Simulation Testbed for the Accelerated Development of Autonomic Mobility Systems

Michal Jakob[1], Michal Čertický[1], Zbyněk Moler[1]

Abstract— **We present an open-source simulation testbed for emerging *autonomic mobility systems*, in which transport vehicles and other transport resources are automatically managed to serve a dynamically changing transport demand. The testbed, based on the AgentPolis framework, allows testing and evaluating the various multiagent planning, coordination and resource allocation algorithms for the control and management of autonomic mobility systems. 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 up to the analysis and interpretation of simulation results. The testbed aims to accelerate the development of control algorithms for autonomic mobility systems and to facilitate their mutual comparison on well-defined benchmark scenarios.**

I. 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 [5]. 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. Implementing such principles requires the development of multiagent 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 algorithms is a challenging task due to many internal and external interdependencies that affect the overall system behaviour.

Simulation modelling and multi-agent simulations in particular [16] are well-established approaches for analysing the behaviour of complex socio-technical systems and have already been applied in the field of mobility systems (see Section II). No attempt, however, has been so far made to provide a more universal simulation tool for studying the behaviour of a wider range of autonomic mobility systems.

[1]{jakob, certicky, moler}@agents.fel.cvut.cz, Agent Technology Center, Faculty of Electrical Engineering, Dept. of Computer Science and Engineering, Czech Technical University, Prague, Czech Republic

The presented testbed, based on the versatile *AgentPolis*[1] [8] transport simulation framework, is our attempt to fill this gap. The testbed allows experimenting with algorithms for the management and control of autonomic mobility systems and assessing their performance in different configurations and scenarios. The testbed supports all stages of experimentation process, from the implementation of tested control algorithms and the definition of experiment scenarios, through simulation execution up to the analysis and interpretation of simulation results.

To facilitate its use, the testbed provides extension points that allows control algorithms to be plugged-in easily by implementing several predefined abstract 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 that can be used for mutual comparison of different control algorithms.

II. Related Work

The general idea of employing simulation testbeds to accelerate the development of multi-agent control mechanisms was put forward e.g. by [13], [9].

Focusing specifically on transport, [7] employed an agent-based simulation, developed completely from scratch, to study operational characteristics of a multimodal transport system integrating scheduled and flexible on-demand services. Demand-responsive transport systems were also studied in [15]. Real-time taxi sharing schemes have been also evaluated using simulation [12], [4]) as were on-demand cooperative paratransit services with strong emphasis on resource allocation, e.g. [6] or [10].

A common attribute of the above simulations of autonomic mobility systems is that these simulations were developed from scratch using general-purpose programming languages (most often C++ or Java). This is because none of the existing general (such as AnyLogic[2]) as well as transport-specific simulation tools (such MATSIM[3]) has proved suitable for simulation-based assessment of autonomic mobility systems.

III. Testbed Overview

The proposed simulation testbed is built upon the versatile transport simulation framework AgentPolis, which

[1]http://agentpolis.org
[2]http://www.anylogic.com
[3]http://www.matsim.org

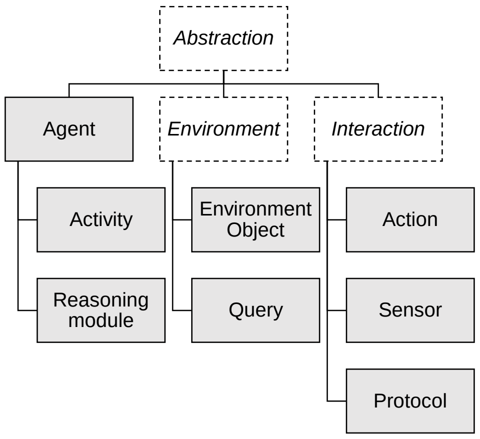


Fig. 1: Modelling abstractions of the AgentPolis framework. The concepts in the white, dashed-outline boxes only provide grouping and are not used as modelling abstractions.

provides abstractions, code libraries and software tools for building and experimenting with agent-based models of interaction-rich multimodal transport systems. The framework consists of the following components:

- *Modelling abstraction ontology* which provides a unifying set of concepts for expressing agent-based simulation models. The abstractions refine the more general multiagent systems concepts and make them expressible in object-oriented programming languages.
- *Modelling element library* which contains concrete implementations of the modelling abstractions chosen so as to represent the elements frequently used in real-world transport models.
- *Simulation engine*, based on the discrete event simulation approach, which provides the runtime functionality for simulating AgentPolis models.
- *Simulation tools* which support the deployment and use of AgentPolis models in real-world conditions by providing data import, scenario configuration and simulation result analysis and visualization capabilities.

A. Modelling Abstractions

Modelling abstractions of the AgentPolis framework (figure 1) covers several dozens of modelling elements that are used to build specific simulation models, including our proposed AgentPolis testbed:

- *Agents.* The concept of the *agent* in AgentPolis needs to be defined loosely, since it covers a wide variety of different types of actors (taxi drivers, travellers, dispatchers, etc.). During its lifecycle, an agent usually needs to make decisions that require executing complex algorithms, including the ones that we want to evaluate. In accordance with the conceptual architecture of the AgentPolis framework, these algorithms are encapsulated into so-called *reasoning modules*.
- *Reasoning modules* constitute the major source of intelligent behaviour of modelled transport system. In addition to the standard pathfinding module of AgentPolis, our testbed contains interchangeable implementations of the coordination or resource allocation mechanisms that we want to evaluate. In practice, the reasoning modules are Java classes that can be easily rewritten to implement a wide variety of algorithms, or even call external tools or solvers.
- *Sensors and queries* process the percepts from the environment and allow agents to be informed about the state of the environment and events happening during simulation execution.
- *Activities* provide the abstraction for defining agent's actual behaviour in the simulation. Technically, activities are reactive control structures determining which *actions* the agent executes in response to certain events (e.g. `DriveVehicle`).
- *Actions* provide a low-level abstraction for modelling how agents actually manipulate the environment (e.g. `MoveVehicle`).
- *Environment objects.* The environment models the physical context in which 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`).
- *Communication protocols* are the abstraction for modelling inter-agent communication by means of message passing. Currently, the testbed provides `1-to-1 messaging`, `1-to-many messaging` and `auction` protocols, that should be sufficient for the purposes of the coordination or resource allocation algorithms. However, additional protocols can be added to the framework if they are needed.

Detailed description of modelling abstractions and corresponding model elements is given in [8].

B. Control Logic Implementation

AgentPolis is a versatile framework for creating simulation models of a very wide range of transport systems. In addition to supporting the modelling of autonomic mobility systems, AgentPolis can also be used for activity-based transport planning models, modelling the security transport systems or urban goods delivery logistics [8]. Although all the power and flexibility of the AgentPolis framework is accessible to the users of the AgentPolis testbed, it is normally hidden and only the relevant parts of framework are exposed through a facade of APIs and interface designed specifically for modelling and assessing autonomic mobility systems.

Unless the tested control algorithm has some special features, its incorporation into the testbed only requires implementing several classes and methods. For example, to evaluate a centralized coordination mechanism for an on-demand mobility system, the user needs to extend the `DispatchingLogic` class and implement its `processNewRequest()` method.

The testbed supports both centralized and decentralized control mechanisms. For example in the case on-demand

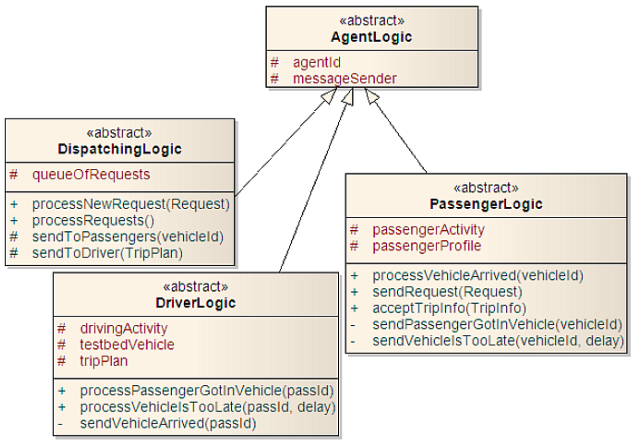

«abstract»
AgentLogic
# agentId
# messageSender
«abstract»
DispatchingLogic
# queueOfRequests
+ processNewRequest(Request)
+ processRequests()
# sendToPassengers(vehicleId)
# sendToDriver(TripPlan)
«abstract»
PassengerLogic
# passengerActivity
# passengerProfile
+ processVehicleArrived(vehicleId)
+ sendRequest(Request)
+ acceptTripInfo(TripInfo)
- sendPassengerGotInVehicle(vehicleId)
- sendVehicleIsTooLate(vehicleId, delay)
«abstract»
DriverLogic
# drivingActivity
# testbedVehicle
# tripPlan
+ processPassengerGotInVehicle(passId)
+ processVehicleIsTooLate(passId, delay)
- sendVehicleArrived(passId)

Fig. 2: Abstract classes that need to implemented for testing control algorithms for on-demand transport.

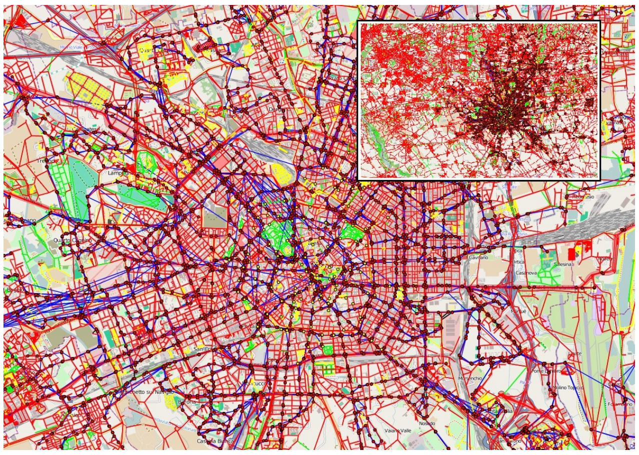


Fig. 4: An integrated transport graph combining road and public transport networks (Milano, Italy).

mobility system, the behaviour of driver agents can be governed either 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 the `DriverLogic` or `DispatchingLogic` classes (see Figure 2). The 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 [11]).

Coordination algorithms can further be divided into two categories. *Online* algorithms (sometimes called "dynamic", for example in [3]) process the travel demand (request) when it is announced. On the other hand, *offline* (or "static") algorithms need to know the demand in advance. The AgentPolis testbed grants the agents (drivers or dispatchers) the access to requests only after they are announced by the travellers. Nevertheless, to also cater for the requirements of offline algorithms, there are several benchmarks in which the travel demand is announced long time in advance.

IV. Testbed Experiment Process

After the tested algorithm is incorporated into the framework, the actual experimentation using the testbed follows a three-step process, as depicted in Figure 3.

A. Scenario Definition and Setup

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

- *Road network* – The road network in the experiment area represented in the *OpenStreetMap(OSM))*[4] format.
- *Traffic intensities (optional)* – The time-dependent traffic intensities on individual road segments affecting the travel speed, represented in JavaScript Object Notation (JSON). The ability to import traffic intensities allows more accurate assessment without the necessity to simulate all vehicles in the transport systems.
- *Vehicle fleet* – Description (in JSON) of all relevant vehicles with their properties including the capacity, initial position, fuel consumption, CO_2 emissions or non-standard equipment (e.g. wheelchair accessibility).
- *Public transport timetables (optional)* – Description of public transport schedules, routes and stations in the *General Transit Feed Specification (GTFS)*[5] format.
- *Travel demand* – The exact representation in JSON format of travel demand containing all the traveller agents with their associated trip details: origin and destination coordinates, time windows, announcement time and special requirements.

The AgentPolis testbed includes a *benchmark importer*, which loads such benchmark package, constructs the road network and creates all the environment objects and agents according to it. The testbed also includes a random *benchmark generator*, which allows users to generate their own randomized scenarios with custom numbers and types of vehicles, travellers and their distributions.

Real-word Data Import: To facilitate working with real-word scenarios, the testbed includes tools for importing map (OSM) and public transport (GTFS) data, and for automatically cross-reference between them. This way, the information about road, cyclepath and footpath networks and public transport routes and timetables can be easily incorporated (see Figure 4 for an example). All data imported into the testbed can be automatically checked for consistency in order to prevent hard-to-trace errors caused by invalid data.

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

[5]`http://developers.google.com/transit/gtfs/reference`

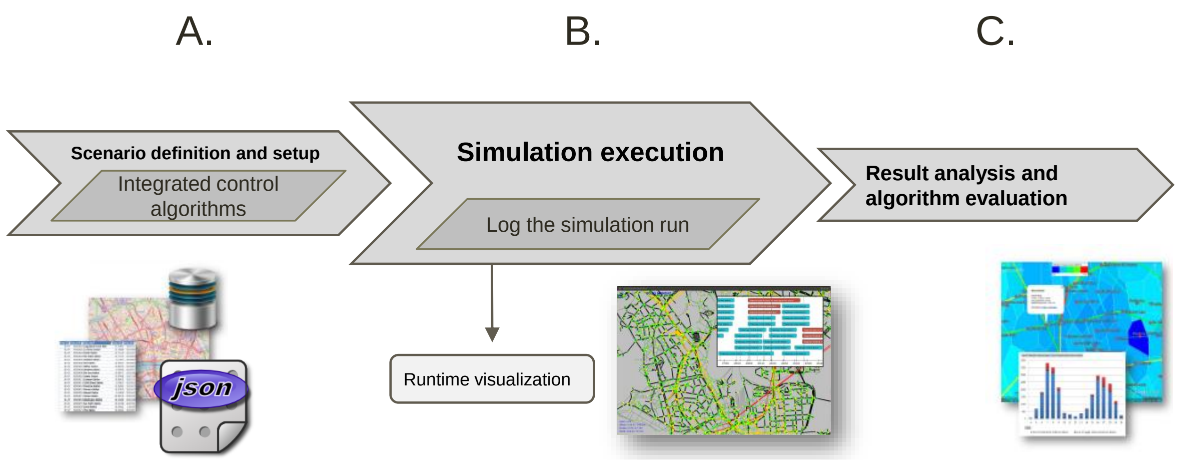

A.
B.
C.
Scenario definition and setup
Integrated control algorithms
Simulation execution
Log the simulation run
Result analysis and algorithm evaluation
Runtime visualization
json

Fig. 3: Three-step process of the experiment (setup, simulation, evaluation).

Design of Experiments: Robust evaluation of a control algorithm under a sufficiently wide range of circumstances may require thousands of simulation runs. To speed up the evaluation process, the testbed provides a simulation management tool leveraging the *design of experiments* methods that generate simulation configurations in a way so that maximum information about the behaviour of the tested algorithm is obtained using a minimum number of simulation runs. The effectiveness of the simulation management tool is further increased by the ability to execute simulations in parallel on high-performance computing grid infrastructures.

B. 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 [1] 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 (see Figure 5). 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.

Event Logging: To record simulation progress, in order to analyse and evaluate the algorithm's performance, the framework provides a logging mechanism that allows detailed recording of all the low-level simulation events. Recorded event log is later used to compute the aggregate metrics and analyse or visualize the simulation run and algorithm's behaviour.

Run-time Visualization: The experiment is also presented visually on the run-time, using the internal visualization component of AgentPolis. It is capable of displaying the transport network and agents within our model, along with a convenient visualization of all the actual events (see figure 5).

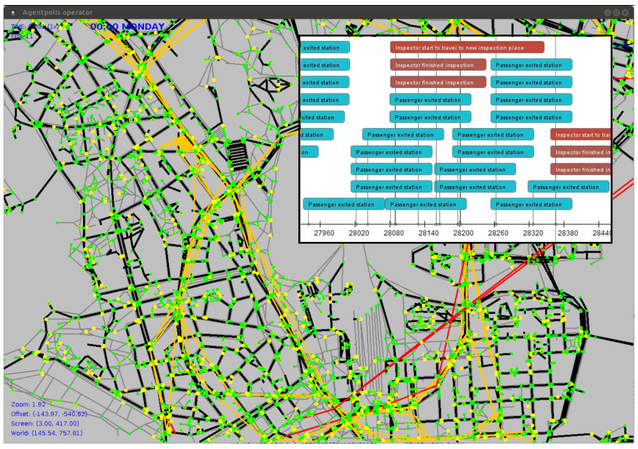


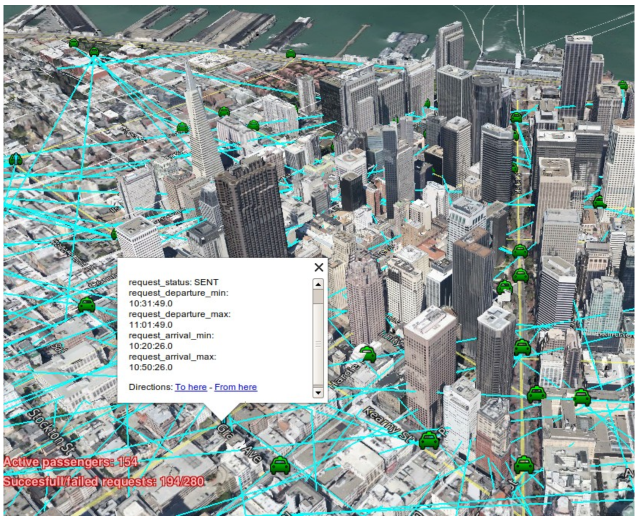
Fig. 5: Runtime view of a running AgentPolis simulation model. Road (black), pedestrian (grey), tram (yellow) and metro (red) networks and `UrbanCitizen` (green) and `PTDriver` (yellow) agents are shown. Simulation events are depicted in the overlay window.

C. 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 two tools for visualizing simulation runs and results in the geospatial and temporal context.


```
request_status: SENT
request_departure_min:
10:31:49.0
request_departure_max:
11:01:49.0
request_arrival_min:
10:20:26.0
request_arrival_max:
10:50:26.0

Directions: To here - From here
```

Active passengers: 154
Succesfull/failed requests: 194/280

Fig. 6: Simulation run of on-demand transport coordination 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.

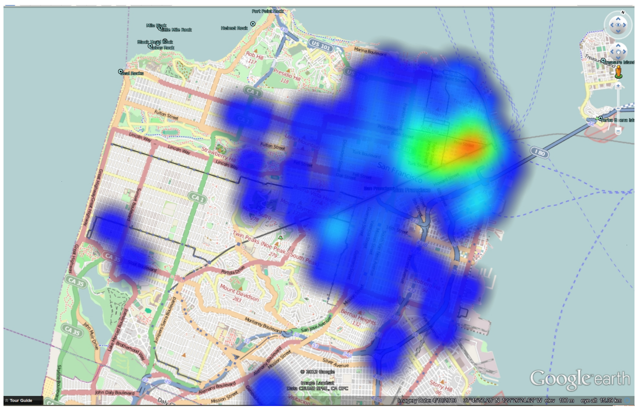
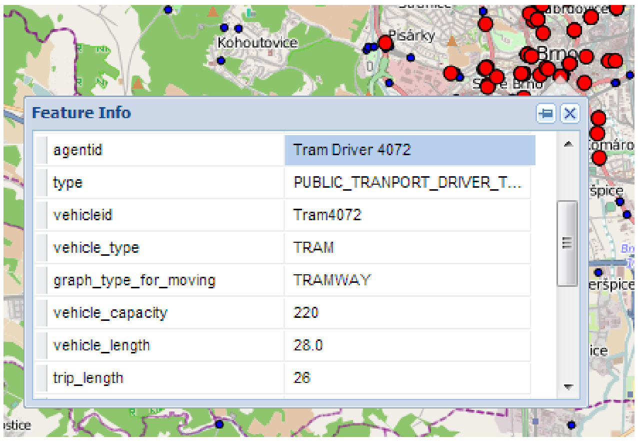


Fig. 7: Heat map representing the spatial distribution of successful trip requests shown in Google Earth.

The interactive geobrowser *Google Earth*[6] can display the progress of a simulation run exported in *Keyhole Markup Language* (KML)[7]. It is capable of displaying relatively large number of agents, along with simple geometry and screen overlays, over a realistic satellite imagery and 3D model of the environment (see figure 6). Google Earth can further be used to display the values of metrics as they vary across different areas. For example, Figure 7 shows a heat map representing the spatial distribution of successful passenger trip requests in the simulation of an on-demand mobility system.

OpenGeo[8] is an alternative JavaScript-based geovisualization framework that can be used to display the experiment progress and results in a standard web browser. It allows displaying a variety of details about every agent in the model at any point in time (see figure 8).

[6]http://earth.google.com/
[7]http://developers.google.com/kml/
[8]http://opengeo.org/


Feature Info

agentid	Tram Driver 4072
type	PUBLIC_TRANPORT_DRIVER_T...
vehicleid	Tram4072
vehicle_type	TRAM
graph_type_for_moving	TRAMWAY
vehicle_capacity	220
vehicle_length	28.0
trip_length	26

Fig. 8: Details about a specific agent displayed in the OpenGeo frontend.

Together, the AgentPolis visualization tools allow interactively browsing and analysing simulation results at different spatial and temporal resolutions. This assists the researchers during the development and debugging process, and allows him to find out how the tested algorithms perform under different conditions.

V. Example Study

An example application of the AgentPolis testbed we show how it was used to evaluate a basic algorithm for decentralized taxi sharing service, described in [2].

A. Scenario Setup

The scenario used for the evaluation was built using two datasets containing real-world historical data from San Francisco Bay Area (SF), 22nd of May 2008. From the first dataset [14], including the GPS traces of 536 taxis collected in SF, we extracted the travel demand (10,000 requests) and a list of taxis and used them to generate the traveller agents and taxi drivers, along with their necessary attributes.

The second dataset, obtained from the Freeway Performance Measurement System (PeMS) [17], contained, among other useful information, the maximum and average speeds on individual segments of the SF road network, measured in 5-minute intervals. Transport network itself was imported from the public OSM dataset[9].

B. Algorithm Implementation

To incorporate the taxi sharing control algorithm into the testbed, we modified the class implementing the decision making of the *driver* agent, since the algorithm is decentralized and based on the interactions between drivers and passengers. We extended the abstract class `DriverLogic` and implemented the `processRequests()` method.

[9]http://openstreetmap.org/

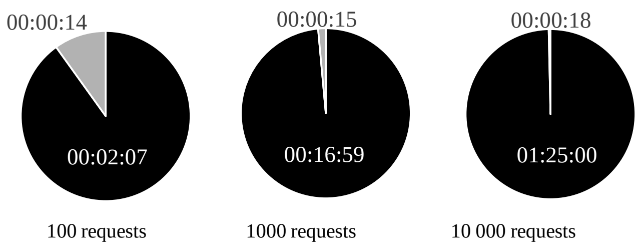


Fig. 9: Run times of the simulation with 100, 1000 and 10000 requests. Dark portions of the graph correspond to the time spent by the coordination control algorithm, while the light portions depict the time needed to run simulation testbed.

C. Simulation Results

One simulation run, representing 24 hours of simulation time, with 10000 requests and 536 drivers, took 1h25min of wall-clock time, out of which only 18 seconds were spent by the simulation and the rest spent by the coordination control algorithm itself. The simulation used an average of 7GB and maximum of 8.5GB RAM on a system with Intel Core i7 CPU 960 @ 3.20GHz processor, 24GB RAM, running a 64bit release of Ubuntu 12.04 OS.

Aggregate metrics (described in Section IV) were computed and confirmed our expectations that due to its simplicity, the algorithm would not perform well under the real-world conditions of the scenario. For example, the median of passenger travel time was 1h35min, which is not satisfactory.

Similar experiment with 536 drivers but only 1000 requests needed an average of 2.5GB RAM and took 17 minutes, out of which only 15 seconds were spent by the simulation. The simplest experiment, with only 100 requests, required an average of 0.75GB RAM and took 2min21s, out of which the simulation took only 14 seconds. It appears that the simulation overhead does not grow significantly with the increasing number of agents. See Figure 9 for the comparison of the simulation overhead for different scenario sizes.

VI. Conclusion

We have developed a software testbed, based on the AgentPolis framework, for simulation-based evaluation of autonomic mobility systems. The testbed allows the users to incorporate their own control algorithms, evaluate them with respect to range performance metrics and compare their performance to alternative control algorithms under identical conditions using benchmark scenarios based on realistic real-world or synthetic data. The testbed will soon be made publicly available on the AgentPolis website `http://agentpolis.org/`.

References

[1] J. Banks, J. S. Carson, B. L. Nelson, D. M. Nicol, et al. *Discrete-event system simulation.* Pearson Prentice Hall Upper Saddle River, NJ, 2005.

[2] L. Čanda. Simulační testbed pro algoritmy poptávkové přepravy. Master's thesis, České vysoké učení technické v Praze, 2013.

[3] J.-F. Cordeau and G. Laporte. The dial-a-ride problem: models and algorithms. *Annals of Operations Research*, 153(1):29–46, 2007.

[4] P. M. d'Orey, R. Fernandes, and M. Ferreira. Empirical evaluation of a dynamic and distributed taxi-sharing system. In *Proceedings of the 15th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 140–146, 2012.

[5] P. Drewe. What about time in urban planning & design in the ict age. In *Proceedings of the CORP conference*, 2005.

[6] 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.

[7] M. Horn. Multi-modal and demand-responsive passenger transport systems: a modelling framework with embedded control systems. *Transportation Research Part A: Policy and Practice*, 36(2):167–188, 2002.

[8] M. Jakob and Z. Moler. Modular framework for simulation modelling of interaction-rich transport systems. In *Proceedings of 15th International IEEE Conference on Intelligent Transportation Systems*. IEEE, 2013.

[9] M. Jakob, M. Pechoucek, M. Cap, P. Novák, and O. Vanek. Mixed-reality testbeds for incremental development of hart applications. *IEEE Intelligent Systems*, 27(2):19–25, 2012.

[10] 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.

[11] E. Kamar and E. Horvitz. Collaboration and shared plans in the open world: Studies of ridesharing. In *IJCAI*, volume 9, page 187, 2009.

[12] 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.

[13] 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.

[14] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser. CRAWDAD data set epfl/mobility (v. 2009-02-24). Downloaded from http://crawdad.cs.dartmouth.edu/epfl/mobility, Feb. 2009.

[15] 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.

[16] A. M. Uhrmacher and D. Weyns. *Multi-Agent systems: Simulation and applications*. CRC Press, 2010.

[17] P. P. Varaiya. *Freeway performance measurement system: Final report*. California PATH Program, Institute of Transportation Studies, University of California at Berkeley, 2001.