24th International Congress on Modelling and Simulation, Sydney, NSW, Australia, 5 to 10 December 2021
mssanz.org.au/modsim2021

Discrete event modeling and simulation of smart parking conflict management

A. Dominici [a] , **L. Capocchi** [a], **E. De Gentili** [a] **and J-F. Santucci** [a]

[a]*SPE UMR CNRS 6134, University of Corsica, Campus Grimaldi, 20250 Corte (France)*

Email: dominici.antoine.p@gmail.com

Abstract: More cities are choosing to use Smart Parking software tools to solve their parking problems. Due to the price inflation of land within the city centers, adding new parking lots cause more problems then they offer solutions. This is why the solution of providing information to users has become essential. It allows with much less financial means, to ensure that the various car parks already in place can be used to their full potential Lin et al. [2017]. The smart parking approach presented in this paper is a complex discrete-event system whose components can be described by finite state automata reacting to internal or external events. The DEVS Zeigler et al. [2018] (Discrete-EVent system Specification) formalism makes it possible to model this system and to simulate it both in real time and in simulated time.

During our previous research Dominici et al. [2020], a system combining discrete-event simulation and artificial intelligence to determine the time at which a place will be released from its user has been developed. To do this we have classified the different classes according to the estimated time before their release. In this paper, based on our previous work, we want to create a system to direct a driver looking for a place according to the release times of the different places available to him. We must also take into account the competition between drivers wanting to park so that they do not interfere with each other and therefore do not increase the time to find a space due to access conflicts.

To prove that such an approach is possible and interesting to achieve with a discrete-event system, we propose to simulate the evolution of class-based sensors previously constructed in our previous research Dominici et al. [2020]. Then we add a system allowing users to simulate finding a place while being in conflict with other users, all this while applying different conflict management policies in order to determine which would be the most suitable for a real situation.

Each ”Space” atomic models is associated with a class before to start the simulation. This parking slot will change state at regular intervals depending on its class which is modeled using the DEVS time advance function. In our current application, a driver has the ability to search for one (or a set of) place(s) (inside an area) in order to maximize the chances of finding one of them available. In order to simulate the users, we have to create an environment in which a driver can move thanks to a model called ”Travel”. We also need to place the sensors on this virtual environment. Once this is done, each of the instantiated users will therefore move according to one or another strategy to one place. However these can come into conflict by coveting the same place. We have created an atomic model ”Access Conflict Management” which has (according to different policies) to manage potential conflicts occurring during the simulation.

We have therefore done a significant amount of simulation of the evolution of parking spaces for a dynamic and random environment. The simulations were performed with different policies. We noticed that the effectiveness of the policy used depended on the needs. A minimum distance policy will then be more interesting when used in the short term, while a policy taking into account the places chosen and avoiding redundancy and inter-blocking is much more interesting in the long term.

Our results allowed us to see that an application of the simulation in the previous case presented advantages at all points. However the applied policies being too simplistic it does not allow to solve all the problem related to the parking in the cities. In the future, we will therefore have to rely on more advanced methods such as reinforcement learning in order to obtain much more effective policy. We will also try to apply these methods to real cases in the smart parking application which will soon be available in the city.

Keywords: *Discrete-event, modeling, simulation, conflict management, Internet Of Things, smart parking*

1 INTRODUCTION

Modeling and Simulation (M&S) is a discipline which aims to reproduce the behavior of a real system from the most possible realistic model. Smart parking is a system for optimizing the occupancy of parking spaces based on specifications that include the behavior of drivers. One of the challenges in this area lie in determining a reliable model capable of resolving cumulative parking conflicts that appear when there are many drivers looking for parking in a dynamic environment system where user behavior is paramount. Thanks to sensor network systems, it is currently possible to provide parking spaces with sensors in order to know their availability in real time Agarwal et al. [2021]. This acquisition also makes it possible to constitute a database which can be used to develop supervised artificial intelligence models Agarwal et al. [2021] (neural networks, decision tree, vector machine support, etc.) used for the classification of parking spaces according to their frequency of occupation.

This paper presents a discrete-event modeling and simulation approach dedicated to propose conflict management strategies based on the estimated travel time to reach desired places around a specific area. The DEVS (Discrete EVent system Specification) Zeigler et al. [2018] is used since (i) it is discrete-event oriented and therefore an effective solution to the management of time advances in your simulation (ii) its hierarchical modular aspect allows you to conclude a conflict model working from the isolated user in specific models.

More and more cities are choosing to use Smart Parking to solve their parking problems. Indeed, because of the inflation of land prices in city centers, adding new parking lots poses a lot of problems and offers cities very few solutions. This is why the solution of providing information to users has become so important because it allows, with much less financial means, to ensure that the various car parks already in place can be used to their full potential Lin et al. [2017]. This choice to inform drivers of the different spaces available has therefore given rise to multiple man-machine interfaces allowing these different spaces to be displayed in a simplified manner Alkhuraiji et al. [2020]. But the Smart Parking solutions come with some problems with the conflicts that drivers coveting the same place could cause resulting in completely saturated parking areas Piovesan et al. [2016]. This problem has also emerged thanks to multiple solutions using all different technologies Melnyk et al. [2019]. Two solutions in general have emerged: prediction Lin et al. [2019] and reservation Boudali and Ouada [2017]. In our previous work, we developed a prediction system based on machine learning. Nevertheless these systems have a very results-oriented view and decision-making cannot be taken in a holistic way with policies to be dynamic. To do this, we are going, on the basis of our previous work, to create a system to manage conflicts between users in order to shorten the time spent searching for places, but also to ensure that everyone has a chance to find one.

The smart parking is a complex discrete-event system whose components can be described by finite state automata reacting to internal or external events. The DEVS formalism makes it possible to model this system and to simulate it both in real time and in simulated time. For example, it is possible to model an occupancy sensor by an atomic model composed of two states: occupied, unoccupied, the lifetime of which depends on membership of the classes.During our previous research Dominici et al. [2020] a system combining discrete-event simulation and artificial intelligence to determine the time at which a place will be released from its user has been developed. To do this we have classified the different classes according to the estimated time before their release. In this paper, we want to create a system to help a driver looking for a place according to the release times of the different places available to him. We must also take into account the competition between drivers wanting to park so that they do not interfere with each other and therefore do not increase the time to find a space due to conflicts.

To prove that such a approach would be possible and interesting to achieve with a discrete-event system, we will simulate the evolution of class-based sensors previously constructed Dominici et al. [2020]. Then we will add a system allowing users to simulate finding a place while being in conflict with other users, all this while applying different conflict management policies in order to determine which would be the most suitable for a real situation. We will also make sure that the model is closest to reality so that the results are the most optimistic.

2 MODELING AND SIMULATION OF SMART PARKING CONFLICTS

Figure 1 shows the approach used to classify parking space sensors implemented in Dominici et al. [2020]. A “Space” atomic model can be constructed with states whose lifetime depends on the classes. A new DEVS atomic model has been created in order to simulate a sensor placed on the ground and detects the presence of a car. Depending on the previously defined class, this will have to change its state between free (’NOTBUSY’)

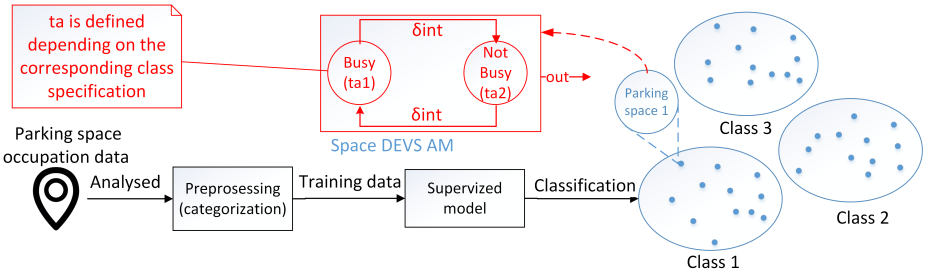

Figure 1. Classification sensor system with the new atomic model ”space”.

and occupied (’BUSY’).

Each Space atomic models are associated with a class before to start the simulation. This place will change state at regular intervals depending on its class which corresponds to a time advance as shown in figure 2. ta1 (resp. ta2) is the time spent in the state BUSY (resp. NOTBUSY). In our current application, a driver has the ability to search for one or a set of places (inside an area) in order to maximize the chances of finding one of them available. Since the spaces can be grouped by area, corresponding to streets or car parks, we must create a coupled model allowing these spaces to be associated.

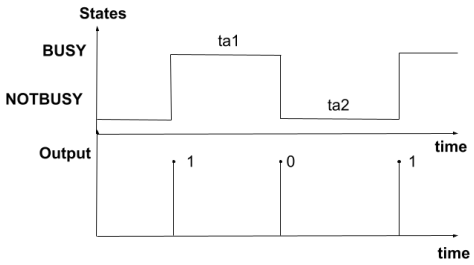

Figure 2. State activity of the Space model.

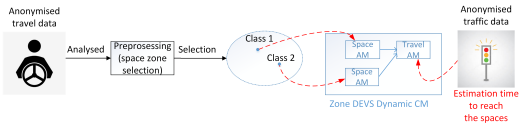

Figure 3. The ”Zone” DEVS coupled model with two Space DEVS atomic models connected to the Travel DEVS atomic model in charge of the esti-mation of the time to reach selected spaces.

Figure 3 shows the coupled dynamic “Zone” model, which includes as many Space atomic models as there are sensors of the parking area desired by the driver. The atomic Travel model makes it possible to estimate the time to reach the desired place from the position of the motorist. It is connected to all Space models in order to have their position as well as their state (’BUSY’, ’NOTBUSY’).

2.1 The Space Map

In order to be able to place the sensors in space, we first need to define it as a spatial environment. Since mapping was not the core of the proposed work, we opted for the abstract creation of an environment with no limits. The coordinates on this virtual map are latitude and longitude. So we can assign each sensor a defined position in space. The user will therefore be able to move around freely on this map and without restriction in this environment. The non-presence of a road on this map does not present any concern here from a realist point of view because even if constraints do exist in reality, the only value of interest to us here is the estimated time necessary for a user to reach a place. The propose space map is used as a way to calculate this travel time. In reality, this map will be replaced by the location where the user is and the estimated time to reach a place will be calculated from a formula, this time taking into account the physical constraints of the real world.

2.2 Driver and Travel Models

Now we need to simulate the behavior of a driver. The one asked to be able to park on a set of defined places. It also has a position on its environment. This driver therefore receives at a time t all the information relating to the surrounding places and must therefore transmit to the model whose role is to direct it all the information necessary for its proper movement within its environment.

The main function of the Travel model is to direct users to a free parking space as quickly as possible. In the case of reality, the user evolves in his environment of his own will by following the dictated advice but here in a simulated environment the movement of this one will be simulated. The Travel atomic model has two main functions:

- It must retrieve information from the user regarding the desired parking slots, such as their states and positions. With this information and thanks to a shortest path algorithm, the user will therefore choose as the target the free space closest to himself. Depending on the chosen place he will have to update his time advance according to the time needed to join the new space. In a real case, a user is using our app while driving to find a nearby parking space. However, many other users are in the same situation as him. Offering him a seat when another user is more likely to have it because his estimated travel time is much shorter than his would be a waste of time. It is therefore necessary to prevent these conflicts which can lengthen the search time for a place thanks to the conflict manager who, thanks to one policy or another, can allocate places to one user or another.
- Once the spot is chosen, it must, thanks to internal transitions, move the user through the simulated environment to the place previously chosen. This functionality makes it possible both to advance the user but also to advance the simulation time because moving the user forward amounts to reducing the time to reach, and therefore the time advance before the next internal transition.

Figure 4, shows the finite state machine of the Travel model. The model begins with the IDLE state before receiving the first information from a driver. Once the information has been collected, it will check the status of all the places of interest to the driver . In the event that one or more places are available, it will switch to the $DriveToGoal$ state which is aimed at a place with a $sigma$ corresponding the time necessary to reach the place (discrete time). At each simulation step, the model will simulate an approach of the drivers towards the directed place thus reducing the time advance and thus the $sigma$. If the $sigma$ reaches 0 while the drivers are in this state, it means that the driver has reached the place and managed to park. In the event that no space is available, the model switches to the $DriveToSearch$ state and proceeds to the nearest space while it finds a free space.

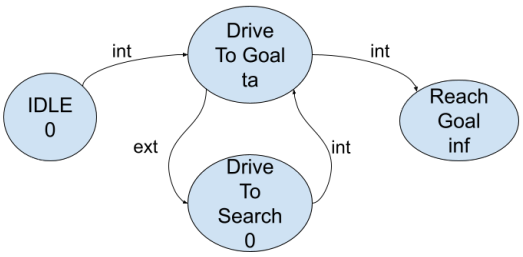

Figure 4. State automaton of the Travel DEVS atomic model.

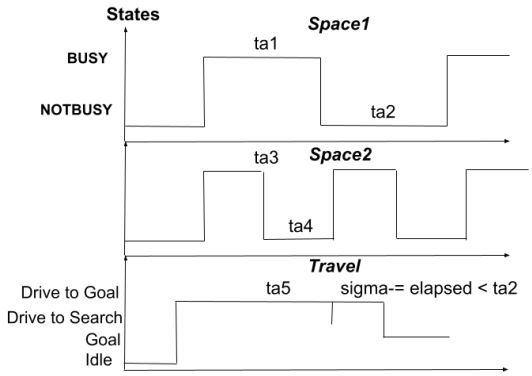

Figure 5. State activity of the two Space models and the Travel model.

The $sigma$ in this state is infinite because riders will search for a place as long as it is not parked. The model's activity will therefore continue until the user finds a place.

Figure 5 shows the example of a driver with a selected area comprising two sensors. He chooses Space1 while in the IDLE state. It enters the Drive_to_goal state with a lifetime which is the estimated time to reach the place. When it receives an event from Space1, it goes to Drive_to_search and it updates sigma according to the lifespan of the new state of Space1, if $ta5 - elapsed = sigma < ta$ and the new state of Space1 is 'NOTBUSY', the place is still suitable. ta is the time advance function that estimates the time of futuristic events that are going to happen on the basis of a list of events. In our case the ta here can correspond to the

time travel of a user going on a parking slot. $elapsed$ is the elapsed time where $elapsed$ is the actual time taken from the start to a specific point in the simulation.

Otherwise Travel returns to the $DriveTogoal$ state considering the second sensor buffered from δ_{ext} transition function to recalculate a new time advance (ta5) and so on. δ_{ext} is an external transition function which changes the state depending on the received external event. It reacts according to an external event and can, like an internal transition function, change the current state of the model. The advancement of time is here induced by the movement of users in the virtual environment.

2.3 Conflict Manager Model

Space access conflicts arise when at least two users covet at least one and the same space (Figure 6). A new model of conflict management is introduced to signal that the place is coveted by several users and to propose resolution strategies such as assigning the place to the one who is closest to the estimated time. However, the conflict management model must define strategies and simulate its strategies to validate them and propose them to drivers.

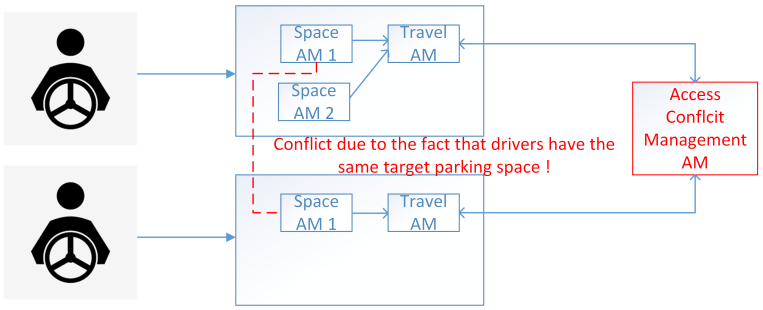

Figure 6. Access Conflict Management resolving conflict between tow drivers that select the Space1.

The model retrieves the decisions made by each of the different travel models corresponding to each user and will verify the presence or absence of a conflict. This model has three different states. The model exits in the IDLE state the first time one of the travel models is received and therefore enters the SAFE state without any conflict. Each time a new message is received, it checks that no conflict is in progress. If a conflict arises it goes into the CONFLICT state or according to different strategies in place it will react and make the conflict disappear. We have 3 main strategies of conflict management:

- The $FirstToAsk$: The first to request a place will have priority over this one and any other Travel model targeting the place finds itself in conflict.
- The $MinimumTA$: The travel which has the shortest distance to cover and therefore the shortest TA will be allocated the place. The others will therefore be in conflict
- The $SpaceChoosedComparison$: In the case of two users searching for à parking slot where one of the users called “A” wants one single place and the other calling “B” wants multiple ones, the algorithm will give the place to the user “A” if they have the parking slot wanted by both of them. Thanks to this we can ensure that each user always has a chance to find a parking space among those they like.

2.4 DEVSimPy Simulation Model

Figure 7 shows the DEVSimPy Simulation model of the proposed approach. DEVSimPy Capocchi [n.d.] is an advanced wxPython General User Interface for the M&S of systems based on the DEVS formalism. With DEVSimPy, a system can be modeled by interconnecting atomic and coupled DEVS models instantiated from libraries. First, we have the Space atomic models which can also be grouped into a Zone coupled model. It changes state based on a previously performed classification. The UserLink atomic model allows the data from the sensors to be pooled and transmitted to the various users (drivers). The user model transmits the places that suit it as well as personal data to the Travel model. The travel model retrieves data from the

user, chosen a place among that desired according to different policies. It then passes its decision to the AccessConflictManagement model which reacts to different conflicts between users by using different kinds of algorithms.

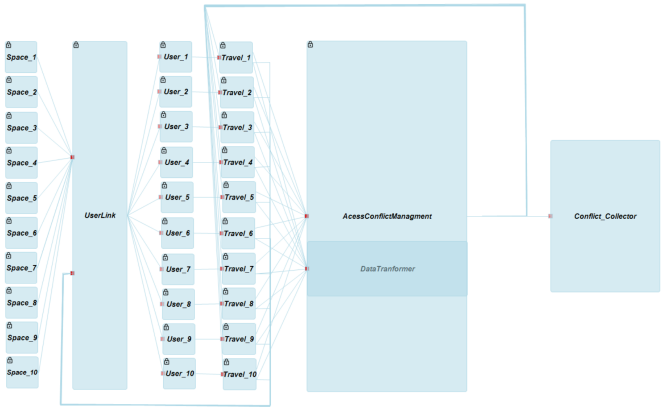

Figure 7. The DEVSimPy simulation model that includes all the different DEVS atomic models and their interconnection. 10 drivers and spaces have been considered. The Conflict Collector model has been used to collect the simulation outputs used for the results analysis.

3 Simulation Results Analysis

In order to generate data to verify the interest of such a system in a real case the following simulation setting are considered: (i) The position of the sensors and their base states have been randomly generated on a 100x100 grid. 10 sensors will be placed in the diagram (ii) The position of the drivers as well as the desired places has been chosen randomly. Their positions can vary on a 100x100 grid. 10 users will be set up followed by the unique travel model for each user (iii) A data storage model will be put in place to record the various data.

The two simulation times (ST) 100 and 200 have been considered where ST is the total time for which the simulation performs data generation cycles thanks in our case to the "space" model. These two simulation times were set up according to the arrangement of the sensors and the drivers who will therefore be placed randomly on a grid. The maximum time for a user to join a place is 200 if both are placed on either side of this abstract map. For this we have decided for $ST = 200$. The $ST = 100$ aims to study the results of the different methods in a short time. The simulation will therefore be performed 100 times by different ST and methods.

Figure 8 depicts that the Space Choosed Comparison strategy is more effective over a longer simulation time because it takes into account the choice of users and prevents some users from being left without space at the end of the simulation. However, taking only very slightly into account the distances between the seats of the

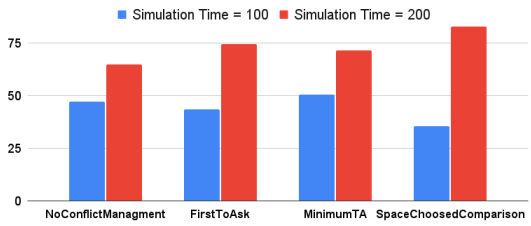

Figure 8. Percent of places taken with different conflict strategy.

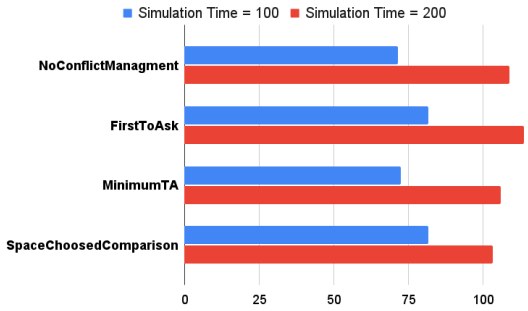

Figure 9. Average of time spending by the drivers to find a space depending on the conflict strategy.

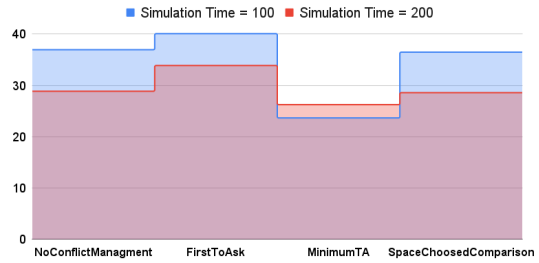

Figure 10. Average number of movement of drivers depending on the conflict strategy.

users are efficiency on a short ST is r educed. We can also notice that because the model travel has a shorter path algorithm, the differences between the $FirstToAsk$ and the $MinimunTA$ are very low. However, as you can see in figure 9 and 10, the $MinimumTA$'s strategy of organizing parking as a whole reduces both the time to find a place but also the number of movements to reach a p lace. The $MinimumTA$ strategy is also very effective on reduced times because it allows users globally to find the place closest to them while avoiding conflicts. The model without conflict management is quite efficient on a short ST as well, which proves that conflicts redirecting users to another place can therefore waste time, which is not negligible on a short ST. However on longer ST we can notice that the use of a conflict manager is essential.

4 Conclusion and Future Work

In this paper a simulation-based conflict management model for smart parking has been p roposed. A DEVS simulation model composed with Space, Drivers, Travel and Conflict Manager models has been implemented in the DEVSimPy framework to test strategies providing solutions to the drivers that want park their cars. However, our system does not allow the dynamic generation of models which reduces our fields of application for real cases. We are therefore going to work on a dynamic DEVS model that will allow us to use our Conflict Manager with our Smart Parking application. We will also test these methods in real cases by slightly modifying the work already done. We have acquired parking space occupancy data for the city of Bastia (France). We plan to replace the Space model by these real occupancy data associated with additional traffic information to test the simulation model in real time. In addition, we would like to check whether a reinforcement learning Kaelbling et al. [1996] algorithm can be combined with DEVS to improve the capabilities of the proposed approach Capocchi et al. [2018].

Acknowledgments

The research leading to these results has received funding from the "Coeur de Ville" Program `https://www.cohesion-territoires.gouv.fr/programme-action-coeur-de-ville`.

References

Agarwal, Y., Ratnani, P., Shah, U. and Jain, P. [2021], Iot based smart parking system, *in* '2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)', pp. 464–470.

Alkhuraiji, S. et al. [2020], 'Design and implementation of an android smart parking mobile application', *TEM Journal* **9**(4), 1357–1363.

Boudali, I. and Ouada, M. B. [2017], 'Smart parking reservation system based on distributed multicriteria approach', *Applied Artificial Intelligence* **31**(5-6), 518–537.

Capocchi, L. [n.d.], 'DEVSimPy', `https://github.com/capocchi/DEVSimPy`. Online; accessed March 10 2021.

Capocchi, L., Santucci, J. F. and Zeigler, B. P. [2018], Discrete event modeling and simulation aspects to improve machine learning systems, *in* '4th International Conference on Universal Village, UV 2018, Boston, MA, USA, October 21-24, 2018', IEEE, pp. 1–6. **URL:** *https://doi.org/10.1109/UV.2018.8642161*

Dominici, A., Capocchi, L., De Gentili, E. and Santucci, J.-F. [2020], Towards a combination of discrete-event simulation with machine learning for smart-parking, *in* 'Proceedings of the 2020 Summer Simulation Conference', pp. 1–12.

Kaelbling, L. P., Littman, M. L. and Moore, A. W. [1996], 'Reinforcement learning: A survey', *Journal of artificial intelligence research* **4**, 237–285.

Lin, J., Chen, S.-Y., Chang, C.-Y. and Chen, G. [2019], 'Spa: Smart parking algorithm based on driver behavior and parking traffic predictions', *IEEE Access* **7**, 34275–34288.

Lin, T., Rivano, H. and Le Mouël, F. [2017], 'A survey of smart parking solutions', *IEEE Transactions on Intelligent Transportation Systems* **18**(12), 3229–3253.

Melnyk, P., Djahel, S. and Nait-Abdesselam, F. [2019], Towards a smart parking management system for smart cities, *in* '2019 IEEE International Smart Cities Conference (ISC2)', pp. 542–546.

Piovesan, N., Turi, L., Toigo, E., Martinez, B. and Rossi, M. [2016], 'Data analytics for smart parking applications', *Sensors* **16**(10), 1575.

Zeigler, B. P., Muzy, A. and Kofman, E. [2018], *Theory of modeling and simulation: discrete event & iterative system computational foundations*, Academic press.