Towards Discrete-Event, Aggregating, and Relational Control Interfaces for Traffic Simulation

Zhuoxiao Meng[*]
Anibal Siguenza-Torres[†]
Mingyue Gao
Margherita Grossi
Alexander Wieder
Xiaorui Du[‡]
Stefano Bortoli
Huawei Munich Research Center
Intelligent Cloud Technologies Laboratory
Munich, Germany
{firstname.lastname}@huawei.com

Christoph Sommer
TU Dresden
Faculty of Computer Science
Dresden, Germany
cms-labs.org/people/sommer

Alois Knoll
Technical University of Munich
Department of Informatics
Munich, Germany
knoll@mytum.de

ABSTRACT

The use of IoT and AI/ML to extract insights for Data-Driven Decision-Making (DDDM) in Intelligent Traffic Systems (ITS) is becoming increasingly popular. While simulation is a cost-effective and safe way to evaluate such approaches, existing simulators are often impractical due to inefficient control interfaces. In this work, we propose a *Discrete-Event, Aggregating, and Relational Control Interfaces* (DAR-CI) framework for achieving efficient traffic management simulations through a coupled approach. It enables a non-blocking interaction mode based on a discrete-event synchronization architecture. The overhead caused by data exchange is substantially reduced by supporting the direct retrieval of temporal metrics, data batch processing and customized in-situ aggregation. Combined with flexible, extendable, easy-to-understand, and implementation-friendly semantic specifications, we propose DAR-CI to serve as a universal tool for the traffic simulation community, taking the use and control of traffic simulation to a new level. A proof-of-concept study on the simulation of an adaptive traffic light control system demonstrates a 9.53X speedup compared to TraCI, a widely used protocol for controlling traffic simulators.

CCS CONCEPTS

• **Computing methodologies → Modeling and simulation**; **Interactive simulation**; *Simulation tools*; *Discrete-event simulation*; *Real-time simulation.*

[*] Also with Department of Informatics, Technical University of Munich.
[†] Also with Department of Informatics, Technical University of Munich.
[‡] Also with Department of Informatics, Technical University of Munich.

KEYWORDS

Traffic Simulation, Interactive Simulation, Traffic Control Interface (TraCI), Co-simulation

ACM Reference Format:
Zhuoxiao Meng, Anibal Siguenza-Torres, Mingyue Gao, Margherita Grossi, Alexander Wieder, Xiaorui Du, Stefano Bortoli, Christoph Sommer, and Alois Knoll. 2023. Towards Discrete-Event, Aggregating, and Relational Control Interfaces for Traffic Simulation. In *ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS '23), June 21–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3573900.3591116

1 INTRODUCTION

The Internet of Things (IoT) has facilitated the collection of data through physical devices, which can then be processed using technologies such as Artificial Intelligence (AI) and Machine Learning (ML) to extract valuable insights for Data-Driven Decision-Making (DDDM). In Intelligent Traffic Systems (ITS), this approach has gained popularity, enabling traffic engineers and researchers to utilize up-to-the-minute traffic data collected from road sensors, cameras, and floating vehicles to identify traffic patterns and bottlenecks. Real-time adjustments, such as optimizing traffic signals [2, 4, 16, 18], adjusting speed limits [15, 33], and rerouting traffic [13], can then be made, even automatically, to improve the efficiency and safety of transportation systems.

Prior to implementing these algorithms in real-world traffic management systems, thorough testing and evaluation must be conducted. Using simulations is a cost-effective and safe approach for this purpose. A mature simulator, under the control of external systems, can be used to execute the algorithms being tested within a closed-loop environment. However, today's simulators are often impractical for the task at hand due to their inefficient control interfaces. The challenges can be divided into two primary areas: the requirement for efficient synchronization, and the need for flexible and scalable data processing during runtime.

In terms of the synchronization model, state-of-the-art transportation simulators primarily use a conservative step-based approach, which needs synchronization at every interaction point.

However, in a DDDM scenario, the high frequency of interactions caused by data collection can result in an overwhelming amount of message flow, leading to significant overhead. Alternatively, an optimistic approach that allows for full asynchronicity and causality violations exists [9, 14], but implementing a rollback mechanism to address missing decision-making or data collection times is complex, error-prone, and not supported by most traffic simulators. Predicting when to roll back in a dynamic interactive simulation is difficult as well, and storing too many snapshots may result in high memory usage, which is impractical for large-scale scenarios.

Processing simulation data during runtime poses another challenge. The objective is to dynamically instruct the running simulation, allowing it to store and output the data of interest while avoiding any redundancy. However, due to the temporal and dynamic nature of simulation data, achieving accurate guidance can be challenging, which gives rise to the *targeted and online* simulation data extraction problem [25]. An essential yet challenging aspect of this problem is performing temporal operations and on-the-fly data-processing within queries. For example, filtering or computing data to retrieve results over a specific time interval.

Moreover, the existing query approaches often rely on an ID-based key-value approach without considering the relationships between data objects. For example, to retrieve the speed of cars on a specific road in a traffic simulation often involves two separate queries: first, querying the IDs of the cars on the road, and then sending another query to retrieve the speed of each car with the given IDs. A relational database can provide a structured representation of the relationships between data objects, which would result in more efficient querying. However, it is typically designed to manage static data which does not change frequently. In order to cope with the dynamic, volatile data generated by simulations in real time, additional treatments should be taken into account.

In this study, we propose DAR-CI (Discrete Aggregate and Relational Control interface) to address the two aforementioned challenges. First, we use a discrete-event-based approach to enable a non-blocking interaction with the running simulation. It enables the simulation to run asynchronously with the coupled controller when feasible, especially during data collection. To prevent the need for rollback, we incorporate events at critical time points, such as data collection and decision-making, to avoid missed opportunities. Our second contribution is the definition of generic control interfaces that enable users to adapt this framework to their specific needs. We introduce a query mechanism based on relational algebra logic which enables the extraction, selection, and aggregation of simulation data at runtime as needed. In addition, we provide a well-defined semantic specification, i.e., a language and platform-neutral protocol under a non-restrictive license[1] that standardizes communication between the traffic simulation and external applications.

The rest of this paper is organized as follows: Section 2 reviews related work, while Section 3 and 4 provide details on the system architecture and semantics of DAR-CI, respectively. We evaluate DAR-CI by comparing its performance to the state-of-the-art tool TraCI in a simulation of two practical use cases (Section 5). Finally, we conclude the paper in Section 6.

[1]DAR-CI protocol repository is hosted on Zenodo: https://doi.org/10.5281/zenodo.7842682

2 RELATED WORK

2.1 State-of-the-Art Controllable Traffic Simulators

This section presents the current state of controllable traffic simulation[2], noting that there is a high demand for interacting with and controlling a traffic simulation *during* runtime [28], but only a few established simulators offer built-in support for this feature. These simulators include VISSIM [8], AIMSUN [3], SUMO [17], and CityMoS [36]. VISSIM and AIMSUN provide API-based solutions that require controller code to be embedded in the same process space as the traffic simulator. This in-process architecture is not horizontally scalable and lacks language independence. In contrast, SUMO and CityMos use a client-server architecture with a public, standardized protocol specification known as Traffic Control Interface (TraCI) [35]. Due to its open source nature, TraCI has been widely adopted by researchers and practitioners in the traffic simulation community and beyond (e.g., network simulators [19, 26]).

However, all these simulators can only interact synchronously in the step-based approach (see Section 1), and lack flexibility and scalability in data processing. We illustrate this limitation using TraCI as an example. The controller first requests the simulation to advance to a specific time step (t_1) and waits. When the simulation has advanced to t_1, the controller is notified and then it treats the simulation snapshot as a static database, sending queries accordingly. After processing the queries, the controller requests the simulation to unpause and advance to the next time step of interest (t_2) and repeats the query process. However, between t_1 and t_2, the simulation is non-interoperable. This step-based interaction mode has some advantages, such as simplicity, but its disadvantages become apparent when applied to DDDM applications. In order to collect data over a period of time, the TraCI controller must pause the simulation at each time step to retrieve the simulation state, resulting in frequent communication between the controller and the simulation, and hence, major impact on performance.

From a semantic perspective, TraCI requests have a limited scope because they can only access one mobility primitive at a time, with the object of interest identified solely by its ID. For example, a moving vehicle may have multiple attributes describing its state, including its speed, location, direction, and route. However, only one attribute of an object can be retrieved or modified within each TraCI request. The scalability is thus limited, resulting in significant network overhead in large-scale simulations.

2.2 Approaches Towards Simulation Interoperability

Our research is relevant to the concept of interoperability, which involves the seamless exchange of information and collaboration among different systems or components. In this section, we investigate various techniques to attain interoperability in simulation

[2]Only microscopic simulators, which can provide details of vehicle dynamics, traffic lights, etc., are under consideration

systems, utilizing the three-layer model proposed by Tolk [29], which includes the technical layer, informational layer, and organizational layer.

The technical layer concerns the physical ability of the simulator to facilitate information exchange. First of all, it is essential to differentiate between synchronous and asynchronous communication. Synchronous communication involves the client and server waiting for the completion of any outstanding requests before issuing new requests. However, asynchronous communication does not require this kind of waiting. For our specific requirements, the use of asynchronous communication is more appropriate because the controller requests typically do not require an immediate response, e.g., acquisition of simulation data for a future time. Recent trends in this layer also include the use of web-based service-oriented architecture (SOA) [34], involving various communication technologies such as RESTful [1, 7], SOAP [31], and gRPC[3] [11].

The information layer requires a common understanding of the data being exchanged, as seen in examples such as Protocol Data Unit (PDU) in the Distributed Interactive Simulation (DIS) standard [6] utilized in the warfare domain, and the Federation Object Model (FOM) in the High-Level Architecture (HLA) standard [5]. While TraCI remains the exclusive protocol for traffic simulation, its binary format can present challenges in implementation and debugging. Instead of implementing protocols in programming languages on all target platforms, Interface Description Language (IDL) are used to define protocols once and generate code in various languages automatically, such as the use of Google's Protocol Buffers (protobuf)[4] in [21, 22].

The coordination and collaboration of inter-organizational processes is managed by the organizational layer. The DEVS specification [30] provides a formalism for coupling event-based simulations that is applicable to the majority of traffic simulations [32]. Within DEVS specification, simulations maintain a chronologically ordered list of events based on their execution time. Interoperability is achieved by dynamically manipulating the composition and ordering of events within this list, such as inserting or removing events.

Additionally, there exist general architecture frameworks such as HLA that offer a standardized approach for integrating multiple simulators through the use of the Runtime Infrastructure (RTI) middleware. This is particularly well suited for DEVS-based co-simulation [10] and is widely used in traffic simulation. MOSAIC [23], which follows the HLA concept but tailored specifically for traffic domain, enables various simulations to work together to simulate ITS. While our research aims to improve the efficiency of control interfaces in traffic simulation, especially for data collection, it does not seek to provide an alternative to HLA or MOSAIC. Instead, our efforts contribute to simplifying and optimizing the integration of traffic simulation into the standard co-simulation frameworks mentioned above.

2.3 Online Simulation Data Extraction

With respect to data extraction techniques for simulations, Schützel and Uhrmacher [25] consider simulation data from two dimensions:

[3]https://grpc.io/
[4]https://protobuf.dev/

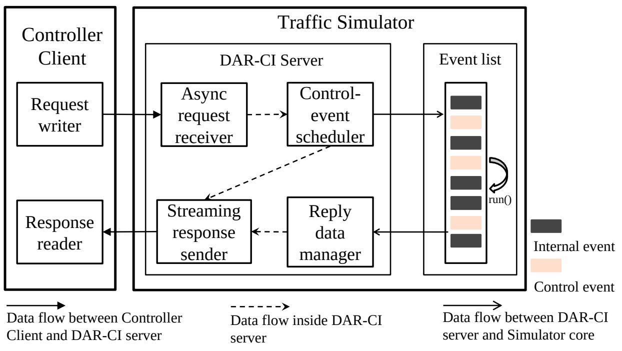


Figure 1: The DAR-CI system architecture allows for client-server interaction, where the client sends requests that are transformed into events and added to the simulator's updated event list.

structural and sequential (i.e., time). They emphasize that the data should be selectable, extractable, and aggregatable at runtime in both dimensions, but designing a query language that incorporates all of these functions is a challenging task. They have also proposed two different query languages in [24] and [12], respectively. However, neither of them can perform tasks such as time windowed aggregation.

Particularly interesting is the work of Zehe et al. [37], who use *SimuSQL*, a SQL-like query language, to reduce the cost of storing simulation data in the cloud. They proposed an optimization approach to track data status and transfer only updated data, thereby reducing I/O overhead. However, the query language still lacks support for temporal operations. The study also does not address synchronization, as online intervention in the simulation is beyond the scope of the research.

Another example is the HLA, where essential data is predetermined and sent to RTI during simulation. Nevertheless, this approach is inflexible when it comes to selecting data of interest and frequently leads to the transmission of redundant data. Also, transferring data to the middleware can result in additional communication and synchronization overhead.

3 SYSTEM ARCHITECTURE

Fig. 1 illustrates the proposed system architecture, which employs a client-server approach. Within the traffic simulator, a DAR-CI server is integrated to handle requests from external controller clients and to provide corresponding responses. Asynchronous operation is a key feature of the system for efficiency, but it also poses challenges due to synchronization issues. Ensuring that the system works as intended requires careful consideration and design. The four-step process, inspired by [27], outlines the steps that the simulation should follow when receiving a request, and it includes the following:

Admission: Verify the validity of the received request.

Operation: Determine the operations included in the request.

Sequencing: Establish when the operations should be performed.

Responding: Determine what and how the responses (e.g., simulation data) should be sent.

Moreover, it is important to have a **synchronization** mechanism in place to prevent the coupling approach from missing critical simulation time points. The following sections provide a comprehensive overview of DAR-CI's design and how it addresses the challenges for handling external requests in the asynchronous coupling approach.

3.1 Admission

The Event-based approach is widely used in traffic simulation, also including fixed time-step based traffic simulators. These simulators typically maintain a sorted event list that contains internal events representing state changes in the simulated system, such as vehicle arrivals, lane changes, and traffic signal changes, as illustrated in Fig. 1. The simulation process progresses by continuously executing events from this list in the order of their scheduled execution time.

To enable run-time interoperability with these simulations, we introduce external control events (also known as input events according to DEVS formalism [30]) in addition to the internal events. Each control event represents an atomic external operation that the simulation can allow, such as `VehicleStateRetrievalEvent` and `VehicleStateUpdateEvent`. By placing these control events in their proper positions (i.e., at the appropriate execution time) within the event list, we can manipulate the simulation's behavior.

The basic process of DAR-CI is as follows (see Fig. 1): a request receiver constantly listens for requests asynchronously to the execution of the simulation. Upon receiving a request, it is first parsed and converted into multiple control events based on its semantics. Each control event must have an explicit execution time specified in the request, indicating when the operations should occur. The **Control event scheduler** is then responsible for scheduling these control events into the event list, making the operations are performed with the simulation update.

Since optimistic synchronization with rollback is not suitable for our purposes (see Section 1) and to avoid causality violations, operations in a DAR-CI request are restricted to be executed only **later** than the current simulation time. Therefore, the time values specified in the request are compared to the current simulation time t_{now} (i.e., the time value of the previously executed event). All time values in the request must be greater than t_{now} for the request to be considered valid. Otherwise, the scheduling will fail because some operations are too late to be served.

The **Streaming response sender** will immediately notify the requester of the scheduling result. This can help the requester to better organize the remaining requests, for example, an exception can be thrown when the client receives a failure message.

3.2 Operation and Sequencing

The inserted control events are executed in chronological order, along with internal events that are generated by the simulation system itself. However, the execution order of events that have the same execution time requires special consideration. A straightforward method is to use the First In - First Out (FIFO), which means that the first event received will be executed first. However, this approach can introduce ambiguity and inconsistency, causing simulation results to vary unpredictably from run to run.

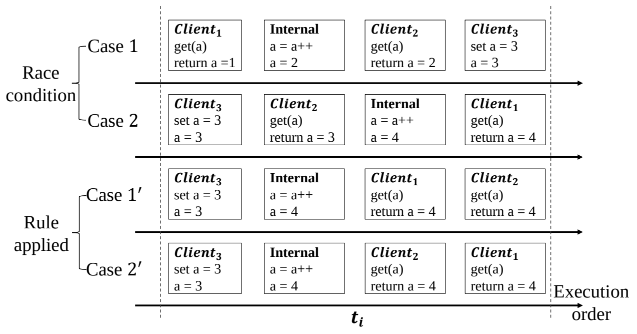

Race condition
Case 1
$Client_1$ get(a) return a =1
Internal a = a++ a = 2
$Client_2$ get(a) return a = 2
$Client_3$ set a = 3 a = 3
Case 2
$Client_3$ set a = 3 a = 3
$Client_2$ get(a) return a = 3
Internal a = a++ a = 4
$Client_1$ get(a) return a = 4
Rule applied
Case 1′
$Client_3$ set a = 3 a = 3
Internal a = a++ a = 4
$Client_1$ get(a) return a = 4
$Client_2$ get(a) return a = 4
Case 2′
$Client_3$ set a = 3 a = 3
Internal a = a++ a = 4
$Client_2$ get(a) return a = 4
$Client_1$ get(a) return a = 4
t_i
Execution order

Figure 2: Apply rules to resolve race conditions due to uncertain execution order of same time step event.

For example, as shown in Figure 2, the simulation has three control events at t_i, each from one concurrent controller client. $Client_1$ and $Client_2$ are trying to retrieve the value of a and $Client_3$ is setting it equal to 3. The order in which these events are received is uncontrollable. An internal event that performs $a{+}{+}$ has also been scheduled at t_i, and using the FIFO method eventually leads to a race condition (see Case 1 and Case 2), resulting in a completely different simulation behavior.

To solve this problem, we define a rule that controls execution order of these "same time step" control events. In our framework, we first categorize the control events according to their operations into Get-events (read only) and Set-events (read and write), which retrieve simulation data and modify it, respectively. To avoid the race condition, Set-events should be executed first within the same time step. After that, internal events should be executed, followed by Get-events. Events within the same category can be executed in any order, but an object can only be modified once in a single time step for all Set-events to prevent conflicting updates. If a conflict occurs, the simulation will notify the sending system to handle the exception. By following this rule, the simulation remains deterministic regardless of the receiving order, as shown in Fig. 2.

3.3 Synchronization

In DAR-CI, the controller and simulation operate independently using an asynchronous approach. As previously mentioned, any control event that arrives after its execution time with respect to the simulation time will not be scheduled because it is already "too late". However, in a coupled simulation, certain critical time points, such as those related to decision-making or data collection, must not be missed. Failure to meet these time points could cause the entire integrated system to fail in achieving its intended simulation purpose.

To avoid this, we define a new type of control event, the Pause-event. It allows the simulation progress to be suspended at a specified time, giving participating controllers the opportunity to "catch up", and the controllers can use *continue requests* to resume the simulation.

The order where the Pause-event should be placed is also crucial, because it could lead to deadlocks if a Pause-event is executed before the other control events at the same execution time step. Therefore, we place the Pause-event at the end of its time step to be executed. Additionally, if multiple pause requests are received targeting at the same simulation time step, only one Pause-event is inserted, and a pause counter (*pc*) is used to track the status after receiving multiple concurrent continue requests. The simulation will resume when *pc* reaches to zero.

3.4 Data Movement

With the event-driven approach, a request can collect a large amount of data over time by inserting multiple events at different times. However, sending every piece of raw data as soon as it is generated would lead to significant network overhead, which is also often unnecessary since the data may not be needed at that moment.

Hence, by using DAR-CI, we allow the requester to specify the desired way of returning data in the request, which facilitates the streaming of data in batches. The **Reply data manager** shown in Fig. 1 is responsible for this. It collects the retrieved data from the running simulation, packages it according to the request, and then sends it in batches via the **Streaming response sender**. This not only reduces network overhead by reducing the frequency of data transfers, but also facilitates parallel computing, as the data requester can process data from the previous batch while the simulation produces the next batch.

Additionally, the **Reply data manager** also supports in-situ data processing, which enables the computation of pre-defined traffic statistics and customized data aggregation using an SQL-like approach (see Section 4.4) after the requested data is generated from the simulation.

In summary, DAR-CI provides users with a flexible and efficient approach to data processing and management. It allows users to tailor their data analysis strategies to their specific needs and requirements, while effectively managing network traffic and resources for better performance.

4 SEMANTICS

In this section, we focus on semantically structuring the request and response of DAR-CI to address the informational challenges that arise when interacting with heterogeneous simulation systems, as noted in Section 2. Specifically, we address how to formally describe requests to express the control operations and how to structure responses to provide accurate and meaningful information. Our approach is also a solution to the *targeted and online* simulation data extraction problem [25], as we can build precise and focused requests to query simulation data at runtime.

The design of the semantics is based on the event-based approach, but is not strictly limited to it. As our goal is to provide a universal control protocol for the traffic community, it should be semantically simple, efficient, and flexible. Existing traffic simulators can easily create a wrapper that matches their interface to DAR-CI with very little effort to implement. This can help standardize traffic simulation communication, foster a more collaborative and productive traffic simulation research community, and ultimately lead to more effective and efficient traffic management solutions.

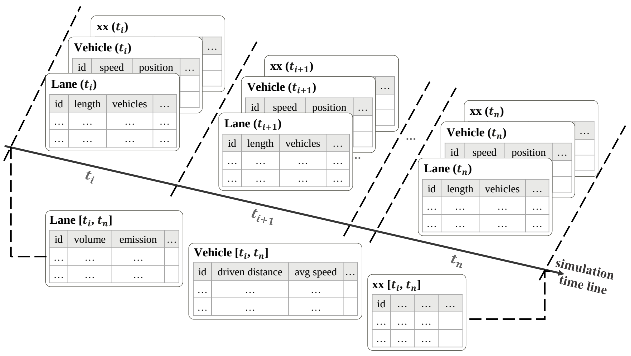


Figure 3: The organization of simulation databases involves updating them with time. Tables representing simulation data for each time step are located above the time line, while aggregated attributes are below.

4.1 Simulation Databases

Fig. 3 illustrates how, at each timestamp, simulation data can be organized into multiple tables containing columns and rows as a relational database. Each table represents a specific type of simulation object, such as a vehicle, lane, or traffic light. Each row represents an instance of that type with a unique identifying key (i.e., ID) and variables for each attribute (e.g., speed, position) written in each column. The state of any simulation data can thus be represented by the combination of Table name, Object ID, Attribute name, and Time step.

Another important but often overlooked type of attribute is the temporally aggregated attributes, such as traffic volume, road emissions, and energy consumption. These attributes are based on a time interval instead of a single time step and are frequently used as standard "domain-specific statistics" in the traffic domain. Including them in the simulation data representation would be beneficial to improve simulation performance and semantic clarity, as shown in Fig. 3. The state of these data can be indicated as Table name, Object Id, Temporally attribute name, and a time interval, e.g., from t_i to t_n.

The distinction between simulation data and a conventional database is quite significant. In a database, the data is typically static, while in a simulation, only previously generated and saved data is accessible. Therefore, when requesting information in a simulation, it is critical to provide temporal details to ensure that the simulator can allocate resources for transmission or post-processing.

4.2 Unary Request-Response

The fundamental component of the DAR-CI request is the **unary-request**, which is composed by five segments that must be completed in a left-to-right order:

$$< \textit{Operation}, \textit{Time}, \textit{Table name}, \textit{Reference by}, \textit{Attributes} >$$

Details are as follows.

1) *Operation*:
Four types of operations can be performed on the simulation side as requested: *data retrieval*, *data updating*, *pause simulation*, and *continue simulation*. Selecting *continue simulation*

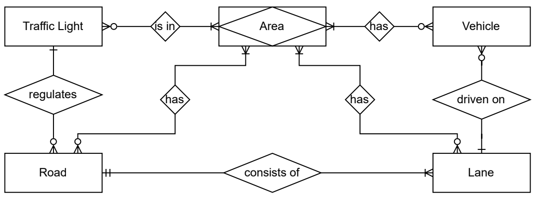


Figure 4: Entity Relation (ER) model of traffic simulation data.

as the operation will complete the request, as it immediately tries to wake up the sleeping simulation thread and is not considered an event. Otherwise, it is necessary to fill in the next parameter, i.e., time before sending the request.

2) Time:
It determines the timing of the operation, i.e., when the operation will occur. Note that for *data retrieval* operation, the time value can also be a 2-tuple, e.g., (t_s, t_e), which indicates that this request is intended to retrieve temporally aggregated attributes that are computed over this time intervals.

3) Table name:
During this step, the operations are either *data retrieval* or *data updating*. As discussed in Section 4.1, simulation data can be modeled as a collection of tables representing the states of various simulation objects at any given timestamp (or time interval). Therefore, the relevant table for the query must be specified.

4) Reference by:
This step is to narrow down the relevant data by specifying the rows representing the simulation instances (also known as agents). In contrast to the traditional approach of locating agents solely by their IDs, DAR-CI employs a predefined relation between simulated objects to support locating instances by the IDs of other associated objects, as illustrated in Fig. 4. For example, if we select the `[vehicle]` table and specify `[Road IDs]` as the references, along with the relation from **Road - Lane - Vehicle**, we can filter out vehicles driving on lanes of these roads as the instances of interest. This design is particularly useful in cases such as rerouting all vehicles on some congested Roads, which can be then expressed in a single request.
In addition, DAR-CI also supports the selection of objects with other relevant information. For instance, it enables the selection of agents within a specific region by sending polygons in a standard format, such as the well-known text (WKT) format.

5) Attributes:
In this step, the targeted data is further refined by specifying the names of the attributes that are of interest. This process is known as projection in the relational database domain, where the required columns are selected from the filtered table generated in the previous step. By specifying the relevant attributes, the dataset is further narrowed down, providing more focused and specific results that match the requirements of the query. For *data retrieval*, the name of all the interested attributes is required, i.e.,

$$[attributeName_1, attributeName_2, ...]$$

For *data updating*, the replacement value should also be given, i.e.,

$$[(attributeName_1, value_1), (attributeName_2, value_2), ...]$$

Table 1: Example of a Compound-Request with data return times in t_{i+1} and t_{i+5}.

Req	Operation	Time	Table name	Reference by	Attributes
Req_1	data retrieval	t_i	A	A_IDs=[1, 2]	[attr_1]
Req_2	data retrieval	t_i	A	A_IDs=[2, 3]	[attr_2]
Req_3	data retrieval	t_i	B	B_IDs=[1, 2]	[attr_3, attr_4]
Req_4	data retrieval	t_{i+5}	A	A_IDs=[1, 2]	[attr_1]
Req_5	pause simulation	t_{i+5}	N/A	N/A	N/A

Regarding the responses, the *data changing*, *pause simulation*, and *continue simulation* operations have the same response, which doesn't carry any semantic meanings. The purpose of this response is simply to inform the requester that the operation has been performed successfully. In contrast, for data retrieval, the response is the extracted target data. As the extracted data is a subset of a relational data table, it can be easily described in CSV, XML, or other data interchange formats. It is important to note that when responding to a data retrieval request, the time value describing when the data was fetched must always be attached to the response. This ensures that the requester is aware of the specific point in time that the data corresponds to, allowing them to accurately interpret and utilize the retrieved data.

4.3 Compound Request-Response

Unary-requests in DAR-CI have some limitations. For example, it cannot cover multiple objects or span across different times, as it is restricted to a single data table. To overcome this limitation, we propose the concept of **Compound-Request**. Formally expressed as follows:

$$< Reqs, T_{ret} >$$

Where *Reqs* is a set of unary-requests that can have different operations, times, table names, and attributes. T_{ret} is a set of data return times, representing when the generated responses (e.g., retrieved data) will be batched and sent.

Tab. 1 provides an example of a compound-request, which contains four data retrieval unary-requests. While Req_1 and Req_2 operate on the same table with the same time step, they retrieve different combinations of instances and attributes. In contrast, Req_3 targets another table (i.e., B), and Req_4 fetches data from a different time. This demonstrates how a compound-request can offer complete access to the entire simulation data, i.e, across different tables and time series. It also illustrates how the requester can precisely specify the desired data by using the *data retrieval* unary-requests and arranging their combinations.

In addition, a compound-request can include different types of unary-requests, such as the *pause simulation* request shown in Tab. 1. This request schedules a pause event that will halt the simulation's progress at time t_{i+5}.

T_{ret} provides flexibility for retrieving data in batches, as discussed in section 3.4. The **Reply data manager** component utilizes these time values to send out the queried data in batches. In the example from Tab. 1, the specified return times are t_{i+1} and t_{i+5}. Therefore, at t_{i+1}, unary responses for req_1, req_2, and req_3 (which were generated before t_{i+1}) are batched and transmitted together. Following that, when the simulation reaches t_{i+5}, responses of req_4 and req_5 are sent. Essentially, this feature enables users to tailor the timing and content of the returned data in batches in the request, as opposed to using a fixed frequency and batch size.

Finally, the semantic structure of DAR-CI requests is well-suited for the event-driven approach described in section 3. Each unary-request can be easily mapped to a *control event* with the time value indicating the execution time, resulting in a relatively straightforward implementation.

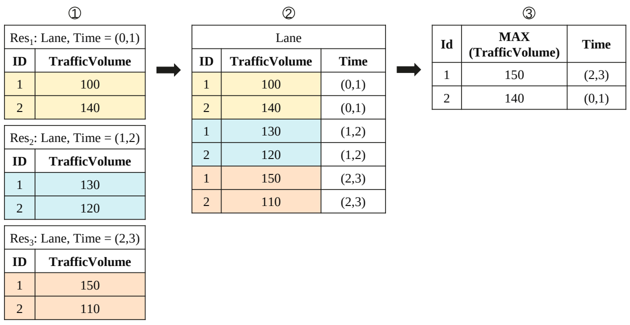
4.4 Possible Extension: In-situ Data Aggregation

DAR-CI is designed such that it is easy and natural to extend. As we have already described, the response of a unary-request is a subset of a relational data table with a time value, and the response of a compound request is a collection of these subsets, which itself becomes a **static temporal relational database**. This opens the possibility for any user defined query acting on the database collected by a compound-request, which can enable in-situ data aggregation, further reducing transfer overhead.

Fig. 5 illustrates how to use a standard SQL statement to perform second query on the data obtained from a compound request. The compound request consists of three unary-requests, each retrieving hourly traffic volumes for lanes with id 1 and 2 for the time intervals 0:00 to 1:00, 1:00 to 2:00, and 2:00 to 3:00, respectively. The collected data from each unary response is first stored (①) and then integrated into a data table, the table "Lane", and use a time column to label each row, indicating when the data corresponded to (②). With the following SQL statement, the peak traffic volumes and the times when the peaks occurred in both lanes during these three hours are filtered out (③):

```
SELECT ID, Time, MAX(TrafficVolume) FROM Lane GROUP BY ID
```

Compared to sending the entire data table out, the cost of transferring the aggregated data directly is significantly smaller. Given that query languages in static (temporal) databases are quite well established, our study does not include how to implement them in DAR-CI. Instead, what DAR-CI provides is a way to extract data across different objects and time series during dynamic simulations and form them into a static database, laying the groundwork for further use of the existing query languages.



①

Res$_1$: Lane, Time = (0,1)

ID	TrafficVolume
1	100
2	140

Res$_2$: Lane, Time = (1,2)

ID	TrafficVolume
1	130
2	120

Res$_3$: Lane, Time = (2,3)

ID	TrafficVolume
1	150
2	110

②

Lane

ID	TrafficVolume	Time
1	100	(0,1)
2	140	(0,1)
1	130	(1,2)
2	120	(1,2)
1	150	(2,3)
2	110	(2,3)

③

Id	MAX (TrafficVolume)	Time
1	150	(2,3)
2	140	(0,1)

Figure 5: Using SQL to do a second query on data collected from a compound-request. Hourly traffic volume data with units of vehicles is first integrated into one table. The SQL statement filters the data to find the peak traffic volumes and corresponding times for lanes 1 and 2 over the three-hour period. The time column in the table is with the unit hour.

5 CASE STUDY

We conducted two case studies to assess the advantages and disadvantages of DAR-CI: a dynamic Lane Traffic Volume Data (LTVD) query system and an adaptive traffic light control system. Both cases were tested on a grid road network with 100 intersections and 5000 lanes. DAR-CI is implemented on top of the traffic simulator CityMoS [36], using gRPC as the communication framework.

The baseline for our tests is a simulation of a day-long traffic without any runtime interaction, which takes 339 seconds to finish, resulting in a speedup of 255 times compared to the physical time (24 hours). The time step for the simulation is 250 milliseconds. We will refer to the time steps with $t_i = i \cdot \textit{time step}$ for $i \in [0, n]$. The controller client and the simulator are placed on separate machines with identical hardware configurations (Intel(R) Xeon(R) CPU E5-2630 v3 and 32GB of memory), connected by a local network. The order of magnitude of the network latency is 0.2 milliseconds and the bandwidth is 1 Gbits/sec . We also ran each use case using TraCI for comparison.

5.1 Lane Traffic Volume Data Query

LTVD refers to the number of vehicles that pass through a given lane in a given time period $T \in (t_s, t_e]$. This scenario is testing the impact of repeatedly retrieving the LTVD value at fixed time intervals ΔT from the running simulation. For instance, if $\Delta T = 5s$, the LTVD value will be generated for each period of 0-5 seconds, 5-10 seconds, 10-15 seconds, and so on. Consequently the time intervals will correspond to $T_j = (j \cdot \Delta T, (j+1)\Delta T]$ for $j \in (0, n)$.

The LTVD for a lane is calculated by counting the number of vehicles which have left the lane in the specified time interval, or formally expressed $\sum_{t=t_s}^{t_e} |V_{t_i} \setminus V_{t_{i+1}}|$, where V_{t_i} is the set of vehicles in the t_i time step, and $V_{t_{i+1}}$ is the set of vehicles of the subsequent time step. Therefore, to compute this metric for a lane, the list of vehicles present on it for each time step in the time interval is required. If there is a missing time step, a vehicle leaving the lane could be missed from the count.

Four controllers were implemented and evaluated:

TraCI_norm is implemented with TraCI (see Section 2.1). At each time step, and for each interested lane, `LastStepVehicleIds`[5] is sent individually to retrieve the IDs of the vehicles present on that lane. Subsequently, the controller calculates the LTVD incrementally using the methods described above.

TraCI_sub is also a TraCI-based approach but uses the subscription[6] feature for better performance. Instead of continuously active querying, it subscribes to `LastStepVehicleIds` for all relevant lanes after the simulation starts. The subscribed data (i.e.,

[5]https://sumo.dlr.de/docs/TraCI/Lane_Value_Retrieval.html
[6]https://sumo.dlr.de/docs/TraCI/Object_Variable_Subscription.html

Table 2: Compound Request for retrieving Data for DAR-CI_raw in Time Interval $(t_s, t_e]$. Data return at $\frac{t_s+t_e}{2}$ and t_e.

Operation	Time	Table name	Reference by	Attributes
Data Retrieval	t_{s+1}	Lane	Lanes' IDs	LastStepVehicleIds
Data Retrieval	t_{s+2}	Lane	Lanes' IDs	LastStepVehicleIds
⋮	⋮	⋮	⋮	⋮
Data Retrieval	t_e	Lane	Lanes' IDs	LastStepVehicleIds
Pause Simulation	t_e	N/A	N/A	N/A

Table 3: Compound Request for retrieving Data for DAR-CI_opt in Time Interval $(t_s, t_e]$. Data return at t_e.

Operation	Time	Table name	Reference by	Attributes
Data Retrieval	$(t_s, t_e]$	Lane	Lanes' IDs	TrafficVolume
Pause Simulation	t_e	N/A	N/A	N/A

vehicles' IDs on each lane) of the just-finished time step are then automatically sent back **in one message** to the requester whenever the simulation is paused. However, this still requires that the simulation be paused at each time step to collect the data for the entire time period.

DAR-CI_raw uses DAR-CI to interact with the simulation. It collects the same data and uses the same algorithm as TraCI_norm and TraCI_sub to compute the LTVD. The data is collected by a compound request for each time interval. As shown in Tab. 2, it consists of a series of unary requests, with each request querying the vehicle IDs on the lanes at each time step. A pause request is also added at the end of the interval to ensure that the simulator will not run further than t_e, which guarantees that the simulator will not reject the next requests. Once the computation is done and the request for the next interval is scheduled, the controller sends a continue request to resume the simulation. To facilitate efficient data processing, the returned data is split as evenly as possible into two batches to be returned. The first batch is returned at the midpoint of the current time interval, and the second batch is sent at the end of the interval.

DAR-CI_opt operates similarly to DAR-CI_raw, but the difference lies in the content of the compound request (see Tab. 3). As the LTVD is a commonly used metric, we integrated the aforementioned computation logic directly into the DAR-CI server. In this case, the LTVD value is requested directly by DAR-CI_opt by giving the desired time interval and lane IDs, and the calculation is performed in-situ within the simulation.

5.2 Performance Analysis

To evaluate the performance of different controllers for retrieving the LTVD value, we conducted a series of experiments where we varied the length of ΔT and the number of lanes (n_l). The results of these experiments are presented in Figure 6.

TraCI uses a step-based synchronous approach. Therefore, it provides consistent speedup values at different ΔT for the same n_l. DAR-CI shows better performance as ΔT increases. This is because a longer ΔT results in fewer requests and lower overhead. We observed that for ΔT greater than or equal to 5 seconds, DAR-CI consistently outperforms TraCI. For example, when we compared

Table 4: Scalability analysis: The values represent the additional time required to retrieve the data of an increased number of lanes relative to retrieve a single lane. The closer the value to one, the better the scalability of the controller.

Controller	d_{25}	d_{50}
TraCI_norm	6.60	11.57
TraCI_sub	1.42	1.71
DAR-CI_raw	1.11	1.21
DAR-CI_opt	1.02	1.05

DAR-CI_raw and TraCI_sub with $\Delta T = 300s$ and $n_l = 50$, DAR-CI_raw exhibited a remarkable 223% improvement in performance over TraCI_sub, even though both of them actually received the same data.

However, when dealing with small intervals, DAR-CI must pause the simulation at a very fine granularity to validate requests for the next time interval. In the extreme case, i.e., $\Delta T = 250ms$, the control strategy falls back to the "time step mechanism". Due to its more complex request-response behavior, DAR-CI underperforms compared to TraCI under such cases. For example, each DAR-CI request always has at least two responses, with the first response providing information about the validity of the request and the rest with the actual contents. Whereas in TraCI, it is not needed for its synchronous approach since the request is always assumed to be valid. This additional overhead may make DAR-CI unsuitable for use cases that require high-frequency synchronization. However, in the context of our targeted DDDM scenarios, which typically involve long-term data collection with ΔT on the order of seconds and minutes, this is not a limitation.

Another aspect to consider is the scalability. In the context of our experiment, it refers to the ability of the integrated simulation system to handle an increasing number of queried lanes without sacrificing performance. To measure it quantitatively, for each client, we compare the additional time required for the simulation as the number of queried lanes increases. The following formula is used:

$$d_N = \frac{\sum_{i=1}^{n} WCT(n_l = N, \Delta T_i)}{\sum_{i=1}^{n} WCT(n_l = 1, \Delta T_i)}$$

where WCT denotes the wall clock time used for the simulation of retrieving LTVD with N queried lanes and the different used time intervals ΔT_i. The results is depicted in Tab. 4.

The TraCI_norm is particularly sensitive to scale and exhibits the most significant decrease in degradation. This is due to its limitation of allowing only a single object to be queried per request, which leads to an increased number of messages exchanged as the number of lanes queried increases. In contrast, the TraCI_sub has better scalability due to its ability to consolidate data from all lanes of each time step into a single message to be sent, ensuring consistent number of messages exchanged in all the runs. However, the performance degradation is still notable due to larger message sizes and higher computational efforts.

While the DAR-CI_raw can also encounter challenges in scaling with an increase in data size and computational requirements, it

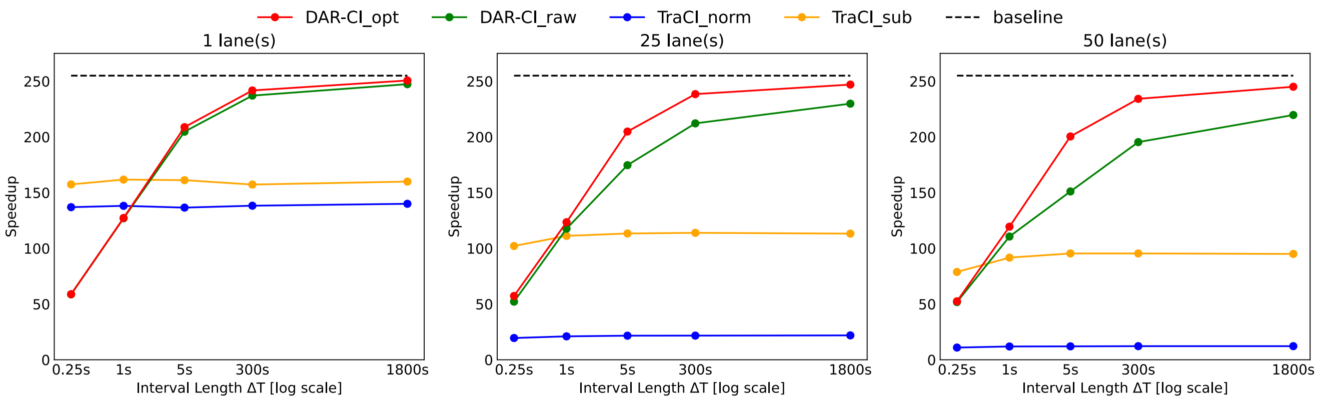


Figure 6: Speedup of simulating LTVD retrieval scenarios with different controllers. The LTVD was varied with different ΔT and n_l. The Speedup is the ratio of the simulation wall clock time and the physical time (i.e., 24 hours). As a baseline, the simulation was run without any runtime interaction, and it achieved a speedup of 255 times.

demonstrates better scalability than TraCI_sub due to its batch processing capability. In our experiment, we divided the data for each time interval into two evenly sized batches for transmission. This approach not only reduces the frequency of information exchange compared to TraCI_sub, but also enables parallel computing. Specifically, the communication and computation overhead for issuing the first batch of data can be overlapped with simulation updating, minimizing the performance degradation due to scaling. The DAR-CI_opt performs the best in terms of scalability. This is because it allows computations to be performed in-situ, only the calculated results need to be transferred. As a result, variations in message size between different lanes numbers are relatively minor and do not significantly impact performance.

5.3 An Adaptive Traffic Light System

In the previous section, we illustrate how DAR-CI can significantly improve performance through LTVD experiments. In this section, we present a practical application of DAR-CI in a real-world DDDM scenario. Specifically, a complete pipeline is shown where DAR-CI is used not only to retrieve data from the simulation, but also to perform interventions at specific points in time.

In our previous research [20], we introduced an algorithm that uses real-time data and machine learning to dynamically optimize the timing of traffic lights at intersections. The algorithm takes collected traffic metrics (e.g., LTVD) as input and adjusts the traffic light programs accordingly at the beginning of each traffic light cycle[7]. To evaluate this optimization approach, we used DAR-CI to reproduce and evaluate the algorithm in a coupled simulation environment. Specifically, a controller client is created for each traffic light that uses the optimization algorithm, and then these clients are coupled with the CityMoS traffic simulator. A sequence diagram, as shown in Fig.7, illustrates the interaction flow between the traffic light clients and the simulation. It shows how DAR-CI's multi-client support enables seamless coordination and synchronization in a complex simulation environment.

[7]A traffic light cycle is a time interval consisting of a sequence of traffic signal phases, with each phase allowing traffic to flow in a particular direction or stopping it to allow traffic from other directions to proceed.

We conducted experiments assuming that LTVD is the input for the optimization algorithm. Specifically, the client collected the LTVD value for each lane at the intersection during each signal cycle. At the end of each cycle, a new signal timing plan for the next cycle is sent back to the simulation. Similar to our previous experiments, in addition to querying the LTVD data directly in the request (i.e., **DAR-CI_opt**), three additional approaches were implemented: using TraCI without subscription (**TraCI_norm**), using TraCI with subscription (**TraCI_sub**), and querying raw data with DAR-CI (**DAR-CI_raw**). For each approach, we tested 2 scenarios in different scale, i.e., with 5 optimized intersections and 100 intersections, respectively. Each intersection has 12 lanes. The optimized traffic light cycle is constrained within a duration of 200 ± 30 seconds. As a result, with 100 intersections, on average every 4.3 seconds a traffic light ends its cycle. All the approaches were single-threaded, including the two with DAR-CI, which use a coroutine for each traffic light client, and therefore did not benefit from hardware parallelism.

The results, shown in Tab. 5, show that DAR-CI outperforms TraCI in all tested approaches. When requesting the same data and performing the same computations, DAR-CI_raw is 100 times faster than TraCI_norm and almost 6 times faster than TraCI_sub. As expected, DAR-CI_opt proves to be the best performing controller due to its ability to directly retrieve LTVD with in-situ data processing. With 100 intersections, it achieves a speedup of 165.39, representing 64.85% of the baseline, and is 9.53 times faster than TraCI_sub. Also, DAR-CI_opt is the least affected by scale of all the approaches.

6 CONCLUSION AND FUTURE WORK

This paper presents *Discrete-Event, Aggregating, and Relational Control Interfaces* (DAR-CI), to improve the flexibility and efficiency of coupled traffic simulation, particularly in the context of Data-Driven Decision-Making (DDDM) scenarios. We propose to use

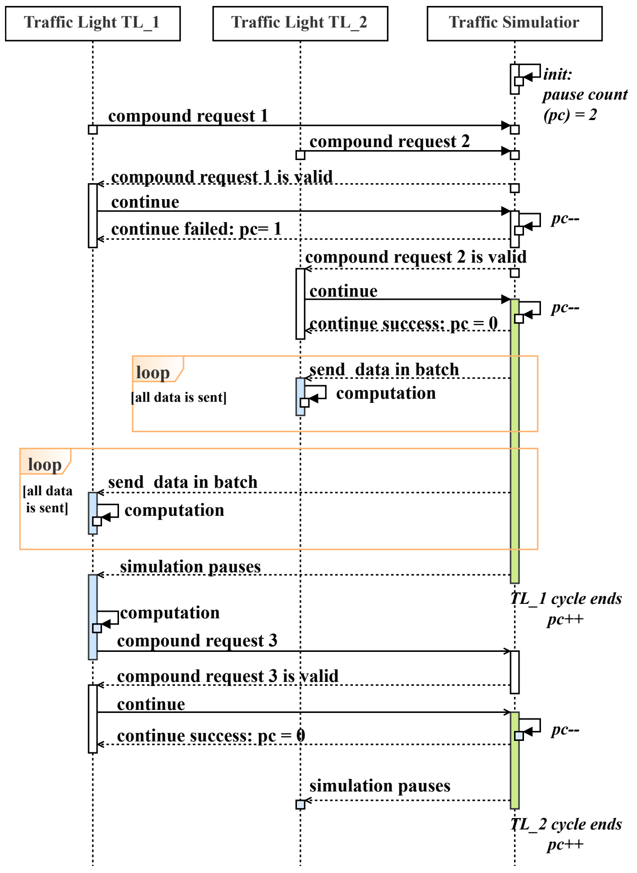


Figure 7: At the initial stage, the traffic simulator pauses and sets the pause count (*pc*) to the number of traffic light controller clients (2 in this case). Each controller client sends a compound-request, which includes three operations: updating the traffic light signal timings for the upcoming cycle, querying data for the cycle, and requesting the simulation to pause at the end of the cycle. The DAR-CI server schedules the requests and sends a confirmation message to the controller clients. Once the controller clients receive the confirmation message, they send a continue request, reducing the *pc* by 1. The simulation continues only when the *pc* reaches 0. During the simulation, queried data is sent back to the controller clients, and the clients can perform computation in parallel. If a traffic light cycle ends, the simulation pauses, and the corresponding controller client optimizes the signal timings with a new compound request to the traffic simulator. Due to space limitation, the diagram is only shown up to TL_2 cycle ends. The life bar is color-coded to provide visual cues: green means the simulation is running, blue for client-side data processing, and white for others.

Table 5: Speedup of simulating adaptive traffic light systems with different controllers.

Intersections	TraCI_norm	TraCI_sub	DAR-CI_raw	DAR-CI_opt
5	23.41	95.87	227.56	243.73
100	1.01	17.35	101.60	165.39

a discrete-event-based approach that allows for asynchronous interactions. Unlike existing approaches, the simulation can keep running without having to wait for data to be queried, reducing the idle processing time and the communication overhead, thus resulting in higher performance. In addition, DAR-CI utilizes rich and flexible semantic expressions to construct requests and responses. It incorporates a relational data model that facilitates precise data retrieval, ensuring that only the required simulation data is retrieved and transmitted. Moreover, DAR-CI allows for greater control over the data processing strategy (batching with customized size or in-situ computation) to optimally meet the requirements of different use cases. These features have been implemented in the CityMoS traffic simulator, resulting in improved and more efficient traffic management scenarios simulations.

Other use cases can also benefit from DAR-CI. For instance, its asynchronous model allows for plug-and-play and human-in-the-loop simulation, meaning that any intervention can be made during the runtime, without the need for pre-definition. This can be used to create immersive simulations of traffic scenarios, enabling transportation authorities and planners to observe and understand simulation results in real-time and experiment with What-If strategies in a controlled environment.

Although DAR-CI has shown promising results in traffic simulations, there are limitations to our current approach that we plan to address in future research. One such limitation is the current time-based constraints of DAR-CI. Each unary request must explicitly have a time value to indicate when the operation should occur. However, this approach can be challenging in situations where the timing is uncertain, such as in cases of traffic congestion or accidents. In such scenarios, a time step-based condition check is still required to determine whether the request should be applied or not, which is cumbersome and limits the performance of DAR-CI. Another area of research we are pursuing is extending DAR-CI to serve as a control system for distributed traffic simulations, which is a common practice for large-scale simulations. However, this is a challenging task due to the inherent complexity of distributed systems. Multiple event lists are simulated simultaneously under non-shared memory conditions, making it difficult to coordinate interventions accurately and efficiently. Data queries may also span different simulation instances, which requires an efficient and flexible semantics for distributed simulation databases. In addition, we're also prioritizing optimal query processing as part of our research. This includes identifying and reducing redundant queries to the same data, as well as implementing caching mechanisms to minimize duplicate data output.

REFERENCES

[1] Khaldoon Al-Zoubi and Gabriel Wainer. 2013. RISE: A general simulation interoperability middleware container. *J. Parallel and Distrib. Comput.* 73, 5 (May

2013), 580–594. https://doi.org/10.1016/j.jpdc.2013.01.014

[2] Zhiguang Cao, Siwei Jiang, Jie Zhang, and Hongliang Guo. 2017. A Unified Framework for Vehicle Rerouting and Traffic Light Control to Reduce Traffic Congestion. *IEEE Transactions on Intelligent Transportation Systems* 18, 7 (July 2017), 1958–1973. https://doi.org/10.1109/TITS.2016.2613997

[3] Jordi Casas, Jaime L. Ferrer, David Garcia, Josep Perarnau, and Alex Torday. 2010. Traffic Simulation with Aimsun. *Fundamentals of Traffic Simulation* (2010), 173–232. https://doi.org/10.1007/978-1-4419-6142-6_5

[4] Mustafa Coskun, Abdelkader Baggag, and Sanjay Chawla. 2018. Deep Reinforcement Learning for Traffic Light Optimization. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE. https://doi.org/10.1109/icdmw.2018.00088

[5] Judith S. Dahmann. 1997. High Level Architecture for simulation. In *1st International Workshop on Distributed Interactive Simulation and Real Time Applications*. IEEE, 9–14. https://doi.org/10.1109/IDSRTA.1997.568652

[6] Paul K. Davis. 1995. Distributed interactive simulation in the evolution of DoD warfare modeling and simulation. *Proc. IEEE* 83, 8 (1995), 1138–1155. https://doi.org/10.1109/5.400454

[7] Mostafa D. Fard and Hessam S. Sarjoughian. 2021. A Restful Persistent DEVS-Based Interaction Model For The Componentized Weap and Leap Restful Frameworks. In *2021 Winter Simulation Conference (WSC)*. IEEE. https://doi.org/10.1109/WSC52266.2021.9715348

[8] Martin Fellendorf and Peter Vortisch. 2010. Microscopic Traffic Flow Simulator VISSIM. In *Fundamentals of Traffic Simulation*. Springer, 63–93. https://doi.org/10.1007/978-1-4419-6142-6_2

[9] Richard M. Fujimoto. 1990. Parallel discrete event simulation. *Commun. ACM* 33, 10 (Oct. 1990), 30–53. https://doi.org/10.1145/84537.84545

[10] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. 2018. Co-Simulation: A Survey. *ACM Computing Surveys (CSUR)* 51, 3 (May 2018), 1–33. https://doi.org/10.1145/3179993

[11] Tobias Hardes, Dalisha Logan, Touhid Hossain Pritom, and Christoph Sommer. 2022. Towards an Open Source Fully Modular Multi Unmanned Aerial Vehicle Simulation Framework. In *2022 IEEE 42nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE. https://doi.org/10.1109/ICDCSW56584.2022.00060

[12] Tobias Helms, Jan Himmelspach, Carsten Maus, Oliver Rower, Johannes Schutzel, and Adelinde M. Uhrmacher. 2012. Toward a language for the flexible observation of simulations. In *2012 Winter Simulation Conference (WSC)*. IEEE. https://doi.org/10.1109/wsc.2012.6465073

[13] Celine Jacob and Baher Abdulhai. 2005. Integrated Traffic Corridor Control Using Machine Learning. In *2005 IEEE International Conference on Systems, Man and Cybernetics*. IEEE. https://doi.org/10.1109/icsmc.2005.1571683

[14] David R. Jefferson. 1985. Virtual time. *ACM Transactions on Programming Languages and Systems* 7, 3 (July 1985), 404–425. https://doi.org/10.1145/3916.3988

[15] Zhibin Li, Pan Liu, Chengcheng Xu, Hui Duan, and Wei Wang. 2017. Reinforcement Learning-Based Variable Speed Limit Control Strategy to Reduce Traffic Congestion at Freeway Recurrent Bottlenecks. *IEEE Transactions on Intelligent Transportation Systems* 18, 11 (Nov. 2017), 3204–3217. https://doi.org/10.1109/tits.2017.2687620

[16] Xiaoyuan Liang, Xunsheng Du, Guiling Wang, and Zhu Han. 2019. A Deep Reinforcement Learning Network for Traffic Light Cycle Control. *IEEE Transactions on Vehicular Technology* 68, 2 (Feb. 2019), 1243–1253. https://doi.org/10.1109/tvt.2018.2890726

[17] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. 2018. Microscopic Traffic Simulation using SUMO. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. https://doi.org/10.1109/ITSC.2018.8569938

[18] Tuo Mao, Adriana-Simona Mihaita, Fang Chen, and Hai L. Vu. 2022. Boosted Genetic Algorithm Using Machine Learning for Traffic Control Optimization. *IEEE Transactions on Intelligent Transportation Systems* 23, 7 (July 2022), 7112–7141. https://doi.org/10.1109/tits.2021.3066958

[19] Michal Piorkowski, Maxim Raya, A Lezama Lugo, Panagiotis Papadimitratos, Matthias Grossglauser, and J-P Hubaux. 2008. TraNS: realistic joint traffic and network simulator for VANETs. *ACM SIGMOBILE Mobile Computing and Communications Review* 12, 1 (Jan. 2008), 31–33. https://doi.org/10.1145/1374512.1374522

[20] Radu Tudoran, Stefano Bortoli, Cristian Axenie, Mohamad Al Hajj Hassan, Goetz Brasche, Hailin Li. 2019. Traffic signal control by spatio-temporal extended search space of traffic states. Patent No. WO2020147920A1, Filed Jan 14th., 2019, Issued Jul. 23th., 2020.

[21] Matthias Rupp, Stefan Schuhbäck, and Lars Wischhof. 2021. Coupling Microscopic Mobility and Mobile Network Emulation for Pedestrian Communication Applications. https://doi.org/10.48550/ARXIV.2109.12018

[22] Anuj Sanghvi and Tony Markel. 2021. Cybersecurity for Electric Vehicle Fast-Charging Infrastructure. In *2021 IEEE Transportation Electrification Conference*. IEEE. https://doi.org/10.1109/itec51675.2021.9490069

[23] Karl Schrab, Maximilian Neubauer, Robert Protzmann, Ilja Radusch, Stamatis Manganiaris, Panagiotis Lytrivis, and Angelos J. Amditis. 2022. Modeling an ITS Management Solution for Mixed Highway Traffic With Eclipse MOSAIC. *IEEE Transactions on Intelligent Transportation Systems* (2022), 1–11. https://doi.org/10.1109/TITS.2022.3204174

[24] Johannes Schützel, Sebastian Stieber, Christian Haubelt, and Adelinde Uhrmacher. 2015. Targeted on-line data extraction with SystemXtract. In *9th EAI International Conference on Simulation Tools and Techniques*. ACM. https://doi.org/10.4108/eai.24-8-2015.2260916

[25] Johannes Schützel and Adelinde M. Uhrmacher. 2015. Targeted Extraction of Simulation Data. In *2015 IEEE/ACM 19th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. IEEE. https://doi.org/10.1109/DS-RT.2015.37

[26] Christoph Sommer, Reinhard German, and Falko Dressler. 2011. Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis. *IEEE Transactions on Mobile Computing* 10, 1 (Jan. 2011), 3–15. https://doi.org/10.1109/TMC.2010.133

[27] Timothy Sprock, Conrad Bock, and Leon F. McGinnis. 2018. Survey and classification of operational control problems in discrete event logistics systems (DELS). *International Journal of Production Research* 57, 15–16 (Dec. 2018), 5215–5238. https://doi.org/10.1080/00207543.2018.1553314

[28] Ivo J. P. M. Timóteo, Miguel R Araújo, Rosaldo JF Rossetti, and Eugenio C Oliveira. 2010. TraSMAPI: An API oriented towards Multi-Agent Systems real-time interaction with multiple Traffic Simulators. In *13th International IEEE Conference on Intelligent Transportation Systems*. IEEE. https://doi.org/10.1109/ITSC.2010.5625238

[29] Andreas Tolk. 2013. Interoperability, Composability, and Their Implications for Distributed Simulation: Towards Mathematical Foundations of Simulation Interoperability. In *2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications*. IEEE. https://doi.org/10.1109/DS-RT.2013.8

[30] Yentl Van Tendeloo and Hans Vangheluwe. 2017. An Introduction to Classic DEVS. *arXiv preprint arXiv:1701.07697* (2017). https://doi.org/10.48550/arXiv.1701.07697

[31] Gabriel A. Wainer, Rami Madhoun, and Khaldoon Al-Zoubi. 2008. Distributed simulation of DEVS and Cell-DEVS models in CD++ using Web-Services. *Simulation Modelling Practice and Theory* 16, 9 (Oct. 2008), 1266–1292. https://doi.org/10.1016/j.simpat.2008.06.012

[32] Thomas A. Wall, Michael O. Rodgers, Richard Fujimoto, and Michael P. Hunter. 2015. A federated simulation method for multi-modal transportation systems: combining a discrete event-based logistics simulator and a discrete time-step-based traffic microsimulator. *Simulation* 91, 2 (2015), 148–163. https://doi.org/10.1177/0037549714564079

[33] Erwin Walraven, Matthijs T.J. Spaan, and Bram Bakker. 2016. Traffic flow optimization: A reinforcement learning approach. *Engineering Applications of Artificial Intelligence* 52 (June 2016), 203–212. https://doi.org/10.1016/j.engappai.2016.01.001

[34] Wenguang Wang, Weiping Wang, Yifan Zhu, and Qun Li. 2010. Service-oriented simulation framework: An overview and unifying methodology. *Simulation* 87, 3 (Dec. 2010), 221–252. https://doi.org/10.1177/0037549710391838

[35] Axel Wegener, Michał Piórkowski, Maxim Raya, Horst Hellbrück, Stefan Fischer, and Jean-Pierre Hubaux. 2008. TraCI: an interface for coupling road traffic and network simulators. In *11th communications and networking simulation symposium*. ACM. https://doi.org/10.1145/1400713.1400740

[36] Daniel Zehe, Suraj Nair, Alois Knoll, and David Eckhoff. 2017. Towards CityMoS: A Coupled City-Scale Mobility Simulation Framework. *5th GI/ITG KuVS Fachgespräch Inter-Vehicle Communication* 2017 (2017).

[37] Daniel Zehe, Vaisagh Viswanathan, Wentong Cai, and Alois Knoll. 2016. Online Data Extraction for Large-Scale Agent-Based Simulations. In *2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. ACM. https://doi.org/10.1145/2901378.2901384