Distributed Simulation to Support Driving Safety Research

Chris Schwarz, Ph.D.
Associate Research Engineer
National Advanced Driving Simulator
The University of Iowa

Distributed Simulation to Support Driving Safety Research

Chris Schwarz, Ph.D.
Associate Research Engineer
National Advanced Driving Simulator
The University of Iowa

David Heitbrink, M.S.
National Advanced Driving Simulator
The University of Iowa

Wanxin Wang
Graduate Research Assistant
National Advanced Driving Simulator
The University of Iowa

A Report on Research Sponsored by SAFER-SIM University Transportation Center

with matching funds provided by Toyota Motor Corporation, and the National Advanced Driving Simulator

June 2017

DISCLAIMER

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. This document is disseminated under the sponsorship of the U.S. Department of Transportation's University Transportation Centers Program, in the interest of information exchange. The U.S. Government assumes no liability for the contents or use thereof.

Acknowledgments

The authors wish to thank the SAFER-SIM staff for helping the project reach a successful conclusion. Thanks to Omar Ahmad for agreeing to contribute NADS simulator hours as a match for the project. Our thanks to Professor Joe Kearney for discussions and guidance generously shared with us.

Table of Contents

Acknowledgments iv
Table of Contents v
List of Figures vii
List of Tables viii
Abstract ix
1 Introduction 1
1.1 Background 1
1.2 Objectives 2
2 Functional Specification 2
2.1 Overview 3
2.2 Scenarios 3
2.2.1 Scenario 1: platooning 3
2.2.2 Scenario 2: confederate 3
2.3 Nongoals 4
2.4 Configuring a simulator network 4
2.5 Starting and stopping simulations 4
2.6 Authoring scenarios 5
2.7 Simulator Data 5
2.8 Open Issues 5
3 Technical Development 6
3.1 Distributed architecture overview 6
3.2 Simulation software 7
3.3 Synchronization of ADO and DDO behaviors 11
3.4 External objects interface library 14
3.5 Coordinated universal time 16
3.6 Future work 18
3.6.1 Lobby setup program 18
3.6.2 Dead-reckoning algorithm 18

3.6.3 Centralized HCSM for ADOs 18

4 Scenario Design 19

5 Conclusions 20

References 22

List of Figures

Figure 1. Distributed simulation architecture showing two simulators. The architecture will support more than two, but an upper limit has not been set yet 7

Figure 2. Overview of component structure 9

Figure 3. Interactions between components for updating external driving object state 10

Figure 4. Interactions between components for updating ADO state 11

Figure 5. Overall algorithm for updating dynamic objects 11

Figure 6. Work flow for updating all dynamic objects 12

Figure 7. Work flow for updating all dynamic objects 13

Figure 8. Component overview after addition of ExternalObjectsInterfaceLib 14

Figure 9. Internal structure of ExternalObjectsInterfaceLib 15

Figure 10. Network data transfer between sender and receiver 16

Figure 11. A set of simulators running in universal time norm (OB stands for overbound; the black dots are for time points generating the state information) 17

Figure 12. Algorithm for comparing time stamps 17

List of Tables

Table 1. Top five pre-crash scenarios of two-vehicle light-vehicle crashes [1] 19

Table 2. Top five priority V2V pre-crash scenarios [15] (FYL = functional years lost)................................ 19

Abstract

Over half of all crashes involve two or more vehicles. While driving simulation provides an effective way to study many crash scenarios in a well-controlled environment, it cannot capture the complex dynamics between two human drivers in the seconds leading up to a multi-car crash or safety-critical event. Examples of multi-driver simulation extend back to the 1980s in military applications, but only in the last five years have several groups begun to be active in this area to study connected and automated vehicles. We present the development of a distributed simulation capability for NADS MiniSim™ simulators. This capability will eventually benefit all MiniSim customers who are interested in performing multi-driver experiments. Technical details about the implementation are documented, as well as future work that remains to achieve a larger vision for cross-platform distributed simulation.

1 Introduction

1.1 Background

Two-vehicle crashes account for 65% of light vehicle crashes, and multi-vehicle crashes (three or more vehicles) account for 7 percent of light vehicle crashes [1]. Among young drivers, 50% of crashes have two or more vehicles involved. While driving simulation provides an effective way to study many crash scenarios in a well-controlled environment, it cannot capture the complex dynamics between two human drivers in the seconds leading up to a multi-car crash or safety-critical event. With few exceptions, most driving simulator laboratories do not have the capability to study multi-driver crashes in detail.

There are several examples of multi-driver distributed simulations in the literature, extending back to at least 1997 [2] for private sector applications, and to the 80s for military simulations. The High Level Architecture (HLA) and Distributed Interactive Simulation (DIS) standards have been developed for military simulations and applied to academic and commercial purposes over the years. A traffic simulation that mixed computer-controlled vehicles with vehicles that were updated by sensors on actual vehicles was demonstrated as an early application of HLA to transportation [3]. A driving simulator was also added to the simulation [4]. The state of the art in advanced distributed simulation using HLA at that time was summarized in 2000 [5]. Large distributed simulations had been constructed for the synthetic theater of war (STOW), however civilian applications were still being thought of as future developments.

Hancock and DeRidder [6] conducted the first research on driver responses to crash scenarios in a distributed driving simulator experiment. They used a head-on collision event over the crest of a hill as well as an intersection with off-angle streets. Linked responses were observed in which the avoidance responses of each driver actually cancelled each other out, leading to collisions. This is the type of result that cannot be obtained on traditional simulators with computer-controlled vehicles. They concluded with a recommendation to the research community:

> *"What is clearly required as a next step in the process is a programmatic and sustained effort on behalf of many researchers in order to take advantage of the opportunity which dynamic, interactive simulation presents."*

Lamentably, this advice was not taken up for the most part until this decade, and many opportunities for research remain.

Recent innovations in intelligent transportation systems and vehicle-to-vehicle (V2V) communications have revived the idea of multi-driver simulations. Platooning has been studied with platoons of four drivers [7]. The drivers were engaged in cognitive secondary tasks, and dependent measures included lane-keeping and gap-keeping (platoon coherence) variables. This research highlights a commercial product, SILAB, that supports multi-driver simulations with up to five drivers [8]. Additionally, Hancock has continued to push forward by developing the Real Time Multiple Seat Simulator (RTMSS) at the University of Central Florida [9]. Elsewhere, the Cyber-Physical System Simulator (CPSS) has been developed with a mixture of commercial and open-source software to create simulations with multiple drivers, V2V networking, and traffic modeling [10]. Others [11] present the development of a Networked Multi-Drivers Simulation Platform at the Intelligent Human-Machine Systems Laboratory at Northeastern University in

Boston, MA. The paper also lists seven recent multi-driver simulation studies in the US, Europe and Asia. Several lessons learned from distributed simulation studies using the multi-driver simulator MoSAIC at the German Aerospace Center were recently presented [12]. They include advice on providing enough research staff, visualizing the location of each driver in the simulation, and noting the challenges of data reduction and analysis of multi-driver events.

Previous research at the National Advanced Driving Simulator (NADS) in this area has been focused on demonstration projects for virtual proving grounds [13]. Two drivers participated in a cooperative agricultural task, one driving a combine and the other a grain cart. The combine was depositing harvested grain into the cart, and the driver of the cart had to maintain relative position and velocity with the combine at all times. One simulator was the NADS-1 and the other was the Cave at the Virtual Reality Applications Center (VRAC) at Iowa State University. Communications took place over the Iowa Communications Network (ICN), a high-speed fiber infrastructure for Iowa internet applications.

It is clear that the last five years have seen increased and accelerating interest in multi-driver simulations focused on connected vehicles and advanced driving assistance systems. Additionally, new types of multi-car interactions are expected, and in fact are already being seen, with the mixture of automated and traditional vehicles on the road. Driver-to-driver signaling is an important type of communication that is usually taken for granted, but which is lost when one driver is “out of the loop,” i.e., not driving. The added capability of seeing the behavior of the driver’s avatar in the vehicle is required to adequately study this type of signaling.

The existing state of the art for single-driver simulators provides computer-controlled behaviors to all other vehicles. These behaviors can be programmed to be quite complex, but it is doubtful that they approach the complexity of human behavior or, if they do, that they represent validated behaviors. Driver modeling is a popular approach for characterizing the behaviors of human drivers, and there is a great need for data that can inform such models. Previous research at the NADS has resulted in driver models for responding to various crash scenarios. Data collected from multi-driver simulations would be expected to provide valuable information for more complex and realistic driver models.

1.2 Objectives

The first objective of this project was to design and implement a distributed simulation capability for the NADS MiniSim™ that could also be deployed in the NADS-1 and NADS-2 simulators without much additional effort. The second objective, which was not achieved, was to recruit subjects and run a small study using a distributed simulation scenario.

2 Functional Specification

According to the CEO of Stack Overflow, Joel Spolsky [14], “A functional specification describes how a product will work entirely from the user’s perspective. It doesn’t care how the thing is implemented. It talks about features. It specifies screens, menus, dialogs, and so on.” A functional specification is a living document that should be revisited through the life of a project. This section shows a snapshot of the functional specification for the distributed simulation capability.

Author: Chris Schwarz

Last Updated: October 20, 2016

2.1 Overview

This spec is not yet complete. I expect that several details of the architecture and implementation will be developed over the course of future projects.

This spec does not discuss the details of the networking layer, or even offer a preference for client-server or peer-to-peer. It merely lays out what the final product should do.

2.2 Scenarios

2.2.1 Scenario 1: platooning

We've been hired to study car platooning behavior. To get as many drivers in the platoon as possible, we use the NADS-1, NADS-2, and every MiniSim in the building we can lay our grubby little hands on. Jeff starts the scenario from the control room and, magically, the scenario starts on all the other simulators too. Each driver gets a customized audio cue to start driving at the appropriate time to bring the cars to the right place at the right time.

Using the instructions they have been given, along with the very helpful audio cues they get from their sim, they are able to merge into a platoon with the other participants and transfer control over to an automated driving system. Just as the drivers are starting to think this platooning thing might be kind of useful, a computer-controlled vehicle zooms up and cuts in to the middle of the platoon. The platoon separates, giving this rude computer the space it needs (but doesn't deserve).

Later, drivers receive customized cues to take back control and then depart the platoon so they can catch their exit.

2.2.2 Scenario 2: confederate

We like our computer-controlled artificial intelligent driving objects (ADOs) (really!), but for one study we want the added realism of having a real person controlling some of them. We call this person a confederate (no, not the flag-waving kind). Our unsuspecting participant, Sally, starts her drive and merges onto a long, long highway. Computer-controlled traffic soon surrounds her, and she is now driving on a busy highway.

Our confederate, Billy Bob, is sitting at his sim ready to go. When the participant passes some pre-defined point in her drive, Billy Bob's simulation starts, and he finds himself parked on an on-ramp and is instructed to merge onto the highway too. Since Billy Bob is a confederate, he knows all about what he's supposed to do and which car belongs to Sally. He catches up to Sally's car and cuts into her lane just in front of her. Aghast at the jerk who cut her off, Sally pumps the brakes and utters a few choice words. His job done, Billy Bob takes the next exit, and his simulation stops.

But wait! No sooner had Billy Bob congratulated himself on a job well done, but he's back like a bad penny. His scenario starts up again, and he is sitting on a different on-ramp. This time he knows he is supposed to merge onto the highway in front of Sally, so he waits for the audio cue

prompting him that it is time to roll. Once he hears it, he starts speeding up on the ramp and looks for Sally's car, adjusts his speed, and zooms onto the highway, forcing her to once again slow down. From Sally's perspective, Billy Bob's car is not the same car that cut her off earlier, so she is under the impression that her highway is full of jerks.

This happens a few more times more during Sally's drive. Sometimes Billy Bob behaves like a jerk, and sometimes like a saint.

2.3 Nongoals

It is not our goal to design and implement a networking layer on our own. Initially, we can use the shared memory layer we have. Later we can replace it with a product like VR-Link or VR Exchange.

At this time, we do not plan to implement new types of scenario triggers that could be imagined in a multi-driver environment.

2.4 Configuring a simulator network

Each driver will be linked with an IP address. The scenario software will know how to map position and speed updates from each IP address with the object corresponding to the correct driver.

Each simulator that is equipped for distributed simulation will be able to create a lobby, but only one lobby needs to be created. Other simulators can then join the lobby through a new lobby screen.

Since scenarios will have been created with specific roles in mind for each driver, each driver and IP must also be linked to a specific scenario vehicle. We'll call these vehicles Driver 1, Driver 2, Driver 3, and so on. As simulators join the lobby, they are assigned to a scenario vehicle in the order that they joined. The lobby screen will include controls on the right to move each driver up or down in the list so that they are mapped to the desired scenario vehicle. This operation would normally be done by an operator or a researcher.

2.5 Starting and stopping simulations

Not every simulation needs to start or stop at the same time. Also, we don't want to have to man every simulator with an operator. Therefore, there should be only one simulator that has an operator, and the operator should start the whole distributed simulation by cycling to run, per normal operating procedure.

However, not every driver will automatically be able to start driving at the same time. Scenario timing may dictate that drivers' vehicles get created at specified places/times in the overall simulation. For this reason, the optimal approach may be to use a trigger that "creates" a driver vehicle just like triggers are currently used to create/destroy ADOs.

Practically, the state at a driver's simulator might be held until a scenario trigger "creates" it. At that time, that simulator cycles through initialize to run. Similarly, a scenario trigger can be used to "destroy" the driver's vehicle, at which time that simulator would cycle from run down to off.

Alternatively, if the operator stops the simulation through the operator interface, then the simulation will terminate for all drivers.

2.6 Authoring scenarios

It would be preferable for scenario authors to be able to design all aspects of the scenario in a single scenario file (or with external references), rather than having to create N separate scenario files for N drivers.

Existing triggers can be predicated on environmental conditions (location, global variable), on ADOs, or on the external driver. Predicates should be generalized to include all the external drivers that could participate in the simulation. The total number of external drivers may be planned ahead of time, while authoring the scenario, with the assumption that the correct number of drivers will be participating in the drive.

Just as it is currently possible to rehearse scenarios using simple dynamics and behavior for the external driver, it should also be possible to rehearse multi-driver scenarios the same way.

2.7 Simulator Data

It is OK for each simulator to collect its own data as an approach that requires minimal modifications to the DAQ subsystem. Moreover, each simulator's DAQ file can collect data regarding its ownship normally, i.e., through the cells including VDS_*, SCC_Lane_Deviation, SCC_Follow_Info, etc. The problem of resolving data from multiple drivers can therefore be relegated to data reduction.

Logstreams should end up being identical in each simulator's DAQ file, except for artifacts that may result from a particular simulator's DAQ subsystem. Other environmental variables (SCC_Traffic_Lights, SCC_DynObj_*) are traditionally reported based on some rules with respect to the ownship on that simulator. It is OK for those variables to differ from simulator to simulator as long as it is possible to identify common objects in each DAQ file using their common CVED ID or name, and as long as the data for common objects is consistent across simulators.

This functionality suggests a design in which each simulator runs its own version of scenario software. Then a higher-level scenario arbitration subsystem may be required as a server to all participating simulators.

2.8 Open Issues

1. Do we keep the MiniSim software intact and run a copy of all subsystems on each sim, or do we break up the subsystems and run some of them on a server?
2. Does the simulator with an operator need to be the one who created the lobby? It would be convenient, but is it necessary?
3. Should the lobby cap the number of joining drivers based on some limit that is specified in the scenario?
4. Long delay at prepos. Is a concern if the driver has to wait a long time before he can join the simulation?
5. What happens when one simulator crashes?

3 Technical Development

This chapter addresses technical details of the distributed simulation development. First we present an overall concept for a distributed architecture. Then we analyze the existing structure of the driving simulator software, then measure the gap between what we have and what we want. Next, a practical solution to fill the gap is designed and divided into sub-solutions. We describe each and then describe remaining gaps that are left to future development.

3.1 Distributed architecture overview

The simulator software that runs the NADS-1, NADS-2 and MiniSim simulators share a common architecture and much of the software is exactly the same, making the problem of linking them in a distributed simulation a little easier. The scope of this project is to produce a solution that works only for NADS simulators. The intent of the project is to take a few steps towards a broader capability that supports cross-platform simulation as well.

A hybrid architecture that shares elements of client-server and peer-to-peer approaches was conceived for this project. Rather than run the entire simulation on every simulator, some simulator systems would be aggregated onto a server to help better maintain coherence of the simulation state. Dynamics and visual rendering would logically be the responsibility of each simulator, as would the hardware interface to the steering wheel and other controls. The driver inputs, as well as the state of the vehicle model, would be published to the network.

A few software components work together to accomplish the job of reading logical road information from the world database, managing simulation objects and controlling the scenario. To minimize network traffic, it would be reasonable to keep a copy of the database on each simulator to directly query the terrain elevation. However, the most reasonable approach for the management of the scenario and all its objects seems to be to keep it centrally located on the server side. Additionally, the responsibilities of the scheduler, advancing the simulator frame and transitioning the simulation through its various states and modes (RUN, STOP, OFF, etc.), seem to fit naturally on the server side.

While some components of the current simulation environment might be adapted easily to a distributed solution, others will need to be replaced or upgraded. Objects in the simulation software are classified as either “external” or “internal.” Internal objects are simulated by the scenario, while external objects are simulated in a different subsystem, usually dynamics. The software has the flexibility to easily allow for multiple external objects, as many as are desired, so this functionality may be used. The current implementation of the scheduler exclusively uses the concept of frames; however, a more robust scheduler would make use of a high-fidelity clock from a network time protocol (NTP) server.

A diagram of the distributed simulation concept is shown in Figure 1. It shows two simulators, however more may be added. The server must pass data relating to the simulator configuration, scenario state, object state, and scheduling to the clients. Meanwhile, the clients must send data relating to driver inputs and vehicle state back to the server.

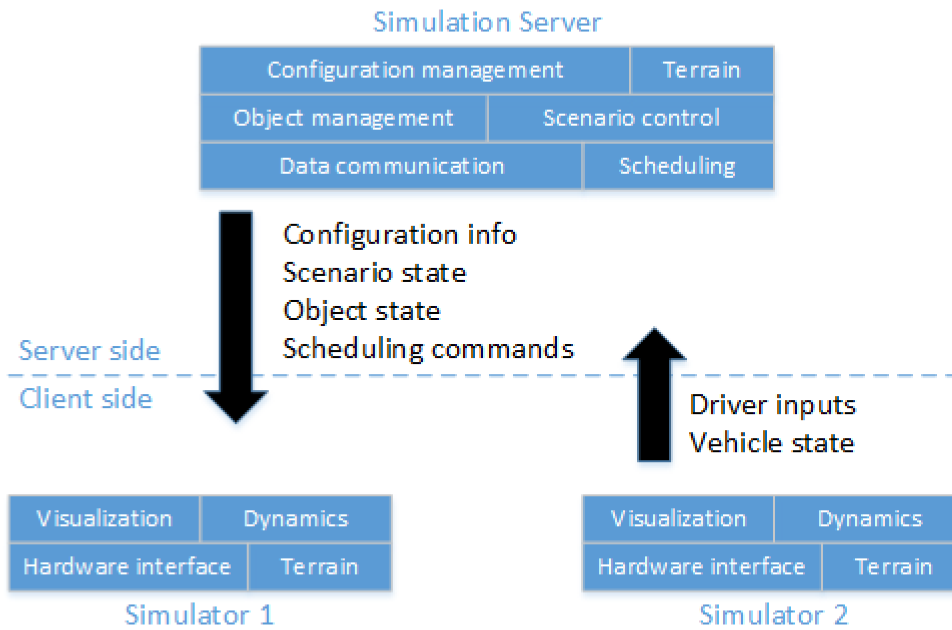

Figure 1. Distributed simulation architecture showing two simulators. The architecture will support more than two, but an upper limit has not been set yet

3.2 Simulation software

The four main components of the NADS MiniSim software that are relevant for distributed simulation are listed as the following:

- CVED: Contains all the objects included in the scenario and provides a dynamic calculation for ADO.
- HCSM: Executes the logic configuration for the scenario, controls the life cycle of the dynamically generated objects, and controls the state of the objects.
- NADSDyna: Calculates the state of the external driving object (drivers' vehicles) based on input from the steering wheels and pedals.
- VisualServer: Visualizes the data contained in CVED.

The overall structure that is relevant to control the scenario is depicted in

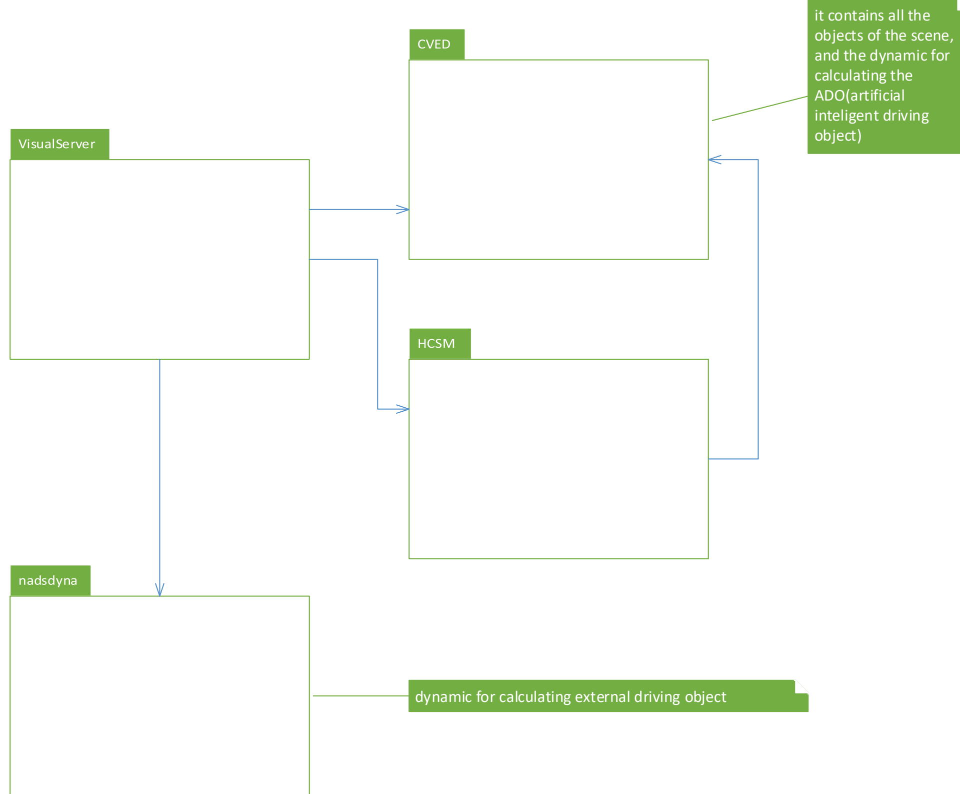

Figure 2. The arrows show the dependent relationships between different components; for example, VisualServer depends on CVED and HCSM, so VisualServer could call the functions provided by CVED and HCSM, while CVED could not call functions of VisualServer, i.e., VisualServer sees CVED, but CVED doesn't see VisualServer.

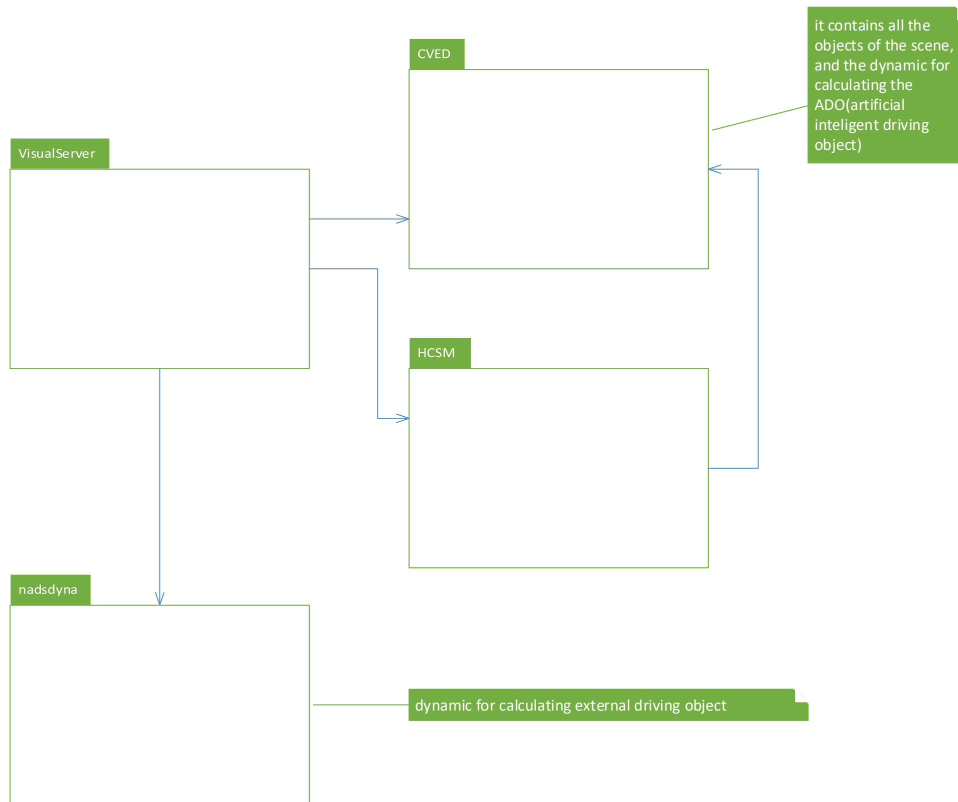

Figure 2. Overview of component structure

The vehicle dynamics used by each driver is provided by NADSDyna, while the dynamics for all traffic objects is provided by CVED. Vehicles are a subset of all dynamic objects that can be created during a simulation. The current existing set of vehicle dynamic objects consists of the following:

- One external driving vehicle (the vehicle controlled by the driver)
- ADO
- DDO (dynamic driving object – no intelligence)

In the connected driving scenario, there would be multiple external driving vehicles. From the perspective of each simulator, there would be one ego vehicle (the vehicle controlled by the driver), and one or more peer-driven vehicles (the vehicles controlled by other drivers). In this new scheme, for a system of N connected simulators, the set of vehicles for each one consists of the following:

- One external driving vehicle (the vehicle controlled by the driver)
- (N-1) peer-driven external driving vehicles (controlled by other drivers) (N: the number of connected simulators driving in a scenario file)
- ADO
- DDO

Information about the external driving vehicles is communicated among simulators using a shared memory approach. The interactions between NADSDyna, the VisualServer and CVED that relate to the external drivers are shown in the sequence diagram of Figure 3.

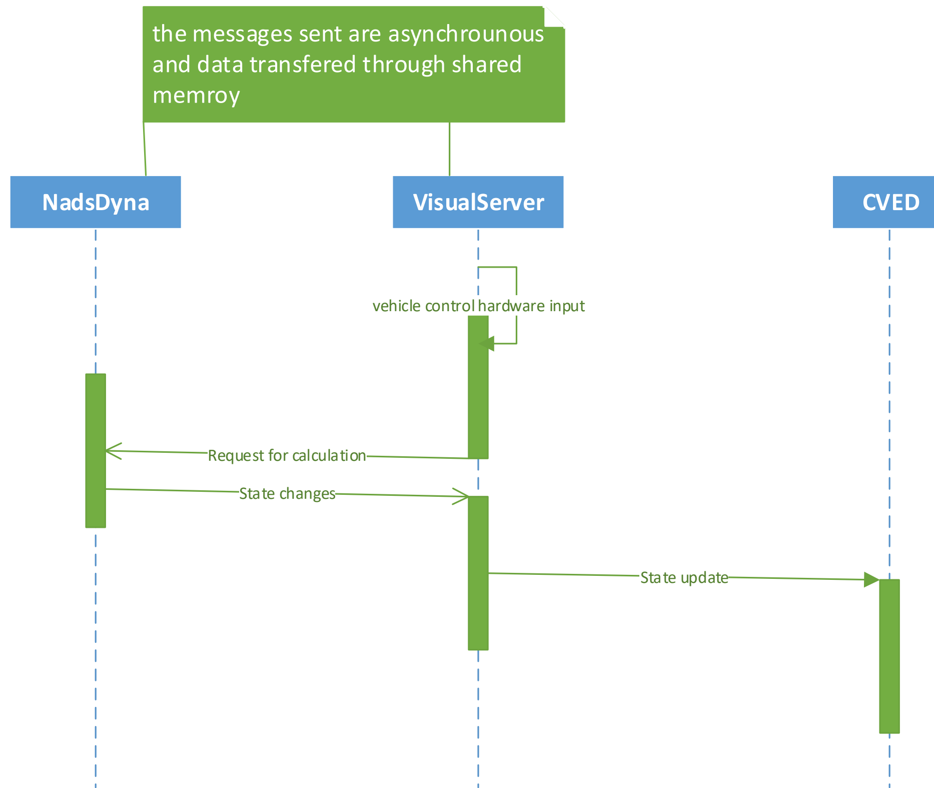

Figure 3. Interactions between components for updating external driving object state

Meanwhile, the ADOs and DDOs are created and managed on each simulator in the distributed system, and ADO and DDO behavior is managed by HCSM, in contrast to external driver dynamics which are managed by NADSDyna. The dynamics are much simpler for ADOs and DDOs than the high-fidelity multibody dynamics models in NADSDyna. The basic state update procedure is shown in the sequence diagram of Figure 4.

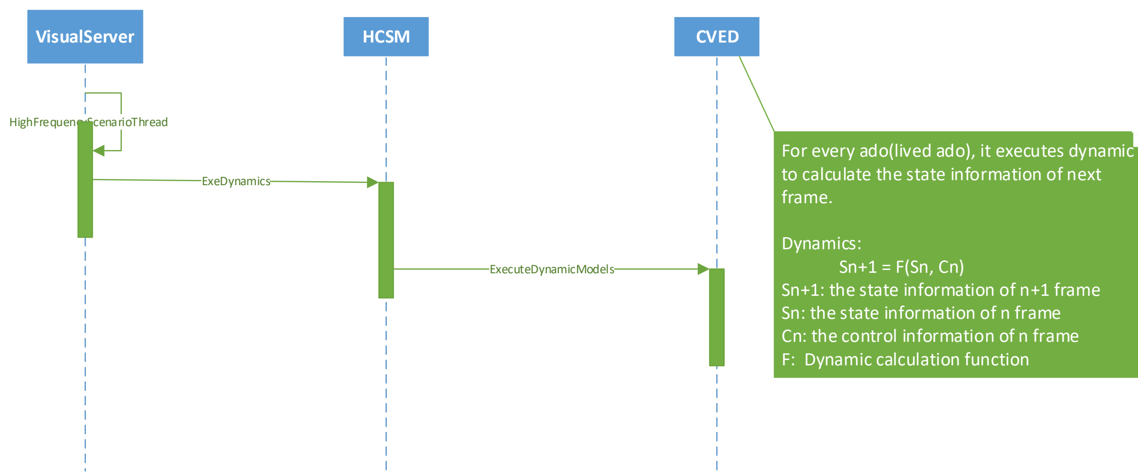

Figure 4. Interactions between components for updating ADO state

Generally, the process of updating the state for dynamic objects can be abstracted to a simple representation regardless of whether it is an external driver, ADO, or DDO. The update procedure could be expressed with the formula

$$S_{n+1} = Update(S_n, C_n)$$

where S is the state, C is the control input, and n is the current frame number. The differences between external driving object, ADO and DDO with respect to this formula are the controlling information and the dynamic calculation. Once the peer driving vehicles are added in, there would be an additional way to perform this update process: rather than doing calculation, the data should be read from the network, and the self-driven vehicle also needs to broadcast its state information in order for peers to read. Thus, the overall algorithm for updating the object states would now be described by the pseudo-code in Figure 5.

```
1  For every dynamic object d do:
2      If d is a self-driven object
3          Sn+1 = Calc_SelfDriven(Sn, Cn)
4          Broadcast the Sn+1 to peer simulators
5      Else if d is a peer-driven object
6          Read Sn+1 of d from network
7      Else if d is an ADO
8          Sn+1 = Calc_ADO(Sn, Cn)
```

Figure 5. Overall algorithm for updating dynamic objects

3.3 Synchronization of ADO and DDO behaviors

The underlying assumption in the approach described above is that all the ADOs and DDOs in the scenario can be managed on each simulator and remain in synchronization with each other. In turn, two lower-level conditions must be true for the assumption to hold. First, the state update procedure must be deterministic. That is, for the same set of inputs, it must generate identical outputs every time and on every simulator. Second, the calls to the state updates along with each object's inputs must be replicated the same way on each simulator.

The work flow for updating dynamic objects is shown in the activity diagram of Figure 6. The first condition, determinism at the function level, can be guaranteed; however, the second one has to do with determinism at the system level and has the potential to cause problems over time. The exact timing of dynamics updates can vary due to different processor speeds and graphic card refresh rates. The result is that some simulators may have more calls to update the states than others. Additionally, minor differences in timing could result in differences in the inputs as the updated positions drift by small amounts. Referring to Figure 6, we can say that the calculations in the green blocks are deterministic; however, we may not be able to guarantee that the activity loop is done the same number of times or at equivalent times on different simulators.

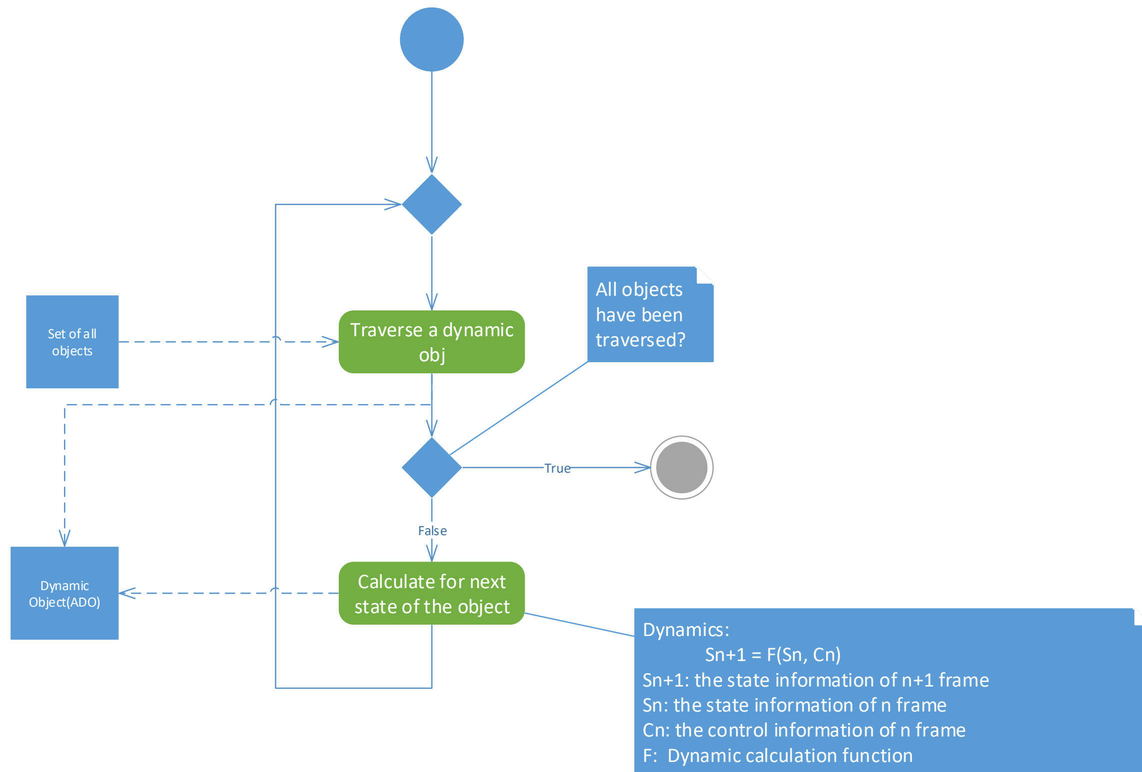

Figure 6. Work flow for updating all dynamic objects

The solution to violations of the assumption is to run all ADO and DDO simulators on a single computer and communicate their states to the simulators over the network. This is similar to what was done with external driving objects. Each externally driven vehicle is controlled by a different instance of NADSDyna, and its state information is broadcast to all the other simulators in the network. A revised update algorithm that accounts for the network activity for external driving objects is depicted in Figure 7.

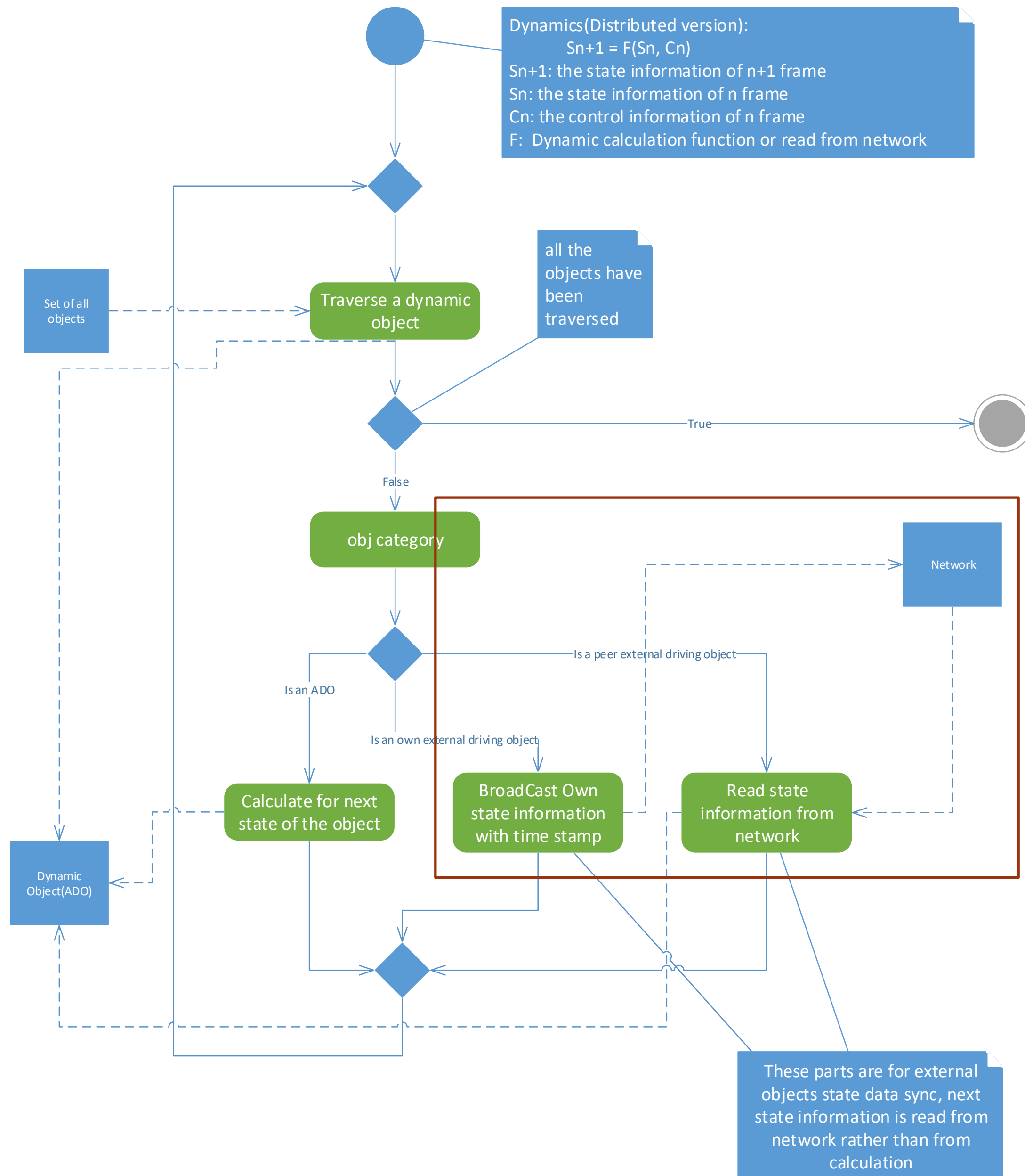

Figure 7. Work flow for updating all dynamic objects

The parts in Figure 7 highlighted with the red rectangle were added to manage the update process for external drivers that exist across a network on a different driving simulator. The implementation was done by adding an external software module to CVED, in which the bulk of this algorithm resides. The addition of this software module adjusts the system overview of Figure 2 to the slightly more complex diagram shown in Figure 8.

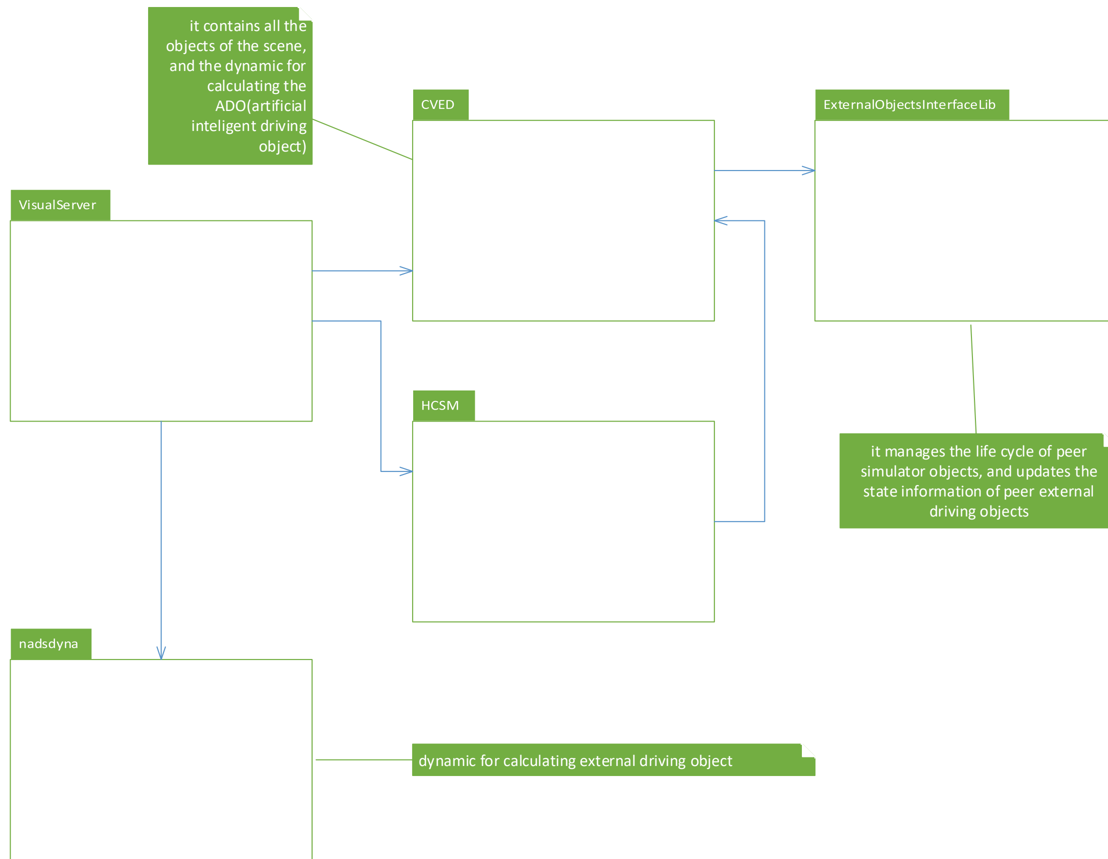

Figure 8. Component overview after addition of ExternalObjectsInterfaceLib

3.4 External objects interface library

This section offers more detail on the implementation of the new external objects interface. It provides the functionality for CVED to update model states that are provided across the simulation network. The network transportation layer was encapsulated so that it could be easily replaced with different implementations when needed. A more detailed diagram showing the interaction between CVED and EmbeddedInterfaceLib is shown in Figure 8.

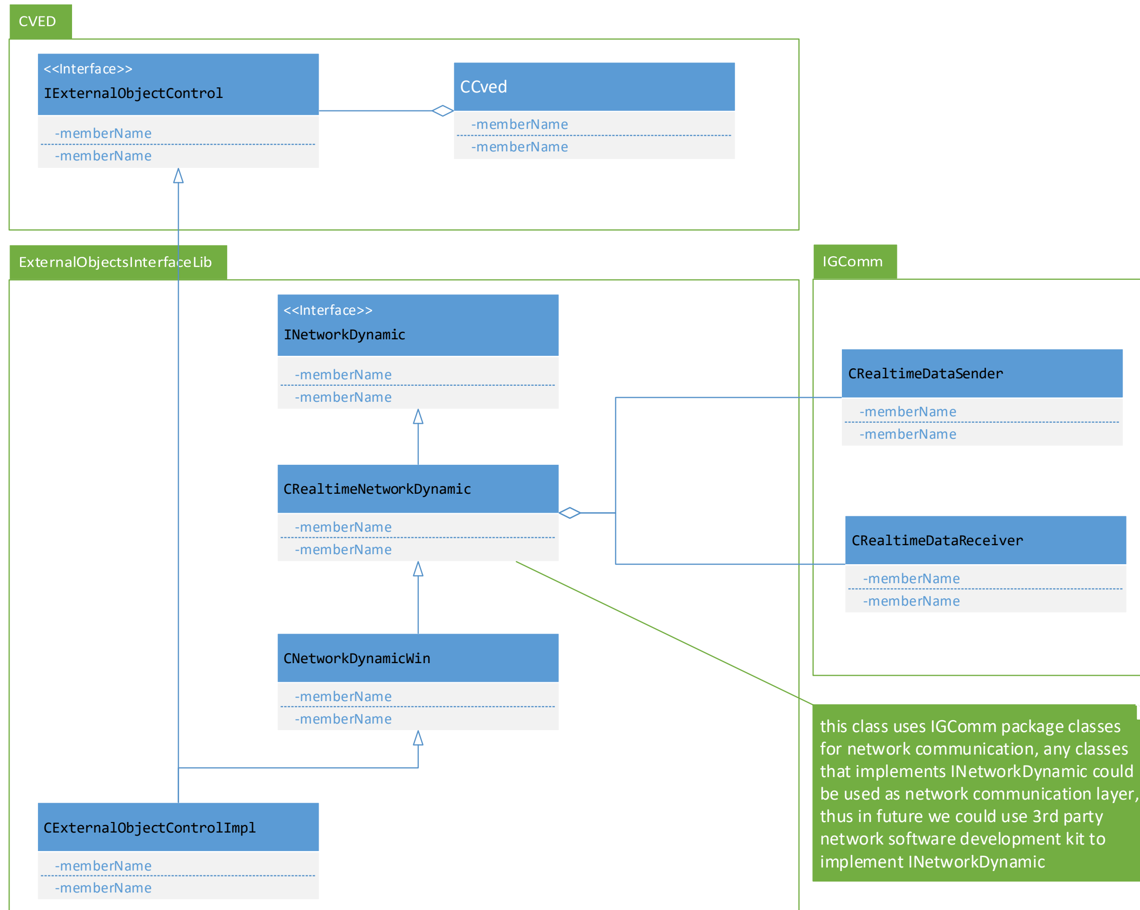

Figure 9. Internal structure of ExternalObjectsInterfaceLib

The network relating process for sending and receiving data works asynchronously. When data is sending, it takes an uncertain amount of time to be received; however, in Figure 9, the receive process is designed in synchronous mode. In order to make the algorithm work in a semi-synchronous mode, a queue data buffer was introduced in which a network listener reacts when data arrives and restores the received data into a queue buffer. This allows reading of object states to work synchronously even though the actual updates to the buffer are asynchronous. If the state is still missing by the time the synchronous update looks for it, then a dead reckoning method can be used to artificially advance the state.

The detail for networking data transportation mechanism is depicted in Figure 10. The implementation is in class CRealtimeNetworkDynamic. The queue buffer was implemented with a min heap, so that the insert could happen in time complexity of O(LogN), pop front in time complexity of O(LogN), and access front in time complexity of O(1). In the following diagram, the three lanes run in parallel, asynchronously.

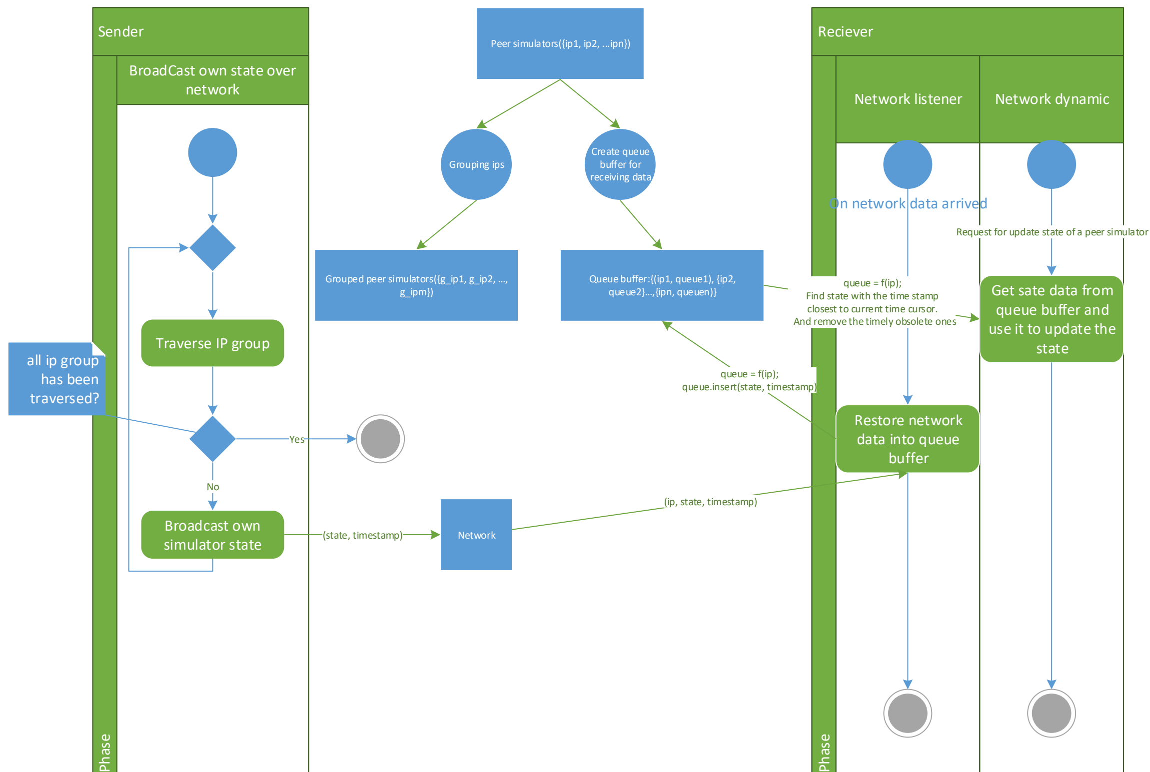

Figure 10. Network data transfer between sender and receiver

The data flow is depicted with green arrow lines, and the control flow is depicted with blue arrow lines. This diagram mixes the data flow diagram and activity diagram, and each of the three lanes is an independent executing unit. A certain event initiates the execution, for example, for “BroadCast Own state over network” lane, the execution is initiated after the driver’s vehicle updates its own state.

3.5 Coordinated universal time

State information coming from peer simulators must be aligned with the ego state information in order to generate a consistent scenario with respect to time; thus, time must be synced across all the simulators as an aligning norm. The problem relevant to time includes two parts:

- Having all computers running in a unified timeline
- Time stamping for the state information broadcasted to peer simulators

The first problem could be solved with an NTP solution. Each simulator would synchronize its time with an NTP server. Since our simulators are running on Windows OS, i.e., all the NTP clients are running Windows, Microsoft NTP would be a primary option. The latest NTP solution recommended by Microsoft involves having an NTP server running on Windows 2006 equipped with a timing appliance (e.g., a GPS device), with NTP clients running on Windows 10 and server and clients existing in a local area network (LAN) environment. We currently work with both NTP server and NTP clients running on Windows 10; the timing error is slim enough to be undetectable visually.

The other problem is the time stamping of the accurate time onto the state information. The precision of Windows is on the order of milliseconds up to tens of milliseconds, and the graphical refresh rate is tied to the display, which makes it around 60 frames per second (fps), resulting in a 16.7 ms frame. Therefore, we need the lower bound on time precision to be on the order of milliseconds as well. The over-bound on time representation is the largest amount of time that can be represented by the software's data type before it needs to reset to zero. If any simulator's time goes over its bound, then it cannot be compared with time on the other simulators. If possible, we should over-bound the precision of time accuracy for the process of aligning the time among simulators.

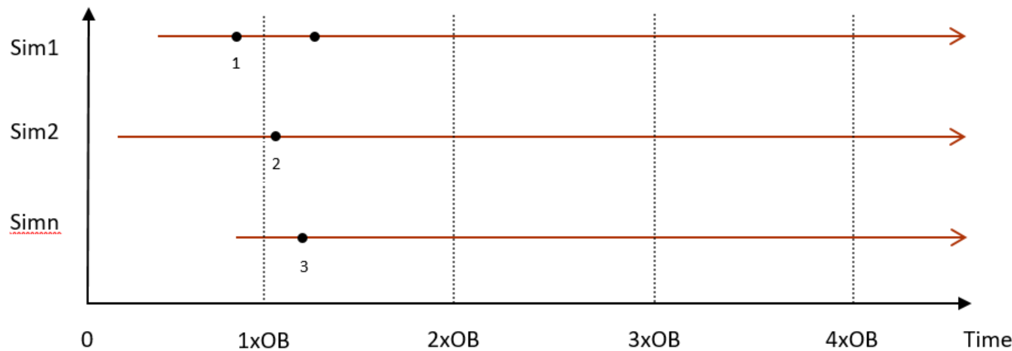

Figure 11. A set of simulators running in universal time norm (OB stands for overbound; the black dots are for time points generating the state information)

There are two solutions to stamp for timing. One is so-called dynamic time stamping: for every time, it generates the state information, dynamically calculates time in bound, and stamps with the calculation result. As depicted in Figure 11, the time points for 1, 2, 3 will be problematic to compare because they are not in same range. In order to solve this problem, we could simply exploit the overbound amount. Pseudo-code for comparing the overbound time is shown in Figure 12.

```
1  bool Less(Timestamp s1, Timestamp s2)
2  {
3      if (number of overbound on s1 < number of overbound on s2)
4          return true;
5      else if(number of overbound on s2 < number of overbound on s1)
6          return false;
7      else
8          return milliseconds encoded in s1 < milliseconds encoded in s2;
9  }
```

Figure 12. Algorithm for comparing time stamps

The other solution is called static time stamping: calculating the starting time and converting it in bound. Every time it generates state information, it calculates the delta in milliseconds since the application started and adds the delta to the original time. This solution ideally has better performance since calculating delta on Windows is faster than calculating system time in bound. In order to solve the overbound problem, we could let the lobby monitor all the simulators and restart them once the time extends over the bound. If we define the overbound limit to be large

enough, for instance a day or a week, it would be very rare for overbounding to become a problem.

The static time-stamping solution was selected from between the two choices and implemented in the MiniSim software. The overbound problem was not solved, since there is not yet a lobby program to manage all the simulators. The overbound limit has been set to one day so that the overbound problem would only occur if the simulation was running at midnight.

The diagrams provided in this chapter model how the system works and how the new distributed simulation features have been added. The code base was written in C/C++, and all new features are also using the same language. The functions that have been implemented do not currently get us to the capabilities described in the functional specification. Features that still need to be added will be described in the next section on future work.

3.6 Future work

3.6.1 Lobby setup program

A lobby setup program is desired to show information about the connected simulators ready to join the distributed simulation system. Similar to networked computer video games, a master is able to set up the game, while other players are able to join the game and wait for it start.

Without a lobby setup program, the configuration for each distributed simulation must be set manually before the start of the simulation. This limitation is not severe for now, and will not become a serious problem until we want to start including remote simulators into the distributed simulation.

3.6.2 Dead-reckoning algorithm

Network latency may cause state data being broadcast to other simulators to be delayed beyond when it is needed. Dead reckoning is a method of compensating for such network delays by predicting what the state should be based on its previous values. Dead reckoning can help to smooth out variable network transmission times.

A simple dead-reckoning algorithm can be created and added to the implementation. Alternatively, a third-party product like VR Link from VT Mak, which provides dead reckoning as a feature, can be used. We plan to try out VR Link since it should help improve the standardization of our solution and prepare us to network with simulators on other platforms.

3.6.3 Centralized HCSM for ADOs

A limitation of the current implementation is that all ADO and DDO objects are simulated on every MiniSim. The assumption is that they will all remain in lockstep with one another during the drive; however, we have discussed why this assumption might not be met due to relative timing inaccuracies. In addition to relative timing differences between simulators, there are parameters for ADO objects that include randomness. The HCSM module is responsible for creating, managing, and finally destroying ADOs and DDOs throughout their lifecycles. By default, HCSM runs separately on every MiniSim in a distributed simulation.

If we move the HCSM module onto a central server and all the ADOs in the distributed system are controlled by a single instance of HCSM, it will keep scenes consistent across all the connected simulators. The consequence of this approach is that the state information for all ADOs and DDOs needs to be broadcast to all simulators, thus increasing the required network traffic by a large amount.

4 Scenario Design

There are a few types of multi-driver scenarios that were identified in the introduction. They take advantage of the unique advantages of multi-driver studies, as well as current trends in connected and automated vehicles. Previously identified types of multi-driver scenarios include the following:

- Multi-vehicle crashes
- Platooning performance
- Mixed automated/traditional interactions

The third item was recognized to require the addition of avatar visualization, which is outside the proposed scope of work. Of the first two, we currently prefer the study of multi-vehicle crashes. There are still many opportunities for multi-driver simulation studies to contribute information about how multi-vehicle crashes develop. Most importantly, it relates to common crash situations that are responsible for taking thousands of lives each year. The study of Cooperative Advanced Driver Assistance Systems (CADAS) is especially relevant as vehicles become more connected. The top pre-crash scenarios for two-vehicle crashes and for the application of V2V communication are listed in Table 1 and Table 2, respectively.

Table 1. Top five pre-crash scenarios of two-vehicle light-vehicle crashes [1]

No.	Scenario	Frequency	Rel. Freq.
1	Lead Vehicle Stopped	792,000	20.46%
2	Vehicle(s) Turning at Non-Signalized Junction	419,000	10.83%
3	Lead Vehicle Decelerating	347,000	8.96%
4	Vehicle(s) Changing Lanes – Same Direction	295,000	7.62%
5	Straight Crossing Paths at Non-Signalized Junctions	252,000	6.52%

Table 2. Top five priority V2V pre-crash scenarios [15] (FYL = functional years lost)

No.	Scenario	Cost	FYL
1	Straight Crossing Paths at Non-Signalized Junctions	20.4%	20.7%
2	Left Turn Across Path of Opposite Direction Vehicle	15.1%	15.3%
3	Lead Vehicle Stopped	14.8%	14.0%

4	Opposite Direction No Maneuver	14.7%	15.1%
5	Lead Vehicle Decelerating	6.1%	5.8%

However, it is important to focus not only on pre-crash scenarios but on aspects of normal driving as well. In numerous NADS studies about collision warning systems, the actions of other vehicles were manipulated to create fertile conditions for a collision to occur. With two human drivers, it is much more difficult to manipulate the scenario. If both drivers are cautious, they may simply slow their speed to avoid the event entirely. Our goal, therefore, should not be to try and make a crash occur, but rather to set up a situation as best we can and measure the perception of risk by each driver, as well as their willingness to take risk. Additionally, driver performance and risk perception is of great interest during normal driving maneuvers that usually do not result in a collision.

The current implementation of the distributed MiniSim offers an interesting solution to the problem of how to set up inter-driver events. A traffic vehicle can be created in one simulator, but never get created in the other(s), simply due to the fact that different drivers take different routes. Suppose Driver A approaches an intersection from the north, while Driver B approaches from the south. Driver B could pass a roadpad trigger that creates a lead vehicle that he then follows. However, the speed of the lead vehicle could be tied to the speed of Driver A, thereby ensuring that Driver A and Driver B arrive at the intersection at close to the same time. Moreover, Driver B's lead vehicle could drive right through the same intersection and never be seen by Driver A since it was only created on Driver B's simulator. In this approach, we have created a "rabbit" for Driver B to follow that allows the scenario designer to manipulate his speed and, crucially, the relative arrival times at an event.

5 Conclusions

This project was the first step towards a distributed simulation capability at the National Advanced Driving Simulator. Supposedly, some allowances for such functionality had been programmed into the software at the outset, and we had the opportunity to evaluate its status, capabilities and limitations. The two main objectives of the project were to develop the capability in the NADS MiniSim software and to conduct a small pilot study using it. Unfortunately, time constraints did not allow us to complete the pilot study. The technical development was successful; however, we identified a few items that had to be left to future development.

The four main components of the NADS MiniSim software that are relevant for distributed simulation are CVED, HCSM, NADSDyna and VisualServer. Three of these exist in largely the same form in the NADS-1, with the VisualServer being replaced by a subsystem called NIX. CVED contained all the scenario objects. HCSM executes the scenario logic and provides behavioral control for dynamic objects. NADSDyna simulates the drivers' vehicles using high-fidelity multibody dynamics models. The VisualServer glues together CVED and HCSM with visual rendering code and an interface to the simulator control hardware – steering wheel, pedals, and other controls.

The distributed simulator concept we initially proposed had NADSDyna, CVED, image generation, and hardware interfaces running on each simulator. The job of HCSM, managing the evolution of dynamic objects and scenario logic, was delegated to a server. The current distributed simulation solution is still implemented as a peer-to-peer architecture. HCSM runs on every simulator; thus, there may be up to N instances of a specific traffic vehicle on N simulators. It is up to each simulator to simulate vehicles with enough determinism for the different instances to remain synchronized. Ultimately, we feel the need to shift to the client-server approach for traffic and scenario management, given the challenges of tight synchronization in complex and long simulations.

A key shift in mindset has been to use timestamps instead of frame numbers to mark the evolution of time in the simulation. Due to minor differences in graphics cards and display refresh rates, frame updates eventually diverge on different simulators. The use of time from a master clock is the only way to provide a global concept of time to the simulator network. We have used the Microsoft Time Server as an implementation of a coordinated universal time (UTC) server. The timestamps are added to the output data file and will replace frame number as the definitive marker of time.

Another benefit of using UTC is the flexibility of starting individual simulators at different times. While having all simulators start automatically at the same time is not that serious a constraint, having the ability to stagger start times opens up new and interesting possibilities. For example, the main subject could be completing an hour-long drive, and a second confederate driver might only be needed for the last five minutes to create a safety-critical event at the drive's end.

The future of distributed simulation at the University of Iowa looks bright with multiple new projects on the horizon. Next steps for NADS distributed simulators include the following development tasks:

1. Develop a lobby setup program to manage configuration and starting/stopping
2. Implement a dead-reckoning algorithm to mitigate timing discrepancies
3. Centralize the HCSM subsystem for efficient and deterministic management of all traffic vehicles and scenario logic

Additionally, steps towards cross-platform distributed simulation will also be taken. Towards that end, we will begin to explore connectivity with the Hank lab's pedestrian and bicycle simulators. The first step will be to use a third-party tool called VR-Link™ from VT Mak. This software may be all that we need to facilitate networking between the NADS simulation environment and the Unity simulation environment. If it proves to be insufficient in some ways, then a more capable product, called VR-Exchange™, is also available for use.

The use of distributed simulation is more important than ever to meet the growing demands on our transportation system. A critical example that is perfectly suited to distributed simulation is studying the interactions between automated vehicles and vulnerable road users, such as pedestrians and bicyclists. This problem is currently getting significant attention from automakers, and standards committees are already considering the question of how to modify vehicle signal lighting.

References

1. Najm, W., Smith, J., & Yanagisawa, M. (2007). Pre-Crash Scenario Typology for Crash Avoidance Research. Final Report DOT HS 810 767. NHTSA.
2. Mourant, R.R., Qiu, N., & Chiu, S.A. (1997). A distributed virtual driving simulator. In *Virtual Reality Annual International Symposium, IEEE 1997*, 208. doi:10.1109/VRAIS.1997.583072.
3. Schulze, T., Straßburger, S., & Klein, U. (1999). On-line data processing in simulation models: new approaches and possibilities through HLA." In *Proceedings of the 31st Conference on Winter Simulation: Simulation—a Bridge to the Future - Volume 2*, 1602–1609. WSC '99. New York, NY, USA: ACM. doi:10.1145/324898.325339.
4. Schulze, T., Straßburger, S., & Klein, U. (1999). Migration of HLA into civil domains: solutions and prototypes for transportation applications. *Simulation* 73 (5), 296–303. doi:10.1177/003754979907300506.
5. Wilcox, P.A., Burger, A.G., & Hoare, P. (2000). Advanced distributed simulation: a review of developments and their implication for data collection and analysis. *Simulation Practice and Theory* 8 (3–4), 201–31. doi:10.1016/S0928-4869(00)00023-9.
6. Hancock, P.A., & de Ridder, S.N. (2003). Behavioural accident avoidance science: understanding response in collision incipient conditions. *Ergonomics* 46 (12), 1111–35. doi:10.1080/0014013031000136386.
7. Mühlbacher, D., Zimmer, J., Fischer, F., & Krüger H.P. (2011). The multi-driver simulator - a new concept of driving simulation for the analysis of interactions between several drivers. In *Human Centered Automation*, 147–58. Maastricht, the Netherlands: Shaker Publishing.
8. Maag, C., Muhlbacher, D., Mark, C., & Kruger, H.-P. 2012. Studying effects of advanced driver assistance systems (ADAS) on individual and group level using multi-driver simulation. *IEEE Intelligent Transportation Systems Magazine* 4 (3): 45–54. doi:10.1109/MITS.2012.2203231.
9. Sawyer, B.D., & Hancock, P.A. (2012). Development of a linked simulation network to evaluate intelligent transportation system vehicle to vehicle solutions. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 56 (1), 2316–20. doi:10.1177/1071181312561487.
10. Prendinger, H., Miska, M., Gajananan, K., & Nantes, A. (2014). A cyber-physical system simulator for risk-free transport studies." *Computer-Aided Civil and Infrastructure Engineering* 29 (7), 480–95. doi:10.1111/mice.12068.
11. Xu, J., & Lin, Y. (2014). A networked multi-drivers simulation platform for interactive driving behavior study." In *Proceedings of the 2014 DSC*. Paris, France.
12. Oeltze, K., & Schießl, C. (2015). Benefits and challenges of multi-driver simulator studies. *IET Intelligent Transport Systems* 9 (6), 618–25. doi:10.1049/iet-its.2014.0210.
13. He, Y., Papelis, Y., & Balling, O. (2004). Distributed virtual proving ground simulation of cooperative agricultural tasks. In *Proceedings of the 2004 IMAGE Conference*. Scottsdale, AZ.
14. Spolsky, J. (2000). Painless functional specifications – Part 2: What's a spec?" *Joel on Software*. October 3. https://www.joelonsoftware.com/2000/10/03/painless-functional-specifications-part-2-whats-a-spec/.
15. Najm, W.G., Toma, S., & Brewer, J. (2013). Depiction of priority light-vehicle pre-crash scenarios for safety applications based on vehicle-to-vehicle communications. Final Report DOT HS 811 732. http://trid.trb.org/view.aspx?id=1249869.