CoSiFra: a Generic Middleware-Based Interface for Mobile Communication Simulation

Touhid Hossain Pritom, Burkhard Hensel, and Christoph Sommer
TU Dresden, Faculty of Computer Science, Germany
https://www.cms-labs.org/people/ { touhid.pritom , hensel , sommer }

***Abstract*—Simulating modern mobile communication systems demands complex, multi-domain simulation environments to accurately capture the interplay among diverse network components and protocols. For this, researchers frequently integrate formerly independent domain-specific simulators to form heterogeneous simulation environments. However, existing approaches often rely on either tight coupling – resulting in poor flexibility and limited reusability when trying to switch to different simulators – or on custom solutions that lack generality and extensibility. In this paper, we introduce CoSiFra, a generic middleware architecture for coupling independent simulators in federated network simulation. The architecture is built on the principle of modularity, allowing researchers to easily integrate (and switch between) simulators. The proposed concept also employs an event aggregation mechanism over small, configurable time steps to reduce interaction overhead for remote distributed simulators. We conduct a comprehensive runtime evaluation to compare the costs vs. benefits of using a middleware-based coupling approach.**

***Index Terms*—network simulation, federated simulation, heterogeneous co-simulation, vehicle-to-everything, space-air-ground integrated network, satellite simulation.**

I. Introduction

Sixth-generation (6G) wireless networks are envisioned to support an unprecedented variety of applications – ranging from holographic telepresence and remote surgery to massive machine-type communications and fully autonomous systems [1, 2], including ground vehicles, aerial vehicles, and satellites. This vision of 6G networks presents new challenges for researchers in mobile communication simulation, as evaluating such heterogeneous, multi-domain scenarios is inherently complex and no single simulation tool can comprehensively capture all relevant aspects of the system, including networking interactions, diverse mobility models, and environmental factors.

To simulate heterogeneous environments and bridge different ecosystems, researchers typically connect multiple domain-specific simulators via custom application-level interfaces, thereby producing a tightly integrated simulation environment. An example of this strategy is Veins [3], which bidirectionally couples a road-traffic simulator with a discrete-event network simulator to study vehicular communication systems. A wide variety of works can be found in the literature that extend Veins using tight coupling in an effort to support different heterogeneous scenarios, and a detailed discussion is provided in Section II.

Yet such tight integration carries a cost: each coupling is specific to the pair of simulators it connects, making it difficult to substitute or extend individual components without re-engineering the glue layer. Middleware-based architectures address this by loosely coupling simulators through standardized message-passing or co-simulation interfaces, thereby preserving each tool's autonomy and enhancing modularity and portability across simulation environments [4, 5]. Furthermore, many coupling concepts strongly differentiate between the one simulator that is allowed to drive the simulation forward and others that merely follow, or they limit the number of participating simulators to one per domain (e.g., by communication protocol or by participant), further reducing the flexibility and modularity of such a co-simulation.

This paper builds upon our earlier work [6], where we introduced a middleware-based orchestration concept for heterogeneous simulators in mobile communication scenarios. The middleware architecture leverages gRPC as its primary communication protocol for secure, distributed simulation and remote access, and our earlier work focused on the initial design of the middleware by loosely coupling OMNeT++ [7] and SUMO [8].

In this paper, we extend that work by providing a detailed description of the middleware architecture, introducing support for hybrid simulators (single simulators fulfilling multiple roles such as network simulation and mobility simulation), and conducting a comprehensive runtime analysis. Furthermore, unlike our previous work [6], the presented middleware architecture is fully independent of any active simulation component that would be required to drive the simulation forward. We also coupled a simple Python-based Application_sim with the middleware for rapid prototyping of applications while being able to access the full communication stacks of the participating simulators.

In addition, we demonstrate the flexibility of the generic middleware architecture by integrating multiple simulators across ground, air, and space domains. In more detail, we present a use case where a total of five different simulators participate, including two discrete-event network simulators in the same simulation. It couples OMNeT++ (extended by Simu5G [9, 10]) for ground-based and air-based network simulation, SUMO for the mobility of ground vehicles, AirMobiSim [11] for mobility of Unmanned Aerial Vehicles (UAVs), and space_Veins [12] for satellite mobility and ground-space network simulation. Such a co-simulation allows the evaluation of teleoperated vehicle

DS-RT 2026

scenarios in heterogeneous network environments. For instance, 5G communication can be simulated using a single network simulator for terrestrial coverage, while another simulator specialized in satellite communication can be used to provide connectivity in coverage gaps. This enables seamless end-to-end simulation of teleoperation use cases across mixed network domains. Furthermore, the middleware architecture allows the integration of trained AI models to dynamically select optimal routes based on predicted network coverage, enabling research into adaptive routing and connectivity-aware mobility strategies that were previously difficult to study in a unified simulation environment.

To sum up, the main contributions of this paper are as follows:

- A middleware architecture and concept for loosely coupling multiple distributed simulators in mobile communication scenarios using gRPC for distributed simulation and secure simulator interaction.
- Coupling of two different discrete-event network simulators within the same simulation environment.
- Simulation including ground, air, and space nodes within a shared scenario, and an application simulator separated from the network and mobility simulators, implemented in different programming languages for balancing simplicity with efficiency.
- A runtime analysis of the middleware's performance with five different simulators participating, including synchronization overhead and scalability.

The remainder of the paper is organized as follows: Section II reviews related work in the area of co-simulation and middleware-based frameworks. Section III describes the proposed middleware architecture. Section IV discusses the simulation setup and the evaluation of CoSiFra's performance. Finally, Section V concludes the paper and outlines future research directions.

II. Related Work

Zhou et al. [13] highlights the need for a space-air-ground integrated simulation framework for 6G networks and proposes a simulation framework, UltraStar, by integrating NS-3, MATLAB, and the NASA SPICE toolkit to address 6G network simulation. However, the framework is not generic and lacks a middleware-based interface. For realistic future network simulation, it is not only necessary to integrate space, air, and ground entities with network simulators, but also to integrate specialized tools for core networks, Radio Access Network (RAN), and detailed models for beamforming or intelligent surfaces.

Standards such as High-Level Architecture (HLA) [14, 15] exist for loose coupling of simulation components, but they are very generic and cover a wide range of simulation domains, concepts, and controls for coupling simulators. As a result, it will be difficult to integrate all aspects of HLA without increasing the complexity of the integrated framework while remaining generic for a specific field such as mobile communication simulation. One such comparative study was conducted by Steinbrink et al. [16], who performed a twofold comparison: a conceptual and a practical comparison of the HLA and MOSAIK, a framework for smart grid research (not to be confused with "MOSAIC" [17]). In this work [16], the authors praised the flexibility of HLA, but also noted the complexity of integrating components and the framework's learning curve.

On the other hand, MOSAIC extends the principles to multi-domain mobility and communication scenarios, supporting dynamic simulator integration and event synchronization. Both MOSAIC and its predecessor VSimRTI [18] are middleware-based co-simulation frameworks that have significantly advanced the integration of heterogeneous simulators, particularly for Vehicular Ad-hoc Network (VANET) and multi-domain scenarios. VSimRTI was among the first to provide a generic middleware for orchestrating network and mobility simulators, while MOSAIC broadened this approach to support a wider range of simulation components and domains. They share a similar architectural design, but MOSAIC extends the flexibility and scope of VSimRTI to enable more diverse and dynamic simulation setups.

Our CoSiFra architecture draws inspiration from both HLA and MOSAIC in designing the middleware and tailors it for secure remote distributed simulation of mobile communication scenarios. This inspiration allows us to support not only pure network and pure mobility simulators, but also hybrid simulators that serve as both simultaneously. Using an external application simulator helps to centralize information from different simulators, so that not all simulation components need to be aware of each other. The motivation for using an external application simulator is also drawn from the work of MOSAIC [17], which also uses one to handle application logic and integrate information collected by executing independent simulation components. However, using an external application simulator is not unique to the MOSAIC framework; a similar approach is also observed in the work by Härri et al. [19] on iTETRIS, where the authors use a middleware-based approach to couple NS-3 and SUMO for ITS applications. A more detailed discussion on the comparison of CoSiFra and MOSAIC is provided in Section III-C.

There are only a few handpicked frameworks (e.g., MOSAIC, VSimRTI) in the literature that use a middleware-based approach; rather than actively participating in the simulation flow, they focus on orchestrating simulation components and their synchronization. Moreover, very few of them demonstrate the coupling of more than two or three simulators working together, regardless of the coupling type.

On the other hand, there is extensive literature that uses tight coupling to integrate state-of-the-art simulators for VANET and Space-Air-Ground Integrated Network (SAGIN) scenarios, and most clearly emphasizes the importance of coupling simulators to capture diverse aspects of these scenarios through simulation. Several well-known simulators for network and mobility exist, as well as simulation frameworks that couple these simulators for VANET and SAGIN scenarios. For example, OMNeT++ [7] is a widely used discrete-event network

simulator, and Simu5G [9, 10] is a framework of OMNeT++ for 5G and beyond network simulation that relies on the INET [20] framework. SUMO [8] is a popular microscopic traffic simulator, and AirMobiSim [11] is a mobility simulator for UAVs. Veins [3] is a popular framework for simulating Vehicular Ad-hoc Networks (VANETs) scenarios, which couples OMNeT++ and SUMO. A variety of extensions of Veins can be found in the literature to support heterogeneous scenarios; for example, Artery [21, 22] extends Veins to simulate European ITS-G5/C-ITS vehicular communication scenarios. Artery-C [23] extends the Artery framework to support C-V2X communication, including dynamic mode switching and sidelink communication. AirMobiSim_libveins [11] extends Veins for integrated ground-air simulation of urban cities. space_Veins [12] extends Veins to simulate space-ground integrated networks – by integrating satellite mobility with the Simplified General Perturbation 4 (SGP4) model in Veins for network simulation – and supports ground-space communication scenarios. Veins_Carla [24] extends Veins and couples CARLA [25] for integrating sensing into network co-simulation and facilitating Cooperative Autonomous Vehicle (CAV) research. Most of these works could be supported by a generic middleware-based approach, thereby reducing the number of versions of the same framework and increasing the reusability of the simulators and unique simulation components these works have developed.

Moreover, there are unique frameworks that introduce new aspects to network simulation. For example, Buse et al. [26] and Buse and Dressler [27] took a different approach, proposing a middleware-based coordinating framework for Hardware-in-the-Loop (HIL) simulation by integrating a real-time system with Veins and SUMO to analyze Advanced Driver Assistance Systems (ADAS), but their proposed middleware is not generic for integrating diverse domains.

Zubow et al. [28] presents an open-source framework for coupling NS-3 with NVIDIA Sionna to enhance physical-layer realism by outsourcing channel and propagation modeling to Sionna. The framework employs ZeroMQ for inter-process communication and Protocol Buffers for efficient message serialization. Such works are particularly relevant to scenarios in which a communication framework exchanges layer-specific information within models, a challenge that is also an important step toward achieving more realistic and future-oriented network simulations. However, this type of time-sensitive coupling, while effective for tightly integrated setups, poses significant challenges for generic middleware-based approaches, especially those that rely on a time-stepped synchronization model or involve simulators running on remote servers.

III. Architecture

Figure 1 shows the architecture of CoSiFra. CoSiFra denotes a conceptual middleware architecture, not a specific software release, and can therefore be realized through different implementations. CoSiFra is designed to orchestrate and synchronize heterogeneous simulators, enabling unified co-simulation for complex mobile communication scenarios. The middleware architecture is modular and scalable, ensuring flexibility in integrating diverse simulators while maintaining loose coupling.

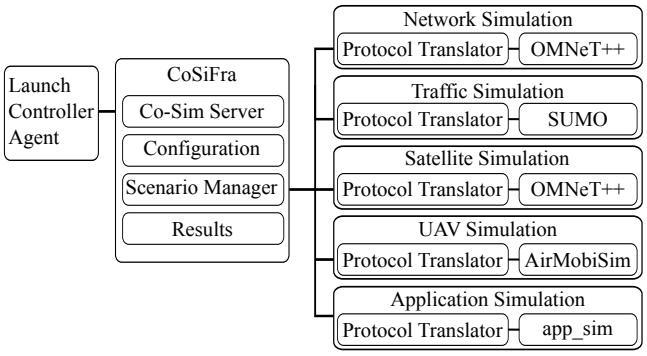

Figure 1. Reference architecture of CoSiFra.

A key architectural goal of the proposed middleware concept is consolidating simulator dependencies, reducing the need to maintain multiple versions of the same library (such as SUMO) across simulators. This not only simplifies the integration of heterogeneous simulators but also mitigates compatibility and maintenance challenges commonly encountered in co-simulation frameworks.

In the following, we describe the key components of the middleware, their interactions, and the role of time and interaction management in achieving seamless orchestration.

A. CoSiFra Overview

At the core of the architecture is the middleware block, which facilitates synchronization, communication, and data exchange among simulators. The design of CoSiFra consists of four key components, each serving a specific purpose in the overall architecture:

- The *Co-Sim Server* is a gRPC-based server that allows external agents – including human users and automated tools – to remotely launch simulations and retrieve results. This interface supports automated workflows, such as AI-driven parameter sweeps, and enables one digital twin to invoke another digital twin for coordinated multi-system analysis. Remote access through the server facilitates integration with broader digital twin or simulation ecosystems.
- The *Configuration Manager* centralizes both global and simulator-specific settings for the middleware architecture. This design allows users to reconfigure and re-execute simulations without modifying the simulator code or the coupling logic. Centralized configuration management supports iterative testing and remote operation.
- The *Scenario Manager* oversees the simulation workflow and consists of two component types:
  - The *Master Scenario Manager* acts as the central orchestrator, coordinating the overall simulation process. Each simulation run creates a new manager instance, which loads the current configuration before execution. This structure separates orchestration logic from simulation components, supporting modular middleware design.

  - A *Dedicated Scenario Manager* handles simulator-specific tasks – such as authentication and service calls – on behalf of the Master Scenario Manager. Encapsulating these details keeps the Master Scenario Manager agnostic and simplifies the integration of new simulators. This approach maintains clear boundaries between orchestration and simulator-specific logic.
- The *Result Manager* collects, processes, and converts outputs from all simulation components. It normalizes heterogeneous data (including logs and performance metrics) for human analysis or machine-driven workflows.

To the right of the middleware block, the simulation layer is represented by individual blocks for the connected simulators. This block is not limited to the simulators shown in Figure 1, as the architecture is designed to be extensible and can accommodate additional simulators as needed.

The interface layer is represented by the connection lines between the CoSiFra block and the simulation layer, which correspond to the communication protocols and interfaces used for data exchange and synchronization between the middleware and simulators. The interface layer can be realized using gRPC-based communication protocols, chosen for their high performance, language-agnostic nature, and support for multiple programming languages, making them suitable for integrating simulators written in different languages. Furthermore, the built-in security features of gRPC, such as authentication and encryption, can help ensure secure communication between the middleware and simulators, especially when they run on remote servers.

The *protocol translator* translates the generic middleware-based protocol into simulator-specific protocols, enabling communication between the middleware and each simulator without altering the existing interface used by each simulator. Furthermore, the protocol translator serves as a modular, self-contained component that facilitates the integration of proprietary simulators, thereby enabling industry-academia collaboration when each partner must retain ownership of their application-specific interface specifications in accordance with intellectual property agreements.

All components of the architecture and the design choices for CoSiFra are optimized for remotely and securely orchestrating simulators tailored to the specific requirements of mobile communication scenarios, such as handling network-specific data and interactions.

B. CoSiFra Interaction and Time Management

Efficient interaction and time management are critical for orchestrating heterogeneous simulators in a unified co-simulation environment. At a very high level, most simulators in mobile communication simulation are either step-based (and they have different time resolutions) or event-based. Most mobility simulators are step-based, whereas network simulators are often event-based to capture the dynamics of wireless communication, which can be on the order of microseconds or even nanoseconds. Coupling simulators with different time resolutions can lead to a loss of coherence, a well-known problem in the co-simulation literature that has been discussed thoroughly by Kempf et al. [29]. When coupling simulators using middleware-based approaches, the middleware typically controls the simulation flow, and its time resolution determines the combined system's time resolution. According to Kempf et al. [29], one possibility is to convert the event-based timing into fixed time steps or to adapt the fixed-time-step side to the event-based method. Furthermore, time-step simulators can be tailored either by using a very small fixed time-step or by using a variable time-step in response to events in the event-based simulator [29].

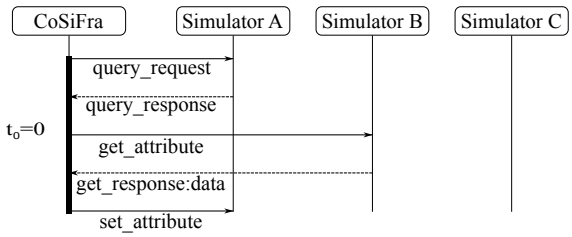

Figure 2. Initial interactions among federates.

There is no one-size-fits-all solution for this: the choice of time management approach depends on the specific requirements of the simulation scenario, the nature of the coupled simulators, and acceptable levels of accuracy and performance.

To better fit CoSiFra for orchestrating remote distributed simulation components where the cost of remote interaction is high, we have chosen to use a fixed time-step approach for time management and to perform event aggregation within the fixed update intervals. This reduces frequent interactions among connected simulation components, since the cost of remote communication can be very high.

In more detail, the middleware defines that the simulation is divided into equal time steps, aligning with the logic of most mobility simulators, which are step-based rather than event-based. While this approach introduces minor but unavoidable errors due to the discrete nature of time steps, it ensures compatibility across simulators and simplifies synchronization. A similar issue arises when coupling a discrete-event simulator with a step-based simulator: the discrete-event simulator must wait for the next synchronization point to reflect changes made by the step-based simulator. This can be mitigated by adjusting the synchronization interval length.

Logical time synchronization is managed by the *Master Scenario Manager*, which enforces a global synchronization barrier at each discrete logical time step, ensuring that no simulator advances beyond the current step until all participating simulators have completed the current step. During the simulation runtime, if any protocol translator fails (or fails to get back with a reply), the middleware stops progress and prints an informative log showing the simulation time and the simulation completion percentage.

Figure 2 shows the initial interaction among the middleware and the federates, where, at the very beginning of the simulation, any federate might request data, such as road network data, from

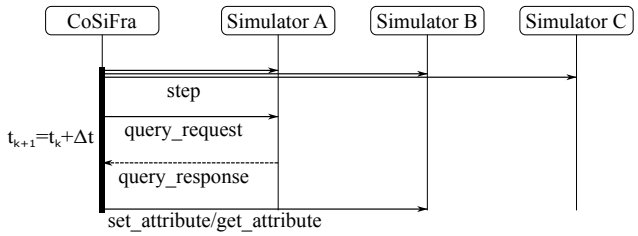

Figure 3. Periodic interactions among federates.

a traffic simulator. The middleware facilitates data exchange among simulators before the main simulation loop starts.

Figure 3 shows the interaction among the middleware and the federates during the main simulation loop. At each simulation step, the middleware instructs all connected simulation components to advance their internal simulation time to a common global synchronization point. Once all simulators have reached this synchronization barrier, the middleware polls each component for any interactions or data exchanges to be performed at that time. These interactions are structured according to a predefined *interaction template (request)*, as shown in Listing 1, ensuring consistent and interoperable communication across the middleware architecture. The *request* template, along with all the interaction templates, is defined using the Google Protocol Buffers (protobuf) Interface Definition Language (IDL) as message instances. The use of language-agnostic protobuf messages allows the middleware to impose a common data structure across interactions, regardless of the simulators' programming languages, and to provide a holistic view of interactions and the data exchanged among simulators. Having a clear understanding of interactions and data exchange is important in a multi-domain orchestrated system, as many existing frameworks lack mechanisms to effectively represent or visualize data flows, hindering users' understanding of the system's behavior.

Furthermore, the polling mechanism for interactions allows the middleware to maintain centralized control over the simulation flow and the interactions among simulators. After exchanging *request* and *response* messages, the middleware ensures that all interactions are served in order, thereby maintaining the simulation's causality. This process of advancing and exchanging interactions repeats in a loop until the simulation is completed. To handle different types of simulation components, CoSiFra uses a type categorization (a hybrid simulator is categorized into more than one type, e.g., *mobility* and *network*). At present, our implementation of CoSiFra allows a maximum combination of three types of categorization for a simulation component: *mobility*, *network*, and *optimizer*. The *optimizer* type is used for simulation components that are not strictly simulators but rather AI-based optimizers (for example, a route optimizer tool). CoSiFra uses this categorization to determine the types of interactions it performs at the synchronization point; for example, it will forward network-related interactions to a hybrid simulator and also be asked to submit mobile host-related status updates via the *GetManagedHosts* gRPC service.

```
message Request {
    enum Attribute {
        SPEED = 0;
        ROUTE = 1;
        SHAPE = 2;
        BOUNDARIES = 3;
        MESSAGE = 4;}
    enum RequestType {
        GET = 0;
        SET = 1;
        CHANGE = 2;
        TX_MESSAGE = 3;  // Transmit
        RX_MESSAGE = 4;  // Received
        DX_MESSAGE = 5;  // Dropped }
    enum Category {
        VEHICLE = 0;
        LANE = 1;
        ROAD = 2;
        POLYGON = 3;
        MAP = 4;
        APPLICATION = 5;
        NETWORK = 6;}
    Attribute attribute = 1;
    bool forAll = 7;
    RequestType type = 2;
    Category category = 3;
    string id = 4;
    ValueType value = 5;
    string subAttributeId = 6;
    string requestId = 8;
    double timestamp = 9;
    EntityType entity_type = 10;
    MessageType message_type = 11;
}
```

Listing 1. Protocol Buffer definition of the *request* message for exchanging interactions.

This service is not invoked for a pure network simulator, as a network simulator is not expected to provide mobility-related information to other simulation components.

C. Comparing Design Decisions of CoSiFra and MOSAIC

In the literature, only a few loosely coupled frameworks are compared to tightly coupled frameworks. Moreover, only a small subset of middleware-based approaches is designed solely for orchestration where the middleware itself does not actively participate in the simulation flow. MOSAIC [17] is a well-established framework in this sector, so we compare the design decisions of CoSiFra with those of MOSAIC.

At the framework level, MOSAIC employs event-based coordination to manage its federates. To accommodate simulators with differing time management paradigms, MOSAIC supports two types of time advance requests: a *time advance request* for step-based simulators and a *next event request* for event-driven simulators. In contrast, CoSiFra adopts a fixed time-step approach for time management, ensuring predictable synchronization across all coupled simulators. This approach simplifies integration between step-based and event-driven simulators and reduces the complexity of time synchronization, though it may introduce some discretization errors. Overall, similar to

MOSAIC, CoSiFra uses conservative time synchronization to avoid causality errors.

The *next event request* approach of MOSAIC can increase the frequency of interaction between the middleware and simulators (e.g., network simulators) when the simulation generates a large number of events, such as in scenarios with frequent beaconing. However, this also enhances its suitability for time-critical applications, as simulators can interact immediately when specific events occur rather than waiting for the next synchronization phase. In contrast, CoSiFra minimizes interaction frequency through event aggregation, thereby reducing the cost and potential delays associated with frequent remote interactions, but at the expense of less immediate synchronization between simulators.

Regarding interaction management, MOSAIC employs an internal custom publish-subscribe mechanism in which all federates subscribe to the predefined interactions they require, and the Run-Time Infrastructure (RTI) forwards these interactions once they are published by other federates. To allow a simulator to process an interaction dynamically, the federate ambassador can use two independent TCP/IP-based channels, one for reading and one for writing to exchange information independently. Using this two-channel model, the simulator can still send information to the federate ambassador without any active polling from the ambassador, which is useful for an event-based simulator, as it can send information to the ambassador as soon as an event occurs. CoSiFra likewise employs a custom internal publish-subscribe mechanism; however, interactions are exchanged at fixed time intervals, and the middleware explicitly polls all connected simulators to collect any pending interaction in the form of a predefined *request* template (see Listing 1). This polling-based approach, combined with event aggregation, reduces the frequency of middleware–simulator interactions, particularly in high-beaconing scenarios, substantially lowers the need for frequent interactions in distributed settings, and requires a single communication channel.

For federate communication, MOSAIC relies on an abstract interface Java class to communicate with a federate via the *federate ambassador*, which requires that developers implement a user-defined class which inherits from the *abstract ambassador* base class provided by the framework to define the federate's behavior. These ambassador implementations are specific to the type of simulator being coupled; for example, extending to an *abstract network ambassador* for coupling network simulators such as OMNeT++ or NS-3, or to a *sumo-ambassador* or *libsumo-ambassador* for configuring the coupling with SUMO. MOSAIC does not impose specific rules governing the implementation of the federate ambassador or its interactions with the simulator. As a result, the implementation of the *federate ambassador* can vary substantially across simulators, and the communication patterns between the ambassador and the simulator can vary considerably. For instance, coupling with OMNeT++ requires the federate ambassador to implement the *abstract network ambassador*; the resulting ambassador maintains two independent TCP/IP-based channels with the OMNeT++ simulator for reading and writing data and commands, a behavior that is fundamentally different from the coupling approach used with SUMO. The lack of standardized rules for *federate ambassador* implementations can complicate the integration of proprietary simulators, particularly in collaborative settings where interface specifications cannot be disclosed. Moreover, MOSAIC does not natively provide security mechanisms for secure remote coupling or simulation-as-a-service, which may limit its applicability in such scenarios. CoSiFra, by contrast, employs a gRPC-based *protocol translator* as a compartmentalized component for integrating proprietary simulators, a design decision that explicitly incorporates security features such as TLS for secure remote communication. However, TLS was not used during collecting results for this paper as a single machine was used for computing the results. Unlike MOSAIC, the protocol translator in CoSiFra allows different project partners to independently implement the respective protocol translators to couple with their proprietary simulators, without disclosing their application-specific interface specifications or requiring knowledge of the middleware's internal logic. This is achieved by implementing only the Remote Procedure Call (RPC) services necessary for the protobuf schema. Consequently, CoSiFra's protocol translator model can offer enhanced modularity and security, facilitating collaboration with industry partners and secure remote simulation.

Concerning simulation configuration, running a simulation with MOSAIC requires users to manage several configuration files beyond the simulator-specific ones, including a federate configuration file that may encapsulate simulator-specific settings, scenario and mapping configuration files for defining the simulation scenario and component mappings, and a runtime configuration file that specifies all participating federates together with their interactions and subscriptions. Managing multiple scattered configuration files can increase the risk of human error and slow down troubleshooting. Additionally, all interactions that a federate may subscribe to must be defined in the MOSAIC codebase as Java classes that inherit from the abstract interaction class. CoSiFra, on the other hand, consolidates all configurations into a single file – except for simulator-specific configuration files – and defines interactions as protobuf messages, providing a holistic, self-contained view of interactions and the data exchanged among simulators. Furthermore, all interactions are conveyed via the predefined *request* protobuf message template, with each interaction distinguished by its category, attribute, and type fields. Consequently, subscribing to a particular interaction requires a simulator to specify only the corresponding category, attribute, and type fields in the configuration file in order to receive the intended interaction. This approach reduces configuration complexity and facilitates quick understanding, particularly in collaborative settings.

In summary, both MOSAIC and CoSiFra are modular and extensible, but, as a consequence of use case and simulation requirements, they differ significantly in fundamental design regarding simulator coupling, particularly in how time and interactions are managed between federates and the middleware.

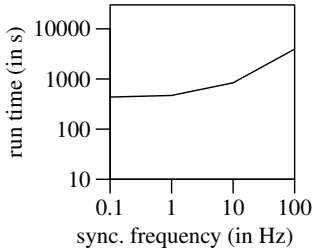

Figure 4. Real simulation completion time against synchronization frequency.

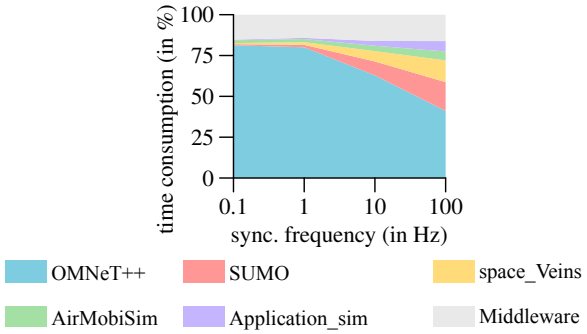

Figure 5. CPU busy time as a percentage of the real simulation completion time against synchronization frequency.

IV. Evaluation

To evaluate CoSiFra, we conducted a runtime analysis. We measured the execution time of CoSiFra by activating all five connected components of the simulation framework to assess potential overhead, particularly in scenarios with more than three coupled simulators, which remain uncommon in the literature. The evaluation also considered varying levels of simulation complexity to analyze scalability and performance under different conditions.

In our simulation scenario, we used OMNeT++ to simulate a 5G network, including an *operator* node connected to the base station to record vehicle data transmitted over the network. SUMO was employed for vehicle mobility simulation, configured to insert 1000 vehicles into the simulation at intervals of 0.5 s. AirMobiSim was used to simulate UAV mobility, with two UAVs configured in the scenario. space_Veins was configured with a single satellite to represent satellite mobility and communication. The application simulator was configured to schedule two distinct messages at each simulation step, targeting both network simulators. Specifically, the application simulator scheduled the first message such that all active ground vehicles sent a status message to the *operator* node. The second message was scheduled by the ground vehicles for transmission to the satellite via space_Veins. To reduce the effect of beaconing message interference, the network simulators scheduled the aggregated beaconing messages with a random delay between 1 ms and 3 ms, where the delay interval was chosen arbitrarily.

To evaluate the performance at different levels of simulation complexity we varied the synchronization frequency from 0.1 Hz to 100 Hz, increasing by a factor of 10 at each step. In tandem with increasing the synchronization frequency we also increased the beaconing message frequency from 0.1 Hz to 100 Hz, increasing by a factor of 10 at each step, thereby raising the overall simulation complexity. Here, shorter synchronization intervals make CoSiFra interact more frequently with the connected simulation components and also put more stress on the CoSiFra codebase. Increasing the beaconing frequency puts more stress on the network simulators' codebase. Both are usually the most time-consuming simulation components. This causes the middleware to wait longer to receive responses from the network simulators and to be processed by other components.

Figure 4 presents the real simulation completion time (wall-clock time) as a function of increasing synchronization frequency. It serves to demonstrate that, as expected, completion time rises with greater simulation complexity.

Figure 5 presents the share of different components in the simulation completion time. For this, the CPU busy time was measured using the Linux utility *time*, which provides the real (wall-clock), user, and system times for each simulator and its protocol translator. For each component, the CPU busy time was calculated as the sum of its user and system times. The real time of the middleware process was used as the overall simulation completion time. To generate the stacked area plot, the busy times of all components were summed and stacked.

The middleware overhead, calculated as the residual time after accounting for all simulator busy times, remained stable across all tested synchronization frequencies (mean values between 14 % and 16 %). While there was a slight trend toward higher overhead at higher frequencies, this effect did not reach statistical significance (ANOVA, $p = 0.066$). The 95 % confidence intervals grow no larger than 6.8 %, which occurs at 1 Hz synchronization frequency.

We can thus conclude that the middleware overhead is largely stable across the tested range of synchronization frequencies.

V. Conclusion and Future Work

In this work, we presented CoSiFra, a middleware architecture for flexible, scalable co-simulation across heterogeneous domains. The proposed middleware architecture enables integration of diverse simulators, including ground mobility (SUMO), aerial mobility (AirMobiSim), satellite and hybrid network-mobility (space_Veins), and multiple network simulators (OMNeT++), through a modular architecture centered on a gRPC-based *Protocol Translator* and a *Master Scenario Manager*.

Although the introduction of a middleware orchestrator incurs some synchronization and communication overhead, this trade-off enables enhanced flexibility, modularity, and extensibility. These qualities are particularly valuable in research and development contexts where adaptability is prioritized over raw execution speed.

It can be intuitively assumed that introducing a middleware-based solution may add overhead, as it adds an extra layer of

code and increases inter-component interfacing. We speculate that future improvements in hardware performance and reductions in communication costs could further mitigate the practical impact of this overhead. Additionally, the simulator execution order in our experiments was sequential, and we expect that parallel execution could further reduce simulation completion time. Nevertheless, our results are intended to quantify the middleware overhead in a conservative setting, even when simulators are executed sequentially. This analysis highlights the trade-off between added flexibility and orchestration overhead in loosely coupled co-simulation. The middleware architecture enables simulation-as-a-service, enabling remote and secure integration of simulators.

Furthermore, the middleware architecture supports integrated simulation with multiple mobility simulators, improving extensibility and maintenance. For scenarios requiring finer-grained simulation, the synchronization interval can be reduced.

Future work will focus on optimizing synchronization mechanisms, enabling parallel execution of simulators, further coupling discrete-event simulators like NS-3, and integrating with AI models for vehicle route optimization. Moreover, we will introduce the coexistence of varied synchronization intervals and event-based execution in CoSiFra.

The current implementation which underpins this study is available as Open Source software.[1]

ACKNOWLEDGMENT

The 6G-TWIN project has received funding from the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation program under Grant Agreement No 101136314.

REFERENCES

[1] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan, and M. Zorzi, "Toward 6G Networks: Use Cases and Technologies," *IEEE Communications Magazine*, vol. 58, no. 3, pp. 55–61, Mar. 2020.

[2] F. Liu et al., "Integrated Sensing and Communications: Toward Dual-Functional Wireless Networks for 6G and Beyond," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 6, pp. 1728–1767, Jun. 2022.

[3] C. Sommer, R. German, and F. Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis," *IEEE Transactions on Mobile Computing*, vol. 10, no. 1, pp. 3–15, Jan. 2011.

[4] N. Mustafee, K. Katsaliaki, and S. J. E. Taylor, "Distributed Approaches to Supply Chain Simulation: A Review," *ACM Transactions on Modeling and Computer Simulation*, vol. 31, no. 4, Aug. 2021.

[5] A. Gazis and E. Katsiri, "Middleware 101: What to know now and for the future," *ACM Queue*, vol. 20, no. 1, pp. 10–23, Feb. 2022.

[6] T. H. Pritom, B. Hensel, M. Franke, and C. Sommer, "Towards a Generic Middleware-Based Interface for Mobile Communication Simulation," in *29th International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2025)*, Prague, Czechia: IEEE, Sep. 2025.

[7] A. Varga and R. Hornig, "An Overview of the OMNeT++ Simulation Environment," in *1st International ICST Conference on Simulation Tools and Techniques for Communications Networks and Systems*, ser. SIMUTOOLS, ICST, 2008.

[8] P. A. Lopez et al., "Microscopic Traffic Simulation using SUMO," in *21st International Conference on Intelligent Transportation Systems (ITSC 2018)*, IEEE, Nov. 2018, pp. 2575–2582.

[1] https://www.cms-labs.org/research/software/cosifra/

[9] G. Nardini, G. Stea, A. Virdis, and D. Sabella, "Simu5G: A System-level Simulator for 5G Networks," in *10th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, Scitepress, 2020, pp. 68–80.

[10] G. Nardini, D. Sabella, G. Stea, P. Thakkar, and A. Virdis, "Simu5G – An OMNeT++ Library for End-to-End Performance Evaluation of 5G Networks," *IEEE Access*, vol. 8, pp. 181 176–181 191, 2020.

[11] T. Hardes, D. Logan, T. H. Pritom, and C. Sommer, "Towards an Open Source Fully Modular Multi Unmanned Aerial Vehicle Simulation Framework," in *15th IEEE International Workshop on Wireless Sensor, Robot and UAV Networks*, Bologna, Italy: IEEE, Jul. 2022, pp. 284–289.

[12] M. Franke and C. Sommer, "Toward Space-Air-Ground Integrated Network Simulation with 4D Topologies," in *19th IEEE/IFIP Conference on Wireless On-Demand Network Systems and Services (WONS 2024)*, Chamonix, France: IEEE, Jan. 2024, pp. 61–68.

[13] H. Zhou et al., "A High-Fidelity and High-Efficiency Simulator for 6G-Integrated Space–Ground Networks," *Elsevier Engineering*, vol. 56, pp. 62–78, Jan. 2026.

[14] J. S. Dahmann, R. M. Fujimoto, and R. M. Weatherly, "The Department of Defense High Level Architecture," in *29th Conference on Winter Simulation*, Atlanta, Georgia, USA: ACM, 1997, pp. 142–149.

[15] J. S. Dahmann and K. L. Morse, "High Level Architecture for simulation: an update," in *2nd International Workshop on Distributed Interactive Simulation and Real-Time Applications*, IEEE, Jul. 1998, pp. 32–40.

[16] C. Steinbrink et al., "Smart grid co-simulation with MOSAIK and HLA: a comparison study," *Springer Computer Science Research and Development*, vol. 33, no. 1–2, pp. 135–143, Sep. 2017.

[17] K. Schrab et al., "Modeling an ITS Management Solution for Mixed Highway Traffic With Eclipse MOSAIC," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 6, pp. 6575–6585, Jun. 2023.

[18] B. Schünemann, "V2X simulation runtime infrastructure VSimRTI: An assessment tool to design smart traffic management systems," *Elsevier Computer Networks*, vol. 55, no. 14, pp. 3189–3198, Oct. 2011.

[19] J. Härri et al., "Modeling and simulating ITS applications with iTETRIS," in *6th ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, ACM, Oct. 2011, pp. 33–40.

[20] L. Mészáros, A. Varga, and M. Kirsche, "INET Framework," in *Recent Advances in Network Simulation*. Springer, 2019, pp. 55–106.

[21] R. Riebl, H.-J. Günther, C. Facchi, and L. Wolf, "Artery: Extending Veins for VANET Applications," in *2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, IEEE, Jun. 2015, pp. 450–456.

[22] R. Riebl, C. Obermaier, and H.-J. Günther, "Artery: Large Scale Simulation Environment for ITS Applications," in *Recent Advances in Network Simulation*. Springer, 2019, pp. 365–406.

[23] A. Hegde and A. Festag, "Artery-C: An OMNeT++ Based Discrete Event Simulation Framework for Cellular V2X," in *23rd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, ser. MSWiM '20, ACM, Nov. 2020, pp. 47–51.

[24] T. Hardes, I. Turcanu, and C. Sommer, "Poster: A Case for Heterogenous Co-Simulation of Cooperative and Autonomous Driving," in *14th IEEE Vehicular Networking Conference (VNC 2023), Poster Session*, Istanbul, Turkey: IEEE, Apr. 2023, pp. 151–152.

[25] A. Dosovitskiy, G. Ros, F. Codevilla, A. M. López, and V. Koltun, "CARLA: An Open Urban Driving Simulator," in *1st Annual Conference on Robot Learning (CoRL)*, Mountain View, California: PMLR, Nov. 2017.

[26] D. S. Buse, M. Schettler, N. Kothe, P. Reinold, C. Sommer, and F. Dressler, "Bridging worlds: Integrating hardware-in-the-loop testing with large-scale VANET simulation," in *2018 14th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, IEEE, Feb. 2018, pp. 33–36.

[27] D. S. Buse and F. Dressler, "Towards Real-Time Interactive V2X Simulation," in *2019 IEEE Vehicular Networking Conference (VNC)*, IEEE, Dec. 2019, pp. 1–8.

[28] A. Zubow, Y. Pilz, S. Rösler, and F. Dressler, "Ns3 meets Sionna: Using Realistic Channels in Network Simulation," arXiv, cs.NI 2412.20524, Dec. 2024.

[29] F. Kempf, P. D. Kremmydas, J.-J. Jensen, A. Freimann, J. Scharnagl, and K. Schilling, "Multi aspect simulation framework for distributed control of networked satellite formations," in *72nd International Astronautical Congress (IAC 2021)*, Paper No. IAC-21-D1,4B,8,x64693, Dubai, United Arab Emirates: International Astronautical Federation (IAF), Oct. 2021.