TRAPPed in Traffic? A Self-Adaptive Framework for Decentralized Traffic Optimization

*Authors ordered alphabetically

Ilias Gerostathopoulos
Department of Software and Systems Engineering
Technical University of Munich
Munich, Germany
gerostat@in.tum.de

Evangelos Pournaras
Professorship of Computational Social Science
ETH Zurich
Zurich, Switzerland
epournaras@ethz.ch

***Abstract*—Optimizing the traffic flow in a city is a challenging problem, especially in a future traffic system of self-driving cars and sharing vehicles. This is due to the interactions between the individual traffic agents (vehicles) that compete for the use of the common infrastructure (streets) given traffic dynamics such as stop-and-go effects, changing lanes, and other. The goal of this paper is to provide a solution to the above problem that works in a fully decentralized and participatory way, i.e. autonomous agents collaborate without a centralized data collector and arbitrator. Such a solution should be scalable, privacy-preserving, and flexible with respect to the degree of autonomy of agents. A self-adaptive framework to support this research is introduced: TRAPP – Traffic Reconfigurations via Adaptive Participatory Planning. The framework relies on a microscopic traffic simulator, SUMO, for simulating urban mobility scenarios, and on a decentralized multi-agent planning system, EPOS, for decentralized combinatorial optimization, applied here in traffic flows. A data-driven interoperation of the two tools in the proposed framework allows high modularity and customization for experimenting with different scenarios, optimization objectives and agents' behavior and as such providing new perspectives for resilient future traffic infrastructures.**

***Index Terms*—self-adaptation, optimization, multi-agent system, traffic, planning, framework**

I. Introduction

Optimizing the traffic flow in a city is a challenging problem that is amenable to self-adaptation approaches. Indeed, real-time traffic control—viewed as a self-adaptation challenge—is the problem of monitoring the current traffic state, analyzing the state to identify traffic jams (and potentially predicting such bottlenecks as well as slow-downs in the near future), planning measures that can regulate or steer driving behavior in order to avoid negative externalities or further optimize the traffic flow, and applying these measures in the real world [1]. In this view, the managed system comprises the vehicles together with their drivers and passengers, and the managing system is responsible for performing changes in the navigation of cars in order to optimize the overall traffic flow.

From the perspective of self-adaptation, traffic control is a challenging problem, since it is very complex to plan ahead for all possible situations and corresponding actions. There is high uncertainty, e.g., on when a certain event that warrants a collective adaptation, such as a traffic accident, might occur. At the same time, it is equally challenging to prescribe an adaptation action when such an event occurs and a road segment is blocked: Shall only some of the vehicles be routed from one of the alternative routes or all of them? Upon re-opening a road segment, shall all vehicles be navigated through that road again? The complexity of traffic coordination is relevant in the case of regular traffic flow as well, however in exceptional situations it becomes even more prominent.

Another question concerns the optimization objective behind real-time traffic control. In many cases, optimizing traffic concerns the maximization of throughput in streets and the minimization of trip times. However, instead of or together with minimizing the time of the average trip, an objective could be to avoid too long trips. As another possible objective, public authorities may also care for balancing the utilization of streets in order to reduce the overall decay and hence maintenance costs in the road network.

A solution to traffic flow optimization should consider the different optimization objectives in the traffic domain and should be able to deal with the real-life incidents by self-adaptive collaborative planning from the traffic agents—drivers and, in the future, self-driving vehicles. In particular, it should allow the drivers to collectively plan towards a solution of overall benefit and should tailor the collective planning process to the specifics of the situation at hand (e.g. high traffic that necessitates increased and more regular collaboration between the agents).

To this end, in this paper we provide a self-adaptive framework that can be used for research in this challenging domain. Traffic Reconfigurations via Adaptive Participatory Planning (TRAPP) combines scalable decentralized planning with a self-adaptation loop that oversees and tailors the decentralized planning process to the specifics of the situation at hand. Technically, it provides an interoperation of SUMO and EPOS, two existing frameworks. The former is a state-of-the-art simulation environment for traffic dynamics and therefore

domain-specific. The latter is a recent advance in decentralized combinatorial optimization problems for multi-agents systems. Their integration comes with three extensible and customizable adaptation strategies.

Succinctly, the contributions of this paper are: (i) introduction of a novel open-source self-adaptive framework for traffic flow optimization; (ii) showcasing the applicability of EPOS in traffic optimization; and (iii) supporting experimention with decentralized participatory optimization in SUMO.

The rest of the paper is structured as follows. Section 2 lists the requirements for a self-adaptive framework for traffic optimization, Section 3 gives an overview of the framework while Section 4 explains the way SUMO and EPOS interoperate in the framework. Then, Section 5 presents the three proposed self-adaptation strategies and Section VI details our initial experience with the framework. Finally, Section VII compares our work to related work and Section VIII concludes.

II. System Requirements

A system that supports research in self-adaptive traffic optimization should satisfy the following main requirements:

1) **Realistic traffic simulation**. The system should support the realistic simulation of traffic that captures the domain dynamics, including stop-and-go effects, vehicles acceleration/deceleration, traffic light effects, traffic jams, etc.
2) **Scalable traffic simulation**. The system should scale in the number of cars, drivers, and streets in order to be used with real-life scenarios.
3) **Local and global optimization objectives**. The system should be able to work with both global traffic optimization objectives (e.g. optimize for the average ride) and local ones (e.g. avoid certain streets and routes).
4) **Different self-adaptation strategies**. The system should support different strategies that can be applied at runtime in order to deal with different situations that warrant adaptation (e.g. traffic accidents, road blockages, increased traffic) and continue to satisfy its objectives.
5) **Extensibility and customizability**. The system should offer abstractions and modularity that make it easy for researchers to tailor it to their needs, e.g. by creating new self-adaptation strategies.
6) **Ease of use**. The system should offer an easy way for specifying traffic scenarios and self-adaptation strategies by externalizing its configuration. It should also provide feedback on the result of a self-adaptation action via logging and visualization capabilities.

In the following, we present our framework that integrates two existing tools, SUMO for traffic simulation and EPOS for decentralized planning, in order to meet these requirements.

III. Overview of the Framework

An overview of TRAPP is depicted in Figure 1. As typical in self-adaptation approaches, TRAPP comprises a managed and a managing subsystem. The managed subsystem contains the traffic simulator SUMO (domain-specific) and the decentralized optimization tool EPOS (domain-independent). In a nutshell, SUMO provides EPOS with a list of possible plans (routes to follow) for each agent (vehicle in our simulation).[1] EPOS outputs the selected plan for each agent; these are picked up and executed by SUMO to progress the simulation. The above process repeats periodically.

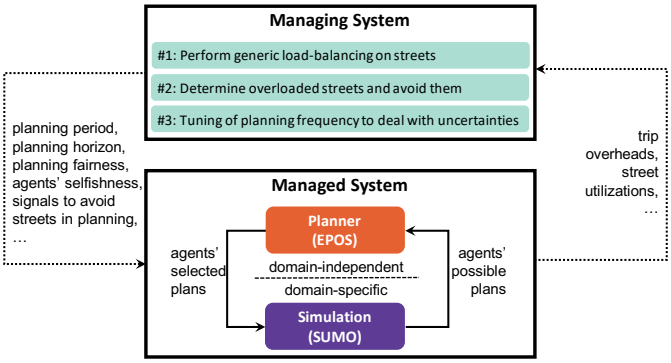

Fig. 1. Overview of the TRAPP framework.

At the outer loop, the managing system monitors data related to trip overheads and street utilizations. A trip overhead is the ratio of the actual duration for a trip that takes place in the simulation to the theoretical duration of a trip when physical dynamics and the traffic situation are ignored and the car travels with the maximum speed allowed at each street. A street utilization is the percentage of the street length that is occupied by cars. Trip overheads are obtained at the end of every trip, while street utilizations on every simulation tick.

The managing system runs periodical adaptation cycles, which, in correspondence to the MAPE-K loop [2], *monitor* data, *analyze* them for detecting traffic problems or irregularities, *plan* corresponding actions to tune the way participatory planning takes place, and finally *execute* the adaptation actions by configuring EPOS accordingly. The next times EPOS is invoked, it uses the updated configuration. What can be changed at runtime via an adaptation strategy (tunable runtime parameters) is described in detail in Section IV-C.

As a proof of concept, this paper presents three concrete adaptation strategies, namely with (1) performing load balancing in the streets in order to reduce overall trip overheads and street utilizations, (2) avoiding overutilized streets, (3) tuning the planning frequency and horizon as a response to traffic events that necessitate more frequent and accurate planning. We detail on these three self-adaptation strategies and on how to create different strategies in Section V.

A. User's Perspective

The user of the framework needs to set a number of parameters, conveniently gathered in a single configuration file (`app.Config.py`).

[1] We are currently supporting three different plans per agent, each generated with a different router. However, the list of plans per agent can be extended by implementing other routers.

In particular, the user needs to select the values for some general parameters related to whether to use the graphical user interface (GUI) of SUMO, whether to print debug information, and which random seed to use. Also, the user has to specify the location of the EPOS jar. With regard to simulation-related parameters, the user needs to specify the path to a SUMO configuration file and a SUMO network file, along with the number of cars and the total simulation time. With regard to adaptation-related parameters, the user needs to specify whether the simulation should start with an EPOS invocation, along with the adaptation period, i.e. how often the managing system should be invoked. Finally, the user needs to select initial values for all the runtime parameters related to planning, described in detail in Section IV-C.

With respect to the self-adaptation strategies, the user can either choose one of the three available strategies or provide their own strategies by providing a Python class with one method for each of the Monitor, Analyze, Plan, and Execute phases (following the MAPE-K model) and integrate them to the framework as explained in Section V.

Once a simulation starts, the user can observe the simulation via the SUMO GUI and the information regarding EPOS self-adaptation loop execution in the console. Once a simulation ends, they can inspect the graphs that may have been generated in support of analyzing the self-adaptation strategy of choice. We note that the data visualization and analysis part is highly specific to the self-adaptation strategy, optimization objective, and evaluation strategy followed. The user can always create custom graphs that analyze the results of a run—we provide a jupyter notebook as a starting point.

IV. Technical Integration

In this section, we first give an overview of the two main tools that TRAPP brings together, SUMO and EPOS, and then delve into the details of the integration we performed.

A. SUMO and TraCI

Simulation of Urban Mobility (SUMO) is the most comprehensive open-source microscopic traffic simulator [3]. It has been successfully used in simulating scenarios that span mid-sized European cities and hundrends of thousands of vehicles [4], [5]. SUMO simulations are realistic since they involve different maps (provided as input to the simulation), multiple lanes per street, acceleration and deceleration of cars in traffic lights and intersections, different car driving styles, speed limits per street, and other features. It also offers a Graphical User Interface (GUI) for observing the simulation (Figure 2).

Although SUMO is written in C++, TraCI (Traffic Control Interface) is a Python interface to SUMO that can be used for controlling a simulation through Python. TraCI offers a convenient way of both monitoring different aspects of the simulation (e.g., number of cars on each street, vehicle speeds) and enacting changes to a running simulation. For example, TraCI can be used for dynamically changing the route of a vehicle or blocking a lane or street as the simulation runs.

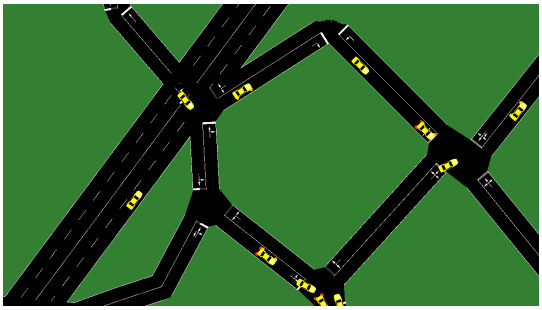

Fig. 2. Graphical User Interface of SUMO.

Similar to our previous work [6], we have created a Python wrapper over TraCI that allows us to start a SUMO simulation with a prescribed number of vehicles on a provided map and continuously route cars to their destinations using different Python-coded routers. Both the initial positions and the destinations of all cars are randomly chosen. When a car reaches its destination, it randomly picks another destination.

A car can choose between one of three available routers to obtain a route. The first router routes cars based on the minimum distance to the destination, the second by selecting the streets with maximum speed, and the third by considering both the maximum speed and the street length in routing. Unless otherwise specified, cars pick one of the routers at random when looking for a new route. The goal of using participatory planning in this setting is for each car to select a route among the ones generated from the different routers that satisfies to the degree possible both local and global objectives.

B. EPOS

EPOS, the *Economic Planning and Optimized Selections* [7] is a Java-based general-purpose decentralized multi-agent system that efficiently solves complex multi-objective combinatorial problems via a collective participatory learning approach. The agents in EPOS self-organize in hierarchical structures, i.e. tree topologies, over which they can perform efficient aggregation and decision-making in an iterative fashion: consecutive child-parent interactions in the bottom-up phase, followed by parent-child interactions in the top-down phase.

In this paper an agent is a smart vehicle, i.e. a self-driving car, that autonomously generates a set of finite number of possible plans that represent the scheduling or allocation of resources for consumption or production. In this paper a possible plan represents the utilization of the traffic links, i.e. streets, that this vehicle is expected to use over a future horizon. In other words, a possible plan encodes a followed route and how much of the traffic network capacity this route utilizes. Figure 3 illustrates how plans are modeled so that EPOS can interoperate with SUMO. For each agent, multiple plans are generated corresponding to alternative routes for a vehicle to reach its destination. These alternative plans provide the necessary flexibility to each vehicle to contribute to a traffic equilibrium that is beneficial system-wide. Each plan is associated with a cost that represents the preferences of the driver

or passengers. For instance, cost can measure the distance of each route, the average speed, the fuel consumption, or the safety of the route.

EPOS is designed to perform collective participatory learning such that each agent selects in a fully decentralized fashion a plan that satisfies multiple complex objectives, i.e. minimization of quadratic cost functions that require agents' coordination and cannot be minimized locally. In this paper, EPOS has the capability to minimize the variance of the utilization over the different road segments as the means to (i) potentially limit traffic congestion, (ii) limit the concentration of traffic and noise pollution as well as (iii) limit the decay and damages of the road network. Such collective objective may oppose the drivers' or passengers' objectives. Selecting the plan with the lowest cost can result in the tragedy of the commons, i.e. selfish agents choosing the shortest route may experience high traffic congestion due to imbalanced utilization of the network. Finally, fairness objectives can also be met in order to preserve an equality in terms of costs payed for the selected plans from the different agents. EPOS is designed to self-regulate all these orthogonal objectives based on local agents' parameters that govern trade-offs.

C. Integration

The plans and their associated costs for each agent are generated by SUMO, followed by a decentralized plan selection in EPOS, and finally the execution of the selected plans within SUMO (inner loop in Figure 1). For now, no agents are modeled as disobedient, i.e. agents always accept the plans of EPOS. The execution of the plans results in a new, eventually optimized state of the traffic flow, given the status of the traffic. At the next planning period the above workflow is repeated.

Independently, a self-adaptation loop (outer loop in Figure 1) is responsible for adapting the way decentralized planning is performed by EPOS to deal with runtime changes. The orchestration of this self-adaptation logic between EPOS and SUMO is performed by exposing and contextualizing a number of application-independent EPOS parameters that are controlled during the runtime of the framework. These parameters are outlined as follows:

- `planning_period`: It determines how often the EPOS planning process will be invoked in the simulation.
- `planning_steps`: It determines the time resolution of an EPOS planning process and corresponds to the time blocks of Figure 3.
- `planning_step_horizon`: It determines the time duration of a planning step. Therefore the total planning horizon is calculated as `planning_step_horizon` $*$ `planning_steps`.
- `alpha`: It determines how fair the selected plans of EPOS are in terms of how equal their costs are among the agents' population. The parameter receives value in the range $[0, 1]$, where 0 represents no fairness optimization and 1 the highest priority in fairness optimization.
- `beta`: It determines the average discomfort that the selected plans of EPOS cause to the agents. Discomfort may refer to an undesired effect that a plan causes to the driver of the vehicle, for instance, longer traveling times or routes with tolls. The parameter receives value in the range $[0, 1]$, where 0 holds for no comfort optimization and 1 holds for the highest priority in comfort optimization, i.e., the plan with the lowest cost is always selected for each agent (which then acts as a *selfish* agent).
- `globalCostFunction`: It determines the objective of the decentralized combinatorial optimization. Two types of objectives are supported: (i) a balancing objective, i.e. minimization of the variance (`VAR`) as the means to load-balance the utilization in the traffic network over time, as well as (ii) three matching objectives, i.e. minimization of cross correlation (`XCORR`), residuals of sum squares (`RSS`) and root mean square error (`RMSE`). The matching objectives require an input target plan for the matching. This signal encodes the steering scenario, for instance, avoiding certain traffic links in which an accident happened.

V. Self-Adaptation Strategies

To illustrate the use of the TRAPP framework, we detail here three adaptation strategies we implemented and provide a cookbook of implementing different ones.

A. Generic Load-balancing Strategy

The goal of this strategy is to balance the cars in the traffic network in order to potentially decrease trip times and limit traffic congestion.

Monitor. The overheads of trips in the last adaptation period are monitored.

Analysis. The median of the monitored trip overheads is calculated. If it is higher than a prescribed threshold (e.g. 2), the planning phase is triggered, otherwise the strategy aborts.

Planning. A value for the `beta` parameter of EPOS, corresponding to agents' preferences for the route comfort, is selected that is lower than the current one. This allows for balancing of cars in streets to take place even if this causes discomfort to certain agents. The strategy may consider results from previous invocations in order to set the appropriate value. Such historical information can be stored in the `Knowledge` of TRAPP to be used across self-adaptation invocations.

Execution. `globalCostFunction` is set to `VAR` and `beta` is set to the value selected in the previous phase.

B. Avoiding Overloaded Streets Strategy

Here, the goal is to determine which street or streets are overloaded and steer traffic to avoid using these streets. This might also correspond to the real-life case of street closures due to accidents or extreme weather conditions.

Monitor. The utilization of all streets over the last adaptation period is monitored. By default, TRAPP provides the highest possible resolution in monitoring this, i.e. one utilization value for each street for each simulation tick.

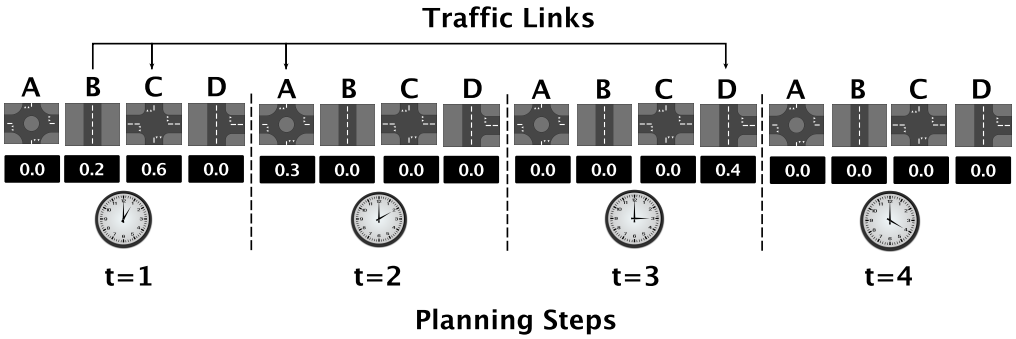

Fig. 3. The traffic plan of a vehicle is determined by a sequence of real values in the range $[0, 1]$ representing the traffic link utilization created during a specific time interval by the vehicle. The sequence is split in blocks representing planning steps. Each block contains the utilization values for each traffic link of the network. In this example, the vehicle follows a route that consist of the traffic links B-C-A-D.

Analysis. The average utilization of each street is calculated. If the average utilization of a street is higher than a prescribed threshold (e.g. 80%), the street is marked as overloaded.

Planning. A target plan signal is created to steer vehicles towards avoiding the overloaded streets. Such a signal in EPOS is a sequence of `0`'s or `1`'s of the same length as an agent's plan, with `0` indicating that a street should be avoided. We provide convenience methods for creating such signals.

Execution. `globalCostFunction` is set to `XCORR`, `RSS`, or `RMSE` and the signal sequence is written to a file to be used by EPOS in its next invocation. In the experimental evaluation of this paper, the `XCORR` value has been used[2].

C. Tuning Planning Frequency Strategy

The goal of this strategy is to tune the planning frequency and horizon in order for planning to better deal with the uncertainties of the environment. Since planning itself comes with a communication cost, agents should replan only as frequently as needed and not more often. However, in case of a major traffic event such as a holiday or a popular match, planning should be invoked more frequently to better control the traffic flows.

Monitor. Either major events are directly monitored or the street utilizations predicted in the latest EPOS invocation are obtained in order to be compared to the actual, monitored street utilizations since that invocation.

Analysis. For each street, the difference between its average utilization and its predicted utilization by EPOS are compared. Then, the average of these differences is computed, which stands as a metric of how accurate planning has been in the last period. If this metric is higher that a prescribed threshold, the planning phase is triggered.

Planning. A planning period is selected that is lower than the current one. Also, the planning step horizon and the number of planning steps may be reduced to increase the accuracy of planning. Note that the total planning horizon can even be lower than the planning period.

Execution. The new values for the EPOS parameters of `planning_period`, `planning_steps`, and `planning_step_horizon` are set.

D. Creating and Registering a Custom Strategy

To create a custom strategy, the user needs to create a new Python class that inherits from the `Strategy` class and implement the four MAPE-K phases as methods. The user has access to convenience methods that monitor trip overhead and street utilization over the last adaptation period (`Util` class). Once a new strategy is implemented, it needs to be "registered" to the framework (by changing a single file), and then it can be selected in `app.Config.py` and used in a self-adaptation loop that goes through all the prescribed MAPE-K phases.

VI. Experience

We report now on our so far experience with experimenting with the generic load balancing strategy (Section V-A). We have been experimenting by deploying 600 cars in the city of Eichstätt with 1131 streets. The main question is how changes in the agents discomfort (modeled by the `beta` parameter of EPOS) can lead to overall more efficient traffic flows it terms of a balanced utilization of streets. To investigate this, we first consider the baseline where agents are selfish (`beta`=1) and always choose their most preferred plan. We then consider a load-balancing strategy that assumes that agents have intrinsic interests to preserve a balanced utilization of the traffic network (`beta`=0). Finally, we consider a load-balancing strategy in which only a small fraction of the agents contribute to the public good (`beta`=0.9).

After setting the threshold for the median of trip overheads to 1.5 and the other parameters to the values shown in Table I, each of the three simulation scenarios ran for 1000 ticks. We observe that an adaptation action is already taken in the first run of the adaptation loop (tick=100), due to the low threshold. The generic load balancing strategy set the `globalCostFunction` to `VAR` and updated the `beta` parameter to 1, 0, and 0.9, respectively.

[2]Note that setting `globalCostFunction` to `VAR` does not require any target plan signal. The choice between `XCORR`, `RSS`, or `RMSE` depends on the scalarization of the plan values and costs and a choice is usually made empirically [8].

TABLE I
EXPERIMENTS WITH PLANNING_PERIOD 100, ADAPTATION_PERIOD 100, PLANNING_STEP_HORIZON 50, PLANNING_STEPS 2, ALPHA 0.

Beta	Median of Trip Overhead	Variance of Street Utilization
1.0	1.86688881491 (**baseline**)	0.00421584184249 (**baseline**)
0.0	1.86697727145 (**+ 0.005%**)	0.00367241577052 (**– 12.890%**)
0.9	1.85167395638 (**– 0.815%**)	0.00408485498667 (**– 3.107%**)

The results, depicted in Table I, indicate that setting `beta` to 0 decreased the variance of street utilization (the global objective of EPOS) with a low sacrifice of 0.005% higher trip overheads. Setting `beta` to 0.9 yielded positive results in both metrics. A thorough analysis of Pareto optimal `alpha` and `beta` values based on TRAPP is part of future work.

VII. COMPARISON TO RELATED WORK

Modeling and simulation software tools for domain-dependent infrastructural networks and complex techno-socio-economic systems are subject of active research: The pervasiveness of the Internet of Things challenges their controlability. Modeling and mitigation of energy blackouts and traffic congestions become more complex. A high degree of shared interconnections and network interdependencies perplex further their dynamics and understanding [9]: Cascading power failures triggered by individual component failures can influence other energized networks, e.g. communication networks and water networks with energized valves. The interoperation and interaction of domain-specific toolkits such as SUMO with domain-independent models and tools, i.e. multi-agent platforms, optimization engines, machine learning frameworks are the means to cope with such challenges by decreasing development costs, simplifying interdisciplinary team collaboration as well as increasing the resusability, modularity and applicability of various general-purpose community artifacts.

Such self-adaptive software systems have been studied in related work. For instance, the multi-agent systems framework of JADE is earlier used for modeling the driver by providing a *strategic layer* for route choices and adaptive learning behavior, i.e. Q-learning [10]. JADE interoperates with SUMO that provides the tactic layer of the simulation: accelerating and breaking actions, lane-changing behavior and other events. Such interoperation is achieved via the Traffic Simulation Manager Application Programming Interface (TraSMAPI) and the Traffic Control Interface (TraCI). In contrast, this paper introduces a data-driven versatile modeling and interoperation between SUMO and EPOS with higher flexibility on determining multiple strategic and tactic layers via the modeling of possible plans and their cost. This can be crucial as Q-learning may not provide fast learning convergence, i.e. it does not have explicit access to all information residing in SUMO.

Similarly, an agent-based integration approach between MATSim [11] and a Belief, Desire and Intention (BDI) system is proposed for extensive modeling of agents' decision making as well as reactivity to the environment [12]. Self-adaptation focuses on modeling agents with predetermined static plans, while other agents are intelligent and reactive to the environment, as such the scenarios of bushfire evacuation or taxis can be effectively modeled. In contrast to this paper, the configurable mechanisms for collective intelligence in such heterogeneous scenarios are not addressed. In other words, decoupling the agents' individual intelligence from the physical traffic simulation is not adequate by itself to solve practical problems that arise when different agents interact in a dynamic environment. Invocation to collective learning capabilities can further enhance self-adaptation.

Traffic models usually focus on a specific aspect or objective, for instance, travel demand and reduction of trips overhead, load-balanced traffic networks and flows, and reduction of carbon emissions [13]. Nevertheless, such traffic phenomena require integrated modeling and analysis, whose complexity often results in low interoperability and ad-hoc solutions. Centralized transport simulation frameworks such as POLARIS [13] address requirements for higher abstraction, modularity and interoperability, however, usability requires continuous maintenance, wrapping of new domain-specific models, and integration of legacy software. In contrast, decentralized multi-agent systems such as EPOS can model multiple objectives and agents' behavior by design without a requirement for domain knowledge. Interoperation with such systems simplifies the integrated analysis and decreases the development effort for domain-specific models and simulations.

Multimodal simulation scenarios for SUMO are the focus of the SCRUM methodology [14], while other approaches focus on the design of domain-specific modeling languages [15] or the integration of traffic demand data derived from different contexts [16]. The SFINA framework integrates different domain-specific backends, including MATSim, to higher-level interdependent flow network simulations, e.g. mitigation of cascading failures in power-to-power or power-to-water interdependent networks [9], [17]. None of these methodologies addresses self-adaptation via decentralized optimization and learning of multi-agent systems.

VIII. CONCLUSION

In this paper, we have presented a research framework – TRAPP – that can be used for experimenting with self-adaptive participatory planning in the traffic domain. We believe that it can be both extended and configured to different research needs. We intend to use it in investigating novel strategies for traffic agents that cooperate in an adaptive way in optimizing traffic flows in cities. We aspire to make TRAPP useful to other researchers in the self-adaptive systems community.

IX. ARTIFACT AVAILABILITY

The source code of the artifact, along with an installation, getting started guide and a pre-installed Virtual Machine are available at https://github.com/iliasger/TRAPP.

ACKNOWLEDGMENT

This work is part of the Virtual Mobility World (ViM) project and has been funded by the Bavarian Ministry of Economic Affairs, Regional Development and Energy (StMWi) through the Centre Digitisation.Bavaria, an initiative of the Bavarian State Government.

REFERENCES

[1] D. Helbing and B. Tilch, "Generalized force model of traffic dynamics," *Physical review E*, vol. 58, no. 1, p. 133, 1998.

[2] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[3] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using sumo," in *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. [Online]. Available: https://elib.dlr.de/124092/

[4] L. Codecá, R. Frank, S. Faye, and T. Engel, "Luxembourg SUMO Traffic (LuST) Scenario: Traffic Demand Evaluation," *IEEE Intelligent Transportation Systems Magazine*, vol. 9, no. 2, pp. 52–63, 2017.

[5] L. Codec and J. Hrri, "Towards multimodal mobility simulation of c-its: The monaco sumo traffic scenario," in *2017 IEEE Vehicular Networking Conference (VNC)*, Nov 2017, pp. 97–100.

[6] S. Schmid, I. Gerostathopoulos, C. Prehofer, and T. Bures, "Self-Adaptation Based on Big Data Analytics: A Model Problem and Tool," in *Proc. of SEAMS 2017*. IEEE: IEEE, May 2017, pp. 102–108.

[7] E. Pournaras, P. Pilgerstorfer, and T. Asikis, "Decentralized collective learning for self-managed sharing economies," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 13, no. 2, p. 10, 2018.

[8] E. Pournaras, M. Yao, and D. Helbing, "Self-regulating supply-demand systems," *Future Generation Computer Systems*, 2017.

[9] E. Pournaras, M. Ballandies, D. Acharya, M. Thapa, and B.-E. Brandt, "Prototyping self-managed interdependent networks–self-healing synergies against cascading failures," in *13th International Symposium on Software Engineering for Adaptive and Self-managing Systems (SEAMS 2018)*, 2018.

[10] G. Soares, Z. Kokkinogenis, J. L. Macedo, and R. J. Rossetti, "Agent-based traffic simulation using sumo and jade: an integrated platform for artificial transportation systems," in *Simulation of Urban MObility User Conference*. Springer, 2013, pp. 44–61.

[11] A. Horni, K. Nagel, and K. W. Axhausen, *The multi-agent transport simulation MATSim*. Ubiquity Press London:, 2016.

[12] L. Padgham, K. Nagel, D. Singh, and Q. Chen, "Integrating bdi agents into a matsim simulation," in *Proceedings of the Twenty-first European Conference on Artificial Intelligence*. IOS Press, 2014, pp. 681–686.

[13] V. Sokolov, J. Auld, and M. Hope, "A flexible framework for developing integrated models of transportation systems using an agent-based approach," *Procedia Computer Science*, vol. 10, pp. 854–859, 2012.

[14] A. F. Acosta, J. E. Espinosa, and J. Espinosa, "Application of the scrum software methodology for extending simulation of urban mobility (sumo) tools," in *Simulating Urban Traffic Scenarios*. Springer, 2019, pp. 3–15.

[15] A. Fernández-Isabel and R. Fuentes-Fernández, "Developing an integrative modelling language for enhancing road traffic simulations." in *FedCSIS*, 2015, pp. 1745–1756.

[16] D. Ziemke, K. Nagel, and C. Bhat, "Integrating cemdap and matsim to increase the transferability of transport demand models," *Transportation Research Record: Journal of the Transportation Research Board*, no. 2493, pp. 117–125, 2015.

[17] E. Pournaras, B.-E. Brandt, M. Thapa, D. Acharya, J. Espejo-Uribe, M. Ballandies, and D. Helbing, "Sfina-simulation framework for intelligent network adaptations," *Simulation Modelling Practice and Theory*, vol. 72, pp. 34–50, 2017.