AATOM: AN AGENT-BASED AIRPORT TERMINAL OPERATIONS MODEL SIMULATOR

Stef Janssen, Alexei Sharpanskykh
Richard Curran, Koen Langendoen

Delft University of Technology
Mekelweg 5, 2628 CD
Delft, The Netherlands
{s.a.m.janssen,o.a.sharpansksykh,r.curran,k.g.langendoen}@tudelft.nl

ABSTRACT

AATOM, the **A**gent-based **A**irport **T**erminal **O**perations **M**odel simulator is open-source, agent-based at its core, and contains several calibrated presets and templates of basic airport terminal components that can readily be used. Agents in this simulator follow the AATOM architecture, an activity-based architecture for human airport agents. This allows analysis based on agent activities, such as shopping and check-in, which is of vital interest for airports. The combination of agent-based modeling and the presence of basic airport terminal components makes AATOM a unique simulator, allowing the modeler to only focus on implementation of important features of their model. The usefulness of AATOM is demonstrated by presenting case studies in the areas of airport security, gate assignment and resilience.

Keywords: Airport terminal, Agent-based modeling, Security, Resilience, Passenger flow

1 INTRODUCTION

Airports are vital components of the global aviation system and are of large value to national infrastructures and economies of countries. They form a complex system in which different stakeholders are involved, and a large number of interactions between actors are observed. With more passengers than ever, it is important that these airports are operated in the best possible way. Simulation of airport processes to analyze and improve them has shown to be a valuable tool to achieve this goal.

It is more and more recognized that human behavior plays an essential role in modern airport operations. Researchers found that human factors are important in the boarding process (Kierzkowski and Kisiel 2017), airport retail revenue (Wu and Chen 2019), airport security (Kirschenbaum 2015), airport signage (Shimada, Yamane, Ohori, Yamada, and Takahashi 2018) and check-in (Lu, Chou, and Ling 2009).

A large set of simulators can be found in literature, but no simulator exists that allows for an easy implementation of both human behavior and airport-specific elements. Agent-based modeling and simulation is a natural approach towards understanding human behavior. General agent-based simulation platforms, such as Netlogo (Tisue and Wilensky 2004), RePast (North, Collier, Ozik, Tatara, Macal, Bragen, and Sydelko 2013), and Gama (Taillandier, Vo, Amouroux, and Drogoul 2010) enable the implementation of any kind of agent-based model. They generally contain a large number of tools to develop agent behavior, visualize simulation traces, and analyze simulation outcomes. As these platforms are general, they do not contain specific

elements for airport terminals, such as a security checkpoint and default behavior for passengers. Implementing airport-specific elements in these simulators may lead to a complicated coding structure, which is harder to extend than a dedicated airport simulator.

More specific pedestrian-oriented simulators exist as well. Menge (Curtis, Best, and Manocha 2016) and PedSim (Gloor 2018) are examples of these simulators. These simulators contain several algorithms to specify pedestrian or passenger behavior, and were shown to be useful to analyze collective walking behavior in practice. As with the general agent-based simulators, these simulators do not contain default airport-specific elements that can readily be utilized.

Furthermore, many airport terminal simulators exist and are in use by airports. For instance, the Pedestrian Dynamics simulation software (InControl 2018), the Pax2Sim simulator (Hub Performance 2012) and the CAST simulator (Airport Research Center 2018) are examples of commercial simulators. They have proven very useful in capacity planning for airports, and provide valuable insights to airport managers. However, all of these simulators are commercial, and therefore not open source. These simulators are also not agent-based, so human behavior and complex interactions between cognitive agents cannot be modeled well.

The AATOM simulator, an Agent-based Airport Terminal Operations Model simulator was developed to fill this gap. The simulator is open-source and contains several basic airport terminal components that can readily be used. AATOM is agent-based at its core, allowing simple implementation of cognitive human behavior. It also contains the AATOM architecture, an activity-based architecture for cognitive airport agents. This paper introduces the AATOM simulator, and discusses its core features in Section 2. Several case studies were previously performed with this simulator, and are outlined in Section 3. Using these case studies, we show the advantages of using AATOM, and how its features were used in practice. Finally, some conclusions and recommendations are provided in Section 4.

2 AATOM SIMULATOR

The AATOM simulator is implemented in Java and includes explicit representations of space and time, allowing the user to model spatial elements of airport terminals. This for instance enables users to analyze the movement of passengers through the airport terminal. As AATOM is an agent-based simulator, human behavior and complex interactions between cognitive agents can naturally be implemented. A visualization of the simulator for a given airport layout is provided in Figure 1.

The AATOM simulator is published on Github (Janssen 2019), and we introduce it in this section. We first provide an overview of its class structure in Section 2.1. The AATOM agent architecture is discussed in Section 2.2, and AATOMs main features are shown in Section 2.3.

2.1 AATOM Class Structure

Following the model-view-controller design pattern (Gamma 1995), the AATOM simulator is split up into four main packages: agent, environment (model), simulation (controller) and GUI (view). Every package of the AATOM simulator can be extended, but the design allows users to focus on their model only. The environment package contains a basic implementations of airport terminal components, and the agent package has a cognitive agent architecture that can be used to easily model human behavior.

An overview of the main classes in the AATOM simulator is provided in Figure 7, shown at the end of this paper. There are several other classes in the simulator, but for clarity only the main structure is provided in this diagram. It is important to note that some classes and packages are combined for simplicity in this diagram, and that the full set of classes is documented in the Javadoc of the simulator.

The central class in the model part of AATOM is *MapComponent*. *MapComponent* is the superclass of every model component of AATOM. A *MapComponent* has a *Position*, an indication if it is destroyed and ensures access to the *Map*. The *Map* class is a container for all *MapComponent*s, which can both be added and removed from it. The *Map* also keeps track of time. All classes and methods in the subsequent text are printed in italics. The four packages are discussed in more detail in the subsequent Sections.

2.1.1 Agent Package

The agent package, as shown in Figure 7, forms the basis of the agent-based simulator, as it encodes the behavior of agents in the simulator. An *Agent*, in its most basic form, inherits all properties of *MapComponent*.

Following standard agent definitions, an *Agent* can perform observations, actions and maintain an internal state. Observations are made through the *getObservation(type)* method. This method allows the agent to observe *MapComponent*s of a specific type on the *Map*. An agent can also *update()* itself. Through this method, actions can be performed and internal states can be altered. Everything an agent does is encoded in child classes of the abstract class *Agent*. Finally, agents can determine if they want to be destroyed through the *wantsToBeDestroyed()* method.

A single type of agent is defined in AATOM: the *HumanAgent*. This agent has both a mass and color. The mass is used for the movement of the *HumanAgent* (see Section 2.2), while the color is used for visualization. The *HumanAgent* has a single added functionality as compared to the *Agent*: it can receive communications through the *communicate(type,comm)* method. This method is called by other *Agent*s that want to communicate information of a certain type to the *HumanAgent*.

A *HumanAgent* following the AATOM architecture is defined as an *AatomHumanAgent*. This agent type, and its airport-specific children types *Operator* and *Passenger* are discussed in more detail in Section 2.2. We will first discuss the remaining packages and classes of the AATOM simulator to provide the reader with an overview of the different elements of the simulator.

Other types of *Agent*s can easily be introduced by extending the right class. For instance, airport visitors who are not *Passenger*s can be added by extending the *HumanAgent* or *AatomHumanAgent* class.

2.1.2 Environment Package

The environment package contains the elements that form the static components of the modeled airport. Combined with the agent package, this environment package forms the model part of AATOM. Several types of environment objects exist: *Flight*s, *Area*s and *PhysicalObject*s are three main examples of these objects.

The *Flight* class forms a central element of the environment. It is of a specific *FlightType* (either departing or arriving) and has an associated *flightTime*. This time refers to, depending on the *FlightType*, either the departure time or arrival time of the *Flight*. It also contains a collection of *checkedIn Passenger*s, and has a *flightSize*. This *flightSize* specifies the expected number of *Passenger*s on the *Flight*. A *Flight* can determine if a *Passenger* is *alreadyCheckedIn(passenger)*, and can *checkIn(passenger)* a *Passenger*. Furthermore, it can determine if the *timeToFlightExceeded()* already.

Another important environment object is that of *Area*. An *Area* specifies the function of a part of the airport terminal. While not shown in the class diagram, several types of *Area*s are defined. Examples

include *GateArea*, *QueuingArea*, and *EntranceArea*. These *Area* types are used by agents to perform certain activities in.

Several types of *PhysicalObject*s exist. *Luggage*, *Chair*, *Sensor* and *Desk* are the four types that are discussed here. *Luggage* has an *owner*, which is a *Passenger* and is of a specific *type* (carry-on or checked). It automatically has the same position as its owner, unless checked at check-in or when scanned by an *XRaySensor*. A *Chair* has a specific *entryPosition*, which is the position where a *HumanAgent* has to be in before it can sit down. It has an *agentSitting* field that specifies what agent is sitting there. A *Desk* has a specific *servingPosition*, in which agents are served by some *Operator*. Furthermore, a *Desk* can be open or closed, specified by the *isOpen* field.

Finally, a *Sensor* is used to observe specific elements that cannot be observed directly by *Operator*s. A *Sensor* has a *SensorState*, which is either active or idle. Furthermore, the *getObservation()* method is used to perform observations. Two specific *Sensor*s are defined: *WalkThroughMetalDetector* and *XRaySensor*. Both of these *Sensor*s are used at the security checkpoint of the airport terminal.

2.1.3 Simulator Package

The simulator package contains the core elements to perform simulations with the above described model implementation, and forms the controller in the design pattern. The main class in the package is the *Simulator* class, which handles the progression of time and calls the agents' *update(timeStep)* method. Four classes are associated with the *Simulator* class: *Map*, *AgentGenerator*, *EndingConditions* and *Logger*.

The *AgentGenerator* is used to generate *Agent*s in the simulator. It is generally used for arrival of *Passenger*s, but can be used for any kind of *Agent* generation. Some example implementations of *AgentGenerator*s are the *BaseAgentGenerator*, which generates *Agent*s following a Poisson distribution, and the *FlightSpecificAgentGenerator*, which generates *Agent*s based on the defined *Flight*s. The *EndingConditions* class defines when the *Simulator* should stop simulating. This can be based on any condition, but some default implementations exist. For instance the *NoPassengerEndingConditions* class stops the simulation when *Passenger*s are no longer present, while the *BaseEndingConditions* class stops the simulation after a predefined number of seconds. *EndingConditions* also specify the return values for the simulation. Finally, the *Logger* logs different elements of the simulation, such as return values (*ReturnValueLogger* class), agent logs (*AgentLogger* class) and *Analyzer* data (*AnalyticsLogger* class). *Analyzers* (not visualized in the diagram), collect temporal data from the simulation. This data for instance includes queue lengths or mean distance covered by passengers, and can later be analyzed. This can either be performed by using the included Matlab implementation, or any other analytic tool.

2.1.4 GUI Package

The GUI package is used to visualize the simulation in real-time. It contains the main *GUI* class, which visualizes the *MapComponent*s on the *Map*. Each *MapComponent* has an associated *MapComponentView* that paints the *MapComponent* on the *MapPanel*. Custom *MapComponentView*s can be added when the user defines a custom *MapComponent* as well. When no *MapComponentView* for a specific *MapComponent* was found, the *MapComponentView* of the superclass of the *MapComponent* is used. For instance, there is no specific *WallView* class, but the more generic *PhysicalObjectView* ensures that all *Wall*s are visualized.

The *GraphCollectionPanel* visualizes the temporal information that is collected by *Analyzer*s. This information is used to form graphs that are visualized in real-time and can for instance be used to track queue

lengths and the number of passengers that missed their flight. Finally, the *ControlPanel* is used to interact with the running simulation, by changing the simulation speed or pausing it.

2.2 Agent Architecture

The AATOM agent architecture conceptualizes different functional modules in human agents. Three layers are distinguished: the operational layer, the tactical layer and the strategical layer. Each of these layers has a set of modules that execute specific tasks. The operational layer handles observations (perception module) and performs actions in the action module. The action module also handles communication with other agents. The belief module (in the tactical layer) maintains a belief based on historical observations, actions and internal states. The tactical layer is also responsible for activity execution (activity module) and navigation (navigation module) based on a plan. Finally, the strategic layer maintains goals (goal module) and generates a plan (planning module) to achieve these goals. An overview of these modules and their relations with each other is shown in Figure 2. More details about the AATOM architecture are provided in a technical report (Janssen, Blok, and Knol 2018).

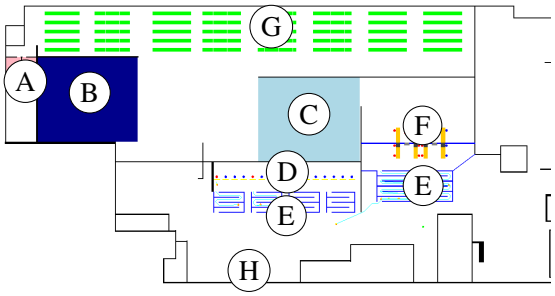

Figure 1: A visualization of the AATOM GUI, showing the outline of an airport, with indicators for different areas. A, B and C are facility areas. D is the check-in area and E are queuing areas. F is the checkpoint area, G is the gate area and H is the entrance area.

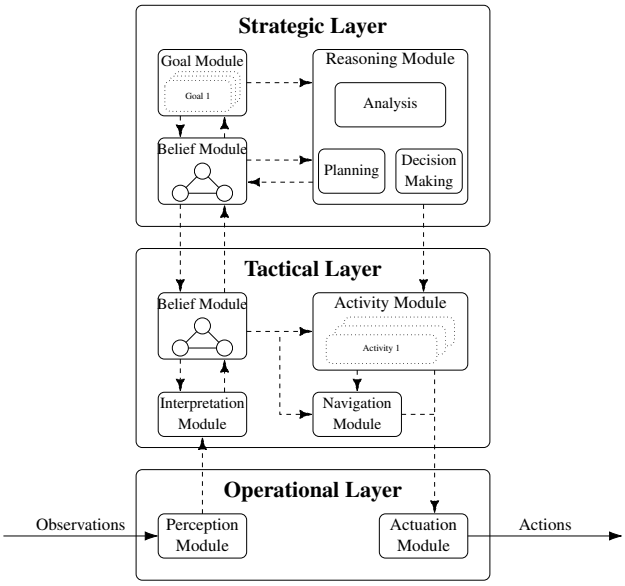

Figure 2: The AATOM architecture and its different modules (Janssen, Blok, and Knol 2018).

The conceptual architecture as provided in Figure 2 is translated to a Java implementation, of which the class diagram is shown in Figure 3.

The *MovementModule* handles the walking of *AatomHumanAgent*s, and some basic implementations of the *MovementModule* are provided. The *HelbingMovementModule* is an implementation of the Social Force Model as proposed by Helbing et al. (2000), while the *BasicMovementModule* ignores the environment and moves the agent in the direction of its goal position. The *CommunicationModule* handles the incoming communication, while the *ObservationModule* handles observation of the environment and other agents. Basic implementations for *Passenger*s and *Operator*s are provided as well. The *CommunicationModule* and the *MovementModule* together form the actuation module as shown in Figure 2.

The *NavigationModule* is used to determine collision-free paths between locations in the airport terminal. Collision-free paths are determined by the *PathFinder* class. Several implementations of the *PathFinder*

class are provided: *JumpPointSearchPathFinder*, which is based on the Jump Point Search algorithm (Harabor and Grastien 2011), and the *AStarPathFinder*, based on the A* algorithm are two examples. The *NavigationModule* additionally detects and handles situations in which the agent is stuck using the *StuckDetector* class. This class observes the previous *Position*s of the *AatomHumanAgent*, and acts if this *Position* was the same for too long.

The *ActivityModule* holds a set of *Activitie*s that the agent can execute. An *Activity* is an executable set of actions and is a central concept in the *AatomHumanAgent*. Example activities include *LuggageCheckActivity*, *CheckpointActivity* and *PassengerCheckInActivity*. The *ActivityModule* is furthermore responsible for a special activity: the *QueueingActivity*. This *Activity* ensures that *AatomHumanAgent*s queue when other *AatomHumanAgent*s are queuing in front of them. The actions of the *Activity* are executed by the modules in the operational layer, as described above.

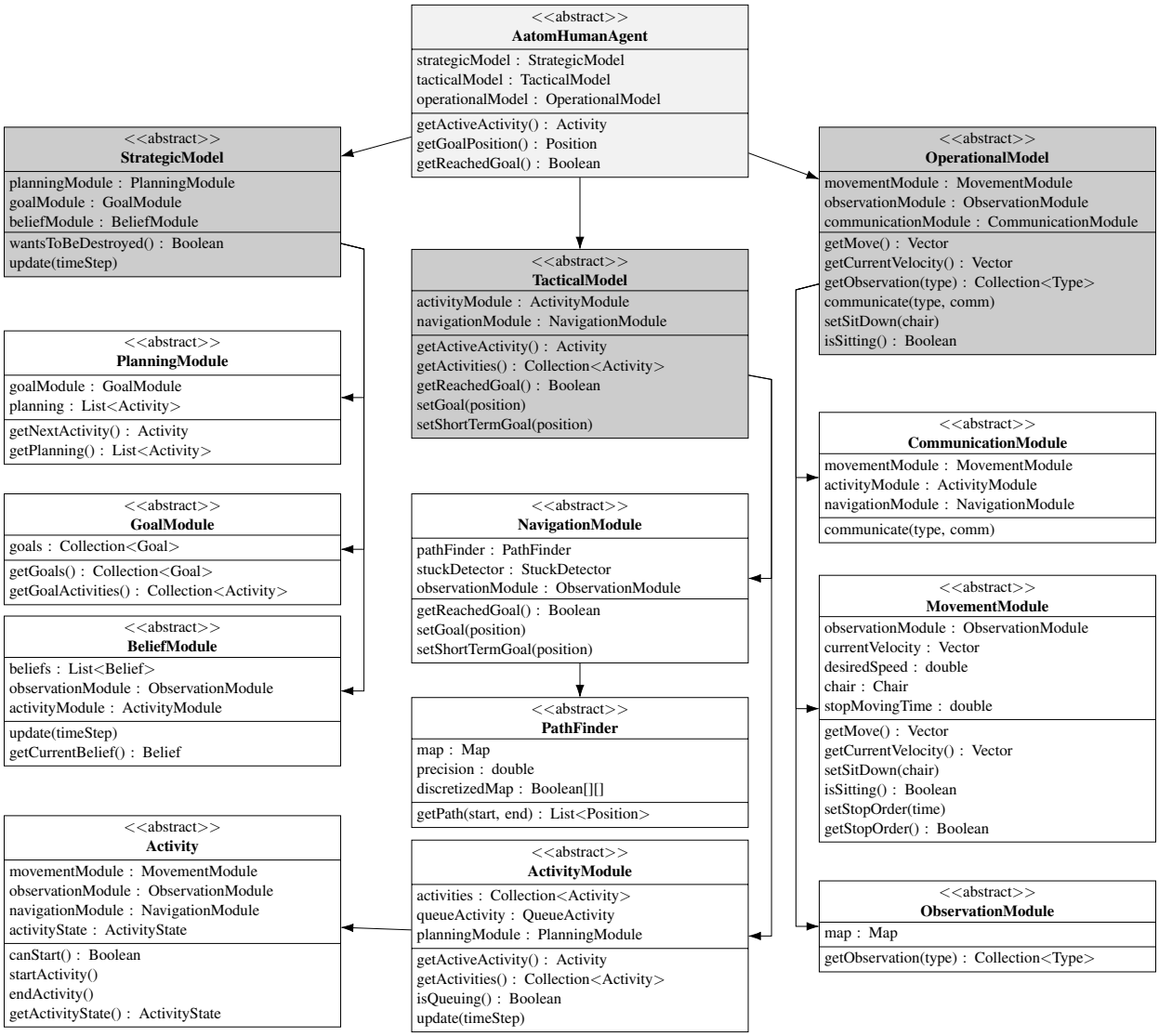

Figure 3: The UML class diagram of the AATOM agent architecture. For brevity, the association labels are omitted in this class diagram, and the intermodule associations are omitted as well. They can be obtained by observing the attributes of each of the modules.

The implementation of the interpretation module and the belief module of the tactical layer fall within the *BeliefModule*. This *BeliefModule* processes and saves information that is relevant for the agent. This for instance includes the current and past locations and the activity that were executed. The *GoalModule* is responsible for maintaining *Goal*s of agents. *Goal*s are always related to *Activitie*s and can be described in two forms. The first form states that an *Activity* should be finished before a certain time, while the second form indicates that the *Activity* should be finished before another *Activity*. Based on the defined *Goal*s, the *PlanningModule* generates plans for the *AatomHumanAgent*. A plan is a sequence of *Activitie*s that are executed by the agent. This plan is subsequently used by the *ActivityModule*.

2.3 AATOM Features

One of the key advantages of AATOM over other simulators (see Section 1) is the availability of airport-specific components. The *ModelComponentBuilder* class is essential in this respect. This class allows the user to add a set of *MapComponent*s to the *Map*. These sets for instance represent a check-in area (using the *checkInArea(...)* method), a checkpoint area (using the *checkpoint(...)* method) and a gate area (using the *gate(...)* method). Using these model components, an airport can readily be built in just a few lines of code.

Many important airport terminal processes were calibrated using manually collected airport data. For instance, the check-in times of passengers was calibrated observing 250 passengers checking in at a regional airport. Different processing times for checkpoint subprocesses, such as luggage drop, were also calibrated using manually collected airport data. These are implemented in the specific *Activity* classes that agents execute. Users can specify their own distributions in these classes as well.

Experiments can be conducted using the *Experimenter* class. The *Experimenter* constructor takes two inputs: *List<String[]>* and *Class<?>*. The first input is used to specify the set of input arguments to be used in the experiment, while the second argument is the *Main* class that handles these inputs. The *Experimenter* class automatically runs n simulation configurations in parallel, where n is the number of available cores for the system. This makes AATOM also suitable for performing experiments in computational clusters.

Output, generated by the *Logger* class, can be analyzed using any data analysis toolbox, as output is generated in plain text files. Basic analysis and plotting functionality is implemented in Matlab and is provided in the AATOM distribution. To facilitate new users of AATOM, a comprehensive tutorial is provided. This tutorial explains the basics of coding in AATOM, and gives detailed explanations about the underlying structure of the code. This tutorial can be used to explore the most important features of AATOM as well.

3 CASE STUDIES

The authors used AATOM to analyze different performance aspects of airport terminals. Security-related work is discussed in Section 3.1. A case study on gate assignment is discussed in Section 3.2 and a case study on resilience is introduced in Section 3.3. For each of the case studies, the usefulness of AATOM is discussed as well.

3.1 Airport Security

The AATOM simulator has been used to estimate security checkpoint performance at a regional airport (Knol, Sharpanskykh, and Janssen 2019). Security checkpoint performance was analyzed in two dimensions: proportion of missed illegal items and queuing time. Security operators were modeled using cognitive agent models, in which decision making and fatigue were represented. Decision making was modeled following the Ratcliff diffusion model (Ratcliff and McKoon 1998) and fatigue was modeled fol-

lowing McCauley's fatigue model (McCauley 2013). Results showed a security checkpoint performance curve with three different regions. The first region indicates low security performance and the second region shows an improvement in both the proportion of missed illegal items and queuing time. Finally, the third region shows a trade-off between the two dimensions.

In another security-related effort we modeled an Improvised Explosive Device (IED) attack in the open areas of the airport terminal (Janssen, Sharpanskykh, and Curran 2019). Behavior Detection Employees (BDEs) were modeled and their effectiveness with respect to preventing an IED attack was analyzed. Results show that airports should attempt to spread passengers across the available space as much as possible to reduce the impact of an IED attack (Figure 4).

The AATOM simulator was especially useful in analyzing security, as an architecture for cognitive agents was already present in the simulator. This allowed for a simple implementation of the Ratcliff diffusion model and McCauley's fatigue model. Furthermore, airport-specific elements, such as the security checkpoint and check-in desks, were implemented and calibrated in the simulator already.

3.2 Gate Assignment

A prominent problem that is well studied in airport literature is that of gate assignment (Deng, Zhao, Yang, Xiong, Sun, and Li 2017). Passengers are often only considered for their static walking distances, but congestion at for instance the security checkpoint is also of major influence on the best gate assignment. Using AATOM, different gate assignments of a local airport were evaluated, while explicitly taking into account the queuing time of passengers (Spans 2018). The layout of the airport, as implemented in AATOM, is shown in Figure 5. Using a differential evolution algorithm the best gate assignment was finally obtained. Furthermore, to improve the speed of the gate assignment, meta-models using regression and a Gaussian radial basis function were used. These models improve the speed of the optimization, but do not always find the same optimal solutions of the differential evolution algorithm.

The AATOM simulator was found to be useful in this project, as it can be used for a holistic simulation of the entire airport. Furthermore, integration with other methods, such as the differential evolution algorithm, was made more simple due to the open-source character of AATOM.

3.3 Resilience

Large disruptions in the aviation sector, such as volcano eruptions or software malfunctions, happen frequently and their impact is generally very costly (Kurtz 2016). To address this problem, airports recently started to improve the resilience of their operations, where resilience is understood as 'the intrinsic ability of a system to adjust its functioning prior to, during, or following changes and disturbances' (Hollnagel 2011). To aid this development, we proposed a formalization of the adaptive capacity of resilience of the air transport system, while specifically focusing on its ability to anticipate (Blok, Sharpanskykh, and Vert 2018). In that work, the effects of anticipation and the corresponding adaptive actions in the context of security operations was analyzed. It was found that proper anticipation of security agents significantly reduce the risk of system saturation, where the saturation was defined as the queue length exceeding a specific threshold. Figure 6 shows the probability that the security checkpoint queue reaches a saturated state for different times and checkpoint configurations.

The AATOM simulator proved useful in this work, as the entire anticipation framework could be implemented in the simulator in fewer than 200 lines of code. In addition, the security checkpoint was already implemented, and flights could be delayed to analyze the effect of an anticipating security agent. Finally,

the performance of the system could easily be analyzed as all relevant variables, such as the number of passengers that miss their flight, were logged automatically.

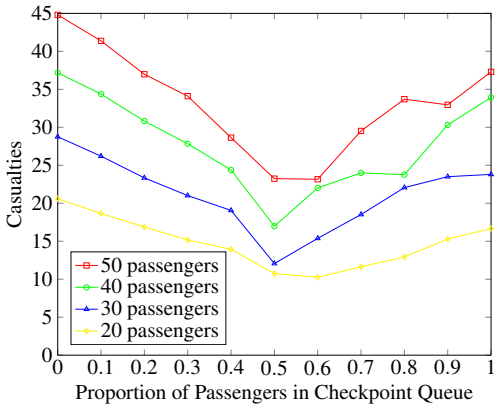

Figure 4: Relationship between the ratio of queue lengths and the number of casualties under different passenger loads (Janssen, Sharpanskykh, and Curran 2019).

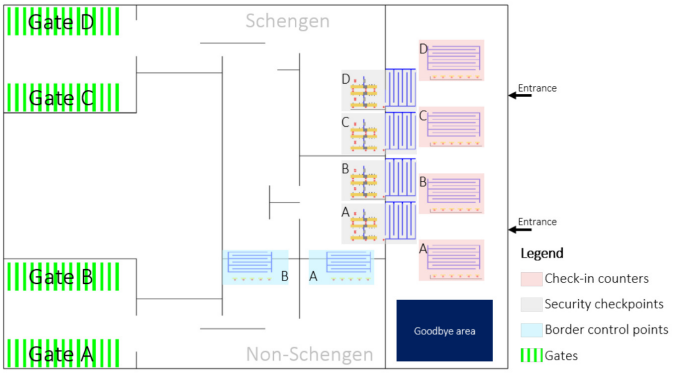

Figure 5: The layout of the airport as used in the work of Spans (2018).

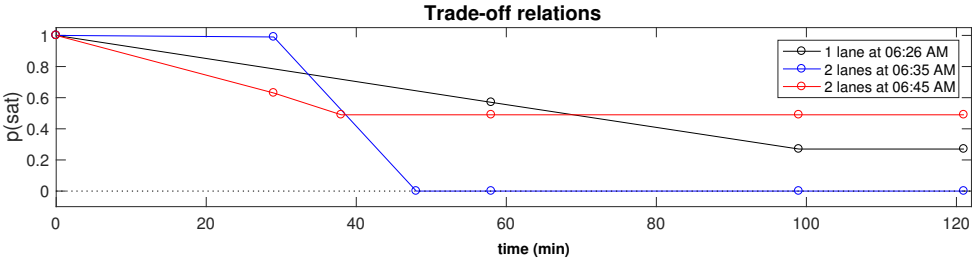

Figure 6: The probability of queue saturation over time at the security checkpoint queue (Blok, Sharpanskykh, and Vert 2018).

4 CONCLUSION & FUTURE WORK

AATOM is an open source Java-based Agent-based Airport Terminal Operations Model simulator. It allows the modeler to focus on the implementation of important features of this model, as calibrated presets and templates for most airport elements are already present. Furthermore, a basic cognitive agent architecture, called the AATOM architecture, was implemented and described. This architecture is activity-based, and contains three layers: operational, tactical and strategical. This allows analysis based on agent activities, such as shopping and check-in, which is of vital interest for airports.

The simulator was used for experimentation in a variety of case studies, ranging from security to efficiency and resilience. The above-described combination of features made AATOM a valuable tool to investigate the properties of airport terminals in these areas.

AATOM can be extended in several directions. First, its stability can be improved by extending its active user base. This leads to a timely identification and resolution of bugs and lacking features. In addition, more different airport-specific features can be implemented. This can for instance be specific sensor types, employee types, disruption scenarios, and complete airport configurations.

ACKNOWLEDGEMENTS

The authors would like to thank Arjan van den Berg, Anne-Nynke Blok, Nardo Gelsomina, Arthur Knol, Régis van der Sommen, and Jeroen Spans for their valuable contributions to the development of AATOM.

REFERENCES

Airport Research Center 2018. "CAST Software. Optimize investments, reduce operational costs". https://arc.de/cast-simulation/. Accessed 2018-23-11.

Blok, A.-N., A. Sharpanskykh, and M. Vert. 2018. "Formal and computational modeling of anticipation mechanisms of resilience in the complex sociotechnical air transport system". *Complex Adaptive Systems Modeling* vol. 6 (1), pp. 7.

Curtis, S., A. Best, and D. Manocha. 2016. "Menge: A modular framework for simulating crowd movement". *Collective Dynamics* vol. 1, pp. 1–40.

Deng, W., H. Zhao, X. Yang, J. Xiong, M. Sun, and B. Li. 2017. "Study on an improved adaptive PSO algorithm for solving multi-objective gate assignment". *Applied Soft Computing* vol. 59, pp. 288–302.

Gamma, E. 1995. *Design patterns: elements of reusable object-oriented software*. Pearson Education India.

Gloor, Christian 2018. "Pedsim: Pedestrian crowd simulation". http://pedsim.silmaril.org. Accessed 2018-23-11.

Harabor, D., and A. Grastien. 2011. "Online graph pruning for pathfinding on grid maps". In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pp. 1114–1119. AAAI Press.

Helbing, D., I. Farkas, and T. Vicsek. 2000. "Simulating dynamical features of escape panic". *Nature* vol. 407 (6803), pp. 487.

Hollnagel, E. 2011. "RAG-The resilience analysis grid". *Resilience engineering in practice: a guidebook. Ashgate Publishing Limited, Farnham, Surrey*, pp. 275–296.

Hub Performance 2012. "PAX2SIM". http://www.hubperformance.com/. Accessed 2018-23-11.

InControl 2018. "Airport Passenger Flows". https://www.incontrolsim.com/application-areas/airports-passenger-flow/. Accessed 2018-23-11.

Janssen, Stef 2019. "AATOM: Initial Release". https://doi.org/10.5281/zenodo.2582733. Accessed 2019-06-17.

Janssen, Stef and Blok, Anne-Nynke and Knol, Arthur 2018. "AATOM - An Agent-based Airport Terminal Operations Model". Techn. rep. http://stefjanssen.com/AATOMarchticeture.pdf. Accessed 2019-06-17.

Janssen, S., A. Sharpanskykh, and R. Curran. 2019. "Agent-based modelling and analysis of security and efficiency in airport terminals". *Transportation Research Part C: Emerging Technologies* vol. 100, pp. 142–160.

Kierzkowski, A., and T. Kisiel. 2017. "The human factor in the passenger boarding process at the airport". *Procedia Engineering* vol. 187, pp. 348–355.

Kirschenbaum, A. A. 2015. "The social foundations of airport security". *Journal of Air Transport Management* vol. 48, pp. 34–41.

Knol, A., A. Sharpanskykh, and S. Janssen. 2019. "Analyzing airport security checkpoint performance using cognitive agent models". *Journal of Air Transport Management* vol. 75, pp. 39–50.

Kurtz, Annalyn 2016. "Delta Malfunction on Land Keeps a Fleet of Planes From the Sky, The New York Times, Accessed: 2019-01-11". https://www.nytimes.com/2016/08/09/business/delta-air-lines-delays-computer-failure.html. Accessed 2019-06-17.

Lu, J.-L., H.-Y. Chou, and P.-C. Ling. 2009. "Investigating passengers' intentions to use technology-based self check-in services". *Transportation Research Part E: Logistics and Transportation Review* vol. 45 (2), pp. 345–356.

McCauley, P. e. a. 2013. "Dynamic Circadian Modulation in a Biomathematical Model for the Effects of Sleep and Sleep Loss on Waking Neurobehavioral Performance". *Sleep* vol. 36 (12), pp. 1987–1997.

North, M. J., N. T. Collier, J. Ozik, E. R. Tatara, C. M. Macal, M. Bragen, and P. Sydelko. 2013. "Complex adaptive systems modeling with Repast Simphony". *Complex adaptive systems modeling* vol. 1 (1), pp. 3.

Ratcliff, R., and G. McKoon. 1998. "Modeling response times for two-choice decisions". *Neural Comput.* vol. 9, pp. 347–356.

Shimada, E., S. Yamane, K. Ohori, H. Yamada, and S. Takahashi. 2018. "Agent Based Simulation for Evaluation of Signage System Considering Expression Form in Airport Passenger Terminals and Other Large Facilities". In *International Conference on Principles and Practice of Multi-Agent Systems*, pp. 621–629. Springer.

Spans, Jeroen 2018. "An integrated approach for efficient and resilient airport management". http://resolver.tudelft.nl/uuid:53253d33-c89d-4be7-a222-7d1fb54f8ffb. Accessed 2019-06-17.

Taillandier, P., D.-A. Vo, E. Amouroux, and A. Drogoul. 2010. "GAMA: a simulation platform that integrates geographical information data, agent-based modeling and multi-scale control". In *International Conference on Principles and Practice of Multi-Agent Systems*, pp. 242–258. Springer.

Tisue, S., and U. Wilensky. 2004. "Netlogo: A simple environment for modeling complexity". In *International conference on complex systems*, Volume 21, pp. 16–21. Boston, MA.

Wu, C.-L., and Y. Chen. 2019. "Effects of passenger characteristics and terminal layout on airport retail revenue: an agent-based simulation approach". *Transportation Planning and Technology* vol. 42 (2), pp. 167–186.

AUTHOR BIOGRAPHIES

STEF JANSSEN works as a PhD candidate at Delft University of Technology in the areas of agent-based modelling and security risk management. He holds an MSc in Operations Research and a BSc in Knowledge Engineering from Maastricht Univeristy. His email address is s.a.m.janssen@tudelft.nl.

ALEXEI SHARPANSKYKH received his PhD degree in the area of artificial intelligence from VU University Amsterdam in 2008. After that, he worked as a postdoctoral researcher in the areas of Complex Adaptive Systems and artificial intelligence at the same university. Since 2013, he has been working as an assistant professor in air transport at Aerospace Engineering faculty of Delft University of Technology. His email address is o.a.sharpanskykh@tudelft.nl.

RICHARD CURRAN is Full Professor at Delft University of Technology and head of the Air Transport and Operations section. Among various editorial positions he is also the Editor in Chief of the Journal of Aerospace Operations and General Chair and founder of the Air Transport and Operations Symposium (ATOS). His email address is r.curran@tudelft.nl.

KOEN LANGENDOEN is a full professor of computer science with the EEMCS faculty of Delft University of Technology. He holds the chair on Embedded and Networked Systems. He has participated as principal and co-principal investigator in numerous national (Dutch) and EU research projects, including D2S2, COMMIT, RELATE, WISEBED, CONET, and RELYonIT. His email address is k.g.langendoen@tudelft.nl.

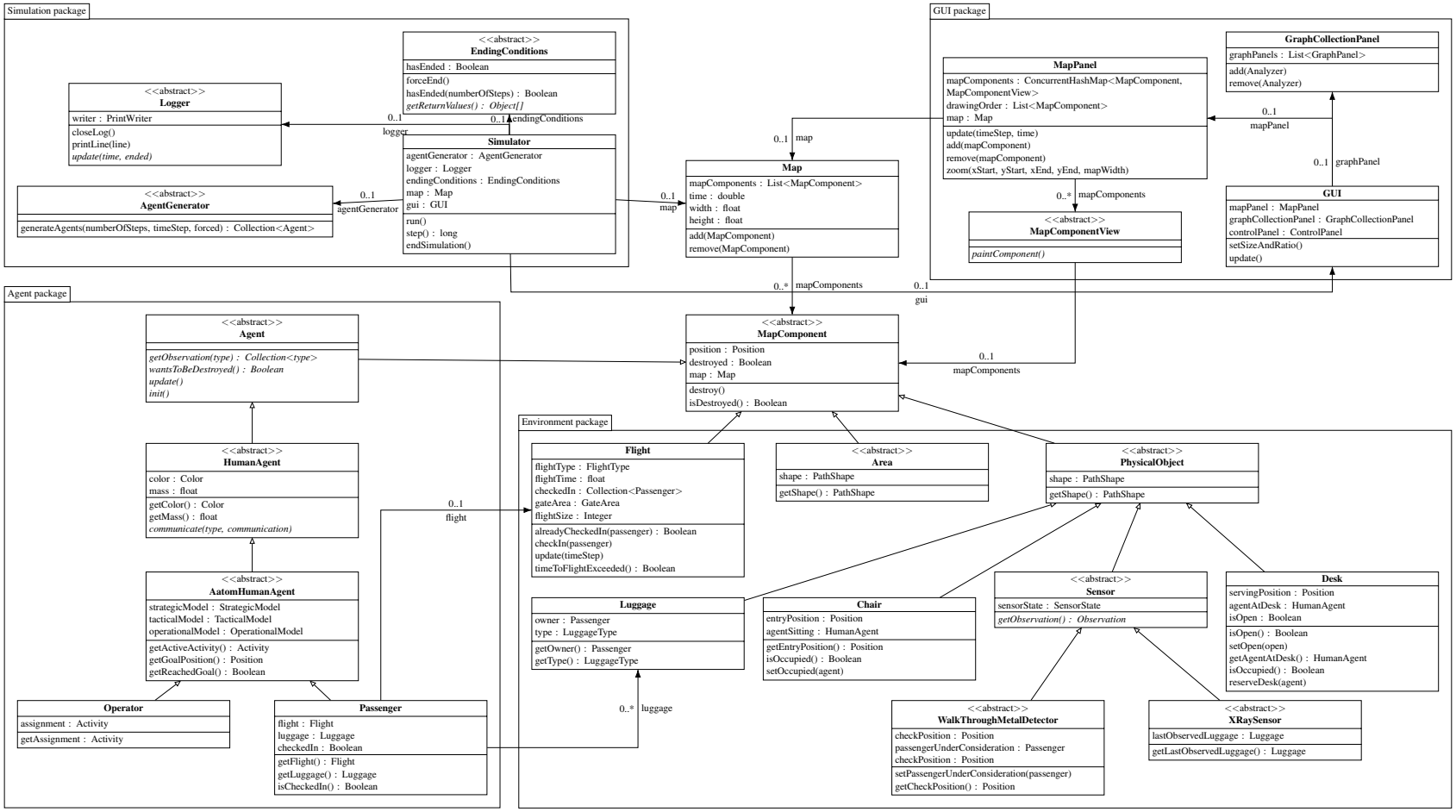

Figure 7: The UML class diagram of AATOM.