. This is to some extent comparable to having a memory within the simulated entity (i.e. in a token, process execution or agent) to base decisions on.

Schedule-based simulation with special regards to the clinical domain has been addressed to a much lesser extent. On a regional level, Miksch et al. (2014) have presented an Agent-Based Simulation framework for simulating medical service provision in the context of epidemiology. However, their model is purely abstract, with no reference to a space layout. Likewise, Einzinger et al. (2014) have simulated whole

health care systems using routine health care data from reimbursement claims as a basis. On the local level of an individual clinic, we are so far the only group that has been doing schedule-based simulation, to the best of our knowledge (Wurzer, Lorenz and Pferzinger 2012).

Typically, hospital simulations use process notations (e.g. Simeone et al. 2012) or other forms of flow graphs (Friesen and McLeod 2014) to supply the logic behind the simulated patient flow. They are applied in late stages of planning, i.e. within the finished floor-plans, for detailing of processes and quantification of resources. By contrast, our method is applied in early phases of hospital planning, for establishing and comparing several preliminary space layouts (of which several variations may exist, see Wurzer 2013). To be fair, these space layouts are still very approximate, and thus running simulations in them merely gives qualitative rather than quantitative results (Wurzer and Lorenz 2013). Nevertheless, applying simulation early has the advantage that the design can be changed easily, in order to avoid planning errors.

3. PROPOSED METHOD

Our method is based on *treatment chains* (figure 1a), i.e. sequences of functional units that a patient visits during his clinical encounter. Each *Functional Unit* (Figure 1b) is a capacity-constrained resource which receives these visits (immediately or after queuing). The service duration is given in the treatment chain (e.g. in figure 1a: 4 time units in Neurosurgery followed by 30 time units in Rehab).

Functional Units have named purpose ("function", e.g. Examination), a boundary (closed polyline) and can be nested, thus forming a spatial hierarchy. They are the elements of the space layout, i.e. named spaces augmented by their role as resource. On a regional level, we may model each clinic as Functional Unit having its departments as sub-units (e.g. in figure 1: sub-units Neurosurgery, Orthopedics and General Surgery in unit North Tyneside, Neurorehabilitation and Neuropsychiatry in Walkergate Park).

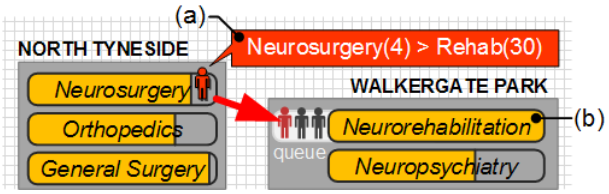

Fig. 1. (a) Treatment chains as list of Functional Units to visit. Durations spent in each Functional Unit are given in parentheses. (b) Functional Units as resources having a finite capacity, after which a queue is formed.

For each of the functional units, utilization and occupancy of its underlying space is recorded over time (the latter through a raster of a given resolution, e.g. 0.5m). Each agent, acting on its individual treatment chain, furthermore records durations spent queuing, receiving service and spending time transitioning. Functional Units furthermore directing the agents furthermore write the intermediate goals into the agents' histories. All of these values are later used in the comparison of different planning scenarios (see section 4.2), i.e. different configurations of Functional Units in which the same treatment chains are simulated.

3.1 Processes moved into agents

If we consider a process simulation based on nodes and directed edges, we have a class structure as shown in figure 2a: A process is specified by nodes and connecting edges, or more precisely: Subclasses of *Node* which have a specified behavior, as for example a *DecisionNode* that chooses an outgoing edge for the process Execution, possibly based upon data that is contained in *Execution:context* (think of this as a key-value container).

Given this structure, the previously stated problem (patients sent from one unit to any other unit if required by treatment) maps to the following two issues:

1 In order to model processes realistically, one would need to introduce additional edges that lead to nodes invoking a process in another unit of the hospital. Referral processes such as 'emergency treatment' will hence fan out to include every possible department. Likewise, the department processes must also introduce edges for back-referral.

2 Within each process changed in this respect, the decision logic must now take the additional edges into account. There are two places in the class structure that are affected, first: the data contained in *Execution:context* and second: the node's decision logic called upon by *execute()*.

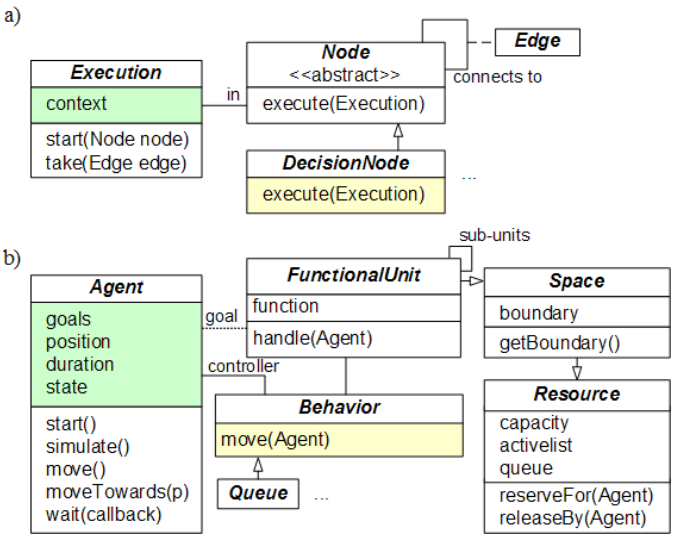

Fig. 2. Class diagram for (a) process simulation (b) agent-based simulation based on treatment chains.

Both issues make the processes difficult to read: Additional edges clutter the process description; additional logic that concerns another process is hard to understand. Furthermore, process descriptions are cumbersome to change: In the worst case, one has to go through *each process* to see whether it is affected. Such changes are, however, very frequent in early

planning, where hospital units are created, split, merged and so forth.

"What if one could free the process of explicit transitions and shift this responsibility to another entity?", was our primary thought when looking at this problem. In an agent-based simulation, the *Execution* becomes an *Agent* (see figure 2b). It is typical that each agent has a list of consecutive goals to which it moves, where each of the goals carries some additional significance (e.g. waypoint, service point where interaction happens). In our model, the treatment chain becomes this goal list. Furthermore, each goal references either a *FunctionalUnit*, (a *Space* with a polyline boundary which is also a capacity-limited *Resource* within the simulation) or a *Waypoint* (a point, not shown in figure 2b).

Each Agent starts at the first goal, which is typically the entrance of a hospital. In case this entrance is not present in the chain, one can always prepend it (preprocessing). Goals coming after that are the actual visited Functional Units, except the last one, which is the exit. Below algorithms give the functions used by agents to process their chain:

```
function Agent:start() {
  if goals not empty {
    fetch next goal and duration
    position = center of goal
  }
}
function Agent:simulate() {
  if controller is set {
    controller.move(myself)
  }else{
    myself.move()
}
function Agent:move(){
  if goals is empty {
    remove myself from simulation
  } else {
    if goal refers to a FunctionalUnit {
      fu = get FunctionalUnit for goal
      if inside fu.boundary {
        fu.handleAgent(myself)
      }else{
        moveTowards(fu.center)
      }
    }else{ // goal is a Waypoint
      wp = get Waypoint for goal
      if position is wp.center {
        wait for Agent.duration
        fetch next goal and duration
      }else{
        moveTowards(wp.center)
      }
    }
  }
}
```

The agent always starts at the center of the first goal and moves, with each step of the simulation, towards the of the next goal. In case the goal is a waypoint, then the agent moves to its center and fetches the next goal. If it refers to a FunctionalUnit, we first move the agent until it reaches the boundary and hand control over via the *FunctionalUnit:handleAgent(Agent)* function, which resets the state of the agent and sets its controller to the FunctionalUnit's *Behavior*, which from now on governs the movement of the agent:

```
function FunctionalUnit:handle (Agent) {
  agent.state = INITIAL STATE
  agent.controller = my behavior
}
```

The default behavior, *Queue*, lets the agent queue at the boundary of the Functional Unit until it is free. Then, it moves the agent to its center and lets it wait for the given duration. As convention, this is always 0 for entrances and exits (first goal and last goal of a treatment chain), since we want arrivals and exits to happen instantaneously. As last step, the Agent's controller is cleared and it resumes normal operation using the next goal:

```
function Queue:move(Agent) {
  fu = get FunctionalUnit for myself
  if Agent.state is INITIAL STATE {
     fu.reserveFor(Agent)
     Agent.state = ACQUIRED RESOURCE
  }else{
     if Agent.position is fu.center {
       wait for Agent.duration
       fu.releaseBy(Agent)
       clear Agent.controller
       Agent.fetch next goal and duration
     }else{
       Agent.moveTowards(fu.center)
     }
  }
}
```

The reason for outsourcing the Functional Unit's behavior is that there are a variety of options to choose from. For example, the Functional Unit can have a waiting zone where the queuing actually takes place. It could have a prioritization (e.g. Manchester Triage) and so on, which are scenarios that are hard to describe for the general case but easy to implement in an own sub-class of *Behavior* which pulls the strings.

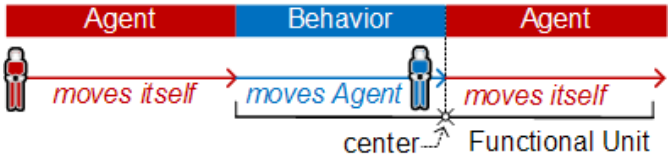

Fig. 3. Control logic of the method: An *Agent* moves until it reaches its goal *FunctionalUnit*, at which spot the Functional Unit's *Behavior* takes over until resource utilization has been simulated.

A summary of the control logic shared between Agent and the Behavior acting on behalf of its Functional Unit is given in figure 3. During the travel to a Function Unit, the agent is

in control. By entering the Functional Unit, control passes to its Behavior until the agent's utilization of the Functional Unit has been fully simulated. The nice thing about this approach is that Function Units can now be characterized independently of their process, i.e. have agents say *what* is utilized and have spaces state *how* this is happening. The resulting design may be saved as a template (Christiansen and Bruce 2008), to be reused in other designs that are subject to different treatment chains.

3.2 Functional Unit Hierarchies

Functional Units are composed into a hierarchical space layout where each sub-unit is completely contained in its parent unit. The hierarchy enables us to specify capacities either *implicitly* or *explicitly* (refer to Figure 4a):

- Implicit Functional Units specify a capacity greater than one, signifying that there are multiple resources that can be served simultaneously (left part of Figure 4a). Often though, planners want to see these resources explicitly - as they mostly represent spaces. Therefore (see right part of Figure 4a), users can place k Functional Units with the same function, which is the same as specifying one Functional Unit of capacity k. The difference between the two options is that, visually, the agents move to the center of each Functional Unit, and hence the sub-units can be thought of as individual "rooms" serving a common purpose. As caveat, the implementation of *FunctionalUnit:getBoundary()* must either return its actual boundary when capacities are implicitly stated, or the parent boundary for the explicit case. The reason is that the agent has to stop at some point at the border of a Functional Unit, which is the parent's boundary in that case.
- At run-time, each agent must resolve the name of its goal so that it can get the corresponding Functional Unit. For example (see Figure 4b), consider a clinic having an Emergency Department (ED) and a Neurology (N), both with sub-units for Examination (EX). If an agents' treatment chain contains "EX", then name resolution will give different results depending on where the agent stands: We first look at sub-spaces, then consider the higher hierarchical levels until a Functional Unit with the correct function has been found (in all other cases, we raise an error). For Neurology, this would choose the examination room in that area, while the Emergency Department would give its examination room to the agent. If it is required to switch between departments, one must prepend the department name (here: N > EX > ED > EX).

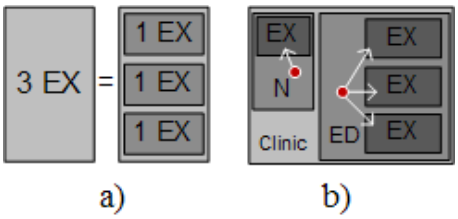

Fig.4. (a) Implicit and explicit capacities. (b) Function resolution at runtime through search in the hierarchy below the current Functional Unit of the agent.

3.3 Comparing different layouts

In order to compare different layouts, we rely on three types of data:

- The utilization of each Functional Unit (see left in figure 5a) leading to an utilization graph (right in figure 5a), e.g. as plot over time or as piechart showing the share of time that the Functional Unit was overutilized.
- The occupancy of each Functional Unit (left in figure 5b), increased when an agent enters the boundary. This can be shown e.g. absolute or according to the increase per hour (see right in figure 5b).
- Agent histories record the time spent utilizing, transitioning and queuing (left in figure 5c). Additionally, each agent has a specified type (e.g. trauma patient, regular patient). Histories can be grouped by that type, in order to arrive at the maximum queuing times before Functional Units or the maximal treatments.

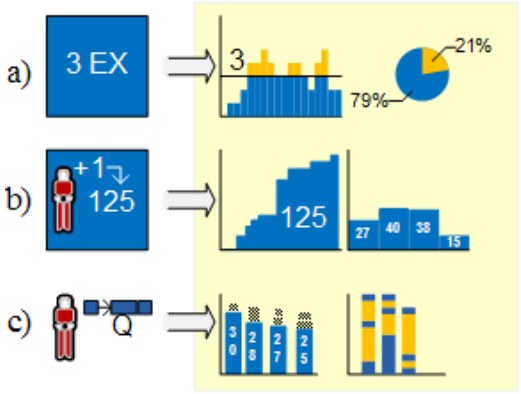

Fig.5. (a) Utilization and (b) occupancy of Functional Units as well as (c) agent histories lead to a visualization that can be used to characterize each given scenario.

In order to compare different layouts, we use side-by-side view in which Functional Units can be dragged in from two different layout trees (refer to figure 6): The left tree depicts the current layout, the right a comparison layout. The user can choose to drag in units from the same tree to also perform a comparison in one layout.

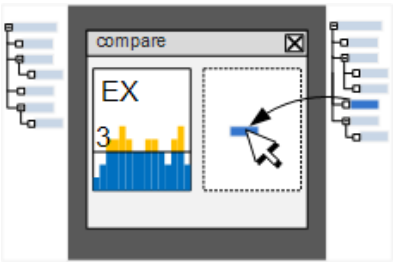

Fig.6. Comparing two layouts side-by-side.

4. DISCUSSION

The presented simulation has been implemented as software and is currently being tested with sample clinics. We herein wish to share some interesting observations that we have made during that phase.

4.1 Importing Data

Treatment chains are imported from Hospital Information Systems, or rather: they are exported as spreadsheet and interpreted by our application. An example for this is given in table 1: Here we have two anonymized patients given by id 1234 (a trauma patient) and 2345 (a regular patient). The first has the treatment chain [Entrance]>1ED2ADM(1 minute)>CXRAY(0 minutes)>[Exit] and the second one [Entrance]>1ED2ADM(0 minutes)>[Exit]. The problem is here that we have only limited information concerning the durations spent in both cases. We know for sure that the patient 1234 has spent 1 minute in 1ED2ADM (which is the Emergency Departments second admission counter), but we do not know how long he was in CXRAY (central radiology), since there is no following timestamp on which to base durations on. The same is true for the second patient, who has only one goal in his/her chain.

Table 1. Sample data in a typical format

timestamp	id	unit	type
2014-09-01 13:30	1234	1ED2ADM	TRAUMA
2014-09-01 13:31	2345	1ED2ADM	REGULAR
2014-09-01 13:32	1234	CXRAY	TRAUMA

We have learned that this is not the exception but the rule for such data. In order to compensate these "zero durations", we have included a field in *FunctionalUnit* that gives the *typical average duration of service*, which is used in such a case. Also, the duration between two timestamps may include waiting times; we have included a Boolean *override duration* flag in each Functional Unit, for cases where this is typical (classical example: x-ray examinations take anything between 3 to 5 minutes, but waiting times may take hours).

4.2 Comparison between two layouts

We have designed different visualizations for being able to depict the recorded simulation histories stated earlier (see section 3.3). The design options were shown to planners, in order to have an expert opinion on their value and be able to implement one option fully. We are currently in the phase where we have completed several of these interviews, and wish to share some preliminary results of these.

At first, we had an "overall" comparison that computed a score for the topmost Functional Units, taking all subunits into consideration. This was, however, not informative, since planners want to be able to "drill into" the data and consider sub-units individually. Also, the expectation of always comparing two different layouts was wrong, surprisingly, as planners also wanted to compare the performance of functional units *of the same layout*. Thus, we now aim at giving both options in one common comparison interface (sketched in Figure 6).

We also found that individual treatment chains were of less interest; what is considered most important are the spaces and their utilization (queues!), followed by occupancy. An extension to visualization of causalities (think: who traveled through a specified Functional Unit, where was he before and where did he go afterwards), in line with Wurzer and Lorenz (2014) is being considered at the moment.

4.3 Relationship with other tools on the market

Our tool is insofar unprecedented as there is currently no simulation based on treatment chains, and no simulation serving as driver for preliminary design (i.e. before and after the tendering phase of an architectural project. Some examples of competing applications are: Programs that offer "sketch-like" Computer Aided Design functionalities (e.g. the ONUMA System [http://www.onuma.com] which provides a web-based editing environment that can be used for early design) or discrete simulation packages that can be used for modeling passages between hospital service units (e.g. FlexSim Healthcare [http://healthcare.flexsim.com]). However, both types of systems do not form a closed cycle - one can import the spatial layout into a simulation but the reverse way of using the simulation results (e.g. space utilization and occupancy for determination of space sizes) is not possible. This is where we see the value of our own software.

5. CONCLUSIONS

We have presented a schedule-based hospital simulation method which relies on treatment chains stemming from a Hospital Information System. In contrast to classical process-based simulation, we have no notion of edges; the transitioning between nodes is rather based on an agent's prescribed schedule, making it easy to capture the real flow between different units, departments or even hospitals. The approach has been implemented as our own simulation suite, ((MODYPLAN)), which we will bring to the market in spring of next year.

ACKNOWLEDGEMENTS

This paper was financed by the ZIT MODYPLAN project, under the call Smart Vienna 2012.

REFERENCES

Christiansen, C., and Bruce, S. (2008). A Template for Change, *Healthcare Design Magazine*, Issue February 2008. www.healthcaredesignmagazine.com/article/template-change [accessed 1st October 2014]

Einzinger, P., Jung, R., and Pfeffer, N. (2012). Modeling Health Care Systems – An Approach Using Routine Health Care Data, *Preprints of MathMod 2012 (Full Paper Volume)*, 500:1-500:6. seth.asc.tuwien.ac.at/proc12/full_paper/Contribution500.pdf [accessed 1st August 2014]

Friesen, M.R., and McLeod, R.D. (2014). A Survey of Agent-Based Modeling of Hospital Environmentss (sic!), *IEEE Access*, (2), 227-233.

Goldstein, R., Tessier, A., and Khan, A. (2010). Schedule-Calibrated Occupant Behavior Simulation, *Proceedings of SimAUD 2010*, 3-10.

Miksch, F., Urach, C., Einzinger, P., and Zauner, G. (2014). A Flexible Agent-Based Framework for Infectious Disease Modeling, *Lecture Notes in Computer Science 8407*, 36-45.

Neufert, P. (2000). *Bauentwurfslehre*, 36th edition.

Page, J. (2007). *Simulating Occupant Presence and Behaviour in Buildings*, PhD thesis, Ecole Polytechnique Federale de Lausanne, Switzerland.

Simeone, D., Kalay Y.E., Schaumann, D., and Hong, S.W. (2012). An Event-Based Model to Simulate Human Behaviour in Built Environments, *Proceedings of eCAADe 2012 (Volume 1)*, pp. 525-532.

Tabak, V. (2008). *User Simulation of Space Utilisation*, PhD thesis, Einhoven University of Technology, Netherlands.

Wurzer, G., and Lorenz, W. E., 2012. Pre-Tender Hospital Simulation Using Naive Diagrams As Models, *Proceedings of the International Workshop on Innovative Simulation for Health Care 2012*, Dime Università Di Genova, 157-162.

Wurzer, G. (2013), Versioning in Early-Stage Hospital Simulation, *Proceedings of the 2nd International Workshop on Innovative Simulation for Health Care (sic!)*, Dime Università di Genova, ISBN: 978-88-97999-26-3, 7 pages.

Wurzer, G., and Lorenz, W. E. (2013). From Quantities to Qualities in Early-Stage Hospital Simulation, *Proceedings of the 2nd International Workshop on Innovative Simulation for Health Care 2013 Athens*, 22-27.

Wurzer, G., and Lorenz, W.E. (2014), Causality in Hospital Simulation Based on Utilization Chains, *Proceedings of SimAUD 2014*, Tampa, 161 - 164.

Appendix A. AUTHORS BIOGRAPHIES

Gabriel Wurzer has done his Ph. D. on *Process Visualization and Simulation for Hospital Planning* (Vienna UT, 2011). His research in architectural sciences focuses on tool support for early-stage planning of complex buildings.

Wolfgang Lorenz has a Ph. D. degree in Architecture for his thesis on *Analysis of Fractal Architecture using the Box-Counting Method* (Vienna UT, 2014). He is lecturer on programming for architects and simulation.

Matthias Rößler received his Master's degree on *Coupling of Thermodynamical Systems* (Vienna UT, 2012) and is currently working on simulation models for DWH Simulation Services as well as on clinical information systems.

Irene Hafner has done her Master's degree on *Co-Simulation using the Building Controls Virtual Test-Bed*. Her current research concerns cooperative and multi-rate simulation, healthcare and production planning.

Barbara Glock Barbara Glock has gotten her Master on *A System Dynamics Model of the Prevalance of Obesity in Austria* (Vienna UT, 2014). Her current work is in the field of coupling different modelling methods.

Martin Bruckner is a specialist on Agent-Based Modeling currently working for DWH Simulation services. He has completed his PhD in that same subject and is currently in the way of defending it before Vienna UT.

Niki Popper has a degree in Mathematics and is CEO of DWH Simulation Services, a professional simulation company based in Vienna. He is also lecturer in mathematical simulation at Vienna UT.