NON-HLA DISTRIBUTED SIMULATION INFRASTRUCTURE

Jan Voracek [a], Jiri Penzes [b], Antonin Kavicka [c]

[a] Faculty of Electrical Engineering and Informatics, University of Pardubice
[b] Faculty of Electrical Engineering and Informatics, University of Pardubice
[c] Faculty of Electrical Engineering and Informatics, University of Pardubice

[a] jan@voracek.net, [b] jirkapenzes@gmail.com, [c] antonin.kavicka@upce.cz

ABSTRACT

Distributed Simulation Infrastructure (DISIS) represents a general layer specialized in communication of distributed simulation engines. The layer supports a multiplatform approach, facilitates the synchronization and communication of simulators (involved in a distributed simulation model) and it is easy to use. The current version of DISIS supports discrete event simulation with conservative synchronization method. Applications of DISIS seem to be quite convenient for industrial applications because of high degree of flexibility.

Keywords: distributed simulation, Non-HLA communication, discrete-event simulation

1. INTRODUCTION

Computer simulation is an effective approach to study the behaviour of various systems over time. Distributed simulation works by distributing the calculation to a cluster of computing nodes that are interconnected in a communication network. This approach has a number of benefits, e.g., the simulation calculation can be easily scaled or, in some cases, more cost-effective resources can be used, etc. The major disadvantage of distributed simulation is the requirement to keep causality which means that the communication between simulators needs to be implemented. The communication becomes nontrivial when the logical processes in the simulation model are implemented in different programming languages / platforms. Each such platform usually supports different high-level communication protocols (like Java RMI, Windows Communication Foundation, etc.) and interconnecting logical processes can become very difficult to implement when a low-level communication (like TCP) needs to be used. In addition, it is also necessary to agree on a synchronization algorithm to preserve time causality of interconnected logical processes. The main goal of DISIS is to minimize these problems. (Fujimoto 2000, Topper 2002)

2. HLA AND DISIS

The proposed general DISIS layer can remind us of High-Level Architecture (HLA). Standard HLA, which was originally developed for military purposes (Kuhl and Dahmann 2000), allows the interconnection of several simulations into a larger simulation unit. It is indeed a very powerful and complex tool. The biggest asset of HLA lies in its universality, which is compensated by strict adherence to standards. The main idea of HLA is a federation – a system that encapsulates all simulators. In the federation, there are described, inter alia, all objects that participate in simulation, or defined methods of data exchange. Every member of the federation – a federate – has to be designed according to Simulation Object Model (SOM), which describes the functionality of the member (Kuhl and Dahmann 2000). Federate also has to be able to receive information from other members, manage its local virtual time and time of other members of federation and also take responsibility for data ownership transfers. This is just a short list of rules of HLA architecture requirements for the federation and its members. The federation has to contain Runtime Infrastructure (RTI) backend layer, which must lead in providing of services needed for running of distributed simulation to all its members. The comparison of HLA and DISIS terminology is in the following table (Kuhl and Dahmann 2000).

Table 1: Terminology of HLA and DISIS

HLA	DISIS
Federation	Distributed simulation model
Federate	Simulator
RTI	DISIS infrastructure
Federation Object Model	Not required
Object Model Template	Not required
Simulation Object Model	API

Compared to HLA, the main idea of DISIS is not to support various combinations of discrete and continuous simulation, different synchronization algorithms etc. DISIS is in this respect way more restrictive, and therefore much easier for implementation. Because DISIS is not so strictly tied to the simulation engine, the software designer of a logical process has much more freedom with regard to the implementation. There is no need to implement some standard or adjust the implementation of a simulation engine to a specific communication layer. That enables to completely replace the communication layer with another. Regarding to their implementation, simulators are not directly dependent on the DISIS layer. Among other things, this feature allows us an easy switch from the monolithic engine to the distributed one with a minimal intervention in the implementation. For example the complexity of a simulated problem may not be well-known in industrial applications in advance. Thus, with DISIS we can start with monolithic simulation and when we need a higher performance or a higher degree of decomposition, we can easily change the standing model to the distributed one. DISIS only provides basic API for message exchange and takes full responsibility for communication. There is no need to design the very simulator according to some standard (see Implementation) (McLeod Institute of Simulation Sciences 2001).

2.1. Other alternatives

Beside HLA there are also other solutions that solve the problem of communication. For example the DDS (Data Distribution Service) or DIS (Distributed Interactive Simulation). Like HLA, they are both very complex. The main disadvantage may be that the serialization is done by the infrastructure, not the simulator. Therefore, the user has to describe all objects exchanged in the simulation by the layer specification. Consequently, such implementation becomes quite difficult (Joshi and Castellote 2006).

3. DISIS

DISIS is implemented as a network of services that are exchanging messages. Simulators of distributed simulation are supposed to be connected to the mentioned services. Each simulation engine communicates only with DISIS service through which it sends messages to other logical processes (Figure 1). This communication runs via given API (Application Programming Interface). The network infrastructure is hidden from the logical processes. Thus, the logical process has no information about the physical location of other processes which it communicates with. When all DISIS services are ready (initialized and paired), each logical process has to obtain a command in order to start the simulation.

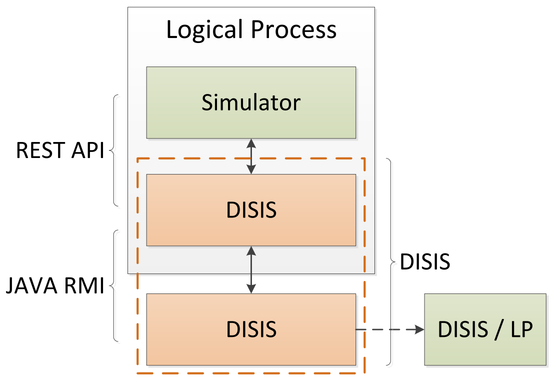

Figure 1: Simulator and DISIS

Every simulator has to communicate with other simulator only through the DISIS service. From the perspective of infrastructure architecture is DISIS a decentralized distributed system that can run on more computing stations. This fact brings us benefits for instance in situation where we have logical processes at geographically distant places (in each location may be one DISIS service) or in the case where we want to control or optimize the message flow.

One case of the use of DISIS infrastructure is a situation where each simulator has its own DISIS service through which it communicates with others (Figure 2).

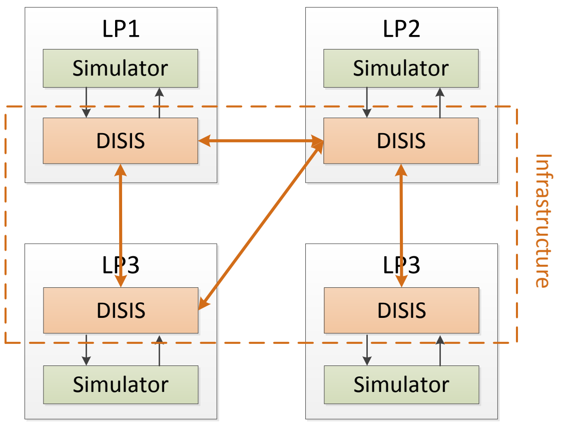

Figure 2: DISIS infrastructure

Figure 2 implies two kinds of messages exchanged in the DISIS network: (i) simulators communicate with DISIS services and (ii) the DISIS services communicate with each other. The simulator has no information about internal messages and cannot interfere with their flow. Internal messages are used only to transfer information between DISIS services. The message contains this set of information:

- sender (DISIS service),
- recipient (DISIS service),
- message timestamp,
- data.

Attribute data usually contains a message sent by logical process during simulation. However DISIS services can exchange messages that are not directly related to the simulation – such messages are called service messages. Amount of service messages in the overall communication is minimal due to optimization. Service messages are sent especially before running the simulation itself because of the initial initialization and exchange information on the locations of all simulators connected to DISIS network.

The DISIS layer brings even more advantages compared to direct interconnection of logical processes. Thanks to the fact that all communications are made via DISIS, it is possible to control the flow of exchanged messages. There is a potential to optimize the communication for maximum efficiency in the large networks (similarly as in computer networks). An example might be two simulators where we know that they will have higher communication overhead during the simulation process. Then it is better if they share one DISIS service (Figure 3).

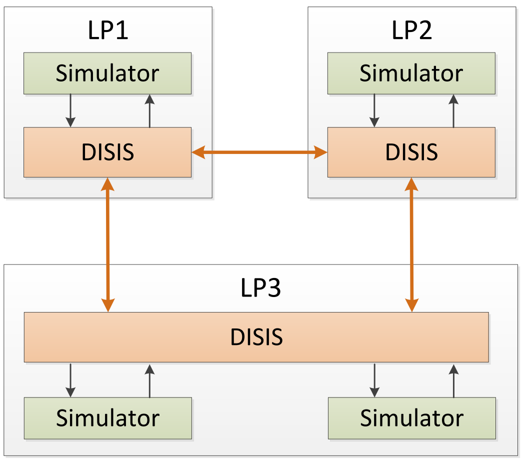

Figure 3: Two simulators sharing one DISIS service

In case of simulation model where we don't expect too high interaction between logical processes or if they are all on local network, we can use a centralized architecture with only one DISIS service (Figure 4).

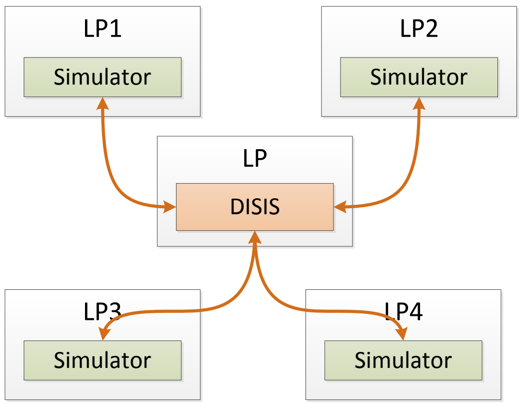

Figure 4: Centralized DISIS network

The DISIS infrastructure architecture is designed by using configuration files. Each DISIS service has its own configuration. It is necessary to have the configuration ready before running the infrastructure. The configuration contains basic information about the configured DISIS service and information about remote DISIS service it will directly communicate with.

The start of DISIS is a very important part. Simulators cannot connect to the DISIS network before all DISIS services are ready. The ready-state in our terminology means the moment when all DISIS services are connected and they established communication with their immediate surroundings. Until this moment, the network is not available for simulators and the simulation cannot be started. The start-up process is realized by these steps performed by each DISIS service:

1. load configuration,
2. initialize local DISIS service,
3. search for remote DISIS services,
4. connect to remote DISIS services,
5. send ready message,
6. wait for simulators.

The connection to a remote DISIS service is successful when the ready message is received. If there are all DISIS services ready, the simulators can connect.

Let's look closer to the realization of distributed simulation using DISIS. Consider the basic simulation model, which consists of two simulators that are supposed to communicate with each other and so create a distributed simulation model (Figure 4).

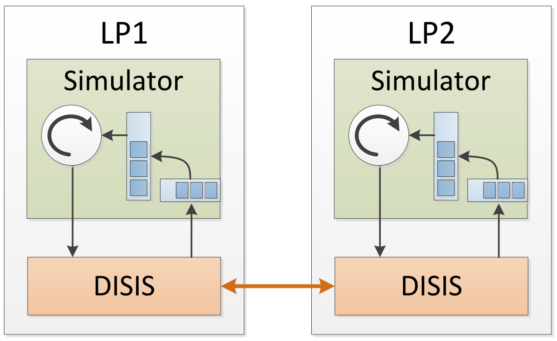

Figure 5: Two simulators

Each simulator communicates through its own DISIS service. DISIS supports only discrete simulation with conservative synchronization method and the actual simulation works on the principle of simulating a life cycle of an entity. In other words, except the timestamp, simulators exchange also an entity and information about what will happen with the entity. In DISIS there is used an algorithm of sending null-messages on request to guarantee the causality. The responsibility of sending null-message requests is taken by DISIS. The simulator doesn't need to worry about sending null-message requests. The simulator works in a standard way – it contains a local event calendar and input queue for every remote simulator it wants communicate to (Figure 5). All messages from simulators go through DISIS, and therefore it is possible to detect the right moment when to send a null-message request (Figure 6). There is one condition for the simulator in order to have everything working all right. It has to inform the DISIS about every change of its local virtual time. If this condition is met, the DISIS knows the exact moment when to send the null-message request.

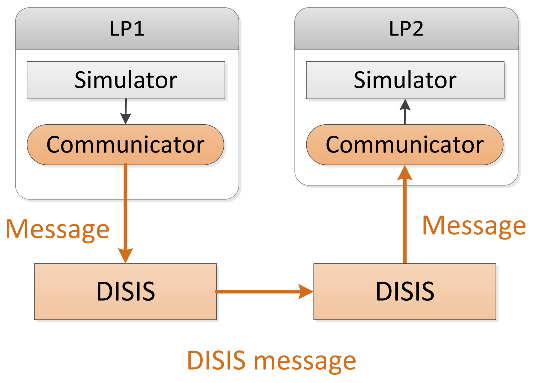

Figure 6: Message transferred from LP1 to LP2

The DISIS infrastructure allows to connect an "observer module". This module can be connected to a DISIS like standard simulator with the exception that it only consumes data (Figure 7).

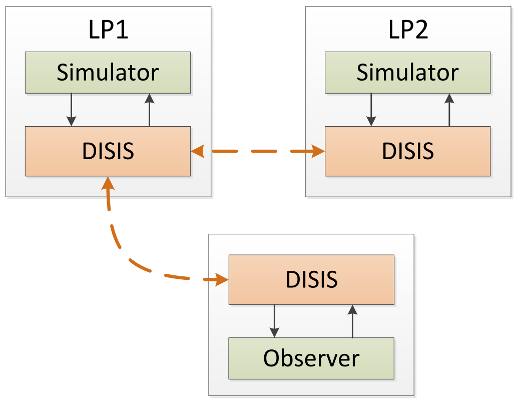

Figure 7: Connected analytic module

This module can be located anywhere. It will be receiving messages from DISIS so it can for example do some analysis of current state of simulation or act like an animation engine.

4. IMPLEMENTATION

The main pillar of DISIS is DISIS-API (Distributed Simulation Infrastructure – Application Programming Interface), which has every DISIS service and through which the simulator communicates with the service. This interface declares four functions:

- connect(RestSimulatorInfo),
- send-message(RestMessage),
- register-observer(ObserverInfo, Observables[]),
- update-simulation-timestamp(RestSimulatorInfo, double).

The connect function has to be called by the simulator as first, because it's used to connect to the DISIS network. Until this function is called, other functions are unavailable. The simulator sends basic information about itself:

- title (name in human-readable form),
- remote-name (unique network name),
- description (short description, not required),
- end-point-address (network address of the simulator),
- surrounding-simulators (simulators with whom it wants to communicate).

The names of surrounding simulators are necessary for monitoring incoming messages. Since we know the timestamps of all messages from remote simulators and the local virtual time of given simulator in DISIS service, we can detect a situation when it is necessary to send null-message request. Thus, the user is partly shielded from the synchronization algorithm.

As its name implies, the function send-message is used to send messages. The message has to contain the following information:

- from (sender – simulator),
- to (recipient – simulator),
- timestamp,
- message (message content).

The function register-observer mediates the connection of observer module. Its parameters are information about the module (like in the case of connect function) and list of message types that it wants to observe.

The last operation update-simulation-timestamp is used to update local virtual time. It is necessary that every simulator calls this function after every change of its local virtual time.

The communication between simulator and DISIS service is implemented using REST-API, since REST is available across many platforms. The disadvantage of this approach is the HTTP on background. Because it is request-response protocol, the HTTP server has to run on both sides – DISIS service and simulator. The simulator has its own REST-API, which supports these four operations:

- start-simulation,
- stop-simulation,
- process-message,
- null-message-request.

The *start-simulation* function is called at the start of simulation calculation – when all simulators are properly connected to DISIS. On the other hand, the second operation *stop-simulation* is used for an explicit stopping of the simulation (such as instruction from some other simulator).

The function *process-message* is intended for receiving messages. DISIS uses this function for passing messages to simulator.

The last function *null-message-request* is used for sending null-message request.

The previous text describes the establishing communication with DISIS and following messaging, which is done through DISIS-API (Figure 8). It should be recalled that the communication between DISIS and simulator is available only when the entire DISIS infrastructure is ready.

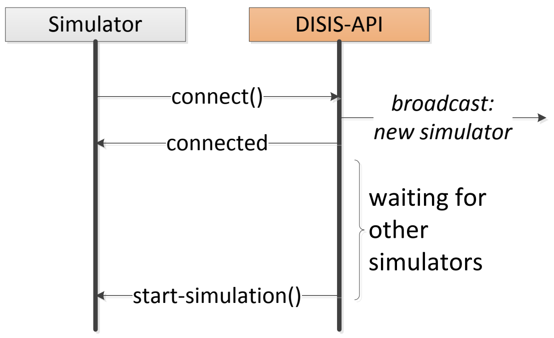

Figure 8: Connection establishing diagram

For the functioning of simulator in DISIS network it is necessary that the simulator meets following requirements:

- It has to send every change of local virtual time to the DISIS service.
- It has to implement REST-API for receiving messages from DISIS service.

DISIS controls the initial impulse that starts the simulation calculation at individual simulators. After running the entire DISIS infrastructure all DISIS services wait for connection of simulators. The DISIS service has no information where the simulators (doesn't know their network location) or what simulators will be connected to it. Information about the connection of each simulator is distributed to all DISIS services. Each service maintains information about which DISIS service is each simulator connected to and the path to given service in the infrastructure (Figure 9). This register is created before starting the simulation.

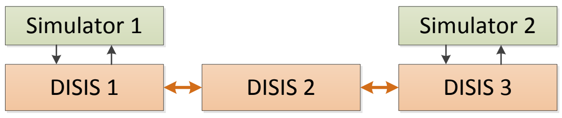

Figure 9: Indirect communication between two simulators

Once all simulators, which given simulator needs for its running, are connected, the DISIS sends its command to start the simulation.

The simulator starts to execute scheduled events and DISIS service updates its information about the simulator's local virtual time and sends null-message requests.

DISIS uses Java RMI (Remote Method Invocation) for internal communication and it is necessary to transform every incoming message from simulator to DISIS message. Thus, DISIS receives message from simulator, transforms it to internal message and looks for service which the target simulator is connected to. The target DISIS service is found in the register mentioned above. Then it sends the message to the target service. The target service transforms the message back (extracts the encapsulated message for simulator) and sends it to the target simulator.

5. CASE STUDY

This chapter will target on a case study, which will reflect a simplified traffic system of a limited scope. There will be demonstrated a process of construction of a simple generic distributed simulation model, that means inputs or outputs of simulation experiments are not closely monitored.

5.1. Crossroad simulation

Let's have a restricted transport segment – crossroad, to which traffic flows from four directions (Figure 10).

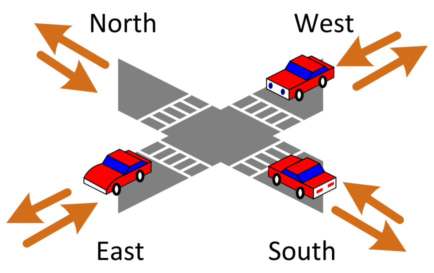

Figure 10: Considered transport segment

The intended case study assumes four traffic input generators and a model of crossroad. One of the possibilities to design an appropriate distributed simulation model is its division into three separated logical parts:

- crossroad,
- traffic – north and south,
- traffic – east and west.

Let's have two simulators that generate, or receive flow of vehicles (in two directions) and a simulator that reflects a model of crossroad. There are several possibilities how to design the communication infrastructure. In this case we used two DISIS services

(Figure 11). One service only handles the crossroad. The second one handles the traffic.

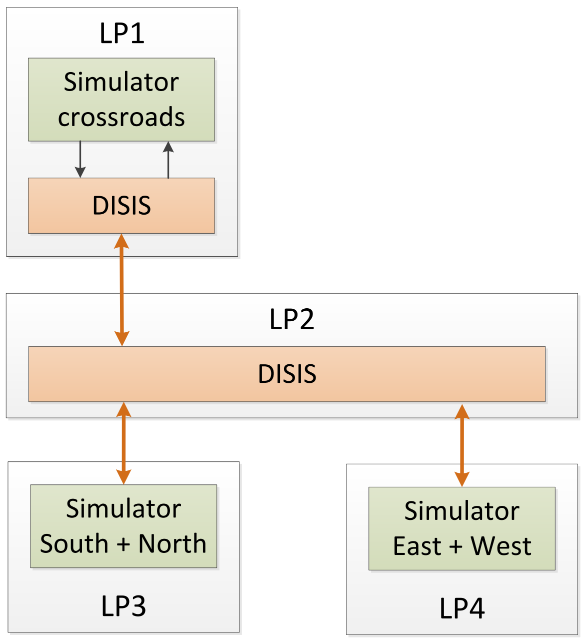

Figure 11: DISIS infrastructure

During the configuration it is necessary to configure properly all simulators and DISIS services. In the intended simulation model we try to find out if it was better to use a crossroad with traffic lights or a traffic roundabout. Replacing the simulator of crossroad with roundabout simulator is very easy. It is possible to use the *observer module* for mining data for analysis, or we can collect the data directly from the simulators.

The described generic simulation model was run in a laboratory of the University of Pardubice – Faculty of Electrical Engineering and Informatics. The computing nodes (on which the logical processes were allocated) were realized by three normal desktop personal computers connected to computer network.

6. CONSLUSION

The paper introduces the problems of communication between simulators across different platforms – especially finding a suitable solution with minimal impact to the original simulator in terms of implementation. The main concept of the HLA standard, which solves this problem to some extent, was also introduced.

Distributed simulation has a number of benefits, e.g., the simulation calculation can be easily scaled or, in some cases, more cost-effective resources can be used. There is a problem if we want to implement the distributed simulation in different programming languages / platforms. The HLA standard could be a possible solution of this problem. However, this standard is very complex and quite difficult to implement. The complexity may not be an advantage when we already have implemented simulators and only want to integrate them to a distributed environment (Fujimoto 2000).

The main topic of the paper is our proposed solution DISIS (Distributed Simulation Infrastructure). The basic concept, main advantages and disadvantages are introduced to the reader.

The issue of different platform is solved by universal API, through which the simulator can communicate with DISIS. All communication between simulators is done through the DISIS infrastructure. This can be used in the simulation process itself – for example to collect data for analysis.

For implementation of DISIS, the Java platform was chosen and the communication interface is designed as REST-API with only small number of necessary operations that are required to run the simulation. It is easy to communicate on almost all commonly used platforms through REST-API. The DISIS requires very little modification of already implemented simulator. Therefore, the integration of DISIS to current solution is relatively simple and fast.

REFERENCES

Fujimoto, R.M, 2000, *Parallel and distributed simulation systems*. New York Wiley-Interscience.

Kuhl, F., Dahmann, J., Weatherly, R., 2000, *Creating computer simulation systems : an introduction to the high level architecture*. Upper Saddle River, NJ Prentice Hall PTR.

Topper, C., 2002, *Parallel and distributed discrete event simulation*. New York Nova Science.

McLeod Institute of Simulation Sciences, 2001. *HLA Module 1. Basic Concepts of the High Level Architecture (HLA)*. California State University HLA Courses. Available from: http://www.ecst.csuchico.edu/~hla/courses.htm [accessed 15 July 2014].

Joshi, R., Castellote, G., 2006, *A Comparison and Mapping of Data Distribution Service and High-Level Architecture*. Santa Clara Real-Time Innovations, Inc.