TraSMAPy: a high-level SUMO API in Python

Ana Barros, João Costa, and João Sousa,
Faculty of Engineering of the University of Porto

In the last few years, the growth of urban traffic volume has become a global problem causing the most significant impact in modern cities. Additionally, trying new approaches and solutions implies interventions on the road, which causes traffic congestion and might cause other problems. In other words, testing in the real world is too expensive and time-consuming to be considered feasible. Fortunately, mobility modelling and traffic simulators provide the technology necessary to tackle these problems. These tools allow the creation of digital twins of urban areas to develop and predict the performance of new mobility solutions like autonomous driving.

One of the most popular and powerful tools for traffic simulation is SUMO, an open-source continuous traffic simulation package designed to handle large networks. Despite its popularity, SUMO can be quite complex, making it hard to reach its full potential. Furthermore, accessing its complete feature set requires programming skills that traffic engineers might lack, as programming might not be their primary area of expertise.

This paper presents TraSMAPy, a Traffic Simulation Management Python API for SUMO, whose goal is to make SUMO accessible to anyone. This API provides real-time interaction with SUMO, a query language for statistics, and a foundation to develop multi-agent system (MAS) solutions. The API is a Python package that interacts with the SUMO simulator. It provides complete, easy-to-read documentation and only requires basic Python programming skills. The MAS approach also offers the appropriate abstraction for the traffic management domain.

***Index Terms*—SUMO, Modelling, Simulation, Traffic management, TraCI, MAS, TraSMAPy**

I. Introduction

FOR years, modern cities have faced a problem related to population growth: as mobility becomes increasingly more accessible, citizens opt to travel long distances daily, e.g., to attend their jobs. This new level of transportation of goods and people turned road networks into a critical part of city planning. Well-planned road networks are essential for large population areas, as they directly impact the city's economy and environment and the well-being of its citizens. Road network management needed to adapt faster to sustain the rapid increase in urban traffic. This circumstance spawns several problems that negatively impact the city and its citizens.

A. Problem Statement

Road networks are an example of systems that are not available for experimentation since any experiment can be too costly, too time-consuming, or cause significant complications. These handicaps call for the usage of simulators.

Simulators can describe current real-world scenarios, simulate *what-if* situations on those scenarios, add new features not yet present in the real world, and predict how a system will evolve into the future. Coupled with these features, simulators can explore these situations/hypotheses without affecting the real world. All these are desirable for the research surrounding the topic of road networks and traffic planning. One of the most popular simulators for research on this topic is the SUMO[1] traffic simulator.

SUMO is an open-source microscopic and continuous traffic simulation package for large networks. It allows intermodal simulation and provides tools for creating diverse scenarios. Despite its popularity, SUMO is far from perfect. In particular, the most common problem is its accessibility.

SUMO is a powerful and complex tool. Accessing SUMO's full potential requires programming skills for using TraCI[2], SUMO's API allowing for real-time manipulation of the simulation. Users without programming skills feel this complexity aggravated. Traffic engineers usually fit this demographic. It is necessary to lower the barrier of entry to access SUMO's features.

B. Introducing TraSMAPy

This document presents TraSMAPy[3], a high-level Python API for SUMO. It provides real-time interaction with SUMO, a query language for statistics, and a foundation for developing multi-agent system (MAS) solutions.

The purpose of this API is to be easy to pick up for non-developers and feature-complete for experienced users. Its primary design philosophy is the extensive use of proxy objects to represent the concepts in the simulation, e.g., vehicles and roads.

The collection of statistics is essential to evaluate the results of any simulation. TraSMAPy provides a statistics module with which users can query for anything in the simulation. Users can also register queries that passively collect and aggregate information so that they can use them later.

C. Document structure

This paper starts by reviewing the existing related work concerning SUMO interfaces, followed by an in-depth look on the developed API. Afterwards, the authors present a simulation example using the API. The authors end the document with notes/suggestions about future works and a conclusion about the work developed.

[1]SUMO website: https://www.eclipse.org/sumo

[2]https://sumo.dlr.de/docs/TraCI.html

[3]https://github.com/JoaoCostaIFG/TraSMAPy

II. Literature review

The current market does not provide a solution for traffic engineers to implement complex simulations without having a programming background/programming experience. In this section, the authors try to identify the closest answers to the gap identified in the market.

In the following sub-sections, the authors discuss the different technologies/products to interact with SUMO, focusing on their features and weakness. For each of these technologies/products, the authors decide if they are appropriate to tackle the problem at hand and, if they are not, why they fail to fill the gap in the market.

A. TraCI

TraCI is the official API client for SUMO. It is the main way to interact with a running SUMO simulation. Upon initializing, SUMO starts acting as a server listening to TCP connections and requests. TraCI is a SUMO client library with bindings for multiple programming languages. The library simplifies the usage of the numerous requests supported by SUMO.

TraCI is a purely functional, stateless API. As such, SUMO keeps the whole simulation state. A string ID uniquely identifies all entities in SUMO. TraCI provides a function for each property that can be set and retrieved for the concepts in the simulation.

TraCI is a good tool, but it has problems that make it unsuitable for general usage:

- It is not feature-complete at the time of writing: it is missing support for multiple SUMO concepts. For example, although it is possible to inspect the existing simulation routes and add new ones through TraCI, it is impossible to do the same with trips.
- TraCI is a low-level API that only provides access to properties of entities and concepts in the simulation. As such, it does not offer abstractions to make it easier to create more complex simulations, for example, road tolls.
- TraCI uses the network stack to make calls, which adds considerable communication overhead when making numerous calls.

Given that TraCI is the official/main way of interacting with SUMO, it is a good candidate on which to base TraSMAPy's implementation.

B. Libsumo

Libsumo[4] is an alternative to TraCI. At the time of writing, it is still in development and not complete. It aims to be a more efficient drop-in replacement to TraCI. Libsumo is a C++ library (with bindings to other programming languages) with function signatures similar to TraCI's, but it does not use the network stack to interact with SUMO. Essentially, it provides the same features as TraCI while eliminating the need for the network to communicate with SUMO by interacting with the simulator directly.

Although Libsumo solves the efficiency problem of TraCI, it brings its own set of extra drawbacks. For the use case targeted by the authors, the flaws are mostly related to cross-platform support. At the time of writing, Libsumo is in an experimental state for all platforms but with additional limitations on Windows, e.g., it does not support running SUMO's GUI on Windows. Given that this work aims to lower the barrier of entry to build simulations on SUMO, making the GUI not supported on the most used Operating System (OS) is already a deal-breaker. Furthermore, it is even more feature-incomplete than TraCI (an issue discussed in the previous chapter).

Fortunately, Libsumo's design allows it to be a drop-in replacement for TraCI. In most cases, moving from TraCI to Libsumo is as simple as changing the library import/include statement. Even though Libsumo is not currently a suitable replacement for TraCI for the use case in question, switching should be easy if it undergoes improvements in the future.

[4]https://sumo.dlr.de/docs/Libsumo.html

C. TraSMAPI

Traffic Simulation Management API (TraSMAPI)[1] is a MAS platform for interacting with microscopic traffic simulators. It collects statistics and provides a framework for developing MAS solutions. TraSMAPI's main objective is to shift the focus from the simulator and its details. This way, users focus on the creation and management of agents. Thus, the agents' implementation is independent of the simulators, allowing the same solution to be tested in various simulators.

The MAS approach in TraSMAPy allows the abstraction of complex domains like SUMO control systems in traffic management. Each agent in the simulation has a simple structure and behaviour, which contributes to more complex system behaviour. So, this approach makes it easier to create more advanced and efficient solutions while still working with simple concepts.

Although TraSMAPI succeeds in simplifying the comparison of different simulators, these abstractions can cause difficulties when working with one simulator alone. For instance, TraSMAPI contains the common concepts in microscopic traffic simulators, like vehicles, edges, and lanes. Nevertheless, it is missing a lot of concepts present in SUMO, so the user might need to program the behaviour of the extra agents.

Another problem is the dependence on TraSMAPI's community for new features. The API should deliver enough features instead of expecting its community to provide them. However, it could also provide a development mode for encouraging contributions and simplifying the process of inserting new features.

Furthermore, to make the API more accessible, TraSMAPI could be written in an easier programming language instead of Java. Thus, using the API would only require basic programming skills.

III. TraSMAPy

TraSMAPy is an API that uses TraCI to access and manipulate simulations on SUMO. The API uses the low-level features offered by TraCI to implement higher-level features and new concepts and make it easier to use. Figure 1 shows how the user interacts with TraSMAPy, which communicates

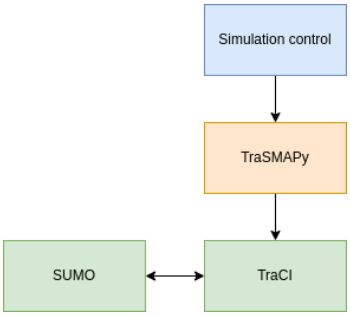

Fig. 1. TraSMAPy interaction with SUMO and TraCI.

with SUMO through TraCI. By using TraSMAPy, the user never has to deal with TraCI.

A. Relation with TraCI

As stated previously, TraSMAPy depends on TraCI. It is almost a wrapper for Traci. The most evident advantage of TraSMAPy relative to TraCI is the representation of concepts/entities of the simulation as objects. For example, a user can obtain an instance of a vehicle in the simulation and access its current speed and colour using the provided getter methods. This simple workflow directly shows the user all operations available on an object of that type and does not require keeping track of an ID.

By grouping TraCI's features, TraSMAPY offers simplified workflows for complex tasks. For example, TraSMAPy provides the ability to create public transport flows with configurable schedules and routes. This task is also possible in TraCI but requires the user to set timers for the schedules and manually create routes and vehicles. TraSMAPy makes these concepts easier to implement, thus making it faster to build a simulation.

An additional advantage of using objects is mainly related to short-lived simulation entities: entities that can exit the simulation during its life cycle, e.g., vehicles. When trying to access a vehicle with a given ID through TraCI, there is no check for whether that vehicle is still on the simulation. When that happens, TraCI throws an exception. However, the only way to know what was the problem that caused the exception is to examine the exception's message because all exceptions thrown by TraCI are the same type. In TraSMAPy, every problem has its exception type, making it easier to recover from these situations.

Unfortunately, because of the reliance on TraCI, TraSMAPy inherits some of its flaws, namely being feature-incomplete and the inefficiency of the network stack used for communication. The most evident consequence of TraCI being feature incomplete is the inability to access some of the entities that can be in a SUMO simulation. For example, container stops are the only stop type not supported by TraCI at the time of writing; if a simulation has a container stop in the network, it is impossible to see it on TraCI and, consequently, on TraSMAPy.

SUMO uses *XML* files to configure the simulation. Working with these configuration files is cumbersome, even for simple

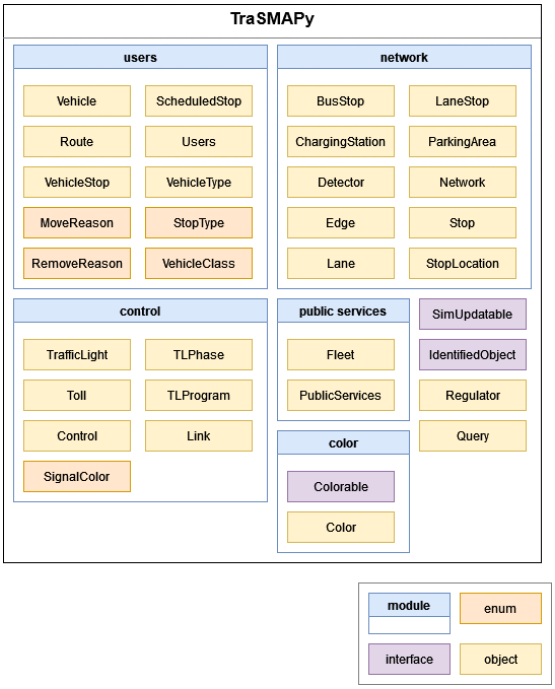

Fig. 2. TraSMAPy architecture.

simulations. When dealing with extensive simulations, for example, when simulating an entire city's traffic, these files can reach thousands of lines. SUMO comes bundled with tools that offer GUIs that generate these files and make it easier to edit them if needed, e.g., *netedit*. TraSMAPy reduces the need to deal with these files. With TraSMAPy, it is possible to create entities/concepts like routes and vehicles and perform actions like scheduling a stop. However, creating new edges and lanes is impossible since those are static to the simulation, and TraCI does not support creating all types of objects. This limitation means that we could not eliminate the dependence on *XML* configuration files.

B. Architecture

TraSMAPy's architecture consists of several modules aggregating groups of concepts. Some concepts consist of proxy objects for TraCI objects. Other concepts introduced by TraSMAPy are new concepts built using the primary/primitive ones. All these work as abstractions for manipulating and extracting information from simulation elements. Figure 2 shows a general view of the different modules present in TraSMAPy.

The *users* module contains the users of the simulation, the vehicles. On the other hand, the *network* module includes concepts related to the network, e.g., lanes, edges, and stops. Concepts related to traffic control, like traffic lights and variable speed signs, fit in the *control* module. Lastly, the *public services* model includes public transport and taxi services, e.g., bus fleets.

C. Statistics

Statistics are essential to evaluate a simulation's performance. TraSMAPy provides utilities that make it easy to query

the network and aggregate statistics. Two query mechanisms are available: Python functions and the Pyflwor[5] query language.

PyFlwor is a Python object query language forked from the original PyFlwor[6], inspired by OQL (Object Query Language), XPath, and XQuery. The queries' syntax is straightforward and allows access to any object in the namespace passed to the query. The main advantage of this query language over using Python functions is its simplicity. This simplicity is advantageous for users unfamiliar with Python or for queries that retrieve values without manipulating them, e.g., obtaining the current speed of all vehicles in the network. Although it is possible to build complex queries in PyFlwor, the authors believe most users will prefer using Python functions when making complex queries.

Users can register queries to aggregate the results of every simulation tick. This way, users can create queries before starting the simulation and then retrieve the results when the simulation is over. These registered queries are named so that users can identify their results on the produced data structure.

IV. A Simulation using TraSMAPy

This section describes the implementation of a simple SUMO example using TraSMAPy. The objective of this example is to build a simple simulation that relies as much as possible on the API, i.e., avoid using XML configuration files unless strictly necessary.

The example's code present in 1 represents a simulation consisting of a simple network with one edge with two lanes. One of the edges is reserved for public transport.

The first step is to open *netedit* to create a single edge with two lanes and export the resulting network into an XML file and the sumo configuration. The user must pass the name of the sumo configuration file to the TraSMAPy object.

To begin with, TraSMAPy makes it possible to access most simulation objects by their ID. In this example, the team uses the network module to obtain a lane object by its ID. The API also provides multiple methods to interact with and manipulate these objects. In this case, the lane method *setAllowed* restricts access to all vehicles that are not from the BUS vehicle class.

Furthermore, in SUMO, every vehicle has a vehicle type. Users usually define these in an XML file. Fortunately, TraSMAPy simplifies the creation of vehicle types using Python code. The example shows how to obtain a *VehicleType* object by its ID and duplicate it to obtain a new *VehicleType* that can be customized.

This new vehicle type inherits the properties of the bus vehicle class. TraSMAPy provides *enums* to assist with SUMO-specific identifiers that are difficult to remember. For example, the *VehicleClass enum* contains the *BUS* value to define a bus vehicle. In this example, the vehicle class of the new vehicle type is *BUS*, and the colour is pink, so it is easier to differentiate the vehicles of different types in the simulation.

Finally, the team creates vehicles using the *createVehicle* method from the *users* module. The while loop represents the simulation loop. In this case, the simulation will continue as long as there are any vehicles in the simulation. The simulation never ends because the spawn of new vehicles occurs inside the loop.

Additionally, to gather statistics about the simulation, the example uses TraSMAPy's query API and PyFlwor to collect information about the lanes' CO2 emissions. The query string starts with the network namespace, followed by the lanes object, and ends with the CO2Emissions property. The example registers this query and prints all collected statistics. The result includes the CO2 emissions for each lane in each tick.

```
traSMAPy = TraSMAPy("config.sumocfg")

# reserve lane to buses
lane = traSMAPy.network.getLane("E2_0")
lane.setAllowed([VehicleClass.BUS])

# create bus vehicle type
default_vtype = traSMAPy.users
  .getVehicleType("DEFAULT_VEHTYPE")
bus_vtype = default_vtype
  .duplicate("BUS_VEHTYPE")
bus_vtype.vehicleClass = VehicleClass.BUS
bus_vtype.color = Color(255, 0, 255)

bus_change = 0.25
i = 0

traSMAPy.users.createVehicle(
    f"b{i}", vehicleType=bus_vtype if
      random() <= bus_change else
      default_vtype
)

queryStr = "network/lanes/CO2Emissions"
traSMAPy.registerQuery("emissions", queryStr)

"""execute the simulation loop"""
while traSMAPy.minExpectedNumber > 0:
    i += 1
    traSMAPy.users.createVehicle(
        f"b{i}", vehicleType=bus_vtype
          if random() <= bus_change
          else default_vtype
    )

    traSMAPy.doSimulationStep()
    print(traSMAPy.collectedStatistics)

traSMAPy.closeSimulation()
```

Listing 1: Example scenario using TraSMAPy

V. Future Work

Although it is already possible to manage complex simulations with the state of TraSMAPy, many more concepts are still possible to implement. However, since TraSMAPy is restricted

[5]PyFlwor: https://github.com/JoaoCostaIFG/pyflwor
[6]Original PyFlwor: https://github.com/timtadh/pyflwor

to the set of TraCI features, it is challenging to create concepts that need features not provided by TraCI. With this, authors believe future work in this area can require extending TraCI.

As of now, there are still features of TraCI (getters/setters) not used in TraSMAPy. Future work could be concentrated on implementation and documenting the missing ones.

Still, the authors have identified concepts and abstractions that can be implemented with the current features offered by TraCI. One of these concepts is the *taxi vehicle*, using SUMO's devices, namely the *taxi device* to turn a vehicle into a taxi. The authors have not implemented taxis because they did not implement people entities in TraSMAPy. Another concept that can be implemented is that of variable speed signs using the ability of TraCI to manipulate the maximum allowed speed in edges and lanes at run time.

Usually, in MAS, norms are implicit as the agents unconsciously adopt them[2]. However, there was a need to come up with a different type of system. These systems are called normative MAS, where norms are explicit, and agents share these norms by communicating with each other or being informed by an authority. As for future work, TraSMAPy could support normative MAS by implementing the concept of a regulator. A Regulator would represent the authority that dictates the norms of the simulation, making it easier to control the agents' behaviour. For example, it could dictate the maximum velocity allowed by vehicles.

Finally, future work should include the maintainability aspect of the API. TraSMAPy depends on TraCI, which is still under development at the time of writing. This dependency means that function calls might change, causing deprecation warnings. TraCI can also add new concepts and features, increasing the set of concepts covered by the API. Thus, future work should include monitoring TraCI updates and modifying TraSMAPy accordingly.

VI. Conclusion

In the end, TraSMAPy ended up working better than existing tools, thanks to its use of proxy objects to represent concepts in the simulation and its ability to simplify complex tasks by grouping TraCI's features. For instance, TraSMAPy enables the creation of public transport flows with configurable schedules and automatic routing. Doing this on bare-bones TraCI would require the user to set schedule timers and manually create routes and vehicles. In addition, using proxy objects allows for easier access to simulation data and makes it easier to recover from errors.

Furthermore, TraSMAPy achieved all planned objectives and became a helpful tool for SUMO simulations. It fills its purpose to facilitate research and analysis of road network and traffic planning scenarios in SUMO. On top of that, it makes it easier for researchers to use the simulator and analyze the results of simulations by providing a high-level, easy-to-understand API. Finally, since TraSMAPy uses TraCI to interface with SUMO, it allows TraSMAPy to evolve with SUMO as new features arrive.

References

[1] I. J. P. M. Timóteo, M. R. Araújo, R. J. F. Rossetti, and E. C. Oliveira, "Using TraSMAPI for the assessment of multi-agent traffic management solutions," *Progress in Artificial Intelligence*, vol. 1, pp. 157–164, July 2012.

[2] A. Morris-Martin, M. De Vos, and J. Padget, "A norm emergence framework for normative mas – position paper," in *Coordination, Organizations, Institutions, Norms, and Ethics for Governance of Multi-Agent Systems XIII* (A. Aler Tubella, S. Cranefield, C. Frantz, F. Meneguzzi, and W. Vasconcelos, eds.), (Cham), pp. 156–174, Springer International Publishing, 2021.