Simulation Testbed for Autonomic Demand-responsive Mobility Systems

Michal Čertický, Michal Jakob, Radek Píbil

Agent Technology Center, Faculty of Electrical Engineering,
Czech Technical University, Prague, Czech Republic
{certicky, jakob, pibil}@agents.fel.cvut.cz

Abstract. In this chapter, we describe an open-source simulation testbed for emerging autonomic mobility systems, in which transport vehicles and other resources are automatically managed to serve a dynamically changing transport demand. The testbed is designed for testing and evaluation of various planning, coordination and resource allocation mechanisms for control and management of autonomic mobility systems. It supports all stages of the experimentation process, from the implementation of tested control mechanisms and the definition of experiment scenarios, through simulation execution, up to the analysis and interpretation of the results. The testbed aims to accelerate the development of control mechanisms for autonomic mobility systems and to facilitate their mutual comparison using well-defined benchmark scenarios. We also demonstrate how it can be used to select the most suitable control mechanism for a specific use case and to approximate operational costs and initial investments needed to deploy a specific autonomic mobility system.

1 Introduction

The increasing deployment of ubiquitous location-aware and internet-connected devices in transport systems enables the realization of autonomic mobility systems, based on continuous, automated management of transport vehicles to serve a dynamically changing passenger transport demand. Several types of such systems appeared in recent years, including real-time *on-demand transport*, real-time *ridesharing* or *dynamically priced taxis* [11]. More highly innovative services are likely to emerge with the wider adoption of electromobility and, most importantly, with the advent of autonomous, self-driving cars.

Due to their inherent complexity, such novel mobility systems need to adopt the autonomic systems principles in order to work reliably, safely and efficiently. They need to be *automatic*, *adaptive* and *aware* [22,31] in order to deal with the situation at hand quickly and efficiently and to maintain long term efficiency under ever-changing conditions. Implementing these principles requires the development of planning, coordination and resource allocation algorithms that would orchestrate the operation of all entities of the transport system in the desirable way. The development of such control mechanisms is a challenging

task due to many internal and external interdependencies that affect the overall system behaviour.

To deal with the complexity of mobility systems, various modelling paradigms have been employed. *Analytical modelling*, *simulation modelling* and *agent-based simulations* in particular [34], as well-established approaches for analysing the behaviour of complex socio-technical systems, have already been applied in this field (see Section 2). The agent-based simulation testbed described in this chapter was designed to meet the research community's demand for an experimentation tool tailored specifically for mobility systems, but at the same time universal enough to analyze and compare their numerous variants.

The presented software is called *Flexible Mobility Services Testbed*[1] and was built on top of a transport simulation framework *AgentPolis*[2]. Even though the testbed is not autonomic itself, it allows and encourages experimenting with a wide variety of mechanisms and algorithms necessary for autonomic mobility systems and lets users assess their performance in different scenarios and configurations. Unlike other simulation platforms, the testbed is fully agent-based, allowing every agent to perceive the environment, adapt and react to it at any point in time. This gives users the freedom to implement a control logic for individual agents in a way that would lead to self-configuration, self-healing, self-optimization or self-protection properties of the whole system. The testbed supports all stages of the experimentation process, from the implementation of tested control algorithms and the definition of experiment scenarios, through simulation execution and visualization up to the analysis and interpretation of simulation results.

To facilitate its use, the testbed provides an easy-to-use application programming interface (API) and allows users to easily plug in their custom control mechanisms by implementing a few predefined abstract Java classes. Furthermore, by providing a standardized way to import and capture key aspects of autonomic transport systems deployment scenarios, including road network maps, vehicle properties or transport demand, the testbed also makes it easy to create benchmark scenarios for mutual comparison of different control mechanisms.

This chapter is divided into six sections: After the overview of related work leading up to brief introduction to agent-based simulation in Section 2, we describe the testbed's architecture and underlying modelling ontology in Section 3. Sections 4 and 5 explain the experimentation process and provide a demonstrative example. Last section merely recapitulates the key ideas and concludes the chapter.

2 Related Work

Since 1970s, we have seen numerous attempts to study mobility and transport systems in general by analytical modelling. An extensive overview of analytical modelling methodology, along with mathematical background can be found in

[1] `http://github.com/agents4its/mobilitytestbed/`
[2] `http://agentpolis.org`

a monograph by Ortuzar and Willumsen [7]. Early models of mobility systems were largely based on mathematical programming and continuous approximations. The former technique relied on detailed data and numerical methods, whereas the latter relied on concise summaries of data and analytic models. Geoffrion [14] advocated the use of simplified analytic models to gain insights into numerical mathematical programming models. In a similar spirit, Hall [16] illustrates applications of discrete and continuous approximations, and notes that continuous approximations are useful to develop models that are easy for humans to interpret and comprehend. Overview and classification of continuous approximation models can be found in [24].

In some of the more recent work, the attention was focused on demand-responsive mobility systems, which were formalized as mathematical abstractions, such as dial-a-ride problem (DARP) [33] or multiple depot vehicle scheduling problem (MDVS) [5] to allow further formal analysis. For example, Heuptmeier et al. [17] and Lipmann et al. [26] studied formal properties of certain algorithms solving DARP and its variant with restricted information. Haghani and Banihashemi [15] addressed the influence of town size on the performance of algorithms for MDVS and its variant with route time constraints.

However, analytic models and theoretical algorithm analyses were often too abstract for expressing relevant aspects in the structure and dynamics of some transport systems. To deal with this shortcoming, the paradigm of simulation modelling was adopted by the transport research community and has been employed in parallel with the analytical approaches. In 1969, Wilson [36] conducted a pioneer simulation-based study of the influence of the service area, demand density and number of vehicles on the behaviour of transport system. Simulations have since then been extensively utilized in the research of transport and mobility, as a powerful tool for the analysis of system's behaviour. To mention a few more examples, Regan et al. [30] studied the performance of freight mobility system using different lead acceptance and assignment strategies, Fu [13] developed a simulation model of an urban paratransit system and Deflorio et al. [8] evaluated demand-responsive transport system under the influence of real-life aspects, such as customer delays and travel time variability.

Simulation modelling also allowed researchers to study DARP or MDVS control mechanisms empirically (in addition to formal algorithmic analysis). Bailey and Clark [3] investigated the performance of one of them in relation with varying vehicle fleet size. Jlassi et al. [20] simulated an ambulance service implemented as dial-a-ride system and Shinoda et al. [32] compared such systems to fixed-route systems under varying circumstances. Diana [9] assessed the effectiveness of scheduling algorithms under different percentages of real time requests and intervals between call-in time and requested pick-up time and Quadrifoglio and Dessouky [28] studied the insertion heuristic scheduling algorithm for Mobility Allowance Shuttle Transit systems, a hybrid transit solution that merges the flexibility of dial-a-ride systems and the low cost operability of fixed-route bus services. More recently, d'Orey et al. [10] used simulations to explore the trade-offs between the satisfaction of drivers and passengers

In these simulations, the system's behaviour could only be centralized – governed in a top-down manner by a single entity or mechanism. Also, any self-initiated interactions, communication or negotiation among individual actors (e.g. passengers) was impossible, severely limiting their level of autonomy. To overcome these limitations, a new paradigm called *Agent-based simulation* was introduced. Agent-based simulation has proven to be a highly valuable tool, especially when studying complex self-organizing systems in many domains [23]. Mobility systems modelled under this paradigm are implemented as multi-agent systems – i.e. composed of autonomous entities termed *agents* situated in a shared environment which they perceive and act upon, in order to achieve their own goals. In the context of mobility, we usually distinguish between three relevant types of agents: *passengers* (announcing transport requests), *drivers* (serving passenger's requests) and *dispatchers* (optional kind of agents who can negotiate with passengers and coordinate the drivers).

Agent-based simulations have been used to study various aspects of mobility, as well as a number of different control mechanisms. Horn [18] employed a simulation, developed completely from scratch, to study operational characteristics of a multimodal transport system integrating conventional timetabled services (buses, trains, etc.) and flexible demand-responsive mobility services (single- and multiple-hire taxis). A combination of traditional and demand-responsive transit was also simulated by Edwards [12]. The impact of zoning vs. non-zoning strategies on demand-responsive mobility were studied by Quadrifoglio and Dessouky in [29]. Real-time taxi sharing schemes and ridesharing have been evaluated by Kamar and Horvitz [21], Lioris et al. [25], or Agatz et al. [1], while the efficiency of traditional taxi services have been studied by Cheng and Nguyen [6].

Control mechanisms that govern the behaviour of mobility systems are usually classified by the concentration of decision making into:

- *centralized* (all the agents controlled by a central entity, e.g. dispatcher)
- *distributed* (agents act based on their mutual, unorganized interactions)
- *hybrid* (combination of those two)

Alternatively, control mechanisms may also be divided based on the structure of transport demand they are dealing with into *static* (all transport requests are known in advance) or *dynamic* (future requests are unknown).

Since these control mechanisms represent a cornerstone of mobility system's behaviour and success, significant amount of research has been invested in them and more is still needed. It therefore makes sense to develop software tools that would assist in this research. The general idea of employing simulation testbeds to accelerate the development of multi-agent control mechanisms was put forward for example by [27]. A common attribute of simulations used in the works above is that they were developed from scratch using general-purpose programming languages (most often C++ or Java), in order to demonstrate only a single specific mechanism. This is because none of the existing general purpose (such as AnyLogic[3]) as well as transport-specific simulation tools (such

[3] `http://www.anylogic.com`

as MATSIM[4] or SUMO[5]) has proven suitable for simulation-based assessment of a wider variety of control mechanisms. The agent-based simulation testbed described in this chapter was created to fill this gap and provide the researchers with a tool necessary to analyze and compare the control mechanisms of various classes without developing their own simulations.

3 Flexible Mobility Services Testbed

An agent-based simulation testbed should not only simulate a mobility system governed by a certain control mechanism – it should be able to support multiple different mechanisms and to compare them under identical benchmark conditions, so that the researchers or public authorities can discover relative strengths and weaknesses of their algorithms and businesses or municipalities can select the most appropriate mechanisms for specific real-world conditions. To satisfy these requirements, a simulation tool should have the following assets:

- Support for *centralized*, *distributed* and *hybrid* control mechanisms.
- Functions or protocols for *direct communication* between agents.
- Means of importing the transport demand for both *static* and *dynamic* cases.
- Agents capable of imposing *custom constraints* on the system (e.g. demand for wheel-chair support in a taxi).
- Simple way of *incorporating* various user-provided algorithms.

Unlike any of the simulations mentioned in Section 2, the open-source *Flexible Mobility Services Testbed* described in this chapter has all of these properties.

3.1 Modelling Ontology of AgentPolis Framework

Transport simulation framework AgentPolis, which serves as the basis for the testbed, provides abstractions, code libraries and software tools for building and experimenting with fully agent-based models of interaction-rich mobility systems. The modelling elements provided by AgentPolis are organized in a modelling ontology and can be grouped to three high-level categories:

- *Agent modelling elements:* The concept of the *agent* in AgentPolis is defined rather loosely in order to support modelling of a wide variety of agents (e.g. `DriverAgent`). The behavior of agents is defined in terms of *activities* – reactive control structures implementing the logic determining which actions or nested activities the agent executes at a certain point in time or in response to sensor information or messages received from other agents (e.g. `DriveVehicle` activity). As part of their behaviour, agents may need to make decisions that require executing complex algorithms, including the ones that comprise the control mechanisms we want to evaluate. In order

[4] `http://www.matsim.org`
[5] `http://www.sumo-sim.org`

to promote reusability, such algorithms are encapsulated into so-called *reasoning modules*. In practice, the reasoning modules are Java classes (e.g. `DriverLogic`) that can be easily rewritten to implement a wide variety of algorithms, or even call external tools or solvers.
- *Environment modelling elements:* The environment models the physical context in which the agents are situated and act. It is represented by a collection of *environment objects*, each representing a fragment of the modelled physical reality (e.g. `Vehicle`), and *queries* that allow agents to be informed about the state of the environment and about the events happening during simulation execution (e.g. `PositionQuery`).
- *Interaction modelling elements:* Modelling complex interactions among the agents or between the agents and the environment is crucial for the analysis of dynamic transport systems. In AgentPolis, agent-environment interactions are realized by *sensors*, which process the percepts from the environment and atomic *actions* that provide a low-level abstraction for modelling how agents actually manipulate the environment (e.g. `MoveVehicle`). Inter-agent interactions are realized by a collection of *communication protocols*. Currently, the testbed provides `1-to-1 messaging`, `1-to-many messaging` and `auction` protocols.

Detailed description of modelling abstractions and corresponding model elements can be found in [19].

3.2 Architecture of the Testbed

Although all the power and flexibility of the AgentPolis framework is accessible to the users of the testbed, it is hidden and only the relevant parts of it are exposed through an API designed specifically for the simulation modelling of demand-responsive mobility systems. The components of the testbed can be broadly divided into three layers (see Figure 1):

- *AgentPolis Transport Simulation Model:* composed of the core simulation engine and the basic transport domain model. This model implements key elements comprising a mobility system, such as road network and vehicles, and basic behavioural logic associated with them. It also provides routing algorithms and communication interfaces designed to simplify the implementation of higher-level simulation logic.
- *Testbed Core:* specializes the general AgentPolis simulator for the specific purpose of modelling demand-responsive transport systems. It implements the model of three types of agents (*Passengers*, *Drivers* and *Dispatchers*) and provides extensible abstractions for defining their behaviour.
- *Control Mechanism:* a user-supplied implementation of a specific control mechanism that is to be experimentally evaluated.

In addition to these three layers, the testbed provides a suite of tools that facilitate creation, execution and evaluation of simulation experiments:

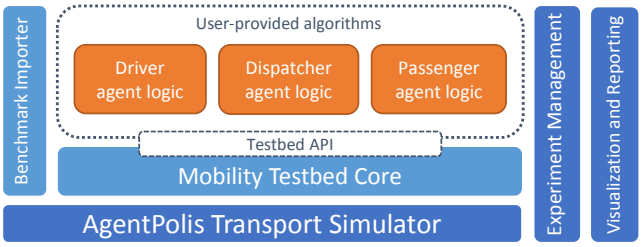

Fig. 1: Testbed's architecture overview.

- *Benchmark importer* loads all the required input data (discussed in detail in Section 4), constructs the graph representation of a road network and creates the environment objects and agents accordingly. All the imported data is automatically checked for consistency in order to prevent hard-to-trace errors. Resulting internal representations are simplified by selectively removing redundant information in order to accelerate the reasoning without losing accuracy.
- *Experiment management* is supported by a benchmark generator and tools for the design of experiments. The *generator* allows users to build their own scenarios covering real-world or fictional locations with custom numbers and types of agents. Agents can be generated either based on real-world data or randomly, using various temporal and spatial distributions.
  Since a robust evaluation of the control mechanism under a sufficiently wide range of circumstances may require many simulation runs, the testbed provides tools for accelerating the evaluation process. In particular, it can use *design of experiments* methods to generate simulation configurations in a way so that maximum information about the behaviour of the control mechanism is obtained using a minimum number of simulation runs.
- The *Analysis and visualization* tools provide a way to interactively browse and review simulation execution and results at different spatial and temporal resolutions. This assists researchers during the development and debugging process, and allows them to find out how the tested mechanisms perform under different conditions. The aggregated results, as well as the visualizations, are generated based on the detailed low-level event log recorded during the simulation, containing all the important events related to passengers (`passGotInVehicle`, `passGotOffVehicle`) and drivers with their vehicles (`vehicleMove`) and all the communication between the agents (e.g. `passSentRequest` or `requestConfirmed`).

3.3 Control Mechanisms

Unless there are too many different kinds of interactions between the agents, the incorporation of studied control mechanism into the testbed only requires implementing several classes and methods. For example, in the most simple case,

the user only needs to extend the `DispatchingLogic` class and implement its `processNewRequest(Request r)` method.

From the perspective of the testbed's user, the classification of control mechanism into *centralized*, *distributed* or *hybrid* class, depends only on whether the driver agent's actions are controlled centrally by (single or multiple) dispatcher agents, locally by the drivers themselves, or the combination of both. The reasoning logic for individual agents and central authorities is implemented by extending specific methods of abstract classes `PassengerLogic`, `DriverLogic` and `DispatchingLogic` (see Tables 1, 2 and 3). Decentralized mechanisms are suitable in situations when communication capabilities are restricted, or when the agents are independent and self-interested but can still benefit from collaboration (e.g. ridesharing).

Dynamic control mechanisms (sometimes called "online", especially in DARP context) process the travel demand requests when they are announced. On the other hand, *static* (or "offline") mechanisms need to know all the requests in advance. The testbed grants the driver or dispatcher agents the access to requests only after they are announced by the passengers. Nevertheless, to also address the requirements of static mechanisms, several benchmarks in which the travel demand is announced long time in advance are available.

4 Experiment Process

After the tested mechanism is incorporated into the framework, the actual experimentation using the testbed follows a three-step process[6] (see Figure 2).

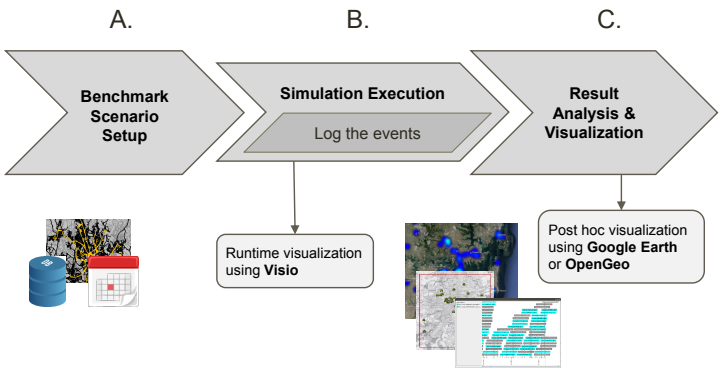

Fig. 2: Three-step process of the experiment (setup, simulation, evaluation).

[6] This section provides a high-level overview of testbed's usage. More detailed tutorial can be found at `http://github.com/agents4its/mobilitytestbed/wiki`.

4.1 Scenario Definition and Setup

First of all, the user needs to set up and *configure the scenario* under which he wants the control mechanism to be evaluated. The scenario is described in terms of a benchmark package, which consists of the following files:

1. *Road network* – The road network in the experiment area represented in the *OpenStreetMap (OSM)*[7] format.

[7] `http://openstreetmap.org/`

`sendRequest(Request r)`: Called by the testbed whenever this passenger is supposed to announce a new travel request `r`, according to input data. The passenger should contact other agents (dispatcher or drivers) within this method. `processProposal(Proposal p)`, `processRejection(Rejection r)`: Two methods that are called when the passenger receives a trip proposal `p` specifying details about the trip (e.g. price or arrival time) or rejection `r` of his older request from a driver or dispatcher. `vehicleArrived(String driverId, String vehicleId)`: Called when a driver arrives to pick the passenger up. Typically, the passenger just gets on board this driver's vehicle.

Table 1: Abstract methods of `PassengerLogic` class, implementing the behaviour of passenger agents.

`processNewRequest(Request r)`: A method called whenever the driver receives a new travel request `r` from a passenger or dispatcher. Here, the driver should react by sending his trip proposal or request rejection. `processNewAcceptance(Proposal p)`: Called when the driver's trip proposal `p` is accepted by the passenger. Here, the driver usually plans his route and starts driving. `processNewRejection(Proposal p)`: If driver's proposal `p` is rejected, the testbed calls this method. `processPassengerGotIn(String passengerId)`: Called when the passenger gets on board this driver's vehicle.

Table 2: Abstract methods of `DriverLogic` class, implementing the behaviour of driver agents.

`processNewRequest(Request r)`, `processNewAcceptance(Proposal p)`, `processNewRejection(Proposal p)`: The methods with similar meaning as in `DriverLogic` class with the exception that the dispatcher usually negotiates with passengers and only sends instructions and routes to drivers.

Table 3: Abstract methods of `DispatchingLogic` class, implementing the behaviour of dispatcher agents.

2. *Driver agents* – Description (in JSON[8]) of all the relevant drivers with their initial positions and the properties of their vehicles including the capacity, fuel consumption, CO_2 emissions or non-standard equipment (e.g. wheelchair accessibility).
3. *Travel demand* – The exact representation (in JSON) of travel demand containing all the passenger agents with their associated trip details: origin and destination coordinates, time windows, announcement time and special requirements.

4.2 Simulation Execution

Once the model is set up, the user invokes the simulation engine to execute the simulation. The AgentPolis engine employs the discrete event simulation approach [4] in which the operation of the target system is modelled as a discrete sequence of (possibly concurrent) events in time. Each event occurs at a particular time, with precision to milliseconds of the simulation time, and marks a change of state of the modelled system. Since there are no changes occurring between consecutive events, the simulation can directly jump in time from one event to the next, which, in most cases, makes it more computationally efficient than time-stepped approaches.

The simulation progress can be presented visually during run-time, using the internal visualization component of AgentPolis. It is capable of displaying the transport network and agents within the model, along with a convenient visualization of all the ongoing events (see Figure 3).

4.3 Result Analysis and Visualization

From the low-level event log recorded during the simulation run, the testbed calculates a range of higher-level, aggregate performance metrics. By default, these include:

- total vehicle distance driven,
- total fuel consumption,
- total values of CO_2 emissions,
- average vehicle productivity (passengers per hour),
- passenger's total travel time statistics (average, median and maximum),
- passenger's on-board ride time statistics,
- passenger's waiting time statistics,
- total runtime of control algorithms.

Additional metrics can be defined. In addition to low-level event logs and highly aggregated metrics, the testbed also provides the means to visualize the simulation runs and results in the geospatial and temporal context, using external tools.

[8] JavaScript Object Notation (JSON): `http://json.org/`

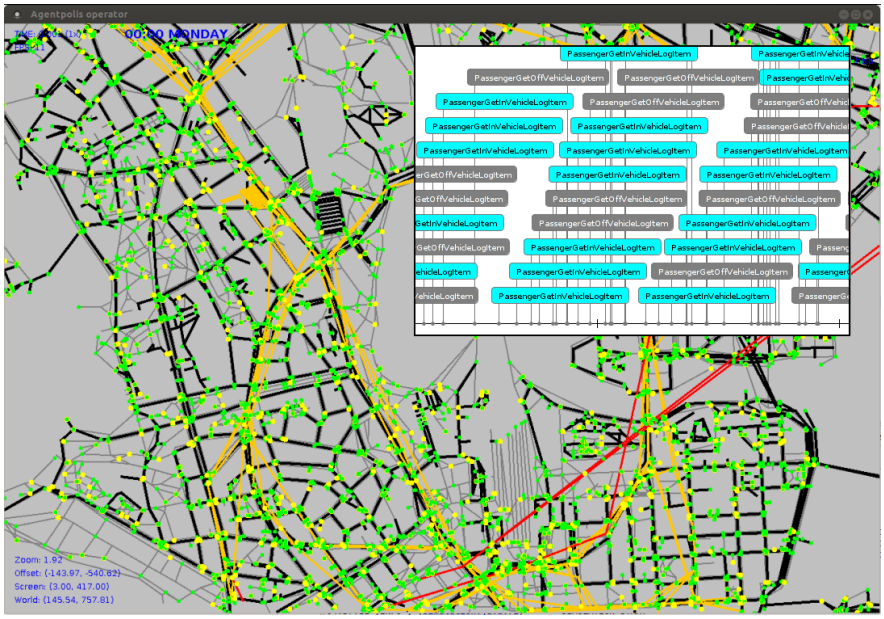

Fig. 3: Runtime view of a running simulation. Road network, `Passenger` and `Driver` agents are shown. Simulation events are depicted in the overlay window.

The interactive geobrowser *Google Earth*[9] can display the log of a simulation run exported in *Keyhole Markup Language* (KML)[10]. It is capable of displaying a large number of agents, along with simple geometry and screen overlays, over a realistic satellite imagery and 3D model of the environment (see Figure 4). Google Earth can be further used to display the values of metrics as they vary across different areas. For example, Figure 5 shows a heat map representing the spatial distribution of successfully served passenger trip requests.

5 Example Study

To demonstrate how the testbed can be used, we implemented and evaluated a control mechanism based on *parallel tabu search heuristic* by Attanasio et al. [2], which is considered a state of the art algorithm for dynamic multi-vehicle dial-a-ride problem (DARP). Mobility systems like taxi sharing schemes governed by centralized dynamic DARP algorithms can be considered autonomic – they are *automatic* (serve incoming requests without external intervention), *adaptive* (modify current plans of driver agents based on new requests) and *aware* (they monitor the state of the system continually and act accordingly).

[9] `http://earth.google.com/`
[10] `http://developers.google.com/kml/`

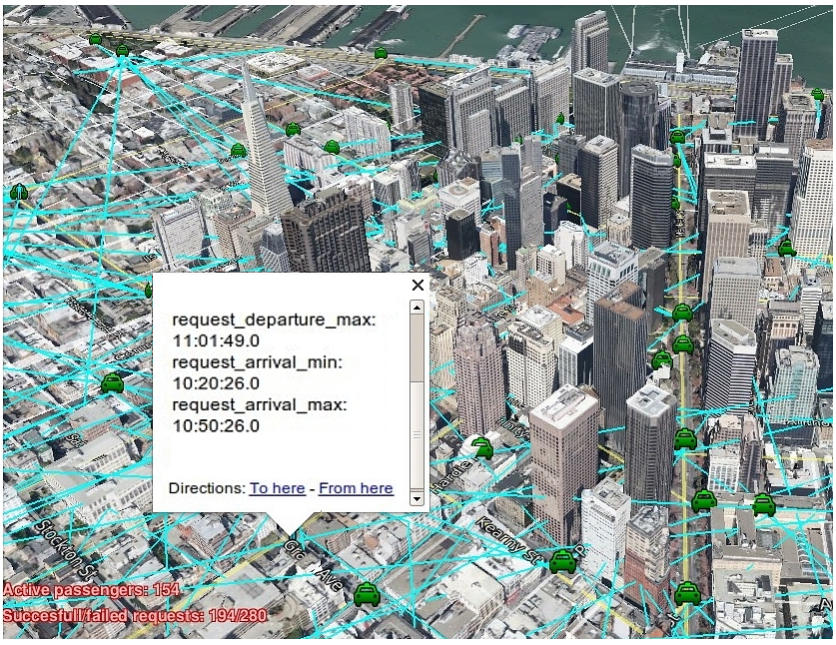

Fig. 4: Simulation run of a taxi sharing scenario exported in KML format and displayed by Google Earth. The input benchmark was based on historical traffic and demand data from San Francisco, 2008.

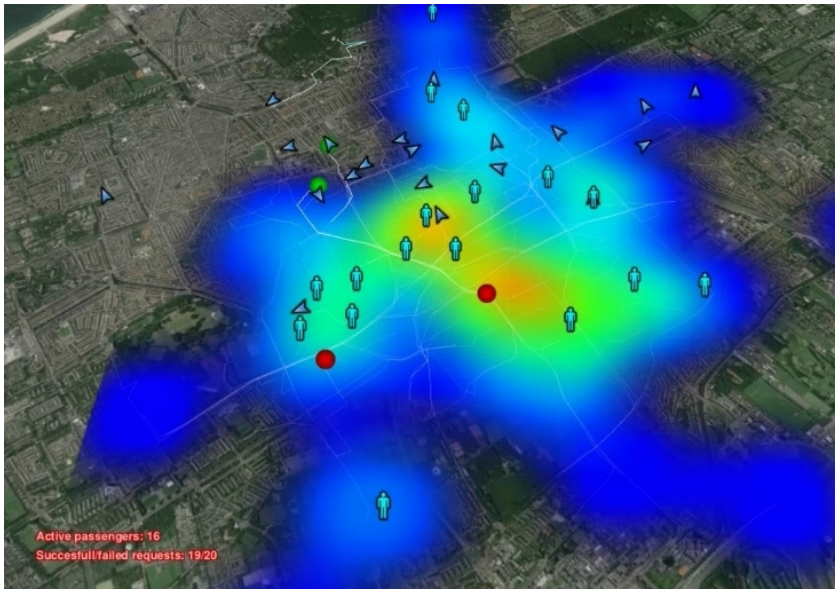

Fig. 5: Heat map representing the spatial distribution of successfully served trip requests in Hague, Netherlands.

5.1 Scenario Setup

Using our *benchmark generator*, we prepared two collections of scenarios[11]: one was situated in the urban core of Hague, Netherlands, covering the area of 80.09 km^2, while the other collection was set in Prague metropolitan area in Czech Republic, spread over 5938.64 km^2. The transport requests of passenger agents were generated with temporal distribution taking into account peak/off-peak hours based on our empiric knowledge, and uniform origin-destination distribution over all the nodes of our transport graph[12]. Each collection contained 24 scenarios – one for every combination of the number of driver agents (10 to 35 drivers, increasing by 5) and request frequency (from 100 up to 175 requests per day, increasing by 25).

5.2 Implementation of the Control Mechanism

Since this particular control mechanism is centralized in the sense that the dispatcher agent has complete power over the behaviour of all the drivers, we only needed to extend the abstract class `DispatchingLogic` and implement its method `processNewRequest(Request r)`, which is called every time the passenger agent announces a transport request.

5.3 Simulation Results

First, we analysed the relation between *request frequency* and *success rate*, computed as a ratio of successfully served requests and all the announced requests, with different *numbers of driver agents*.

We learned that in Prague we would need roughly 30 drivers to satisfy at least 90% of 150 daily requests, whereas in the smaller city of Hague we could maintain the same success rate with only 15 drivers (see Figure 6).

Once we had the estimation of optimal driver count, we were interested in the approximate distance driven by them on a daily basis. According to the experiments, to satisfy 150 requests per day, 30 drivers in Prague would drive 1722.29 km, while 15 drivers in Hague only 753.01 km (see Figure 7).

This way, we were able to estimate operational costs and initial investments needed to serve a specific demand in two different cities. We were also able to study a number of other metrics (see the enumeration in Subsection 4.3) and relations between them.

In addition to performance metrics, we were able to study the visualization of the system's behaviour. During the experiments in Prague, we noticed that, while in the morning the drivers were mainly situated in the city center, after a

[11] Another use case for the testbed can be found in [35], where we compared standard taxi and taxi sharing service in Sydney, Australia.

[12] Users are free to define their own, location-specific temporal distributions of requests based on historical data or empiric knowledge. Also, since version 2.0., the testbed's benchmark generator can use density of certain points of interest within OSM map to distribute requests in space.

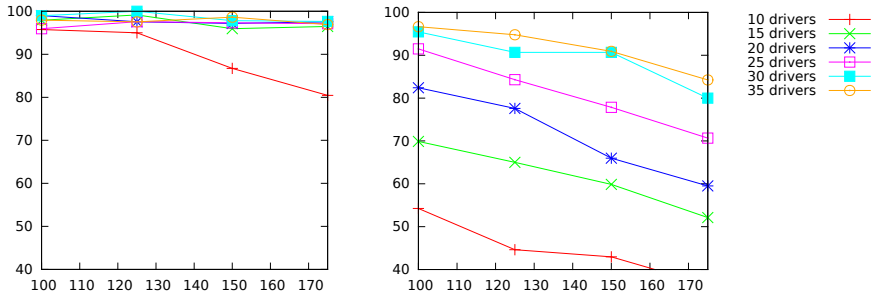

Fig. 6: Success rates in % (axis y) in relation to request frequency (axis x), with different numbers of drivers in Hague, Netherlands (left) and Prague, Czech Republic (right).

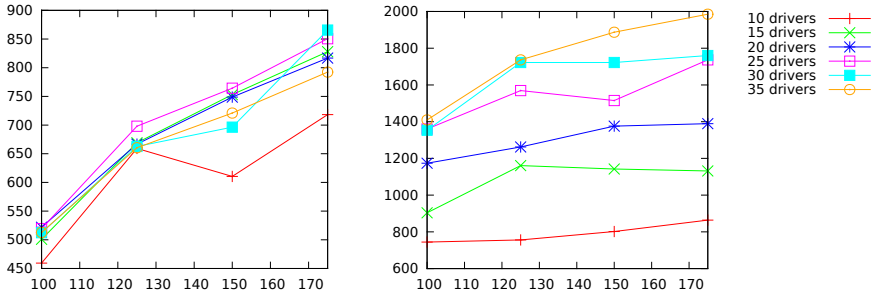

Fig. 7: Total distance in km driven per day (axis y) to serve increasing numbers of requests (axis x) by different numbers of agents in Hague, Netherlands (left) and Prague, Czech Republic (right).

few hours, majority of them served a long-distance request and ended up isolated at the edge of the service area, unable to serve new requests originating in the center quickly enough, because they were too far away (see Figure 8). This lead to significant decrease in service's success rate in the afternoon and evening hours. Discovering this problem via visualization allows us to take steps towards improving the control mechanism (for example by sending secluded drivers back to more busy areas even before the requests are announced).

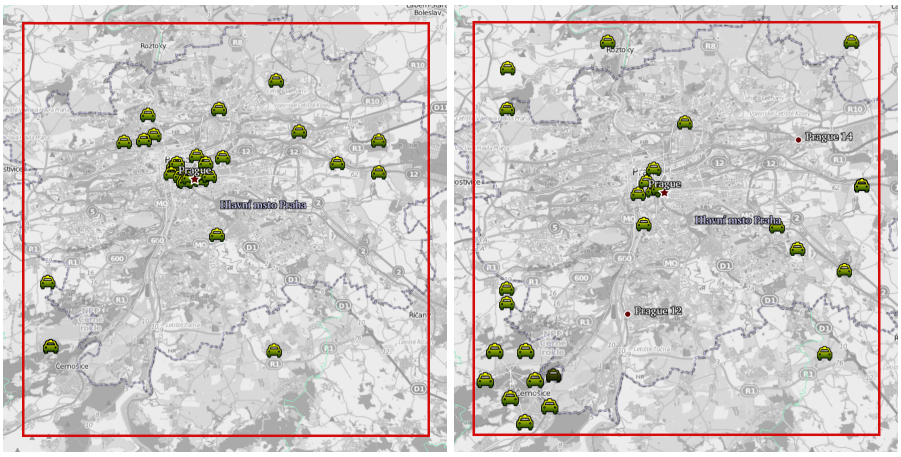

Fig. 8: Spatial distribution of drivers during the experiment in Prague metropolitan area at 5:00 (left) and 23:59 (right). In the later hours, many drivers were left isolated at the edges of the service area, unable to serve the requests originating in the city center quickly enough.

6 Conclusion

The presented testbed for simulation-based evaluation of autonomic mobility systems allows its users to incorporate their own control mechanisms, to evaluate them with respect to a variety of performance metrics and to compare their performance to alternative mechanisms under identical conditions using benchmark scenarios, based on realistic real-world or synthetic data. As such, *Flexible Mobility Services Testbed* can assist researchers in developing new control mechanisms as well as it can help policy makers and transport operators assess the mobility services and schemes prior to their deployment. The testbed is freely available from: `http://github.com/agents4its/mobilitytestbed/`.

7 Acknowledgements

This work was funded by Ministry of Education, Youth and Sports of Czech Republic (grants no. 7E12065 and LD12044), Technology Agency of the Czech Republic (grant no. TE01020155) and by the European Union Seventh Framework Programme FP7/2007-2013 (grant agreement no. 289067).

References

1. N. Agatz, A. L. Erera, M. W. Savelsbergh, and X. Wang. Dynamic ride-sharing: a simulation study in metro atlanta. *Procedia - Social and Behavioral Sciences*, 17(0):532 – 550, 2011. Papers selected for the 19th International Symposium on Transportation and Traffic Theory.
2. A. Attanasio, J.-F. Cordeau, G. Ghiani, and G. Laporte. Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing*, 30(3):377–387, 2004.
3. W. A. Bailey Jr and T. D. Clark Jr. A simulation analysis of demand and fleet size effects on taxicab service rates. In *Proceedings of the 19th conference on Winter simulation*, pages 838–844. ACM, 1987.
4. J. Banks, J. S. Carson, B. L. Nelson, D. M. Nicol, et al. *Discrete-event system simulation*. Pearson Prentice Hall, NJ, 2005.
5. L. Bodin and B. Golden. Classification in vehicle routing and scheduling. *Networks*, 11(2):97–108, 1981.
6. S.-F. Cheng and T. D. Nguyen. Taxisim: A multiagent simulation platform for evaluating taxi fleet operations. In *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology-Volume 02*, pages 14–21, 2011.
7. J. de Dios Ortuzar and L. G. Willumsen. Modelling transport. 1994.
8. F. P. Deflorio, B. Dalla Chiara, A. Murro, and M. A. SpA. Simulation and performance of drts in a realistic environment. In *Proceedings of the 9th Meeting EWGT on Intermodality, Sustainability and Intelligent Transportation Systems and 13th Mini EURO Conference on Handling Uncertainty in the Analysis of Traffic and Transportation Systems, 2002*, 2002.
9. M. Diana. The importance of information flows temporal attributes for the efficient scheduling of dynamic demand responsive transport services. *Journal of advanced Transportation*, 40(1):23–46, 2006.
10. P. M. d'Orey, R. Fernandes, and M. Ferreira. Empirical evaluation of a dynamic and distributed taxi-sharing system. In *Proceedings of ITSC 2012*, pages 140–146. IEEE, 2012.
11. P. Drewe. What about time in urban planning & design in the ict age. In *Proceedings of the CORP conference*, 2005.
12. D. Edwards, A. Elangovan, and K. Watkins. Reaching low-density urban areas with the network-inspired transportation system. In *Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on*, pages 826–831. IEEE, 2012.
13. L. Fu. A simulation model for evaluating advanced dial-a-ride paratransit systems. *Transportation Research Part A: Policy and Practice*, 36(4):291–307, 2002.
14. A. M. Geoffrion. The purpose of mathematical programming is insight, not numbers. *Interfaces*, 7(1):81–92, 1976.

15. A. Haghani and M. Banihashemi. Heuristic approaches for solving large-scale bus transit vehicle scheduling problem with route time constraints. *Transportation Research Part A: Policy and Practice*, 36(4):309 – 333, 2002.
16. R. W. Hall. Discrete models/continuous models. *Omega*, 14(3):213–220, 1986.
17. D. Hauptmeier, S. O. Krumke, and J. Rambau. The online dial-a-ride problem under reasonable load. In *Proceedings of the 4th Italian Conference on Algorithms and Complexity*, CIAC '00, pages 125–136, London, UK, UK, 2000. Springer-Verlag.
18. M. Horn. Multi-modal and demand-responsive passenger transport systems: a modelling framework with embedded control systems. *Transportation Research Part A*, 36(2):167–188, 2002.
19. M. Jakob and Z. Moler. Modular framework for simulation modelling of interaction-rich transport systems. In *Proceedings of ITSC 2013*. IEEE, 2013.
20. J. Jlassi, J. Euchi, and H. Chabchoub. Dial-a-ride and emergency transportation problems in ambulance services. *Computer Science and Engineering*, 2(3):17–23, 2012.
21. E. Kamar and E. Horvitz. Collaboration and shared plans in the open world: Studies of ridesharing. In *Proceedings of IJCAI 2009*, volume 9, page 187, 2009.
22. J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.
23. F. Klügl. *Agent-based simulation engineering*. PhD thesis, Habilitation Thesis, University of Würzburg, 2009.
24. A. Langevin, P. Mbaraga, and J. F. Campbell. Continuous approximation models in freight distribution: An overview. *Transportation Research Part B: Methodological*, 30(3):163–188, 1996.
25. E. Lioris, G. Cohen, and A. de La Fortelle. Overview of a dynamic evaluation of collective taxi systems providing an optimal performance. In *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pages 1110–1115. IEEE, 2010.
26. M. Lipmann, X. Lu, W. E. Paepe, R. A. Sitters, and L. Stougie. On-line dial-a-ride problems under a restricted information model. *Algorithmica*, 40(4):319–329, 2004.
27. M. Pěchouček, M. Jakob, and P. Novák. Towards simulation-aided design of multi-agent systems. In *Programming Multi-Agent Systems*, pages 3–21. Springer, 2012.
28. L. Quadrifoglio and M. Dessouky. Insertion heuristic for scheduling mobility allowance shuttle transit (mast) services: sensitivity to service area. *Computer-Aided Systems in Public Transport, Springer Series: Lecture Notes in Economics and Mathematical Systems*, 600, 2007.
29. L. Quadrifoglio, M. M. Dessouky, and F. Ordóñez. A simulation study of demand responsive transit system design. *Transportation Research Part A: Policy and Practice*, 42(4):718–737, 2008.
30. A. C. Regan, H. S. Mahmassani, and P. Jaillet. Dynamic decision making for commercial fleet operations using real-time information. *Transportation Research Record: Journal of the Transportation Research Board*, 1537(1):91–97, 1996.
31. S. Schuetz, K. Zimmermann, G. Nunzi, S. Schmid, and M. Brunner. Autonomic and decentralized management of wireless access networks. *Network and Service Management, IEEE Transactions on*, 4(2):96–106, 2007.
32. K. Shinoda, I. Noda, M. Ohta, Y. Kumada, and H. Nakashima. Is dial-a-ride bus reasonable in large scale towns? evaluation of usability of dial-a-ride systems by simulation. In *Multi-agent for mass user support*, pages 105–119. Springer, 2004.
33. D. M. Stein. Scheduling dial-a-ride transportation systems. *Transportation Science*, 12(3):232–249, 1978.
34. A. M. Uhrmacher and D. Weyns. *Multi-Agent systems: Simulation and applications*. CRC Press, 2010.

35. M. Čertický, M. Jakob, and R. Píbil. Analyzing on-demand mobility services by agent-based simulation. *Journal of Ubiquitous Systems Pervasive Networks*, 6(1):17–26, 2015.
36. N. H. M. Wilson, J. Sussman, L. Goodman, and B. Hignnet. Simulation of a computer aided routing system (cars). In *Proceedings of the third conference on applications of simulation*, pages 171–183. Winter Simulation Conference, 1969.