MomenTUMv2:
A Modular, Extensible, and Generic Agent-Based Pedestrian Behavior Simulation Framework

Peter M. Kielar[*][†], Daniel H. Biedermann[*] and André Borrmann[*]

August 26, 2016

Abstract

Agent-based pedestrian behavior simulators are computational systems that implement models and theories describing the behavior of individual pedestrians. Pedestrian simulations are performed to evaluate and compare behavior models. Ultimately, pedestrian simulations will provide a method to forecast pedestrian behavior. To develop a pedestrian simulator, multiple infrastructure and non-behavior concepts have to be implemented. Such components can be reused for the development of new simulation models, if the simulator is designed as a modular simulation framework

The existing agent-based pedestrian simulation frameworks are often limited in their capabilities regarding infrastructure flexibility, non-behavior models, pedestrian model requirements, or hybrid modeling. To fill this gap, we developed the agent-based pedestrian behavior simulation framework MomenTUMv2, which was carefully designed based on software engineering paradigms and in-depth background in pedestrian dynamics research. MomenTUMv2 has a number of high-level key features. First, MomenTUMv2 handles models generically, because models are conceptualized as operation providers. Second, MomenTUMv2 is extensible by design, which enables model developers to add models by following interface definitions. Finally, MomenTUMv2 can couple models in almost any modular manner, based on a configurable execution pipeline.

The unique qualities of MomenTUMv2 allows to design simulations without writing code and to add new theory implementations without changing the given infrastructure. To provide evidence for the capabilities of MomenTUMV2, we discuss an example application that shows a multi-hybrid agent-based simulation conducted by means of our framework.

Keywords: Pedestrian Dynamics, Crowd Simulation Framework, Software System

[*]Technische Universität München, Chair of Computational Modeling and Simulation, Arcissstraße 21, 80333 München, Germany

[†]Corresponding author. Email: peter.kielar@tum.de, Tel: +49 89 289-23062

Contents

1 **Introduction** 3

2 **Related work** 4
- 2.1 Pedestrian simulation frameworks . . . 4
- 2.2 General requirements . . . 7

3 **Framework overview** 9
- 3.1 Applied paradigms . . . 10
- 3.2 Packages view . . . 11
- 3.3 Simulation control flow . . . 13

4 **Framework description** 15
- 4.1 Configuration . . . 15
- 4.2 Simulation managers . . . 16
- 4.3 Simulation model types . . . 17
- 4.4 Data contexts . . . 18
- 4.5 Execution order . . . 21

5 **Example application** 23
- 5.1 Simulation scenario . . . 23
- 5.2 Simulation design . . . 26
- 5.3 Simulation execution . . . 31

6 **Upcoming research and developments** 33

7 **Conclusion** 34

1 Introduction

The research on pedestrian dynamics comprises multiple research topics, e.g., laboratory experiments (Zhang and Seyfried, 2014), field studies (Biedermann et al., 2015), behavior modeling (Helbing et al., 2000; Hoogendoorn and Bovy, 2004; Peters and Ennis, 2009; Moussaïd et al., 2011; Canca et al., 2013; Dijkstra et al., 2014; Kielar et al., 2016), theoretical work (Duives et al., 2015), civil engineering aspects (Mayer et al., 2014), validation studies (Campanella et al., 2014), and the work on computational pedestrian simulation frameworks (Curtis et al., 2016). In this report, we approach the former, the research on pedestrian simulation frameworks. We especially focus on agent-based simulations, in which pedestrians are simulated as individuals; thus, on pedestrian research that provides forecasts on behavior from a single pedestrian's perspective. In turn, crowd phenomena emerge by individual behavior of multiple pedestrians (Helbing et al., 2005). The term agent-based simulations originates from research on describing complex systems via simulating the behavior of the numerous single elements of a system (Reynolds, 1987; Bak et al., 1989). In this approach, individual behavior is designed by mathematics, rules, and logics, which define the behavior of the elements. Then, the interaction of the elements result in system-wide phenomena. In microscopic pedestrian simulations, each pedestrian is described as an agent. Thus, agent-based simulations of pedestrian behavior target to predict crowd phenomena by simulating multiple single pedestrians. Understanding and predicting crow phenomena is highly valuable, because such insights can improve pedestrian safety, comfort, and experience during any kind of crowd gatherings (AlGadhi and Mahmassani, 1991; Klüpfel, 2007; Helbing and Mukerji, 2012; Canca et al., 2013), e.g., music events, exhibitions, sport events, or religious festivals.

However, the methods and models that describe pedestrian behavior are numerous (Duives et al., 2013). Consequently, researchers found a concept to categorize pedestrian behavior based on three different but interconnected behavior layers (Hoogendoorn et al., 2001; Hoogendoorn and Bovy, 2004; Bierlaire and Robin, 2009). Operational models describe how pedestrians walk, overtake, accelerate, evade, and finally approach a visible point in space (Blue and Adler, 2001; Helbing et al., 2000; Seitz and Köster, 2012; Alonso-Marroquín et al., 2014). Tactical models forecast how pedestrians perform wayfinding behavior; thus, solving navigational tasks, beginning from the agents current position towards a destination (Höcker et al., 2010; Hartmann, 2010; Kemloh Wagoum et al., 2012; Kielar et al., 2016). We understand tactical models as spatial task solving models (Kielar and Borrmann, 2016a), which describe how pedestrians reach and approach a certain destination location. Thus, tactical models comprise other behavior then wayfinding alone. Queuing models predict the phenomena given in pedestrians queues (Kneidl, 2016). Searching models provide mechanisms to learn spatial systems (Andresen et al., 2016). Arranging models forecast the distribution of pedestrian in front of an attractor (Kielar and Borrmann, 2016a). Finally, the third layer describes strategic models that provide methods to predict how pedestrians choose their next destination they want to visit; thus, providing a goal, which has to be reached by behavior described via spatial task solving models. Furthermore, more specialized models exist in pedestrian dynamics. Meta models allow to address specific non-behavior related contexts, e.g., provide a coupling mechanism in hybrid simulations (Biedermann et al., 2014; Ijaz et al., 2015).

In general, we see models as implementations of theories that describe behavior, decision making, or generic concepts. All of these theories have in common that they define pedestrian dynamics related operations mathematically, which can be implemented in a computational model and executed in a pedestrian simulator. For pedestrian dynamics researchers, computational agent-based pedestrian simulations play an important role, because they bring the theories into a feasible and graspable environment. Only via simulations, forecasts of individual pedestrian behavior can be done. Furthermore, different models can be compared, evaluated, combined, or exchanged in a simulation environment. Hence, the result data of the simulations are the semi-physical evidence that a theory correctly describes pedestrian behavior, if the computed data is successfully matched against real-world measurement data.

To conduct pedestrian simulations, a complex software infrastructure has to be constructed. Well-coded simulation systems empower researchers and practitioners by providing a flexible toolbox that adapts to the needs of its users. Such a modular, extensible, and generic toolbox is MomenTUMv2. In the core, MomenTUMv2 is a fully functional multi-agent simulation framework for pedestrian behavior simulations.

The framework was designed in awareness of the clean code paradigm (Martin, 2009), design patterns (Gamma et al., 1995), and pedestrian dynamics research background. Thus, it provides a well-structured environment that can harbor most, probability all, models currently existing in agent-based pedestrian simulation research. Furthermore, MomenTUMv2 includes transparent interfaces to connect models to data contexts and other models, without creating dependencies, if not mutually demanded. MomenTUMv2 includes multiple built-in infrastructure concepts that improve the overall flexibility of simulations conducted in the framework, e.g., parallel model processing, model execution order configuration options, inherent hybrid model compatibility, scenario modeling tools, and mathematical utility code.

This report provides insights on the framework's software architecture, the used software paradigms, and the internal structures.

This report is structured as follows. In Section 2, we discuss other agent-based pedestrian simulation frameworks and define requirements. Section 3 provides a general overview of MomenTUMv2. We give a more detailed description on the framework's implementation and design in Section 4. In Section 5, we discuss a complex example application based on MomenTUMv2. The framework is still under development; thus, we point to further research and implementations in Section 6. The report is concluded in Section 7.

2 Related work

The research on pedestrian behavior models and computational simulations is tightly coupled with agent-based pedestrian simulators. A large number of pedestrian research papers report on the development of computational models. In most cases, these models have been implemented in an ad-hoc manner, addressing the limited scope of the developed model. However, we focus on pedestrian simulation frameworks; this means, a software that is a platform for agent-based pedestrian simulations and not a specific implementation for one simulation model.

Here, we take a look at existing simulation frameworks and derivate general requirements.

2.1 Pedestrian simulation frameworks

In this Section, we provide a brief overview of pedestrian simulation frameworks found in the scientific research domain. We explicitly do not examine general-purpose agent-based frameworks (e.g., Collier (2001), Luke et al. (2004), or Tianfield et al. (2003) for a review), because we realized that their abstraction demands an implementation of physical space models and pedestrian behavior models within the general-purpose frameworks. Such a reimplementation is equally or even more complex than developing a new agent-based pedestrian simulation framework in an object orientated programming language.

Unfortunately, only a small selection of agent-based pedestrian simulation frameworks can be discussed, because in most cases, the behavior models are described in the publications, but not the simulation framework used.

2.1.1 Framework by Lozano et al. (2007)

The pedestrian simulation framework of Lozano et al. (2007) is a platform for large scale crowd simulations. The framework focuses on distributed computing by means of a networked-server architecture to improve simulation performance. A server, the action server, is the main management unit, which is connected to multiple clients. Each client computes the action of multiple agents. By synchronization procedures, the information are distributed and held up to date on each client via the server.

The framework provides a scalable architecture, but does not enable researchers to introduce or exchange further behavior models. However, a high-level goal was solved by this frameworks: scalable and distributed computing.

2.1.2 Framework by Singh et al. (2009)

Singh et al. (2009) published a computational framework for steering algorithms: *SteerSuite*, which is an agent-based pedestrian simulation framework. They target to provide a platform to compare, evaluate, and share walking behavior algorithms, understanding the need of a common basis to make model implementations more transparent to other researchers. Here, steering can be understood as pedestrian behavior covering operational behavior, but also can comprise tactical and strategic behavior approaches. *SteerSuite* includes modules for testing, benchmarking, visualizing, and simulating.

Testing steering models is done via test case specifications, which resemble simulation configurations. The simulation module runs the simulations based on these configurations. The benchmarking module, which is an analysis module, enables users to calculate metrics based on their simulations.

The authors of *SteerSuite* state that the framework is completely modular regarding any component. Unfortunately, they do not describe how this modularity is achieved and which techniques are used. Nonetheless, the framework descriptions include some interface definitions, which are useful for designing a model in *SteerSuite*.

We see this framework as a useful concept for simulation moving human-sized agents. However, the concept does not comply with all requirements of the pedestrian dynamics research spectrum, especially the layering concepts of pedestrian behavior.

2.1.3 Framework by Torrens et al. (2012)

Torrens et al. (2012) developed an extensible pedestrian simulation framework that handles multiple pedestrian behavior models. Their framework also encompasses a metrics concept that lays the foundation for comparing the different pedestrian behavior models' performance regarding realism of simulated behavior.

The framework comprises multiple core modules: configuration, agent factory, models, GIS, probe, validation, and visualization. The agent factory module defines how pedestrian agents are generated based on agent populations. The configuration provides all parameters needed for a simulation. The model module comprise multiple models for movement behavior. The GIS module functions as an interface to spatial objects and relations; thus, providing spatial queries. The probe module is designed to read and access model-related data during runtime.

The framework integrates multiple mandatory concepts that are fundamental to a pedestrian behavior simulator framework. Furthermore, the given comparison of the movement models provide evidence that different movement concepts can be integrated in a single framework via a common interface concept. The framework provides multiple operational and tactical models; however, both concept are used similarly as movement concept in the framework. Unfortunately, strategical models are not present. Similar to *SteerSuite*, the framework is highly elaborated, but misses to address pedestrian dynamics specific model requirements.

2.1.4 Framework by Kemloh Wagoum et al. (2015)

JuPedSim (Kemloh Wagoum et al., 2015) is a simulation framework for microscopic (agent-based) pedestrian simulations. It comprises four modules: editors, models, reporting, and analysis.

The editor package enables to define scenario geometries, agent populations, and further environmental elements. This input is given to the models package, which computes pedestrian simulations by an operational, tactical, and strategical model. These model implementations can be exchanged. The simulation output can be provided to the report and visualization module. The report model comprises four predefined measurement methods. The report is explicitly to highlight, because it provides a solutions to compare simulation output with real trajectory data within a single tool.

However, *JuPedSim* is not generic in a sense that besides pedestrian behavior models (operational, tactical, and strategical) no further models can be added, e.g., meta models or perception models. Furthermore, Kemloh Wagoum et al. (2015) do not provide insights of how the intercommunication of the behavior models work; thus, there is no concept defined that describes how models are coupled generically.

2.1.5 Framework by van Toll et al. (2015)

A generic five-layer framework for agent-based crowd movement was developed by van Toll et al. (2015). The layers abstract pedestrian behavior from coarse to fine: high-level planning, global planning, route following, local movement, and animation. This hierarchy resembles the three layered structure of strategic, tactical, and operational behavior because route following and local movement is similar to operational behavior, global planning to tactical behavior, and high-level planning to strategic behavior. Additionally, the framework integrates the body animations of the agents, which may not be necessary for pedestrian dynamics related research due to post-processing visualization methods.

The layers are interconnected and lower level layers can initiate computation on higher level layers; however, details on concepts or theories that describe the activation schema are not given. The environment is represented via regions and meshes, which are specific navigation graphs suitable for path planning. Furthermore, agent profiles describe properties of agent populations. Well-defined is the framework's architecture regarding model modularity and the update sequence of the agents' data contexts. Both concepts are highly desired for a framework for pedestrian behavior simulation. Also, parallel processing is integrated by design.

The framework's capabilities are highly useful. However, from a pedestrian dynamics point of view, it misses analysis and meta concepts. Furthermore, a clear explanation of the layers interactions regarding model activation is missing. Such a description would contribute to understand the framework's control flow dynamics.

2.1.6 Framework Vadere

Following the explanations of Seitz (2016), Vadere is a simulation framework developed by multiple researchers at the Technical University of Applied Sciences Munich. Vadere is a microscopic (agent-based) pedestrian simulation framework that provides a platform for students and researchers to implement, compare, and validate models of pedestrian behavior. The framework comprises multiple functionalities: crowd simulations, scenario authoring, configuration via parameter files, result output concepts, post-visualization, post-analysis, and utilities.

The architecture of the framework follows the Model-View-Controller (MVC) pattern. Here, the model describes simulation states, the controllers are simulation models, and the view comprises all user interface. The controllers are models implemented for agent creation, agent removal, agent behavior, output, and post-processing. The controllers are situated in a simulation loop in which they are triggered during simulations. Most models can manipulate the simulation state, which comprises scenario, dynamic elements, attributes, and types.

Each controller is a simulation model and implements specific model interfaces. Furthermore, the interfaces distinguish between active callbacks that can change the simulation states and passive callbacks that can read the states only. Other interfaces are variants of these two types. All simulation models are situated in the simulation loop. Before the simulation loop, a pre-loop is executed for all controllers and similar, a post-loop is run for all controllers after the end of the simulation loop. The model call order is mostly predefined in the simulation loop, which is structured as following. First, the pedestrian create operations are carried out, then the target controllers that change pedestrians states if they trespass an area. Afterwards, the behavior model active callbacks are executed. Then, the output callbacks and other passive callbacks are processed. The order of calling active callbacks is predefined by the user.

Vadere is a well-structured simulation framework for pedestrian behavior simulations and comprises most of the mandatory requirements for pedestrian simulations. The loop and interface concept is especially noteworthy, because it enables to add multiple more simulation models into the framework without changing the framework's core structure. However, the three layered concept for operational, tactical and strategical behavior was not implemented directly, which may introduces problematic hidden dependencies for implementing non-operational behavior models. Furthermore, it is not clear if hybrid simulation can be conducted in Vadere, because it classifies pedestrian sub-types for each model that cannot be used by other models. This contradicts the MVC paradigm of model independence, because the pedestrian state implementations depend directly on the corresponding controllers.

2.1.7 Framework by Curtis et al. (2016)

Menge is a pedestrian behavior simulation framework and was especially designed for the needs of pedestrian dynamics research (Curtis et al., 2016). The framework provides a solution to the simulator implementation challenge of pedestrian dynamics research groups. Simulator developments typically include to implement infrastructure components, which to code is often error prone, time consuming, and not the main research goal. Thus, Menge targets to be a common framework for crowd simulations that is modular, cross-platform, and open source.

Menge comprises multiple modules that are related to three sub-problems: goal selection, plan computation, and plan adaptation. This concept is similar to the pedestrian behavior layer decomposition of strategic, tactical, and operational behavior. The layers are abstractly defined as functions that are connected by output and input. The final result is a velocity adaptation for a pedestrian. On each layer, specific model implementations can be added.

Menge provides the concept of solving the destination choice by a finite state machine approach including the modeling elements: conditions, targets, actions, goals, and goal selectors. The state machine provides a method to integrate strategic behavior and introduces a solution for scripting action logics. However, there are some actions available that should not be integrated within a behavior concept, e.g., agent teleportation for boundary conditions. Within the planning computation, tactical behavior is defined via a velocity component and velocity modifier. In any given state of the state machines, a different velocity theory can be implemented. Operational models will be provided with a directed velocity by the tactical model. Operational behavior is implemented in the plan adaptation module.

There are further modules and functions in Menge. The spatial queries in Menge reads information of the current environmental state. In addition, Menge includes visibility queries to model visual perception. The agent generators, including the state and profile selection, are solutions to define agent populations. The configuration module in Menge is state-of-the-art regarding dynamic configuration via *XML* (eXtensible Markup Language). A task concept provides an interface for any-purpose computations in the agent update phase of simulations. However, we see that a three layered tasks concept is more flexible that includes pre and post simulation cycle tasks.

Menge is well-designed and includes most of the mandatory aspects that a pedestrian simulation framework should include. However, the design approach of focusing on state machines as containers for strategic behavior provides assets and drawbacks. The goal selectors are the strategic models, which are included in the state machine. Thus, the command logic is inverse: strategic models are part of a plan; however, strategic model should generate plans. Nonetheless, the approach improves to script behavior for custom environments, which is especially useful if the behavior processes can be defined beforehand.

2.2 General requirements

Based on our own experience and the literature regarding agent-based pedestrian simulation frameworks, we are able to identify a mandatory set of modules, functionalities, and concepts that a generic, modular, and extensible framework for agent-based pedestrian simulations have to include. Here, we summarize our view on the integral part of such a framework.

We demand that a pedestrian simulation framework should not be limited to a fixed number of models due to the underlying code design. The behavior model types should at least cover the three layer architecture of strategic, tactical, and operational models. The models on these layers should be exchangeable. Furthermore, the interfaces between these models have to be as simple as possible. The execution order of the models should not be given but has to be configurable; however, typically orders may be pre-configured. In addition to the behavior models, exchangeable perception concepts, which describe spatial and visibility queries, should be included. In general, all models within such a framework should be exchangeable.

Configuration of a simulation should be done without including to write code and should provide solutions to define arbitrary configuration parameter sets for models. The configuration should be supported by tools that help to define the scenario layout and the used models.

A visualization should be included, but the use of other, external visualization tools should be possible. Thus, the framework should provide output writer models that can change the container and internal structure of resulting data.

The domain relevant data set for pedestrian simulation comprises at least locations, graphs, lattices, obstacles and pedestrian data. Data containers, e.g., pedestrian agent data contexts, should provide a functionality to be dynamically extended via models because different models may provide specific information sets. Furthermore, the handling of data contexts has to be strictly regulated to guarantee consistency of the data container in the case of multiple access to the data.

Because pedestrian simulations include evaluating and processing of the simulation data, analysis models should be present in a framework. The computations based on the analysis model data can be divers; thus, a generic solutions to define data structures as input and output for analysis models have to be given. However, time-based and pedestrian-based data series are the logical building blocks of these input and output data containers.

It must be possible to generate and initialize various pedestrian agents based on multiple agent-population descriptions during a simulation. The population configurations need to cover all initial states of the agents as well as provide a method to connect agents to specific behavior model data. Additionally, specific models should define how pedestrians are removed when reaching certain locations.

A framework should integrate parallel computation by design. However, to enable model developer to restructure parallel computation due to model demands, parallelism should be defined in the configuration and should be described in the model implementation guidelines. In addition to parallelization, a batch or loop concept improves a simulation framework. Such a concept would distribute or replay the simulation processing based on a configuration input.

Hybrid modeling concepts were not generically addressed in any of the evaluated agent-based pedestrian simulation frameworks. A hybrid model connects two or multiple models that operate on the same behavior layer (Chooramun et al., 2012; Biedermann et al., 2014). These hybrid concepts ensure that the agents may use multiple behavior concepts without creating erroneous simulation results. We define these models as meta models and demand that these models have to be present in a framework for pedestrian simulations. However, since a simulator framework is defined as agent-based, macroscopic models that describe pedestrians as accumulated densities (Hartmann and Von Sivers, 2013; Hoogendoorn et al., 2015) cannot be described flawlessly within an agent-based simulation framework. To do so, the data contexts have to be extended, which introduces a significant modification in the framework architecture.

Group behavior is also fundamental in pedestrian simulations (Peters and Ennis, 2009; Chu and Law, 2013). However, adding a layer describing group behavior seems to be an erroneous approach within a generic agent-based pedestrian simulation framework. This is the case as typical concepts of group behavior mostly build upon individual behavior models. Thus, to generate consistent group behavior, a model that considers groups should be provided on each of the three basic behavior models layers. If agent communication is demanded, this have to be handled via extending the agents data contexts.

Motion visualization, e.g., as shown in ADAPT (Shoulson et al., 2013), should not necessarily be included in an agent-based pedestrian simulation framework, because this is a typical task of post-visualization tools. However, operational models that use a bio-mechanical representation of human bodies should be able to form part of a framework.

Besides the described requirements, often the runtime performance and simulation result correctness are of interest. However, these are not in focus due to the fact that a framework provides a solution to integrate and exchange models, but the models are developed out of the scope of the framework implemented. Thus, incorrect and poorly performing models will slow down the execution of a framework regardless of the frameworks functional correctness.

This list may not be exhausting, but is sufficiently complete to approach the task of developing an agent-based pedestrian simulation framework, which is for the most part feature complete and dedicated to pedestrian behavior research.

3 Framework overview

In this Section, we describe the pedestrian simulation framework MomenTUMv2 on a high level. We will provide a glance on all modules, their interconnections, the software engineering, and the pedestrian dynamics perspective.

The two most important concepts in MomenTUMv2 are that we define a model as an implementation of a theory and we see all models as operation providers.

The core structure of all models that can be integrated in the framework, is shown in Figure 1. Regarding models, the modularity of MomenTUMv2 is always given, because each model can be exchanged or deactivated, and multiple models of the same type can be used in parallel. We define all models in a generic manner; thus, the framework handles all models highly similarly as operators. Also, the framework is further extensible, which means a researcher can implement new models as long as the interface definitions are correctly implemented.

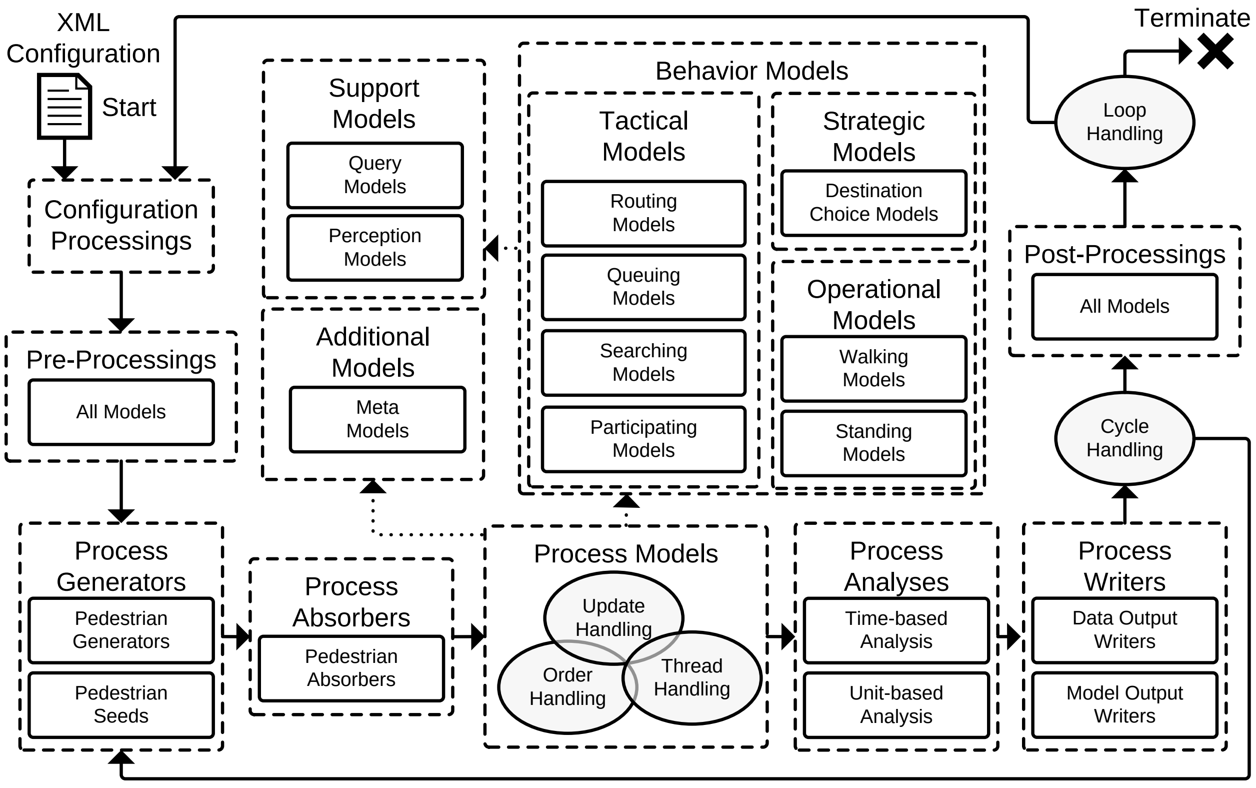


Figure 1: The core structures and flows of the MomenTUMv2 focusing on the modular property of the framework. Each solid rectangle defines a set of generic models, which references to the base-type. Each ellipse is a partially configurable MomenTUMv2 infrastructure component that guides the higher level control flow. Dashed rectangles describe organizational units in the framework. The dotted arrows describe a direct usage dependency. The solid arrows define the control flow. There are two high level control flows within the framework: simulation cycles and framework loops.

The logic units (depicted as dashed boxes) in Figure 1 have following purposes:

- *Configuration Processing* computes the command-line and file configuration inputs and generate all system and simulation objects.
- *Pre-Processing* enables all models to execute operations before the simulation starts.
- *Process Generators* compute pedestrian generation operations; thus, they can add pedestrian agents to a simulation scenario.
- *Process Absorbers* compute pedestrian absorber operations; thus, they can remove pedestrian agents from the simulation scenario.
- *Behavior models* define pedestrian behavior regarding the three layered behavior decomposition.
- *Support models* define additional concepts like spatial and visibility queries, which include perception models.
- *Additional models* define in a general sense meta models, which do not describe pedestrian behavior, but provide further operations related to pedestrians.
- *Process Models* provide an engine to compute pedestrian behavior models in arbitrary manner, including meta models. In classic pedestrian simulation, the models are processed in the order: strategic, tactical, and operational.
- *Process Analyses* compute metrics and analyses data base on input from simulations and other analysis results.
- *Process Writers* compute different output operations on the data given in the simulation.
- *Post-Processing* enables all models to execute operations after the simulations ended.

Figure 1 does not provide details regarding the data contexts. However, the framework hold the data contexts as parallel layer and provide access to the data via service mechanisms. The accessibility of the data is partially restricted regarding the base-type of the models.

3.1 Applied paradigms

In MomenTUMv2, we integrated multiple paradigms of simulation modeling, software engineering, and pedestrian simulation. Here, we provide these high level conceptual guidelines that we applied in the framework.

3.1.1 Simulation paradigms

The simulation framework MomenTUMv2 is an agent-based simulator framework. Thus, we apply the concept of multi-agent simulations (Wooldridge, 2009), in which each pedestrian is simulated as an individual entity. In the framework, the agents are not allowed to activate code independently, e.g., by internal events that activate behavior methods, but are manipulated by models of the simulation. Thus, pedestrians are operands of model operations. To realize this well-defined operation and operand structure for pedestrian behavior manipulation, agents provide a data context, which is the operand of computations. Therefore, operation implementations are the focus of the framework, because they change the simulation state. This concept enables to decouple the pedestrian implementations from the model objects, providing an approach for generic pedestrian simulations. Consequently, the model implementations depend on the pedestrian properties (the context). Beside the pedestrian behavior models, multiple more operations exist, e.g., result output writing or pedestrian generating. These operations are implemented by non-behavior models that follow the same concept of handling pedestrians as operands. Thus, we homogenized the operand and operation approach in every aspect of the framework.

3.1.2 Pedestrian behavior paradigms

MomenTUMv2 is a simulation framework for pedestrian behavior simulations that can integrate multiple different pedestrian dynamics models. Following the concept of hierarchical behavior modeling (Hoogendoorn et al., 2001; Hoogendoorn and Bovy, 2004; Bierlaire and Robin, 2009), pedestrian behavior can be described on three interconnected layers: the strategic layer (destination choice), the tactical layer (wayfinding), and the operational layer (walking). However, we implemented an extended version of this layering concept: the spatial task model (Kielar and Borrmann, 2016a). It describes the idea that besides wayfinding, there are multiple more behaviors that can be categorized as tactical behavior. MomenTUMv2 supports four tactical model types: navigating (to a destination), participating (e.g., find position in a crowd in front of a stage), queuing (e.g., at a counter), and searching (an unknown location). Consequently, we also extended the operational layer of the original concept into a structure that supports walking and standing (e.g., swaying) behavior modeling. MomenTUMv2 also comprises concepts of perception, which subsume visibility queries. Furthermore, non-behavior models are an integral part the framework: pedestrian generation approaches, pedestrian removal approaches, graph generation models, lattice generation models, output writing modules, analysis models, and meta models. In general, MomenTUMv2 can make use of multiple models for each model type and can handle multiple models for each layer simultaneously. Furthermore, models can be deactivated, e.g., a simulation can compute an analysis only.

3.1.3 Software engineering paradigms

We included several software engineering paradigms that help us to design MomenTUMv2 as a generic, modular and extensible software system. We began with constructing a type hierarchy based on a consequent use of the *template method pattern* and the *strategy pattern* (Gamma et al., 1995) to couple abstract model types generically. Furthermore, every model is generated following the *factory pattern* (Gamma et al., 1995) and *dependency injection pattern* (Fowler, 2004) to restrict the type dependencies and organize the object generation procedures in a single module. This approach also enabled us to integrate a strong *separation of concerns* (Parnas, 1972) within the code to ensure flexibility. To coordinate the objects within the framework, we implemented an execution engine and multiple data managers, based on the *mediator pattern* (Gamma et al., 1995). Furthermore, the execution engine integrates the *manager-worker pattern* (Carriero and Gelernter, 1989) by design. The parallel processing schema is supported by a *memento pattern* (Gamma et al., 1995) that resolves read/write conflicts regarding the pedestrian data contexts. An additional challenge is that each simulation may demand different parameters and models. Thus, we applied a variant of the *adaptive object modeling* (Yoder and Johnson, 2002), enabling users to structure their simulation via a *XML* configuration file. Finally, to protect our framework from changes in third-party libraries and simplify the usage of these, we consequently applied wrappers based on a variation of the *adapter pattern* and *facade pattern* (Gamma et al., 1995).

3.2 Packages view

The MomenTUMv2 pedestrian simulation framework comprises eight packages. Furthermore, there are four external modules that are partially dependent on a single core package of MomenTUMv2. The core modules of MomenTUMv2 are written in Java (Oracle, 2016a); however, any object orientated programing languages could have been applied. Our two-dimensional visualization tool is implemented in JavaFx (Oracle, 2016b).

In Figure 2, we provide the package dependencies as directed acyclic graph that includes every package of the framework. If the dependencies are across multiple programming languages, a native representation of the referenced package is given. We implemented the configuration package partially for C#(Microsoft, 2016). In the following Sections, we will shortly describe the contents of each package.

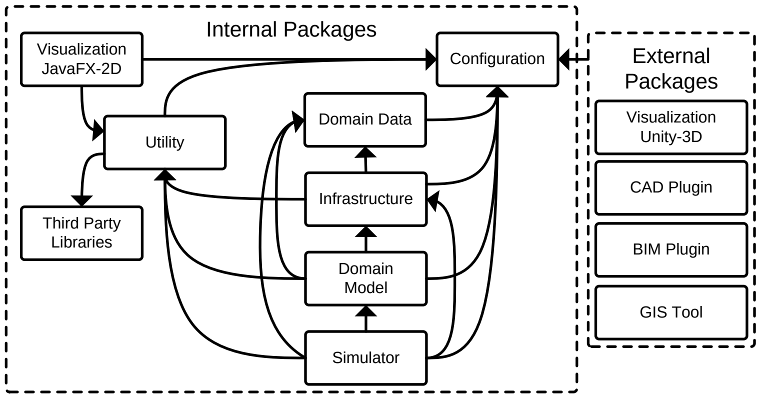


Figure 2: The overview of the MomenTUMv2 packages. The directed arrows defined a *is dependent on* relation. Each solid rectangle describes a software package.

3.2.1 Configuration package

The configuration module comprises all generic and specific classes that define serializable objects that describe the configuration of a simulation. Thus, the configuration defines all possible simulation classes by simple configuration classes, which are used later on to bootstrap the simulation via factories. Therefore, a user needs to create a configuration as initial task for designing a simulation in MomenTUMv2. Because the configuration includes all type-definitions of all elements that can be used in a simulation (e.g., layouts and models), all packages make use of the configuration module.

3.2.2 Utility package

The utility package can be understood as a computational helper module. It comprises all mathematical computation libraries used in MomenTUMv2. We use libraries and implementations of mathematical concepts that were not developed for pedestrian dynamics applications to ensure re-usability within multiple model implementations in MomenTUMv2. The third-party library we use are freely available (Lötzsch et al., 2006; Chew, 2007; Fernández, 2012; Alievsky, 2013; Apache Commons, 2016a,b; dyn4j.org, 2016). They build the underlying basis for most mathematical computations. Besides external third-party libraries, we implemented some concepts in-house, e.g., graph theory algorithms. We did such re-implementations only for specific mathematic domains, which we realized that are of high importance to pedestrian simulation and which we needed full control of.

3.2.3 Domain data package

The domain data package comprises all simulation object specifications that are used in simulations as operands. We have three core data domains: layout data, pedestrian data, and analysis data. Layout data defines the geometrical environment of the pedestrian simulation domain. In the pedestrian data domain, an elaborated type and interface hierarchy is defined, which structures the pedestrian data contexts. The analysis date is a high generic concept incorporating a text-based type system that handles any data that is time-based and pedestrian-based.

3.2.4 Infrastructure package

Some general concepts are integral mechanisms of MomenTUMv2. However, these concepts cannot be completely defined domain independent; thus, they cannot be part of the utility package. The infrastructure package provides the command line argument, the simulation time handling, parallel processing, simulation exceptions, and the model execution engine.

3.2.5 Domain model package

The domain model package is the largest package in MomenTUMv2 because it includes all model implementations, the specific and abstract model types, and object factories. There are multiple sub-packages in the domain model, which subsume each specific modeling domain: generator models, absorber models, layout models, meta models, analysis models, support models, output models, strategic behavior models, tactical behavior models, and operational behavior models. Basically, adding a new model to MomenTUMv2 is done in this package by implementing the theory in the correct sub-package, applying the correct types and interfaces, and extending the corresponding factory.

3.2.6 Simulator package

The simulator packages is the smallest of all packages and defines the main entry point of the application. Therefore, it is dependent on most of the other packages and comprises the high level control flow logic. The simulation packages includes some high level manager classes and output source types that are dependent on simulator package classes.

3.2.7 Visualization 2D package

To present the computation results of MomenTUMv2 simulations, the visualization 2D package is used. It is built on modern user interface technologies (Oracle, 2016b). The package is depends directly on the configuration package, because the layout definitions are needed to create the scenario geometry. Therefore, one data set that the visualization can handle are the layout components of a *XML* simulation configuration. Furthermore, the output models of MomenTUMv2 provide time-dependent simulation output data in the Character-Separated Values (CSV) format. The visualization reads the output data to show the simulation results. These data can originate from pedestrians, models or the analyses. The visualization uses fast data reading methods for fluent output data processing (i.e., sliding window, buffering, and indexing).

3.2.8 External tools

There are other packages that are not the core of MomenTUMv2, the layout tools and the 3D visualization tool.

We implemented a 3D visualization tool based on Unity3D (Unity Technologies, 2016), making use of the ADAPT framework (Shoulson et al., 2013). The visualization 3D tool builds upon previous work (Büchele, 2014). We extended the given code by including the ADAPT framework (Shoulson et al., 2013) and further features. 3D visualization is primarily used for pure demonstration purposes.

For describing two-dimensional geometries, which represent simulation scenarios, Computer Aided Design (CAD), Building Information Modeling (BIM), and Geographical Information System (GIS) models are used. For *CAD* and *BIM*, we developed plugins for the commercial software tools AutoCAD (Autodesk Inc., 2016a) and Revit (Autodesk Inc., 2016b). The plugins can extract data of the geometry models, which are used in the simulations as scenario layout basis, enriched with additional pedestrian dynamics related semantics. For *GIS*, we developed a tool that can read Open Street Map (OSM) (OpenStreetMap Foundation, 2016) *XML* data files and extract specific information relevant for simulation scenarios.

3.3 Simulation control flow

The control flow of the framework can be described in control layers. The control is decomposed on three hierarchies, the framework (high), the simulation (middle), and the models (low). Each layer solves different computational tasks within MomenTUMv2. This approach follows the software engineering paradigm of *separations of concerns* (Parnas, 1972) very tightly. The framework layer solves generic tasks regarding the framework loop. The simulation layer is more detailed and computes specific pedestrian simulation tasks, which are based on the contents of the configuration. Thus, the framework layer gives the control to the simulation layer. The simulation layer comprises a pre-processing and a post-processing phase, both encompassing the simulation cycles processing phase. During all simulation phases, the control is given multiple times to different model implementations; hence, accessing the models' operations. In Figure 3, we provide the overall control flow of MomenTUMv2 in regards to the layers.

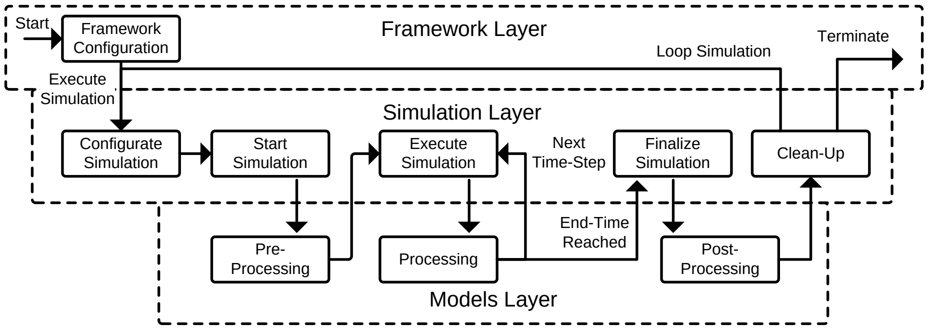


Figure 3: The overview of the MomenTUMv2 control flow, which comprises three hierarchically arranged layers: framework, simulation, and models. Each layer operates on a different level of abstraction.

3.3.1 Framework layer

The highest architectural unit is the framework layer. The framework is the most abstract part of MomenTUMv2 and could also be used for other simulations besides pedestrian simulations. The main tasks of the layer is to process the command-line arguments, start the simulation building and processing routines, and finally re-run the simulation if configured by the user.

3.3.2 Simulation layer

The simulation layer is the core of the pedestrian simulation framework MomenTUMv2. The layer bootstraps all management, data, and model objects, and links the object and executes the simulation phases. Here, the simulation layer does not have access to any specific model types; thus, it is coded completely generic. The simulation layer gives the control to the model objects by calling them in a semi-predefined order. This order ensures that data contexts of simulation objects are manipulated correctly, as intended by the user.

3.3.3 Models layer

The simulation computations are done solely within the models' layer. It is therefore a representative depiction of the control flow schema of all models running in MomenTUMv2. Each model implements a set of operations. The model implementations have access to specific management and simulations objects that provide information and services for the computations. However, models do not have access to other models if not explicitly defined by MomenTUMv2. This independence of models is the core basis for the generic property of MomenTUMv2. However, to communicate, the models operate on data contexts; thus, on the states of data. This ensures that any model can be implemented in MomenTUMv2, as long as they can handle the given data context. Some models may demand specific data that MomenTUMv2 does not provide. To solve this issue, model-referenced properties were implemented to provide a mechanism to add information to the data contexts without changing the framework's implementations.

4 Framework description

In this Section, we provide further details of MomenTUMv2. We begin by a more detailed description of the configuration concepts. Then, we describe the managers of MomenTUMv2. These managers are the main operating objects, which are not dedicated to pedestrian dynamics. As following topic, we address the simulation models in more detail and explain their abstract types. Afterwards, the data context of MomenTUMv2 is presented. Finally, we provide a centerpiece of MomenTUMv2: the model execution engine.

4.1 Configuration

MomenTUMv2 applies command-line arguments as an initial configuration and a file-based configuration as a detailed description of the simulation. Here, we give a brief overview on both configurations.

4.1.1 Command line arguments

MomenTUMv2 is designed as a command-line application. Thus, command-line arguments are the initial communication method between user and software. A command-line argument parser processes the commands and stores them. To do so, we make use of the *JOpt Simple* command-line argument parsing library (Holser, 2016). We address two main commands, the *–config*, followed by a reference to a *XML* file, and the *–loop* command.

The *–config* parameter defines the location where the MomenTUMv2 configuration file is stored. The configuration contains all information to build and run a simulation. The *–loop* parameter activates an internal trigger that forces the framework layer to evaluate the configuration file and check if a simulation should be repeated based on the configuration. If the *–loop* command-line argument is given, the framework will also change the mode of how the configuration file is processed by the simulation layer; thus, slightly changing the configuration each loop. This is useful if a user likes to run multiple simulation experiments in which some parameters change in each framework loop.

4.1.2 Simulation configuration

The configuration of MomenTUMv2 is defined in *XML*. We apply a *XML* to object mapping approach (deserialization) to generate a set of configuration objects. The configuration objects form a hierarchy, which is used as resource for the object factories to generate the simulation objects. For the deserialization, we make use of the *XStream* library (XStream Team, 2016).

Initially, the loop configuration objects are evaluated. The loop configuration defines the number of simulations and redefines specific variables within the configuration text for each simulation. Thus, every time a simulation restarts, parameters will be replaced with new values, slightly altering the configuration.

Besides the loop configuration, the execution order configuration is evaluated. The execution order, if given by the user, redefines the sequence of behavior model execution, including meta models. Thus, simulations that combine model implementations in arbitrary combinations can be defined.

Additionally, some general information are defined in the root tag of the configuration *XML*: the MomenTUMv2 version, the simulation name, the simulation's end time-step in seconds, the simulation's time-step duration and the number of threads for parallel processing.

The configuration comprises *XML* definitions for different model categories: query, perception, operational, walking, standing, tactical, queuing, routing, participating, searching, strategical, analysis, generator, seed, absorber, output, graph, and lattice.

Naturally, layout definitions are given in the layout tag including the elements: area, obstacle, graph, and lattice. The layout elements for areas and obstacles can be sub-typed via attributes and uses point and polygon definitions. A graph comprises edges and vertices, and a lattice is a plain description of cell size and cell type; thus, visualization tools can process graph and cell information. The layout elements can be structured in multiple scenarios.

4.1.3 Model configurations

Each model category tag of the *XML* configuration of MomenTUMv2 defines a set of models for a certain domain. A model comprises a type attribute, name attribute, id attribute, as well as model parameter sub-tags. Thus, all model definitions are structured similarly. The model type has to be defined in the configuration package, the model name is chosen by the user, the model id has to be unique within the category, and the parameters are defined in a generic manner via *XML* tags, which can be translated into a property set. Complex parameter sets can be designed, since the property tags can be nested. The parameters are not parsed and checked regarding data types beforehand; thus, the model implementations need to have access to the parameter name and type by code, to access the properties during simulation runs.

4.2 Simulation managers

The initialization procedure of MomenTUMv2 has multiple functionalities. One is the creation of manager objects, which are mostly independent of the simulation configuration. Each simulation uses management objects, which are services and therefore an integral part of the computations. MomenTUMv2 has five different Managers: component manager, pedestrian manager, the time manager, the scenario manager, and the calling manager. Here, we describe the manager objects and their tasks.

4.2.1 Component manager

The component manager is central to the simulation objects generated. Furthermore, the component manager holds references to all other mangers and creates, organizes, and references all objects that are generated based on the configuration hierarchy.

For object creating, the component manger calls different factories and uses the configuration objects as parameterization. Each factory is responsible to create a specific model domain. Additionally, the component manger links the newly created models with other managers and objects, based on the model's base types.

4.2.2 Pedestrian manager

One important paradigm of the MomenTUMv2 framework is that pedestrians are understand as operands. To ensure this principle, the pedestrian manger is in charge of handling pedestrian related tasks and data.

A pedestrian manager reference is given to all pedestrian generator and pedestrian absorber models to enable communication. If a pedestrian has to be created, e.g., if a pedestrian enters a building scenario, the generator informs the manager about this new pedestrian and provides a set of initial (seed) parameters for the agent. This will trigger calls to all pedestrian behavior models to enable the models to add data to the pedestrian's context. The same concept holds true if a pedestrian is removed, e.g., if a pedestrian leaves a building scenario. Then, the pedestrian manager calls a clean-up method on each model to remove pedestrian related data.

A further important service that the pedestrian manager provides is completely transparent for users of the manager. The manager has a redundant copy of each pedestrian data context. This copy is given in a read only fashion as operand to all model operations, instead of the original pedestrian context. Changes to the copy are transferred to the original data context at the end of the operation within a simulation cycle. This enables an easy implementation of parallelism mechanisms due to disabling all *write-after-write* and similar anomalies.

4.2.3 Time manager

As already indicated above, each simulation runs a fixed number of cycles. To keep track of the current cycle and to compute further time orientated tasks, a time manager is implemented. To solve the tasks, the time manager has access to the time-step duration and keeps track of the runtime-duration of the simulation. The manager also stops the simulation by checking the number of run cycles against the maximal simulation time.

Furthermore, many pedestrian simulation models need access to time-related information. Thus, the time manager implements an interface for a read only service that enables models to access specific time-related information.

The basic information of simulation end-time and time-step duration are given by the *XML* configuration and are put into the time manager during in the initialization phase of the framework.

4.2.4 Scenario manager

The scenario manager handles the geometric elements of a simulation. Thus, it provides access to the scenario related objects and handles the generation of layout elements based on the *XML* configuration. Therefore, models can access the scenario objects by means of the scenario manager.

The elements that the scenario manager organizes are: obstacles (solid and segments), areas (origin, destination, and intermediate), generic properties (size of the bounding box of the scenario), graphs (edges and vertices), and lattices (cells). Each element is defined in two dimensions. The scenario elements are static in their geometric properties. However, they may receive changes in their internal information based on models operations, e.g., graph weights might be altered.

4.2.5 Calling manager

The calling manager is one of the most important objects within the simulation layer. It organizes, parallelizes, and synchronizes calls to models' operations. MomenTUMv2 implements a type hierarchy in addition to callback interfaces. These approaches provide the basis for the generic call concept implemented in the calling manager. Furthermore, the call controller uses configuration objects that defines the user's demands regarding the number of threads and the execution order of pedestrian behavior models.

In the configuration phase of a simulation, the component manger transfers all model references to the calling manager, this is done generically. Then, the calling manager executes specific predefined methods on each model, based on the given model interfaces and the current simulation phase. Therefore, the calling manger provides control to the models layer. Within each simulation cycle, a detailed and partially predefined order of operations is called. We explain details of the model execution ordering in Section 4.5.

The calling manger integrates a threading mechanisms natively, the *manager worker pattern*. Here, the calling manager splits the workload, which are the pedestrian data contexts to process, equally in working bins. Each bin is given to a worker thread, which runs a behavior model operation for each given pedestrian. After all worker threads processed the binds, the controller activates the synchronization between the real and copy pedestrian data contexts, both hold by the pedestrian manager (see Section 4.2.2).

4.3 Simulation model types

MomenTUMv2 provides a set of basic types and sub-types to describe models. We understand models as implementations of theories that compute operations on pedestrian data contexts or in general, operates, uses, and changes data contexts. We distinguish two core types: non-behavior and behavior models. However, non-behavior models are further classified. In general, all models extend a callable type, which is the base class for all models in MomenTUMv2

One concept that all models do have in common, is that they can access a property set, which is defined in the *XML* configuration file hierarchically below the models. A developer can use the property set to define any basic and some complex types as input for each model, without any further implementation effort. Thus, any kind of parameter that can be described in the concept *XML* configuration file is can be accessed in a model implementation.

4.3.1 Non-behavior models

We distinguish three of types non-behavior models.

The first type of non-behavior models does not change the state of pedestrians and operates in the pre- and post-processing phases only. Non-behavior models are scenario oriented models and support models. The scenario oriented models provide important computations that generate graph and lattice objects, which are referenced by the scenario manager. The support models, which are also non-behavior models, are the query and the perception models. Behavior models do have access to query and perception models to use their services.

The second type of non-behavior models has access to all phases in a simulation. Thus, they can run operations during each simulation cycle. The operations of these models do not change pedestrian data contexts, as behavior models do. Nonetheless, they have access to the scenario manager and pedestrian manager to read data. MomenTUMv2 defines multiple non-behavior models that operate on all processing phases: generators, absorber, output, and analysis models.

The third type of non-behavior models operates on the pedestrian data contexts, but does not follow any typical pedestrian behavior paradigm. For example, any kind of meta model that changes the pedestrian data context is such a model. Hence, meta models have unrestricted access to pedestrian data. Furthermore, these models can access the scenario manager to read scenario related data.

4.3.2 Behavior models

Behavior models operate directly on pedestrian contexts and are able to change these. However, the behavior model does have restricted access to the pedestrian data regarding their sub-type. Similar to meta models, behavior models can operate in each of the three simulation phases.

The order, in which behavior models operate, is important for a correct pedestrian behavior simulation. Typically, a strategic model identifies the destination a pedestrian should walk to, then, a spatial task model solves how a pedestrian reaches or approaches the destination, and finally, an operational model changes the position of the pedestrian by computing the motion. This model execution order is repeated in each simulation cycle and drives the pedestrian's behavior by changing the pedestrian's data context. The call manager can change the execution of the behavior models if demanded by the user via the execution ordering configuration. In custom execution pipelines, meta models and execution sequences play a critical role.

4.3.3 Model type hierarchies and interfaces

In Figure 4, we provide the complete model-class inheritance tree. The hierarchy concept solves the demanded modularity and generality of MomenTUMv2. Three classes are not abstract: the classes for operational, strategic, and tactical behavior models. This enables to embed different behavior models in a single behavior object to solve mutual exclusive behavior tasks, e.g, queuing and routing, or walking and standing. Thus, the non-abstract model is a platform to activate the correct sub-model. Therefore, to implement a behavior model, the corresponding sub-model is the base type. From the simulation control layer point of view, multiple non-abstract models can co-exists and are interfaced equally. This concept enables MomenTUMv2 to be extremely modular, without the need of code redesigns.

4.4 Data contexts

MomenTUMv2 provides different data context that are used by models to process their operations. We added three main data domains, the pedestrian data, the layout (scenario) data, and the analysis data. In the following subsections, we give details on each data domain.

4.4.1 Layout data

A simulation scenario comprises of a two-dimensional domain on which multiple geometric concepts are implemented. MomenTUMv2 supports scenarios, obstacles, areas, graphs, and lattices as layout data structures. Multiple instances of each element can coexist in a scenario data context.

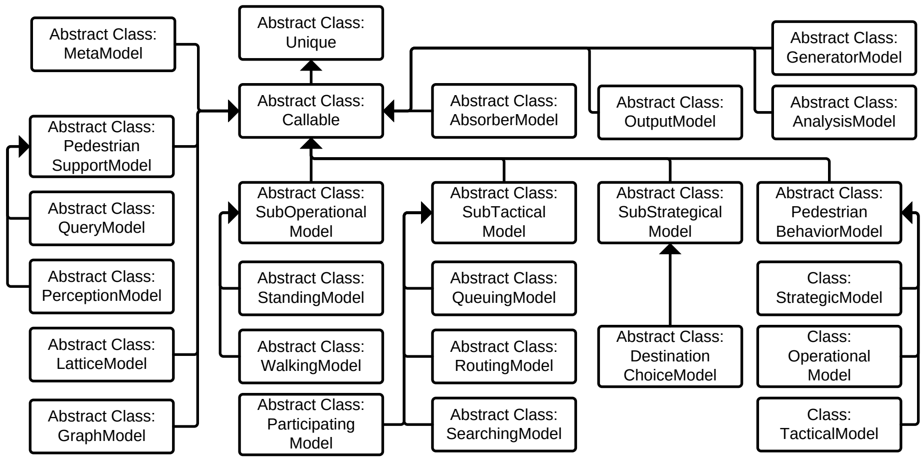

Abstract Class: MetaModel
Abstract Class: Unique
Abstract Class: GeneratorModel
Abstract Class: Callable
Abstract Class: AbsorberModel
Abstract Class: OutputModel
Abstract Class: AnalysisModel
Abstract Class: Pedestrian SupportModel
Abstract Class: QueryModel
Abstract Class: PerceptionModel
Abstract Class: LatticeModel
Abstract Class: GraphModel
Abstract Class: SubOperational Model
Abstract Class: StandingModel
Abstract Class: WalkingModel
Abstract Class: Participating Model
Abstract Class: SubTactical Model
Abstract Class: QueuingModel
Abstract Class: RoutingModel
Abstract Class: SearchingModel
Abstract Class: SubStrategical Model
Abstract Class: Destination ChoiceModel
Abstract Class: Pedestrian BehaviorModel
Class: StrategicModel
Class: Operational Model
Class: TacticalModel

Figure 4: The type hierarchy of MomenTUMv2 for all models. Except of the unique class, all classes implement the *HasProperties* interface, which enables the models to directly read their property sets of the configuration *XML*.

The scenario data context defines the coordinate system of the two-dimensional spatial domain by a minimal and maximal point. The scenario is also the hierarchal root object of all layout data structures.

The obstacles data containers are typical for pedestrian simulations. Obstacles are walls or solids, which are physical barriers for pedestrians. Solid obstacles are defined via closed convex polygons. Wall obstacles are defined via a single segment or a set of segments which are open polygons. Walls have an infinitesimal small thickness.

The area data containers are the base type for origin, destination, and intermediate locations. An area is a convex polygon, which can be traversed by pedestrians. In general, the types for areas are support structures for models; thus, no model implementation is bound to a specific location type. Typically generation models apply their operations in origin areas and absorbers in destination areas. Intermediate areas are multi-functional. For example, they provide a location that pedestrian should interact with, or regions meta models can operate on. Intermediate areas can additionally be defined with a line of interest, which is equal to a segment of the area's polygon. These lines of interest are important for different models, e.g., queuing models use these to model a counter where pedestrians can wait. As additional feature for all areas, each area can be associated to a category, which defines area sets for further use.

In MomenTUMv2, graphs are understood as mathematically defined graphs in which each vertex has a coordinate in the two-dimensional plane. From the implementation point of view, graphs are structured as an object hierarchy, where the root is a graph object, which holds vertex and edge objects as well as supporting functions. Also, further support data structures are given in a graph, e.g., the adjacent vertices of each vertex. Edges and vertices can have weights.

Lattices are data contexts that have versatile use for discretization of the scenario space. In MomenTUMv2, lattices are data objects that predefine an interface to the underlying lattice cells. Furthermore, multiple services regarding the lattice are given, e.g., getting the cell index for a point in the two-dimensional space, occupying and freeing cells, or finding the neighboring cells for a given cell.

The layout data structures are always provided to the models in MomenTUMv2 in a read-only fashion. However, the vertices, the edges and their weights as well as the contents of cells of lattices can be manipulated by models. To reduce conflicts due to parallel processings, write-based operations on lattice and graphs provide a synchronized update interface.

4.4.2 Pedestrian basic data

The pedestrian data context is the interface between all behavioral and meta models and is given for each pedestrian in a simulation. The models change the pedestrian data by applying operations on the internal states of a pedestrian. However, we provide only partial access to the pedestrians data context based on the model type. For example, operational models cannot change the destination location. This ensures that models cannot change the pedestrian data of other behavior domains. In Figure 5, we provide the pedestrian interface hierarchy.

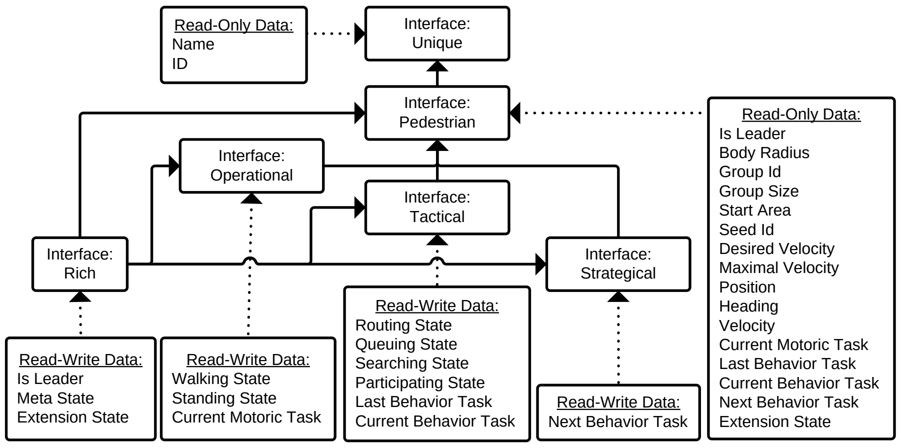


Figure 5: The interface hierarchy of the pedestrian data context. The solid arrows describe the *extends* dependency. The dotted arrows describe the interfaces' properties.

The pedestrian data context is initialized on behalf of a generator model via the pedestrian manager. For generating, an agent population configuration has to be defined in the configuration file, the pedestrian seed. The seed provides basic standard values and distributions for the pedestrian data. However, the data may be enhanced by computing further context manipulations within a generator implementation, e.g., the initial heading.

Each behavior sub-model has a corresponding state in the pedestrian data context. These states are similar for each of the sub-model types. For example, the walking and standing states both have a heading and position properties. Based on the currently activated sub-model, e.g., walking, only the data of the active state is provided. Thus, if a pedestrian is currently walking and a model demands the pedestrian's position, the position of the walking state is given. This ensures a throughout consistent pedestrian data context in the presence of multiple models that change the context. Furthermore, a meta state is given for each pedestrian. This state holds information regarding model types that are currently not allowed to operate on a pedestrian. Typically, this state is empty, defining no restriction. However, in hybrid simulations, in which pedestrian contexts are updated by multiple models of the exact same sub-type, this is a useful tool to define the hybrid models computations rules.

The data context of a pedestrian covers the most basic elements for a pedestrian simulation. However, models may add further context via extension states, which can only be accessed by the model that created it. These additional data can be added by any behavior or meta model if a specific interfaces was implemented. The interface is called by the pedestrian manager on pedestrian creation. Similarly, on pedestrian removal, the manager calls each model to provide an opportunity to clean-up the data extension.

4.4.3 Analysis data

Analysis models compute and measure valuable data based on pedestrian simulation results. However, the types cannot be defined beforehand because different analysis and measurement methods demand different input and output types. Thus, the analysis data is based on a generic type system.

The base element of an analysis data is an analysis element. The element comprises three data properties: the unique identification number of which the data originates from, the simulation cycle number during which the data was acquired, and the data as a number value. An analysis element set is therefore a collection that sorts the analysis elements regarding identification number and simulation cycles. This enables the analysis model developer to access all elements for a single cycle, or the data of a specific identification number for all simulation cycles.

The input or output types are predefined via configuration and have to be valid on runtime, similar as types to interpreted programing languages. The output types are defined equally. The data sources for input data are operand data contexts, specific model data, or file based data. The data source for output data is the analysis model itself; hence, output writer models can use analysis models as data sources.

4.5 Execution order

The control flow presented in Figure 3 describes roughly how the pre- and post-processing as well as the simulation based cycle processing order is defined. However, there are some predefined and some dynamic concepts regarding the processing order, which are tightly coupled to the model sub-types. Furthermore, the processing phase regarding behavior and meta models is generic. This enables researchers to combine models processing orders as they wish.

4.5.1 Pre-processing operation order

The pre-processing of the models follows a clear order. Initially, scenario related operations have to be computed to provide a spatial environment. These layout objects are of high importance and the data basis of model operations that pre-process geometrical based solutions on the scenario. Following the scenario related pre-processing, further model processings can be done in parallel. In Figure 6, we provide the pre-processing operation order dependencies.

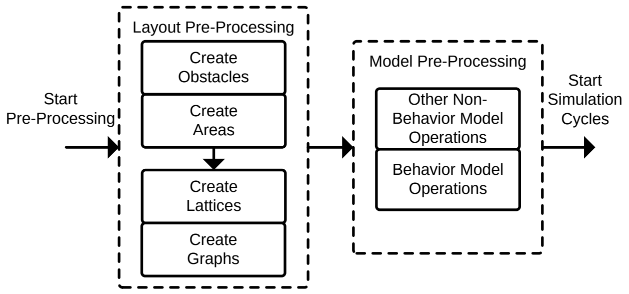


Figure 6: The order of model computations in the pre-processing phase of MomenTUMv2.

4.5.2 Post-processing operation order

In contrary to the pre-processing phase, the post-processing phase is less restricted by dependencies. However, the analysis models can compute post-hock results based on data of the simulations, which output models have to write to output streams after the analysis. The dependencies of the post-processing are given in Figure 7.

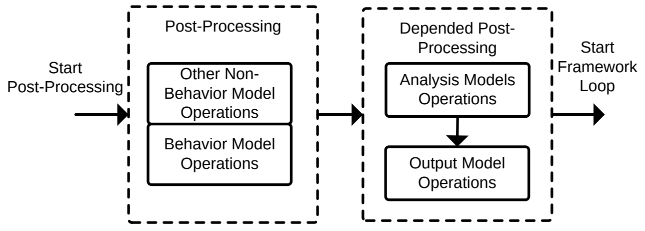


Figure 7: The order of model computations in the post-processing phase of MomenTUMv2.

4.5.3 Processing order

The processing of a cycle in a simulation follows a strict order of model execution and was already shown in Figure 1. However, the operations of behavior models marks the centerpiece of the processings. The behavior model execution can be completely customized, but also provides a standard approach.

MomenTUMv2 enables the classical order of processing the three behavioral layers – strategic, tactical, and operational – without any further configuration definitions. Hence, one has to define a destination choice, a routing, a queuing, a searching, a participating, a walking, and a standing model in the configuration. These behavior models will be executed based on commons sense regarding data dependencies: strategic models first, then a tactical spatial task solving model, and finally an operational model. If models are missing, dummy models are set in action, which functionally do provide the most simplistic pedestrian behavior approach for the corresponding behavior category.

By configuration, the classical concept can be deactivated and a custom behavior model execution order can be defined. This is especially of interest for hybrid modeling or for introducing additional meta models that operate on pedestrian data, but do not follow typical behavior concept in pedestrian dynamics. The execution order is built on arbitrary numbers of execution blocks defined in the *XML* configuration. Each execution block can comprise an ordered set of references to pedestrian behavioral layers. Furthermore, for each block a multiplier value is given. The multiplier defines in the sense of modulo calculus, in which cycle a block should be executed regarding the minimal time-step definition. Every time a block is executed, the models of this block are executed too. Furthermore, the blocks and the model references within these blocks are ordered. This predefines the block-internal and block-external execution order. In each block, references to meta models can be provided, including a multiplier for each meta model. The meta models will be called if a block is called based on the block multiplier and the meta model multiplier.

The execution order starts the operations on pedestrian data contexts for each behavioral or meta model in parallel, based on the number of defined threads. However, only pedestrian data contexts are always thread-safe regarding synchronization dependencies, the layout data containers are not. Nonetheless, each model operation call is given the information on which thread it computes to provide a built-in option to synchronize model. Furthermore, the layout data structures provide atomic operations to help the model developer to ensure synchronized computations. Nonetheless, simulation can always be executed on a single thread only, if necessary.

4.5.4 Operations on pedestrians context

In MomenTUMv2, the operation can change the pedestrian data contexts in different ways. Here, we define all operations regarding pedestrian data context within the processing phases of a simulation

The following operation access points are given for meta models: *on pedestrian creation*, *on pedestrian removal*, *before pedestrian behavior*, *after pedestrian behavior*, and *for all pedestrian*. Pedestrian behavior model do not provide a *for all pedestrians* access point, but the *for a single pedestrian* hook. This method is more granular and operates on a single pedestrian only.

The order of the computations is predefined. If a pedestrian is generated, the models have access to the *on pedestrian creation* operation. If a pedestrian is removed, the models have access to the *on pedestrian removal* operation. During ongoing model execution different interfaces are given. In the processing phase of a meta or behavior model, the *before pedestrian behavior* operation is activated for the complete set of pedestrians first. Afterwards, *for all pedestrian* is activated in parallel for all pedestrians for meta models, or the *for a single pedestrian* operation for behavior models on each pedestrian. Then, the *after pedestrian behavior* operation is executed for all pedestrians for both model types. This operation splitting is especially useful because some models are in need to do pre- and post-operations before actually applying the behavior or meta operations on pedestrians.

The described operation interfaces ensures that all models' demands are covered, even if some models do not make use of all operation hooks.

5 Example application

We report about an example application that uses the described methodologies and paradigms implemented in our agent-based pedestrian simulation framework. Thus, we provide evidence that the MomenTUMv2 works in practice and provide the functionalities as described above. The example application comprises a complex multi-hybrid simulation scenario: the simulation of the event course of the *Back To The Woods* festival (Biedermann et al., 2015; Ehrecke, 2016; Biedermann et al., 2016).

In Section 5.1, we describe the study's simulation scenario. Section 5.2 briefly presents the multiple models used for the simulations and how they were configured in *XML*. Some details of the simulation are given in Section 5.3

5.1 Simulation scenario

The *Back To The Woods* is an one-day festival for approximately 5000 visitors that is held near Munich every year. The visitors of the festival are young adults, which mostly arrive at the event location by subway. However, from the subway station, the visitors still have to walk about 830 meter beeline on streets closed for cars to reach the festival grounds.

Biedermann et al. (2016) presented a theoretical simulation case study, in which the assumption was made that the subway service is not functional and a shuttle bus system has to be introduced to replace the public transport service. Furthermore, the course of the event was simulated over multiple hours. To do so, the study combined multiple models and created a holistic simulation by means of the MomenTUMv2 framework.

In this theoretical scenario, all visitors arrive by the shuttle bus at the bus station in front of the festival entrance area. Figure 8 shows the layout of the simulation scenario connected with an aerial image of the surrounding environment. Starting at the station, the pedestrians walk along the path to the entrance area and line up to enter the event site. Since the festival is sold out beforehand, no tickets are sold at the entrance. Nonetheless, all visitors have to be checked for dangerous objects, their presale-tickets have to be scanned, and they receive their admission wristlets. Since this procedure takes some time, and the visitors arrive in groups according to the schedule of the shuttle buses, crowd congestions easily arise at the entrance area. After a visitor entered the actual event he or she may go to different location on the event: they can go to different bars, to various food stands, to the large dancing area in front of the stage, or to the sanitary facilities. Each pedestrian on this event has individual preferences for these locations. Thus, the visitors change their preference over time and visit other areas of the festival. Furthermore, visitors may leave the event site and heading towards the bus station to travel home.

All geometrical aspects described here to be considered to realize the simulation. To do so, the scenario layout for the simulation was drawn via the CAD-tool Autodesk AutoCAD (Autodesk Inc., 2016a) and transformed via a plug-in tool into the layout elements of XML-format of MomenTUMv2.

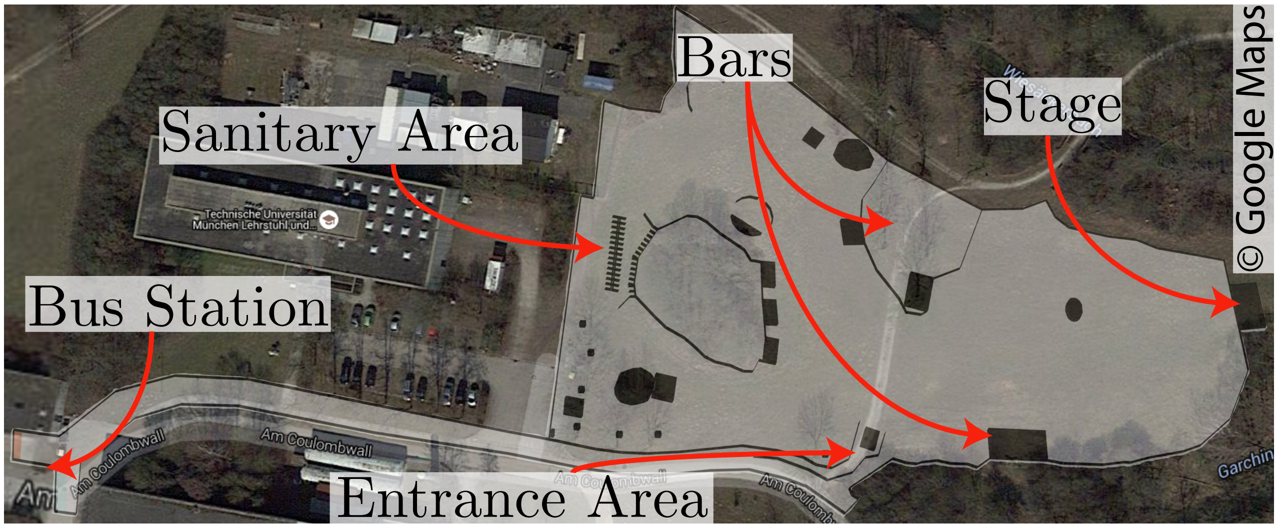


Figure 8: CAD based scenario of the *Back To The Woods* festival mapped on an aerial view.

Listing 1: Example scenario layout configuration.

```
<layouts>
 <scenario id="0" name="BTTW" maxX="149.1" maxY="104.5" minX="-96.5" minY="-6.8">
  <obstacle id="0" name="Wall0" type="Wall">
   <point x="30.9" y="57.6"/>
   <point x="31.8" y="56.9"/>
  </obstacle>
  <obstacle id="1" name="Solid1" type="Solid">
   <point x="17.0" y="31.2"/>
   <point x="18.8" y="31.2"/>
   <point x="17.0" y="32.2"/>
  </obstacle>
  <area id="0" name="Station" type="Origin">
   <point x="-89.1" y="-6.4"/>
   <point x="-87.0" y="-6.7"/>
   <point x="-86.0" y="2.4"/>
  </area>
  <area id="1" name="wc0" category="wc" type="Intermediate">
   <point x="24.8" y="52.8"/>
   <point x="24.3" y="52.0"/>
   <point x="24.2" y="51.5"/>
   <gatheringLine>
    <point x="24.3" y="52.0"/>
    <point x="24.2" y="51.5"/>
   </gatheringLine>
  </area>
  <area id="2" name="Exit" type="Destination">
   <point x="-96.4" y="4.9"/>
   <point x="-95.0" y="3.9"/>
   <point x="-94.1" y="10.1"/>
  </area>
   [...]
 </scenario>
</layouts>
```

Listing 1 is an exemplary extract of the scenario layout for the *Back to the Woods* festival. The *layouts* section contains an arbitrary number of simulation scenarios. A *scenario*-tag has an individual identification number *id* and a coordinate system described by two points $P_{min} = (minX, minY)$, and $P_{max} = (maxX, maxY)$. The static obstacles in Listing 1 are the physical barriers of a simulation scenario. Obstacles can be either of type *Solid* or of type *Wall*. The geometric shape of an obstacle is described by an arbitrary number of *points*, which define a polygon. Other structure elements of the layout are areas with special characteristics. An area of type *origin* is a place to generate new pedestrians during the simulation, whereas areas of type *destination* are locations where pedestrians are removed. Areas of the type *intermediate* are temporary destinations, which pedestrians can visit without being removed from the simulation. These intermediate areas help to model how pedestrians visit different locations on the event site. If many people are heading towards the same location, e.g., a food stands or the sanitary facilities, queues can occur. The *gatheringLine* of an intermediate location defines the orientation at which the queue is built, but also where the line of attention is in front of a stage. The *category*-tag defines a sub-type of an intermediate goal. Figure 9 shows an example of different types of areas on the *Back to the Woods* festival.

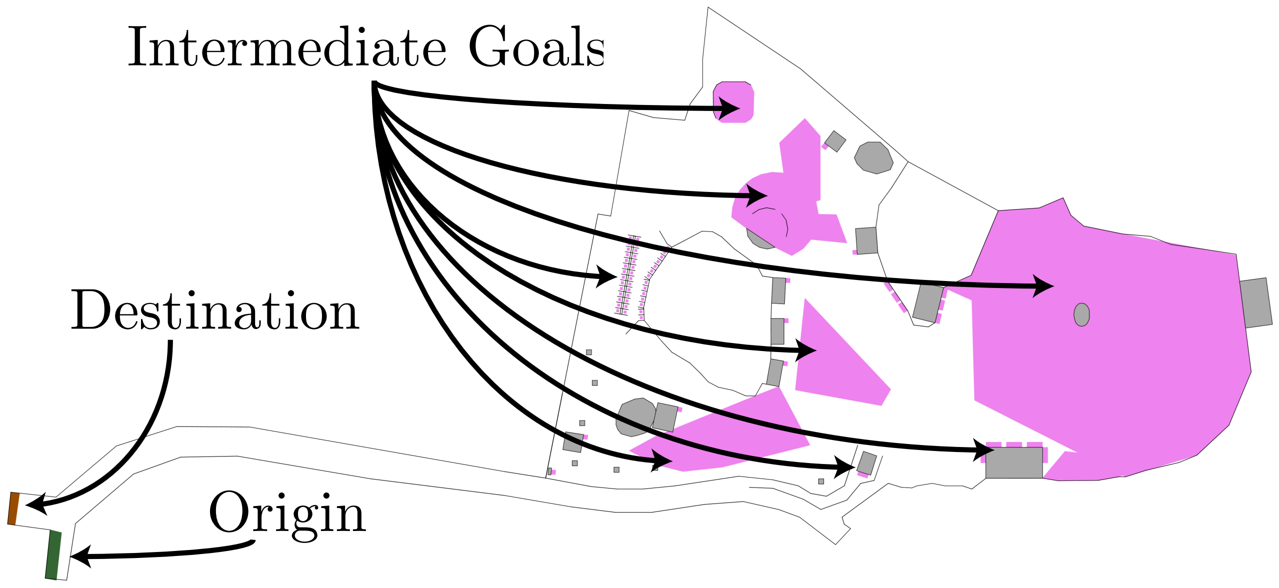


Figure 9: Different types of areas in a simulation scenario

In addition to the geometric layout of the scenario, time-related and generic information have to be defined for a simulation, as shown in Listing 2. The *simulator*-tag is the root element of any configuration in MomenTUMv2 and comprises necessary information for any simulation. The *version* attribute describes the current version of the simulator, *simulationName* the name of the simulation scenario, *simEnd* the total duration of the simulation in seconds and *timeStepDuration* is the duration of a simulation cycle. Furthermore, the number of threads the pedestrian behavior models should operate on is given by the *threads*-attribute.

Listing 2: General simulation configuration example.

```
<simulator version="0.9.0" simulationName="BTTW" simEnd="43200" timeStepDuration="0.05"
    threads="3">
 [...]
</simulator>
```

5.2 Simulation design

The goal of the study by Biedermann et al. (2016) was to show the possibility of a holistic simulation of mobility on a public event. Further, they studied the impact on the event dynamics by introducing a shuttle bus system if the public transport is out of order. Therefore, a large number of singular models have to interact with each other to reach this goal.

In that case study, the visitors of the *Back to the Woods* festival travel to the event by shuttle buses. The arrival time of this public transport service is based on a shuttle bus optimization-simulation model developed by the research group of Prof. Dr. Stefan Ruzika (University Koblenz-Landau). More details about their model can be found at Torchiani et al. (2015) and Biedermann et al. (2016). Their optimization-simulation approach is able to calculate realistic schedules for the arriving times of shuttle buses. Pedestrians were generated at the origin according to the time interval of incoming shuttle buses. Such a generator model can be seen in Listening 3.

Listing 3: Example generator, absorber, and seed configurations.

```
<generator id="0" name="Station" scenarioId="0" origin="0" seed="0" type="Plan">
 <property name="startTime" type="Double" value="0"/>
 <property name="endTime" type="Double" value="Infinity"/>
 <property name="basicHeading" type="Double" value="0"/>
 <property name="maximalPedestrians" type="Integer" value="140"/>
 <property name="safetyDistance" type="Double" value="2.0"/>
 <property name="latticeId" type="Integer" value="0"/>
 <geometry geometryType="Lattice" fillingType="Random"/>
 <complexProperty name="interval" type="List" valueType="Double">
  <entry index="0" value="0"/>
  <entry index="1" value="10"/>
  <entry index="2" value="40"/>
 </complexProperty>
 <complexProperty name="percentage" type="List" valueType="Double">
  <entry index="0" value="0.60"/>
  <entry index="1" value="0.40"/>
  <entry index="2" value="0.0"/>
 </complexProperty>
</generator>
<pedestrianSeed id="0" name="basic" type="NoDistribution">
 <property name="desiredVelocity" type="Double" value="1.34"/>
 <property name="maximalVelocity" type="Double" value="2.7"/>
 <property name="radiusMeter" type="Double" value="0.23"/>
</pedestrianSeed>
<absorber id="0" name="Exit" scenarioId="0" destination="84" type="DestinationSelected">
 <property name="vanishTime" type="Double" value="1.0"/>
</absorber>
```

Different generators, absorbers, and pedestrian seeds can be defined for a simulation. A *generator* of the type *plan* is linked to an origin area and contains a time span from *startTime* to *endTime*, the maximal number of pedestrians that are allowed to be generated, and an optional *latticeId* tag to reference a lattice model. The lattice is used by a generator if the *geometryType* defines that pedestrians are generated inside the cells of a grid. The *safetyDistance* is the minimal distance between newly generated and exiting pedestrian, and the *basicHeading* gives the start heading direction of new pedestrians. In the case study, the incoming pedestrians were generated interval-wise to model the arrival of shuttle buses. An interval example is given in Listing 3. The *interval* property defines the time intervals in which the *plan* typed generator creates pedestrians. The *percentage* defines the share of the maximal number of pedestrians, which are generated in an interval. The definition of the generator intervals are in accordance to the arrival times of the shuttle buses computations.

The tag *pedestrianSeed* defines the pedestrian-agent population based attributes. Other *pedestrianSeed* models can contain additional attributes, e.g. characteristics about groups.

In analogy to generators, absorbers have to be linked to destination areas. The *DestinationSelected* absorber type defines that only pedestrians, who have this area as their current destination, are removed from the scenario. This ensures that no pedestrian, who randomly trespasses the destination area, is deleted. If a pedestrian enters this area and it is his or her destination, the pedestrian will be deleted after a timespan defined in *vanishTime*.

The simulation scenario comprises predefined geometric elements. However, graphs and lattices are additional concepts that pedestrian dynamics models use to compute pedestrian behavior. Graphs and lattices can be added by graph and lattice models as exemplary shown in Listing 4.

Listing 4: Example configuration for graph and lattice models.

```xml
<lattices>
 <lattice id="0" scenarioId="0" latticeType="Quadratic" neigborhoodType="Touching"
     fillingType="ScenarioLayout" cellEdgeSize="0.46"/>
</lattices>

<graphs>
 <graphModel id="0" name="RoutingGraph">
  <graphOperation id="0" name="raw" type="RawGraph" order="0"/>
  <graphOperation id="1" name="seeds" type="VertexCreateSeedBased" order="1"/>
   <graphOperation id="2" name="corners" type="VertexCreateAtCorners" order="2">
    <property name="cornerDistance" type="Double" value="0.7"/>
   </graphOperation>
   [...]
   <graphOperation id="4" name="visibility" type="EdgeCreateAngleBased" order="5">
    <property name="alpha" type="Double" value="30"/>
    <property name="visibilityTolerance" type="Double" value="0.1"/>
   </graphOperation>
  <graphOperation id="5" name="remove" type="EdgeRemoveUnreachable" order="6"/>
  <graphOperation id="6" name="toConfiguration" type="ToConfiguration" order="7">
   <property name="scenarioId" type="Integer" value="0"/>
  </graphOperation>
 </graphModel>
</graphs>
```

For some models, e.g., cellular Figure models, a lattice is necessary. Lattice models are defined by a *lattice*-tag. Further, the *latticeType* defines the grid cells' shape (e.g. quadratic, hexagonal), the *neighborhoodType* defines under which conditions two cells are defined as neighboring cells, and the *cellEdgeSize* defines the length of an edge of a grid cell. Currently, three ways exist to initialize the cells of a lattice. The parameter *Empty* defines every cell as free, *Full* occupies all cells, and *ScenarioLayout* fills only cells, which intersect an obstacle.

Some models use graphs, e.g., routing models use visibility graphs to compute wayfinding behavior. To create graphs, automatic graph calculation models exist in MomenTUMv2, e.g., the method of Kneidl et al. (2012). These models are defined within a *graphModel*-tag via multiple ordered *graphOperation*-tags that define graph manipulation methods (Aumann and Kielar, 2016). Figure 10 shows the visibility graph for the event site of the *Back to the Woods* scenario layout. If the layout of the scenario is quite large and complex, the generation of such visibility graphs takes a large computational effort. Thus, it is useful to generate the graph once and save it via the *ToConfiguration* graph operation. This operation adds the generated graph to the scenario configuration via *graph*-, *vertex*- and *edge*-tags.

The behavior of pedestrians is decomposed into three modular but interacting layers (Hoogendoorn and Bovy, 2004): the strategical, tactical, and operational layer. Listing 5 gives an example of how the strategical, tactical and operational layers are structured in the configuration file. The strategical models are defined directly, but the tactical and operational sub-models are defined later but referenced by id.

In the case study, they used a cognitive based destination choice model developed to describe the decision process for intermediate goals and destinations based on work of Kielar et al. (2014) and Kielar and Borrmann

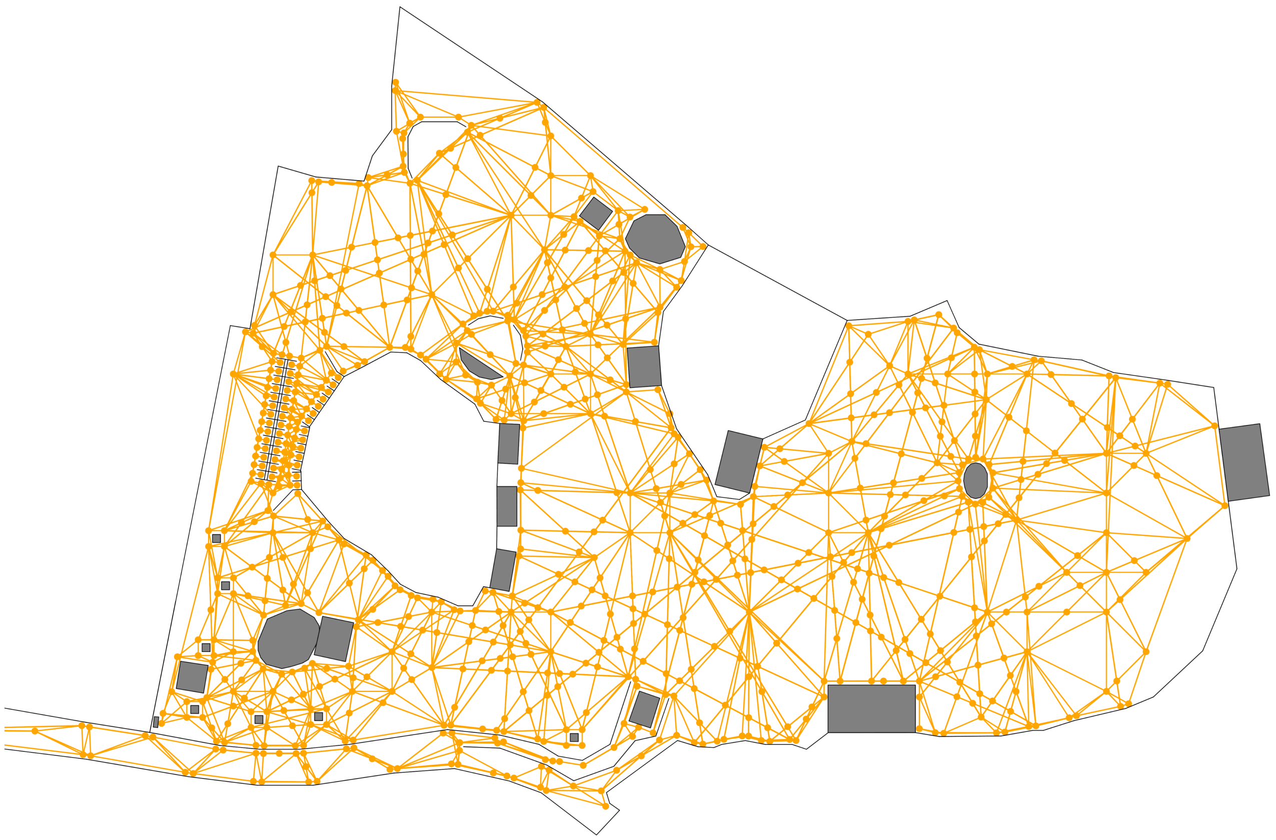


Figure 10: Visibility graph for the event site of the *Back to the Woods* festival.

(2016b). Due to the model's complexity, many characteristic have to be defined. Each intermediate goal has its own properties, e.g. service-times and power of attraction. Thus, for each intermediate goal, individual parameters where defined by a *propertyContainer*, which contains *property*-tags, *complexProperty*-tags, and further *propertyContainer*-tags.

The tactical layer is simulated by means of the spatial task model comprising four individual models (Kielar and Borrmann, 2016a): a participating, a queuing, a routing and a searching model (see Listing 6). The *participating* defines how pedestrians find a position to wait in an area of interest (Kielar and Borrmann, 2016a). A good example is the show on a stage of a public event. All visitors interested to see the show will swarm the stage to look at the artists. The *queuing*-tag contains the parameters necessary for a queuing model. The study used the angular-based queuing model, which is based on approaches presented by Kneidl (2016). Figure 11 shows such a queue at the entrance area of the event site. The unified pedestrian routing model (Kielar et al., 2016) was used as routing model in the event simulation. For the *searching* only a dummy model was used because a sophisticated search concept was not necessary in the event simulation.

In the case study, two different operational walking models were used (see Listing 7): a cellular automation and a continuous space model. As a cellular automation the cellular stock model from Biedermann et al. (2016) was used. All pedestrians in this model have their own stock, which grows over time and a movement is only allowed if the stock has reached a predefined threshold. The second walking model used is the social force model developed by Helbing et al. (2000). In this model, a pedestrian is affected by repulsive forces from other pedestrians and from obstacles. Additionally, a pulling force originates from the destination a pedestrian is heading to. Therefore, his or her movement path is calculated by the superposition of the attractive and all

Listing 5: Example configuration of the behavior model overall configuration.

```
<strategical id="3" name="csc" type="CognitiveSpatialChoice" perceptualModel="0"
    queryModel="0">
 <property name="cognitiveClock" type="Double" value="0.25"/>
 <property name="openingMalus" type="Double" value="0.0"/>
  [...]
 <propertyContainer id="0" name="tickets">
  <property name="preferenceId" type="Integer" value="0"/>
  <property name="oneTime" type="Boolean" value="True"/>
   [...]
 </propertyContainer>
  [...]
</strategical>
 [...]
<tactical id="2" name="tactical"  perceptualModel="0" queryModel="0">
 <participatingReference modelId="0"/>
 <queuingReference modelId="0"/>
 <routingReference modelId="0"/>
 <searchingReference modelId="0"/>
</tactical>
 [...]
<operationalModels>
 <operational id="0" name="operational" perceptualModel="0" queryModel="0">
  <standingReference modelId="0"/>
  <walkingReference modelId="0"/>
 </operational>
 <operational id="1" name="operational" perceptualModel="0" queryModel="0">
  <standingReference modelId="0"/>
  <walkingReference modelId="1"/>
 </operational>
</operationalModels>
```

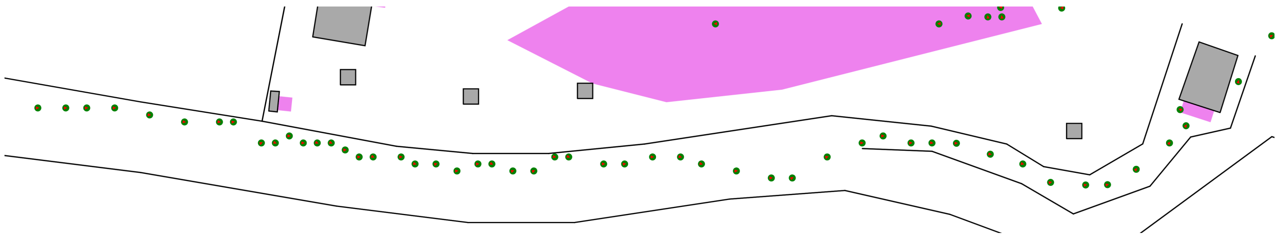


Figure 11: Queue at the entrance area of the music festival.

repulsive forces. Each operational model has two different modes: *walking* and *standing*. For both operational models the standing model *FixedStanding* was chosen. The model ensures, that pedestrians stay on their current position if they changed from walking to standing.

An event simulation is quite complex compared to pure evacuation scenarios, in which all pedestrians are heading towards a set of exit destinations. Additionally, the time-scale of a public event simulation is much longer. In an evacuation scenario, the whole time-span of the scenario is typically in the range of minutes. In contrary, a public event continues over a time-span of several hours with thousands of individual pedestrian agents. This introduces a heavy burden on the computation time. Hybrid approaches help to overcome this issue. Cellular automation calculate fast, but are limited in their spatial resolution by the size of one unit cell, whereas continuous models have a high spatial resolution but include large computational effort (Biedermann et al., 2016). The case study combined the fast cellular stock model with the social force model to reduce the overall computational effort by simulating only potentially dangerous areas of the event with high spatial

Listing 6: Example spatial task solving model configurations.

```
<participating id="0" name="shiftedParticipating" type="ShiftedRandomParticipating">
 <property name="participateDistance" type="Double" value="4.0"/>
 <property name="groupPositionRadius" type="Double" value="2.0"/>
 <complexProperty name="gatheringCrowding" type="List" valueType="String">
  <entry index="1" value="Everywhere"/>
  <entry index="2" value="Close"/>
   [...]
 </complexProperty>
  [...]
</participating>
<queuing id="0" name="angularQueuing" type="AngularQueueing" >
 <property name="queueArc" type="Double" value="30"/>
 <property name="queueDistance" type="Double" value="0.92"/>
  [...]
</queuing>
<routing id="0" name="unifiedRouting" type="UPRM">
 <property name="herding" type="Boolean" value="true"/>
  [...]
 <complexProperty name="resultMode" type="CsvMatrix" valueType="Double">
  <entry file="\UPRM_calibriation.csv" separator=";"/>
 </complexProperty>
  [...]
</routing>
<searching id="0" name="noSearching" type="NoSearching"/>
```

Listing 7: Example operational model configurations.

```
<standing id="0" name="fixedStanding" type="FixedStanding" />
<walkingModels>
 <walking id="0" name="stock" type="StockCellular">
   <property name="scenarioLatticeId" type="Integer" value="0"/>
   <property name="timeStepMultiplicator" type="Integer" value="4"/>
 </walking>
 <walking id="1" name="socialForceModel" type="SocialForce">
  <property name="relaxation_time" type="Double" value="1"/>
  <property name="physical_interaction_kappa" type="Double" value="2.4e5"/>
  <property name="physical_interaction_k" type="Double" value="1.2e5"/>
  <property name="panic_degree" type="Double" value="0.4"/>
   [...]
  </walking>
</walkingModels>
```

resolution (e.g. entrance area). The coupling was done by the generic and zone-based transformation model TransiTUM (Biedermann et al., 2014). An exemplary input data for this meta model is shown in Listing 8.

The TransiTUM meta model is able to transform pedestrians from any cellular automation to any continuous space model and vice versa by using shared transition zones. Figure 12 shows the coexistence of these two model types in one shared simulation. Since the cellular automation and the continuous model can have different time-step durations and since the transformation model has to act in between the simulation cycles, an *executionOrder* has to be defined. Each *executionBlock* has a *multiplier*, which describes how often the block is called in whole multiples of the minimal time-step of this simulation. Furthermore, each *executionBlock* has a *complexProperty*, which contains all models of the execution block. The list includes optional references to strategic, tactical, and operational model configurations. The use of two execution blocks enables the simultaneous use of two pedestrian dynamics models for the same behavior type (here walking). The operations of the TransiTUM models are executed if the *multiplier* of the *blockManipulator* indicates the correct cycle

Listing 8: Example meta model and execution order configuration.

```xml
<meta id="0" name="transiTum" type="TransiTum">
 <propertyContainer name="ConfigurationData" id="0">
  <property name="mesoscopicTimeStepMultiplikator" type="Integer" value="4"/>
  <property name="microscopicTimeStepMultiplikator" type="Integer" value="1"/>
  <property name="maximalVelocity" type="Double" value="2.12"/>
  <property name="transitionAreaFactor" type="Double" value="15"/>
  <property name="scenarioLatticeId" type="Integer" value="0"/>
  <property name="collisionDetectionLatticeId" type="Integer" value="0"/>
 </propertyContainer>
 <propertyContainer name="MultiscaleArea" id="1">
  <property name="areaType" type="String" value="Microscopic"/>
  <property name="radius" type="Double" value="12"/>
  <property name="xPosition" type="Double" value="67"/>
  <property name="yPosition" type="Double" value="13"/>
 </propertyContainer>
  [...]
</meta>

<executionOrder>
 <executionBlock id="1" orderNumber="0" multiplier="1">
  <complexProperty name="models" type="List" valueType="Integer">
   <entry index="0" value="3"/>
   <entry index="1" value="2"/>
   <entry index="2" value="1"/>
  </complexProperty>
 </executionBlock>
 <executionBlock id="0" orderNumber="1" multiplier="4">
  <complexProperty name="models" type="List" valueType="Integer">
   <entry index="0" value="3"/>
   <entry index="1" value="2"/>
   <entry index="2" value="0"/>
  </complexProperty>
  <blockManipulator metaId="0" multiplier="1"/>
 </executionBlock>
</executionOrder>
```

based on modulo calculus. If allowed, the corresponding meta model with the identification number *metaId* is executed.

Another model type are output models (see Listing 9). These models write data to output streams. Different output writers were used in the presented case study. The *csvSingleWriter* output model prints time-dependent *CSV* data to files, e.g., for the pedestrian data and the transition zone data. Furthermore, a *listWriter* was used to write statistical data of the simulation as a list to a file. If the analysis module of the simulator framework is used, more output writers are necessary. The analysis module calculates additional information (e.g. occupancy at intermediate goals or local flows) based on the simulation results. These data are quite useful to obtain helpful insights into the crowd dynamics of the simulated scenario. The *writerSource*-tags define the writer data source; thus, the objects from witch the information are read within the simulation.

5.3 Simulation execution

The start of the simulation is done via the command line. The simulation is started with a set of different options:

Listing 10: Command line to start the simulator.

```
java MomenTUMv2.jar --config "path\BTTW.xml" --loop 1
```

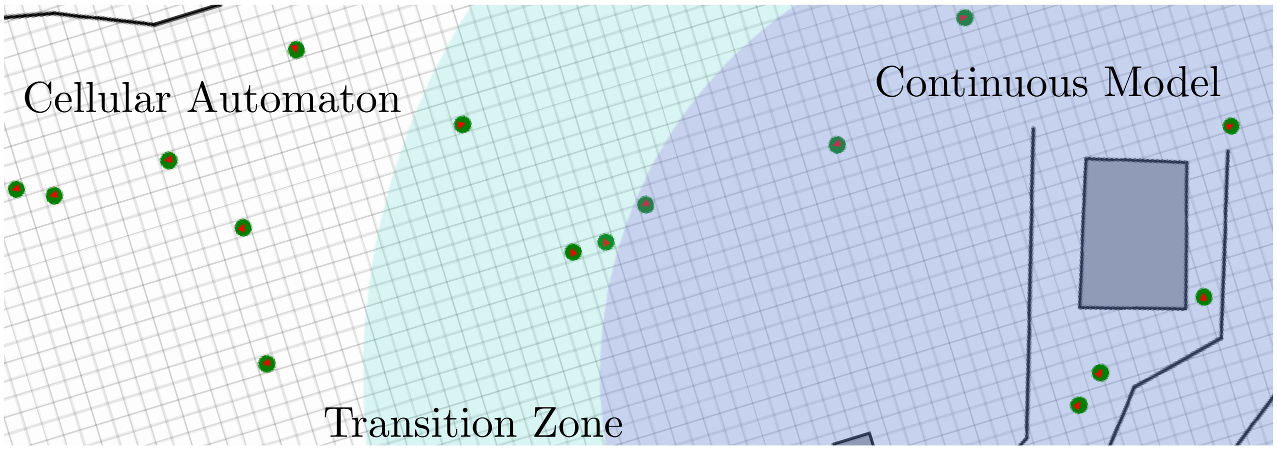


Figure 12: Coexistence of a cellular automation and a continuous agent model enabled by a hybrid approach.

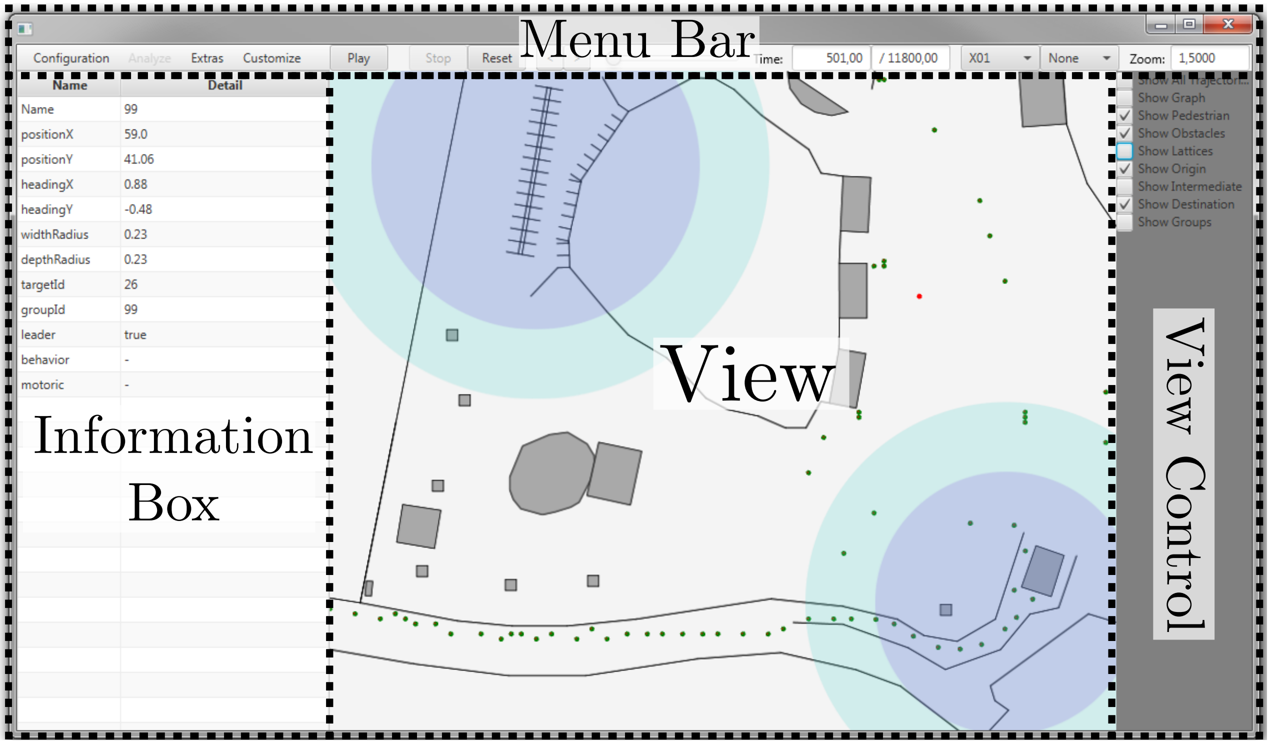
Listing 9: Example output writer model configuration.

```
<outputWriter id="0" name="PedWriter" type="csvSingleWriter">
  <property name="call" type="Integer" value="5"/>
  <property name="file" type="File" value="./BTTW.csv"/>
  <property name="delimiter" type="String" value=";"/>
  <property name="buffer" type="Integer" value="500"/>
  <complexProperty name="heading" type="List" valueType="String">
   <entry index="0" value="timeStep"/>
   <entry index="1" value="id"/>
   <entry index="2" value="x"/>
   <entry index="3" value="y"/>
  </complexProperty>
  <writerSource name="basic" sourceType="Pedestrian">
   <property name="timeStep" type="Format" value="%d"/>
   <property name="id" type="Format" value="%d"/>
   <property name="x" type="Format" value="%.2f"/>
   <property name="y" type="Format" value="%.2f"/>
</outputWriter>
```

The *–config* flag determines the path to the configuration file and the *–loop* flag defines how many simulation loops are executed. In this case, the simulation runs only once.

After a simulation, the written output data needs to be analyzed. Besides the data calculated by analysis models, the visualization module of MomenTUMv2 is quite useful to get insights into the crowd dynamics of a simulation scenario.

Figure 13 shows the individual parts of the visualization module. The *Menu Bar* contains several different commands to control the visualizer. In the *Configuration* menu, the user can load a layout scenario, an output file with pedestrian data, other customized comma separated files (e.g. to visualize the transition zones of the TransiTUM meta model), or close the visualizer. The *Analyze* menu is currently only present as a dummy, but will be linked to the MomenTUMv2 analysis models in the future. The *Extra* menu provide various tools like a record and snapshot function, or a search tool to identify different entities in the view. The *Customize* menu allows the definition of individual coloring of simulation elements and other configurations. Furthermore, the *Menu Bar* contains some buttons and sliders to control the timely flow of the simulation visualization. These control elements are similar to one's of a video player (e.g. scroll forward and backward in time, jump to a given time-step, fast forward, etc.). Since the simulation data for the movement of pedestrians is given in discrete time-steps, options are available for the interpolation between two time-steps (none, linear, and b-spline cubic). The *View* shows all visible objects of the simulation scenario. Which elements are visualized by the *View* can be chosen in the *View Control*. Such elements are pedestrians and their trajectories, graph structures, obstacles, the lattice, origins, intermediate goals, and destinations as well as pedestrian group coloring. If a specific visualized element is selected on the *View*, additional information is shown in the *Information Box*. Such information are for example the pedestrians' identification numbers, their positions and headings, their body radius, all based on the contents of the *CSV* or *XML* data. Furthermore, most of the described functions can be controlled via keyboard shortcuts. An overview of the most important ones is given in Table 1.



Name	Detail
Name	99
positionX	59.0
positionY	41.06
headingX	0.88
headingY	-0.48
widthRadius	0.23
depthRadius	0.23
targetId	26
groupId	99
leader	true
behavior	-
motoric	-

- Show All Trajectori...
- Show Graph
- ✓ Show Pedestrian
- ✓ Show Obstacles
- Show Lattices
- ✓ Show Origin
- Show Intermediate
- ✓ Show Destination
- Show Groups

Figure 13: Visualization module of MomenTUMv2

6 Upcoming research and developments

The framework MomenTUMv2 is still under development. Therefore, some features will be integrated in the future. This will provide a better compatibility with further more pedestrian behavior models.

The data contexts of MomenTUMv2 will be extended to enhance the frameworks capabilities. The layout data will be completed by the third dimension. This extension will enable multi-floor simulations and ensure that models, which can handle the height property, can be added, e.g., escalator meta models or lift meta models. Two-dimensional simulation models will be unaffected because the height properties can simple be ignored. Besides multi-floor concepts, the data contexts package will be improved by multiple more agent contexts and agent managers. For example, we will add agents of the type vehicle, moving obstacle, pedestrian flows, and pedestrian densities. These extensions enable researchers to include further models in MomenTUMv2, such as macroscopic models operating on pedestrian flows.

Table 1: Summary of important keyboard shortcuts.

Shortcut	Functionality
Ctrl + Y	Load layout
Ctrl + S	Load output data of pedestrians
Ctrl + W	Load customized CSV file
Ctrl + D	Load last loaded data file
Ctrl + Q	Quit the visualizer
Ctrl + U	Close the current scenario
Ctrl + F	Search for a specific entity
Ctrl + P	Record video
Ctrl + T	Screen snapshot
Ctrl + C	Visualization Setup
Ctrl + Shift + T	Snapshot Setup
Arrows	Rotate the scenario
Page up/down	Flip scenario orientation
Ctrl + I	Change to isometric 3D-view

The configuration is *XML* based; thus, configuring a simulation by hand can be cumbersome. Hence, the MomenTUMv2 toolkit will be enhanced with some support tools for simulation configurations. This tool will be generic, based on the configuration class definitions. Hence, we will provide a self-organizing tool that creates its own user interface contents to automatically be up to date if new models are introduced.

The number of different models and theories currently implemented in MomenTUMv2 is huge. We have implemented multiple domain related models, which comprise eight operation models, nine tactical models, six strategic models, two meta models, three support models, six output writer models, twelve layout related models, three generator model, two absorber models, and six analysis models. Thereby, all models can be parameterized to provide different behavior. In further publication, we will give an in-depth overview of these model, how to use them in MomenTUMv2, and how to implement new models.

7 Conclusion

Simulations are important for pedestrian dynamics research to prove that mathematically described pedestrian behavior theories can forecast pedestrian behavior. Pedestrian dynamics research can be facilitated by having a framework toolbox in which researchers and practitioners can run pedestrian simulations that comprise multiple individual pedestrians (agent-based). In this report, we provided an in-depth insight of our pedestrian simulator framework MomenTUMv2, which is our simulation framework for conducting pedestrian simulation related research.

MomenTUMv2 is a generic, modular, and extensible framework for agent-based pedestrian simulations. The most important paradigm we applied in designing the framework is that models are understood as implementations of theories. Furthermore, models implement operations that are applied on pedestrians to introduce behavior. This generic view is compatible with almost any agent-based models in pedestrian dynamics. This homogenized approach enabled us to create a simulation framework, which is highly extensible, because new theories are added without much effort, simply by implementing given interfaces. In addition, the system can handle any number of models in any combination. This feature is based on our model execution concept, a configurable pipeline system for model operations.

MomenTUmv2 is still under development; thus, we will improve the framework by providing more agent-types, besides pedestrian agents. Moreover, we will implement useful tools to improve how simulations are configured in MomenTUMv2.

Acknowledgment

We would like to thank our student assistants, who added model implementations to the MomenTUMv2 framework. We especially thank our student assistant Bernd Tornede for implementing the CAD, OSM, BIM, and 3D visualization modules and Quirin Aumann, who extend the graph models concept (Aumann and Kielar, 2016). Furthermore, we thank our students from our Software-Lab curses, which provided the code of their projects as basis for our implementations regarding Unity-3D, AutoCAD, OpenStreetMap, and Revit.

We would like to thank Johannes Schwiebacher, who tested Space Syntax approaches in context of MomenTUMv2, during his Bachelor's thesis (Schwiebacher, 2016), Sven Lauterbach, who is currently working on benchmarking and improving the parallel processing schema of MomenTUMv2 in his Master's thesis, and Daniel Büchele, who created the initial 3D visualization code in his Master's thesis (Büchele, 2014).

We also thank Dr. Angelika Kneidl and Maximilian Bügler, both created the pedestrian simulator MomenTUM(v1), from which we learned Dos and Don'ts.

Finally, we thank all developers, who made their code freely available on the web, without your contributions coding would be a lot harder nowadays.

Funding

This work was supported by the Federal Ministry for Education and Research (Bundesministerium für Bildung und Forschung, BMBF), project MultikOSi, under grant FKZ 13N12823.

References

AlGadhi, S. A. H. and Mahmassani, H. (1991). Simulation of crowd behavior and movement: fundamental relations and application. *Transportation Research Record*, 1320:260–268.

Alievsky, D. (2013). AlgART Java Libraries: arrays and image processing. http://algart.net/java/AlgART/. visited on 2016-07-05.

Alonso-Marroquín, F., Busch, J., Chiew, C., Lozano, C., and Ramírez-Gómez, Á. (2014). Simulation of counterflow pedestrian dynamics using spheropolygons. *Physical Review E*, 90(6):063305.

Andresen, E., Haensel, D., Chraibi, M., and Seyfried, A. (2016). Wayfinding and cognitive maps for pedestrian models. In *11th Conference on Traffic and Granular Flow, 2015*.

Apache Commons (2016a). Commons CSV - Home. https://commons.apache.org/proper/commons-csv/. visited on 2016-07-05.

Apache Commons (2016b). Math - Commons Math: The Apache Commons Mathematics Library. http://commons.apache.org/proper/commons-math/. visited on 2016-07-05.

Aumann, Q. and Kielar, P. M. (2016). A Modular Routing Graph Generation Method for Pedestrian Simulation. In *28. Forum Bauinformatik*.

Autodesk Inc. (2016a). AutoCAD For Mac & Windows — CAD-Software — Autodesk. http://www.autodesk.com/products/autocad/overview. visited on 2016-07-05.

Autodesk Inc. (2016b). Revit-Family — BIM-Software — Autodesk. http://www.autodesk.com/products/revit-family/overview. visited on 2016-07-05.

Bak, P., Chen, K., and Creutz, M. (1989). Self-organized criticality in the game of life. *Nature*, 342(6251):780–782.

Biedermann, D. H., Dietrich, F., Handel, O., Kielar, P. M., and Seitz, M. (2015). Using Raspberry Pi for scientific video observation of pedestrians during a music festival. Technical Report TUM-I1518, Technische Universität München.

Biedermann, D. H., Kielar, P. M., Handel, O., and Borrmann, A. (2014). Towards TransiTUM: A Generic Framework for Multiscale Coupling of Pedestrian Simulation Models based on Transition Zones. *Transportation Research Procedia*, 2:495–500.

Biedermann, D. H., Torchiani, C., Kielar, P. M., Willems, D., Handel, O., Ruzika, S., and Anedr, B. (2016). A hybrid and multiscale approach to model and simulate mobility in the context of public event. In *International Scientific Conference on Mobility and Transport Transforming Urban Mobility*.

Bierlaire, M. and Robin, T. (2009). Pedestrians Choices. In *Pedestrian Behavior. Models, Data Collection and Applications*, pages 1–26. Emerald Group Publishing.

Blue, V. J. and Adler, J. L. (2001). Cellular automata microsimulation for modeling bi-directional pedestrian walkways. *Transportation Research Part B: Methodological*, 35(3):293–312.

Büchele, D. (2014). Visualisierung von Fußgängersimulationsdaten auf Basis einer 3D-Game-Engine. Master's thesis, Technische Universität München.

Campanella, M., Hoogendoorn, S., and Daamen, W. (2014). Quantitative and qualitative validation procedure for general use of pedestrian models. In *6th International Conference on Pedestrian and Evacuation Dynamics*, pages 891–905.

Canca, D., Zarzo, A., Algaba, E., and Barrena, E. (2013). Macroscopic attraction-based simulation of pedestrian mobility: A dynamic individual route-choice approach. *European Journal of Operational Research*, 231(2):428–442.

Carriero, N. and Gelernter, D. (1989). How to write parallel programs: A guide to the perplexed. *ACM Computing Surveys (CSUR)*, 21(3):323–357.

Chew, P. (2007). Voronoi/Delaunay Applet. http://www.cs.cornell.edu/info/people/chew/Delaunay.html. visited on 2016-07-05.

Chooramun, N., Lawrence, P. J., and Galea, E. R. (2012). An agent based evacuation model utilising hybrid space discretisation. *Safety science*, 50(8):1685–1694.

Chu, M. L. and Law, K. (2013). Computational framework incorporating human behaviors for egress simulations. *Journal of Computing in Civil Engineering*, 27(6):699–707.

Collier, N. (2001). Repast: An extensible framework for agent simulation. *Natural Resources and Environmental Issues*, 8:4.

Curtis, S., Best, A., and Manocha, D. (2016). Menge: A Modular Framework for Simulating Crowd Movement. *Collective Dynamics*, 1(0).

Dijkstra, J., Timmermans, H. J. P., and Jessurun, J. (2014). Modeling Planned and Unplanned Store Visits within a Framework for Pedestrian Movement Simulation. *Transportation Research Procedia*, 2:559–566.

Duives, D. C., Daamen, W., and Hoogendoorn, S. P. (2013). State-of-the-art crowd motion simulation models. *Transportation research part C: emerging technologies*, 37:193–209.

Duives, D. C., Daamen, W., and Hoogendoorn, S. P. (2015). Quantification of the level of crowdedness for pedestrian movements. *Physica A: Statistical Mechanics and its Applications*, 427:162–180.

dyn4j.org (2016). dyn4j - Java Collision Detection and Physics Engine. http://www.dyn4j.org/. visited on 2016-07-05.

Ehrecke, L. (2016). Examination of the Professional Support for a Major Public Event. Master's thesis, Technische Universität München.

Fernández, D. (2012). javatuples - Main. http://www.javatuples.org/. visited on 2016-07-05.

Fowler, M. (2004). Inversion of control containers and the dependency injection pattern. http://martinfowler.com/articles/injection.html. visited on 2016-07-05.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Hartmann, D. (2010). Adaptive pedestrian dynamics based on geodesics. *New Journal of Physics*, 12(4):043032.

Hartmann, D. and Von Sivers, I. (2013). Structured first order conservation models for pedestrian dynamics. *Networks & Heterogeneous Media*, 8(4).

Helbing, D., Buzna, L., Johansson, A., and Werner, T. (2005). Self-organized pedestrian crowd dynamics: Experiments, simulations, and design solutions. *Transportation science*, 39(1):1–24.

Helbing, D., Farkas, I., and Vicsek, T. (2000). Simulating dynamical features of escape panic. *Nature*, 407(6803):487–490.

Helbing, D. and Mukerji, P. (2012). Crowd disasters as systemic failures: analysis of the Love Parade disaster. *EPJ Data Science*, 1(1):1–40.

Höcker, M., Berkhahn, V., Kneidl, A., Borrmann, A., and Klein, W. (2010). Graph-based approaches for simulating pedestrian dynamics in building models. *eWork and eBusiness in Architecture, Engineering and Construction*, pages 389–394.

Holser, P. (2016). JOpt Simple - a Java command line parsing library - JOpt Simple. https://pholser.github.io/jopt-simple/index.html. visited on 2016-07-05.

Hoogendoorn, S. P. and Bovy, P. H. L. (2004). Pedestrian route-choice and activity scheduling theory and models. *Transportation Research Part B: Methodological*, 38(2):169–190.

Hoogendoorn, S. P., Bovy, P. H. L., and Daamen, W. (2001). Microscopic pedestrian wayfinding and dynamics modelling. In *1th International Conference on Pedestrian and Evacuation Dynamics*, pages 124–154.

Hoogendoorn, S. P., van Wageningen-Kessels, F., Daamen, W., Duives, D. C., and Sarvi, M. (2015). Continuum theory for pedestrian traffic flow: Local route choice modelling and its implications. *Transportation Research Part C: Emerging Technologies*, 59:183–197.

Ijaz, K., Sohail, S., and Hashish, S. (2015). A Survey of Latest Approaches for Crowd Simulation and Modeling using Hybrid Techniques. In *17th UKSIM-AMSS International Conference on Modelling and Simulation*, pages 111–116.

Kemloh Wagoum, A. U., Chraibi, M., Zhang, J., and Lämmel, G. (2015). JuPedSim: An Open Framework for Simulating and Analyzing the Dynamics of Pedestrians. In *3th Conference of Transportation Research Group of India*.

Kemloh Wagoum, A. U., Seyfried, A., and Holl, S. (2012). Modeling the dynamic route choice of pedestrians to assess the criticality of building evacuation. *Advances in Complex Systems*, 15(07):1250029.

Kielar, P. M., Biedermann, D. H., Kneidl, A., and Borrmann, A. (2016). A Unified Pedestrian Routing Model Combining Multiple Graph-Based Navigation Methods. In *11th Conference on Traffic and Granular Flow, 2015*.

Kielar, P. M. and Borrmann, A. (2016a). Coupling Spatial Task Solving Models to Simulate Complex Pedestrian Behavior Patterns. In *8th International Conference on Pedestrian and Evacuation Dynamics*.

Kielar, P. M. and Borrmann, A. (2016b). Modeling pedestrians interest in locations: A concept to improve simulations of pedestrian destination choice. *Simulation Modelling Practice and Theory*, 61:47–62.

Kielar, P. M., Handel, O., Biedermann, D. H., and Borrmann, A. (2014). Concurrent hierarchical finite state machines for modeling pedestrian behavioral tendencies. *Transportation Research Procedia*, 2:584–593.

Klüpfel, H. (2007). The simulation of crowd dynamics at very large eventsCalibration, empirical data, and validation. In *3th International Conference on Pedestrian and Evacuation Dynamics*, pages 285–296.

Kneidl, A. (2016). How do people queue? A study of different queuing models. In *11th Conference on Traffic and Granular Flow, 2015*.

Kneidl, A., Borrmann, A., and Hartmann, D. (2012). Generation and use of sparse navigation graphs for microscopic pedestrian simulation models. *Advanced Engineering Informatics*, 26(4):669–680.

Lötzsch, M., Risler, M., and Jüngel, M. (2006). XABSL - A Pragmatic Approach to Behavior Engineering. In *IEEE/RSJ International Conference of Intelligent Robots and Systems (IROS)*, pages 5124–5129.

Lozano, M., Morillo, P., Lewis, D., Reiners, D., and Cruz-Neira, C. (2007). A Distributed Framework for Scalable Large-Scale Crowd Simulation. In *2th International Conference on Virtual Reality*, pages 111–121.

Luke, S., Cioffi-Revilla, C., Panait, L., and Sullivan, K. (2004). Mason: A new multi-agent simulation toolkit. In *2004 SwarmFest Workshop*, volume 8, pages 316–327.

Martin, R. C. (2009). *Clean code: a handbook of agile software craftsmanship*. Pearson Education.

Mayer, H., Klein, W., Frey, C., Daum, S., Kielar, P. M., and Borrmann, A. (2014). Pedestrian Simulation based on BIM Data. In *ASHRAE/IBPSA-USA Building Simulation Conference*.

Microsoft (2016). C#. https://msdn.microsoft.com/en-us/library/kx37x362.aspx. visited on 2016-07-05.

Moussaïd, M., Helbing, D., and Theraulaz, G. (2011). How simple rules determine pedestrian behavior and crowd disasters. *Proceedings of the National Academy of Sciences*, 108(17):6884–6888.

OpenStreetMap Foundation (2016). OpenStreetMap. www.openstreetmap.org. visited on 2016-07-05.

Oracle (2016a). Java Software — Oracle. https://www.oracle.com/java/index.html. visited on 2016-07-05.

Oracle (2016b). JavaFX Developer Home. http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html. visited on 2016-07-05.

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058.

Peters, C. and Ennis, C. (2009). Modeling groups of plausible virtual pedestrians. *IEEE Computer Graphics and Applications*, 29(4):54–63.

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH computer graphics*, 21(4):25–34.

Schwiebacher, J. (2016). Berechnung von Fußgünger Ruhezonen mittels Visibility Graph Analysis für Personenstromsimulationen. Master's thesis, Technische Universität München.

Seitz, M. J. (2016). *Simulating pedestrian dynamics : Towards natural locomotion and psychological decision making*. PhD thesis.

Seitz, M. J. and Köster, G. (2012). Natural discretization of pedestrian movement in continuous space. *Physical Review E*, 86(4):046108.

Shoulson, A., Marshak, N., Kapadia, M., and Badler, N. I. (2013). ADAPT: The Agent Development and Prototyping Testbed. In *19th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 9–18.

Singh, S., Kapadia, M., Faloutsos, P., and Reinman, G. (2009). An Open Framework for Developing, Evaluating, and Sharing Steering Algorithms. In *2th International Workshop on Motion in Games*, pages 158–169.

Tianfield, H., Tian, J., and Yao, X. (2003). On the architectures of complex multi-agent systems. In *IEEE/WIC International Conference on Web Intelligence / Intelligent Agent Technology*, pages 195–206.

Torchiani, C., Seitz, M. J., Willems, D., Ruzika, S., and Köster, G. (2015). Fahrgastwechselzeiten von Shuttlebussen. Technical Report TUM-I1517, Technische Universität München.

Torrens, P. M., Nara, A., Li, X., Zhu, H., Griffin, W. A., and Brown, S. B. (2012). An extensible simulation environment and movement metrics for testing walking behavior in agent-based models. *Computers, Environment and Urban Systems*, 36(1):1–17.

Unity Technologies (2016). Unity - Game Engine. https://unity3d.com/. visited on 2016-07-05.

van Toll, W., Jaklin, N., and Geraerts, R. (2015). Towards Believable Crowds: A Generic Multi-Level Framework for Agent Navigation. In *20th Annual Conference of the Advanced School for Computing and Imaging*.

Wooldridge, M. (2009). *An introduction to multiagent systems*. John Wiley & Sons, second edition.

XStream Team (2016). XStream - About XStream. http://x-stream.github.io/. visited on 2016-07-05.

Yoder, J. W. and Johnson, R. (2002). The Adaptive Object-Model Architectural Style. *Software Architecture*, 97:3–27.

Zhang, J. and Seyfried, A. (2014). Comparison of intersecting pedestrian flows based on experiments. *Physica A: Statistical Mechanics and its Applications*, 405:316–325.