KEYWORDS
Agent-based Simulation, Virtual Reality Systems, Representation Alignment

ABSTRACT
In this contribution, we are dealing with the problem of on-demand interfacing agent-based simulations with Virtual Reality (VR) systems. We focus on how to handle and relate information between the two systems and investigate a generic way for mapping the information which describes the state and context of a simulated agent to its 3D counterpart in the VR system. We provide details about the issues involved as well as propose generic representations and procedures that cope with the granularity discrepancies between the two systems. Finally, we present a first prototype and discuss aspects of its performance.

INTRODUCTION
In the field of agent-based modelling, visualisation plays an important role in communicating, analysing and debugging the behaviour of a model. As the technology advances, there is an increasing interest to use more realistic and complex environments such as 3D virtual worlds in order to improve the transparency, understandability and also attractiveness of the running simulation.

Advanced visualisation is particularly attractive for model evaluation. Especially for validation of a multi-agent simulation involving simulated humans, the lack of empirical data motivates searching for new validation methods such as the systematic usage of human experts checking the plausibility of a simulation (Klügl, 2008). Hereby, an immersive perspective - supported by 3D visualisation - is particularly important as the perspective during plausibility checks should be similar to the real-world perspective. A mere 3D visualisation might be useful for many simulations independent of the used paradigm; real immersion into the simulated system during simulation (as the high end of participatory simulation (Berland and Rand, 2009)) is particularly apt for agent-based simulations.

However, setting up and rendering a VR visualisation in addition to the actually intended simulation means additional effort for an addon to the simulation that is just relevant for plausibility checks (and selling the model), but not for the actual deployment runs. Thus, our vision is to create a generic component for connecting an agent-based simulation environment to a VR system which is able to control visualisation in an immersive virtual environment as the CAVE (Cave Automatic Virtual Environment) (Lee et al., 2010). In such a setup a human can observe the simulation's dynamics immersed in the simulation environment. Setting up a VR visualisation for a simulation model shall come at minimal costs. Ideally, all the necessary tasks for connecting a visualisation in a CAVE to an agent-based simulation shall be automated. Thus, a generic coupling between a simulation platform and a VR platform needs to be provided in a way that the simulation is not depending on the VR visualisation. The simulation shall mostly run without the VR platform, therefore embedding the simulation into a 3D environment with all the necessary visualisation overhead, does not make sense. The immersive component shall work as an add-on that is used when necessary, but still can be switched off for the deployment or other test simulation runs.

In the remainder of this paper, we will first give more details on the overall idea and the involved challenges. The next section gives an overview to the overall system setup and more specifically introduces the MiddleLayer-Component. Then we present the results from some initial performance tests of our prototype implementing the described setup. Then, we discuss upcoming challenges if a human interacts with the VR system. Finally, after a discussion of related work, the paper ends with a summary and an outlook to our planned next steps.

SYSTEM COUPLING
We consider the Visualisation System and Simulator as two different systems which represent the same multi-agent model, on different levels of temporal and representational granularities. The Simulator contains the more qualified representation assuming that it is built for validly reproducing the relevant features of the original system. It possesses refined object structures which aims to represent the relevant structural and especially behavioural aspects of the original system. Contrarily, the focus of the Visualisation System is on the realism and plausibility of the 3D object models and their animated behaviour. This means that it needs to display all details that a human observer would expect in the real world. In the following we will deal with particular aspects of the coupling that
individual dynamics of the characters, their interactions have realism. Therefore it becomes necessary to compensate the Simulator just sent the position information to the Visualisation System. To clarify this problem, consider the synchronous operation modes for the two systems, there may be several issues related to the way a character in the Visualisation System can change its appearance and displayed behaviour in runtime. For example, morphing operators can be used for changes in the shape (e.g. pregnancy) and movement of the object model. Animations can be combined in different ways and may have longer duration than what corresponds to a simulation step (that gives the frequency with which new data is arriving at the Visualisation System) yet the animations need to continue or to be connected in a smooth way.

Environmental dynamics A multi-agent simulation may contain global dynamics such as fog or raising temperature, but also discrete signals for example triggering evacuation. These dynamics cannot be assigned to one agent/object model, but should be visualised.

Interaction between the agents Apart from the individual dynamics of the characters, their interactions have to be visualised as well (e.g. characters that talk to each other, wave to each other etc.). Thus, the visualisation must be coordinated. Animations of different agents that the Visualisation System is independently informed about, cannot be independently displayed.

Synchronisation It is important to assure that for a given timestep $t$, both systems have matching situations/scenes. Yet, Simulator and Visualisation System have different time advance functions. The Visualisation System shall render in real-time whereas the Simulator updates the situation usually as fast as possible. The relation to real time may be varying depending on the numbers of agents or events to be treated. Thus, different time granularities are present and there is a clear need to buffer the relevant information while maintaining consistency between the two systems, as well as to interpolate or even extrapolate dynamics.

Interpolation of behaviour As we assume asynchronous operation modes for the two systems, there may be an update delay or insufficient information arrives at Visualisation System. To clarify this problem, consider the following example as illustrated in Fig. 1. At time $t_{Sim} = 0$ an agent $A$ is at position $P_0 = \{x_0, y_0, z_0\}$. Given an action "walk", at time $t_{Sim} = 1$ be is at position $P_1 = \{x_1, y_1, z_1\}$. Between these two positions there is an obstacle that the agent has to avoid, yet it does not have the necessary information how to do as the more abstract Simulator just sent the position information to the Visualisation System. The latter has to maintain the needed realism. Therefore it becomes necessary to compensate this lack of information and estimate a reasonable movement between the given positions. A method which can be used to accomplish this is interpolation, for example using algorithms based on dead reckoning (Beauregard, 2007). The situation becomes critical, if the agent during its interpolated movement interacts with other entities not foreseen in the Simulator representation.

VR world configuration In order to be able to render the scene in the Visualisation System, further configuration in addition to the actually simulated situation is necessary. Each scenario/scene may have specific demands on cameras and lighting. Their positioning in the scene is an important issue and can clearly affect the realism of the scene and therefore also the result of the immersive plausibility checks.

**SYSTEM OVERVIEW**

A general view of our proposed framework for coupling a Simulator and a Visualisation System is presented in Fig. 2.

![Figure 2: Graphical illustration of the overall framework.](image)

We assume a time-stepped Simulator. In every time step, explicit information about the current situation is used to create a full snapshot. As the Visualisation System operates in an independent time regime, there is a need to use a component that buffers information from the Simulator and transfers the right information parts to the Visualisation System when needed. This is the **MiddleLayerComponent**. It constitutes the most important block in the framework because here the core data processing takes place. Its main responsibility is to buffer the different time and object granularities. It translates the incoming simulator data to specific event and actions at the appropriate time thus maintaining the consistency between the two platforms.
We make a distinction between agents and passive entities: as well as the exact number and setups of agents and passive entities with their relevant initial positions as well as the assigned 3D object models. By default, cameras and lights are added following some generic heuristics about the size of the simulated world as the human observer. The list of objects, ObjectList, is composed of information on all the entities currently existing in the scene; agents and passive entities (for example resources or obstacles). Both of them are PhysicalObjects, which means that they have an unique name, position and rotation information.

We make a distinction between agents and passive entities: agents possess an additional attribute which denotes their state. This distinction facilitates mainly the generation of the scene due to fact that the 3D object model for the agents must hold animations that are controlled by the state. The classification agent/passive entity might be different in Simulation and Visualisation; in the later passive entities may appear to be active as well due to a more flexible and elaborated visualisation: A passive object may require an animation (e.g., opening/closing door). Also the environment may have a state which can trigger changes to the rendered scene (e.g., showing changes in day/night or displaying an evacuation signal).

This shared meta model allows the scene to be automatically generated from the Simulator. The modeller only needs to add pointers to the appropriate object models and animations. Based on this data buffer, the information stream from Simulator and the processes that access information for visualisation are decoupled. The memory needs are depending on the richness of the model. If the buffer size is restricted, problems may occur. Therefore we need to identify the actual requirements with respect to the simulated model.

Overall Process
There are two processes that need to be considered: The setup process that causes the main effort for the modeller and the run-time process connecting the two systems during simulation and visualisation. The modeller needs to execute the following tasks:

- Connect physical objects to 3D object models
- Connect animations / animation combinations to agent activities

We assume that the rest can be done automatically: The Simulator may full scene description. This scene describes the overall starting state of the Visualisation System and its configuration. This core material upon contains information about the size of the simulated world as well as the exact number and setups of agents and passive entities with their relevant initial positions as well as the assigned 3D object models. By default, cameras and lights are added following some generic heuristics about the size of the environment or the designation of a particular entity as the human user in the VR system/CAVE.

During simulation, agents execute particular actions or activities that are connected to combinations of animations or other operators. The high-level behaviour specification using activities is a specialty of the simulation platform that we used for the endeavour. The simulated agents are programmed using activity graphs. This structure facilitates the connection to animation combinations as the behaviour program is already structured. Due to this explicit information, the Simulator can generate the appropriate information for each agent and send this information for all agents to the MiddleLayerComponent where it is stored into buffering data structures. Triggered by rendering in VR system, the data structure is accessed and appropriate events are formed and sent for execution in the visualisation side. The execution of events is then completely handled by the Visualisation (e.g., play animation).

**EXPERIMENTS**

**Prototype**
We created a prototype for the MiddleLayerComponent connecting the two systems. This component connects SeSAm (Klügl, 2009), a general modelling and simulation platform and the Horde3D GameEngine (http://hcm-lab.de/projects/GameEngine) implementing the above introduced functionality. In addition, we developed an export function as a SeSAm plugin that generates a complete description of the scene in XML format. After minor configurations, it can be used as an initial scene description for Visualisation System. However, due to the absence of an automatic generation of geometries from the information in the simulation platform, this export is restricted as it assumes that the relevant object models are existing.

During the simulation, all the visible effects of the agents’ actions, are transmitted via client/server communication. The protocol includes information as described above; position, orientation and activity (animation) of each entity in each update. In addition to this, information about environmental state changes or other events are also sent, as well as information the generation or removal of agents from the overall scene. The generation is handled with the same information used for the setup of the scene.

**MiddleLayerComponent** stores the incoming data in the above described buffer. The rendering process of Horde3D sends a request for the appropriate information to be rendered in the current frame. As an answer to this request, the **MiddleLayerComponent** generates all the necessary events so as to display the appropriate real-time scene.

**Scenario**
To evaluate the performance of our prototype, we used two testbed scenarios. The first one is an implementation of the well-known Boids model in 3D (Davison, 2005). The second, is a small evacuation scenario in

**Data structures**
Fig. 3 describes the data structure that forms the basic skeleton for all incoming data relevant for visualisation. A full snapshot GameState of the simulation at every tick is stored in a list of game states, the GameStateList. Every game state has a time-stamp, a list of objects that are populating the scene at that time, relevant information about the environment and information about the state of a human observer. The list of objects ObjectList is composed of information on all the entities currently existing in the scene; agents and passive entities (for example resources or obstacles). Both of them are PhysicalObjects, which means that they have an unique name, position and rotation information.

Also the environment may have a state which can trigger changes to the rendered scene (e.g., showing changes in day/night or displaying an evacuation signal).

This shared meta model allows the scene to be automatically generated from the Simulator. The modeller only needs to add pointers to the appropriate object models and animations. Based on this data buffer, the information stream from Simulator and the processes that access information for visualisation are decoupled. The memory needs are depending on the richness of the model. If the buffer size is restricted, problems may occur. Therefore we need to identify the actual requirements with respect to the simulated model.

**Overall Process**
There are two processes that need to be considered: The setup process that causes the main effort for the modeller and the run-time process connecting the two systems during simulation and visualisation. The modeller needs to execute the following tasks:

- Connect physical objects to 3D object models
- Connect animations / animation combinations to agent activities

We assume that the rest can be done automatically: The Simulator may full scene description. This scene describes the overall starting state of the Visualisation System and its configuration. This core material upon contains information about the size of the simulated world as well as the exact number and setups of agents and passive entities with their relevant initial positions as well as the assigned 3D object models. By default, cameras and lights are added following some generic heuristics about the size of the environment or the designation of a particular entity as the human user in the VR system/CAVE.

During simulation, agents execute particular actions or activities that are connected to combinations of animations or other operators. The high-level behaviour specification using activities is a specialty of the simulation platform that we used for the endeavour. The simulated agents are programmed using activity graphs. This structure facilitates the connection to animation combinations as the behaviour program is already structured. Due to this explicit information, the Simulator can generate the appropriate information for each agent and send this information for all agents to the **MiddleLayerComponent** where it is stored into buffering data structures. Triggered by rendering in VR system, the data structure is accessed and appropriate events are formed and sent for execution in the visualisation side. The execution of events is then completely handled by the Visualisation (e.g., play animation).

**EXPERIMENTS**

**Prototype**
We created a prototype for the **MiddleLayerComponent** connecting the two systems. This component connects SeSAm (Klügl, 2009), a general modelling and simulation platform and the Horde3D GameEngine (http://hcm-lab.de/projects/GameEngine) implementing the above introduced functionality. In addition, we developed an export function as a SeSAm plugin that generates a complete description of the scene in XML format. After minor configurations, it can be used as an initial scene description for Visualisation System. However, due to the absence of an automatic generation of geometries from the information in the simulation platform, this export is restricted as it assumes that the relevant object models are existing.

During the simulation, all the visible effects of the agents’ actions, are transmitted via client/server communication. The protocol includes information as described above; position, orientation and activity (animation) of each entity in each update. In addition to this, information about environmental state changes or other events are also sent, as well as information the generation or removal of agents from the overall scene. The generation is handled with the same information used for the setup of the scene.

**MiddleLayerComponent** stores the incoming data in the above described buffer. The rendering process of Horde3D sends a request for the appropriate information to be rendered in the current frame. As an answer to this request, the **MiddleLayerComponent** generates all the necessary events so as to display the appropriate real-time scene.

**Scenario**
To evaluate the performance of our prototype, we used two testbed scenarios. The first one is an implementation of the well-known Boids model in 3D (Davison, 2005). The second, is a small evacuation scenario in...
which agents randomly move around the environment, communicate/interact (e.g., talk, wave etc.) with other agents if they are nearer than a given distance. At a given time, there is an evacuation signal, which causes all agents to run towards the exit by avoiding collisions. Fig. 4 shows two snapshot pairs of the scenes in the two systems.

**Performance in the Boids Scenario**

Five situations of the same scenario are generated varying in the number of agents: 5, 20, 50, 100 and 500. A human observer is positioned as an obstacle. The agents are exhibiting flocking behavior while avoiding the human observer/obstacle. The simulation ends after timestep 1000, running as fast as possible. At \( t = 100 \), 50 new agents are added, at \( t = 500 \), 50 randomly selected agents are deleted from the simulation for showing how the overall system reacts to generation and deletion of entities.

Fig. 5 gives our performance results. The graph in Fig. 6(a) shows the size of the buffer (GameStateList) depending on the number of agents in the scene in a simulation run of 1000 steps. The Overall Buffer Size shows the amount of data size the Simulator has sent to the MiddleLayerComponent (measurement of atomic information units) whereas the Remaining Data deals with the information that is pending to be executed by the Visualisation in the moment the simulator has ended. It is obvious that the memory consumption is linear, yet it appears that in all cases the Visualisation System can handle the load.

In the following, we evaluate the time consumption of the several operations performed in the MiddleLayerComponent. We identify three main interesting values:

1. **Transfer Time** - It measures the time needed for the data from being sent by the Simulation to reach the socket in the MiddleLayerComponent.

2. **Processing Time** - It measures how much time it takes for the data to be processed and stored in the buffer.

3. **Retrieval Time** - It measures how much time it takes for the data to be retrieved for rendering.

We already described that during runtime, 50 agents are generated and the same number of agents is deleted after some time. The result of this operation is obvious in Fig. 6(c). At \( t = 100 \) the process time shows a significant increase which gradually is getting reduced and stabilizes when the agents are deleted and no additional information is communicated apart from the agents’ position, orientation and particular state. In Fig. 6(d) there is an interesting observation: The cases with 100 and 500 agents, do not really seem to be affected by the creation/deletion of agents but the time to retrieve the data from the buffer increases constantly. This is a result of an overall delay of the simulations update. The read/write process can be seen quite costly when a large amount of agents is involved. In the case of 500 the Visualisation System sometimes had to wait for the simulation to produce the state message. A more efficient simulation platform will replace SeSAm in our future work.

**Performance in the Evacuation Scenario**

In this test case, we demonstrate the generality of our framework. The evacuation scenario is a 2D agent-based simulation. Main difference to previous case is that we need to connect 2D shapes with fine grain object models as well as their relevant animations. Concerning the third dimension, it is not available in the 2D simulation, therefore it has to be given as additional information in visualisation. We have a variety of states/animations that need to be communicated as well as information triggered by the world such as the evacuation signal.

**INTERACTION WITH HUMAN**

The assumption that the human observer stays passive within the VR system when evaluating a simulation is not realistic. Thus, we must expect that the human observer...
This is a bidirectional coupling causing some challenges: the agent in the simulation which is corresponding to the time step that is currently displayed in the visualisation. The human in the simulated timestamp corresponding to the human then actually executes the actions induced by the other agents, then both his/her actions and their results have to be transferred back to the simulation. The System to the Simulator: In the prototype, the immersed human would affect the behaviour of simulated agents at a time when the simulation itself actually computes at a completely different time step. This would require that information flows back from the Visualisation System to the Simulator: In the prototype, the immersed human-avatar corresponds to a static obstacle from the other agents’ point of view. The agents have to avoid the human by moving around him. Yet, if the human avatar interacts with other agents, then both his/her actions and their results have to be transferred back to the simulation. The agent in the simulation which is corresponding to the human then actually executes the actions induced by the human in the simulated timestamp corresponding to the time step that is currently displayed in the visualisation. This is a bidirectional coupling causing some challenges: at least wants to move around in the overall scene or even interact with the simulated agents in a similar way as in real life. This was coined by (Pelechano et al., 2008b) as the necessity of “presence”. Thus, for enabling a human to evaluate the simulation model dynamics, we must take the human immersion in our system seriously. However, the overall complexity rises significantly as the behaviour of the human would affect the behaviour of simulated agents at a time when the simulation itself actually computes at a completely different time step. This would require that information flows back from the Visualisation System to the Simulator: In the prototype, the immersed human-avatar corresponds to a static obstacle from the other agents’ point of view. The agents have to avoid the human by moving around him. Yet, if the human avatar interacts with other agents, then both his/her actions and their results have to be transferred back to the simulation. The agent in the simulation which is corresponding to the human then actually executes the actions induced by the human in the simulated timestamp corresponding to the time step that is currently displayed in the visualisation. This is a bidirectional coupling causing some challenges: An asynchronous operation of the two systems would basically imply the need to have a roll-back function in the simulation side. Imagine a case in which the Simulator is processing the agents’ state in $t_{Sim} = 100$ while the user connected in the Visualisation platform, is changing the flow of actions in $t_{Vis} = 10$. The simulation has to adapt to this change and reset the simulated situation to $t_{Sim} = t_{Vis} = 10$ and restart again. An alternative in some restricted cases might be based on a more intelligent MiddleLayerComponent. Instead of initiating a full roll-back on the Simulator side, the MiddleLayerComponent analyses and adapt all stored snapshots from the currently displayed time to the current simulation time. If there are not too extensive discrepancies, it might be possible to actively align the two representations starting with the human modifications in the Visualisation System. The MiddleLayerComponent calculates the effects and resulting delta in the next-to-render situation. If component possesses enough knowledge about effects of actions, and the time difference between Simulator and Visualisation System is not too large, the effects of the human interaction might be isolated and the Simulator...
might be just forced to restart from a time significantly later than the current visualisation time.

RELATED WORK

3D visualisation in multi-agent simulation made its first breakthrough with the seminal evolving creatures of (Sims, 1994). Despite the promising results, detailed 3D representations of simulated models could mostly be found until recently in domain specific applications such as in pedestrian simulation and 3D visualisation could not but be seriously considered. Recent developments in microscopic pedestrian simulation platforms yielded the need to increase visibility on generic multi-agent simulation and thus they include detailed 3D spatial representations in physically simulated 3D worlds. There is a number of agent simulation platforms specialized for 3D modelling; breve seems to be the most well known open source platform (http://www.spiderland.org/breve). Similarly, there is Repast (http://repast.sourceforge.net/) or MASON (http://cs.gmu.edu/eclab/projects/mason/) that embedded Java3D into their simulation platforms. However, a simple 3D visualisation is not always capable of increasing the understandability of the simulated situation so in a more complex scenario a finer visualisation is more desired leading to the creation of more complex 3D visual representations such as virtual worlds. The combination of virtual worlds and simulation is also prominent in crowd simulation (Pelechano et al., 2008a), (Thalmann and Musse, 2007). This endeavour often involves the use of game engines where apart from the agents’ modelled environment, sensor data and action commands are available for communication between the simulation platform and the game engine (Norling, 2003). In our work we also consider the use of a game engine but as a means for visualising the simulated multi-agent model’s dynamics. One of our main considerations is to provide a solution that is not application specific but rather generic.

In (Avouris, 1992) different multi-agent system architectures for identifying specific challenges are classified when designing interfaces to multiagent simulations but only cases restricted to top-down perspective (bird’s eye view) are considered. In this area where visualisation for agent-based simulation is the main focus, there is also interesting work reported. Namely, (Kornhauser et al., 2009), give design guidelines for adapting general design principles and techniques (e.g., shapes/colours of the agents or use of entity groups) for visualising the dynamics of a multi-agent simulation. Their work involves the creation of 2D models implemented in Netlogo (http://ccl.northwestern.edu/netlogo/). Our research is related to some extent with the creation of interfaces for multi-agent systems but we are mainly interested in deploying the assets of a fine grain visualisation such as virtual reality.

There have been many uses of agent-based frameworks to create interactive simulation, (Vizzari et al., 2008) present in their work a framework for visualising crowd simulations based on interacting situation agents. They do not handle the synchronisation issue but they rather reduce the simulation’s speed so as to visualise properly the simulation’s dynamics and maintain consistency. Our framework presents similarities to the one of (Oijen et al., 2011), who consider the coupling of the two systems and not their embodiment. Nevertheless their vision is to use/include BDI agents in 3D game engines for building intelligent behaviour. The performance of this framework as well as others like (Gemrot et al., 2009) should be also tested in the context of our work that is mainly oriented towards the evaluation of the multiagent model.

Consistency plays a central role in our work. Thereby, a relation can be also found in the area of distributed interactive systems such as multiplayer network games, where consistency has to be assured for presenting the same situation to different users. Several techniques have been developed for avoiding or dealing with inconsistencies coming from latency and jitter (Diot and Gautier, 1999), (Delaney et al., 2006). In our case, we identify the major problems to be the different resolutions and synchronisation problems between two full representations of the same system, not a distributed representation. Nevertheless it is of worth to mention here that the interpolation of agent’s behaviour has similarities with the consistency problem of distributed interactive systems.

SUMMARY AND FUTURE WORK

In this contribution we discussed the concept of adding a sophisticated visualisation to a multi-agent simulation using a generic MiddleLayerComponent for providing a way of easily setting up what is needed for plausibility checks by a human expert from the usual perspective. A first prototype of the overall system is available where a MiddleLayerComponent buffers information from the Simulator. It is accessed at the appropriate times by the Visualisation System. First performance measures show positive results indicating that the connection itself does not have critical performance problems.

However, in the current status just half of the problem is solved. A still critical one is how the overall system shall react on the human observer. The problem already starts with collision avoidance. Agents in the simulation might avoid the human observer but the question how the situation changes initiated by a mobile human in the Virtual World can be fed back into the simulation is not yet answered. This will be the major part of our future work.

Additionally, we will consider situations where the visualisation needs more and faster data than the Simulator and the MiddleLayerComponent can provide. Then, stops in the Visualisation System may hinder the objective of our endeavour; the human observer immersed into a virtual reality that cannot correspond to the real one just because of stops. It might be a good idea if there would be some extrapolation on the visualisation side, assuming that the simulation can later catch up again when e.g. the
number of agents has reduced and thus simulation speed has increased again. How then the state of the Visualisation System can be mapped back to the Simulator is a problem related to the interaction problem discussed above, yet it does not just affect the representation of the human in the simulation, but a number – in the worst case all – agents.

We currently have set up two test cases for the overall framework: the above used boids simulation and a small evacuation scenario. Clearly for showing the scalability of the system in terms of agent complexity and numbers, we need to use the system in several real simulation projects.

REFERENCES


AUTHOR BIOGRAPHIES

ATHANASIA LOULOUDI is PhD student at the Modeling and Simulation Research Center of Örebro University. She holds a Master in Robotics and Intelligent Systems (Örebro University).

FRANZISKA KLÜGL is Professor in Information Technology at Örebro University, Modeling and Simulation Research Center. She holds a PhD in Computer Science and did her Habilitation with focus on Agent-based Simulation Engineering, all at the University of Würzburg in Germany. In 2008, she became Senior Lecturer at Örebro University, Sweden.