# INTERACTIVE DISTRIBUTED SIMULATION REALISED AS A WEB APPLICATION

**Stepan Kartak**

University of Pardubice, Faculty of Electrical Engineering and Informatics

stepan.kartak@student.upce.cz

## ABSTRACT

*This paper is focus on a practical use of a web browser and its modern technologies for realisation of distributed web-based simulations. A web application is realised in the environment of a web browser, providing basic software support and additional services for a user-friendly, simple to realise (above all simply programmable) interactive, distributed, discrete simulation. The user is freed from having to deal with the process of creating a simulation core, synchronisation methods etc., and can fully concentrate on the logic of the solved problem. The paper also describes the realisation of the simulation in a web browser, the appropriate classification and extent of applications which are suitable for this service, as well as the positives and negatives of a web-based simulation realisation.*

Keywords: Distributed Simulation, Web-based simulation, HTML5, WebRTC

## 1. INTRODUCTION

This work focuses on use of a web browser to realise a user-friendly distributed simulation. The aim is to create a web-based application which would provide the user with basic functionality (simulation core, synchronisation, interactive approach …) for realisation of logical processes, composition and configuration of a distributed simulation model, central management and record of data. The user is then freed from tackling the aforementioned fundamental problems, allowing concentration on programming which is the actual problem to be solved.
Web browsers are more than suitable for such an application. Browsers have contained functions allowing solving problems of this classification without third-party plugins since the year 2012 (Kartak, 2015). However, in spite of the progress web browsers have made in the last few years, creating distributed simulation was only possible after making a number of compromises. For this reason, we will operate with a distributed discrete simulation made up of a maximum of 20 logical processes (theoretically, the number is unlimited).

A web browser is suitable for the realisation due to many reasons – one of them being that it is available on practically every device which can connect to a computer network. Algorithms of the solution were programmed in today’s well-known programming language JavaScript. This fact is an advantage as well – high availability.

The foundation of the solution is based on previous work, covering distributed web-based simulation (Kartak 2014; 2015). The previous work was aimed at simulators (simulator applications for testing or education, of workers / dispatchers). This solution contained many compromises (e.g. unshared state space), and is surpassed by the introduced work.

## 2. WEB APPLICATION CHARACTERISTICS

The web application allows the user to define a distributed, synchronised space, made up of logical processes available in the form of web pages.

There are three main parts of solution:

- Distributed and synchronised (memory) space: visible to all logical processes it contains. Configurable.
- Logical process type: prearranged web page, which represents a logical process after it has been configured
- Logical process: an instance of the logical process type; a concrete webpage engaged in the simulation.

The realisation itself presents a user with an editor (Fig. 1), i.e. a 2D area which represents a configurable, shared, distributed space, into which logical processes are incorporated. Any number of logical processes may be placed here (realised as a placeholder icon in the editor, with a practical maximum of 20 logical processes). Every logical process represents a workstation or user space. Every logical process has access to the global state of the whole simulation. The application itself provides services of a simulator and automatic synchronisation of logical processes. Furthermore also provides access to configuration of the 2D space, in which logical processes are placed, and in which the simulation unfolds. All functionality has been designed with interactive behaviour in mind.

The logic itself is programmed by the user alone (i.e. the user programs the behaviour of logical processes). Since the simulator itself and synchronisation methods are prepared in the form of a library, the user must create only discrete activities, and in some cases an animated output.
Due to the limited computational capacity of web browsers (see chapter 3) and high requirements on network infrastructure, use on local networks for discrete distributed simulation is expected.

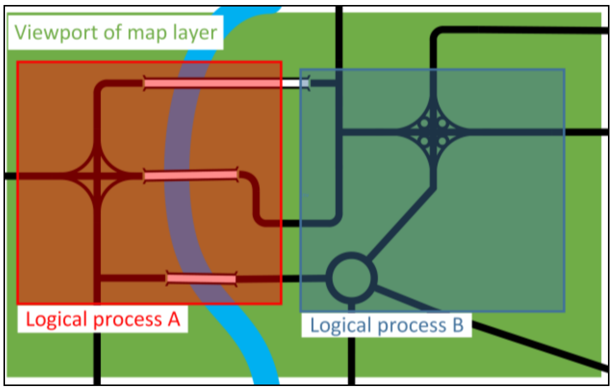


Figure 1: Detail of idea of simulation editor, example of simulation model (the road infrastructure) defined from two logical processes with defined playground 2D area (part of a map).

Considered application/simulation characteristics that can be realised this way:

1. A foundation for „simulator applications“, i.e. applications (interactive simulations) for testing (training, education) of workers / dispatchers. Dispatcher (i.e. one logical process, serviced by a computer/workstation operator). To illustrate, imagine a railway station dispatcher in a region (where there are more station dispatchers). In the same way, a simulation of a technological process / production can be realised. In this case, logical processes represent work areas of individual operators in the technological process. In this scenario, the shared space would represent the scheme of a production process, in which the stations of dispatchers would be located (i.e. logical processes).
2. A simple creation of a multiplayer game, where logical processes represent the area for individual players, and the allocated (configurable) 2D space of simulation represents the game area.
3. Distributed space for data exchange in the frame of a workgroup. This case does not represent an actual simulation. Only the synchronisation methods would be utilised, to keep the memory space for all logical processes up to date.

The primary classification of a possible use can be generally specified as a distributed system which requires an interactive approach, which does not have high computational requirements, and which is based on discrete events.

Interactive approach is represented by user input (keyboard, mouse), which alters the logical process (and the distributed simulation as a whole). The execution of the user input is realised in relation with the online animation output, i.e. the user influences the immediate state of the simulation that is displayed (see Image 9 - screenshot of use case example).

The aim of this work is not to create competition for extensive standards such as DIS, HLA, TENA etc. (IEEE 1278.1-2012; Kuhl et al. 2007), and such a thing is not even possible due to the limitations of a web browser (see chapter 3). The aim is to utilise the availability of a web browser to build a simple, yet functional and rationally available (if we disregard the programming of the necessary logical behaviour) distributed interactive simulation.

## 3. WEB BROWSER ENVIRONMENT SPECIFICATION

A contemporary web browser provides sufficient functional support for the realisation of any application, which would have been, until recently, considered a solely desktop application. To illustrate, take working with files, the 2D and 3D graphic drawing possibilities, and direct server to client communication (WebSocket technology), as well as direct peer-to-peer connection between browsers (WebRTC, W3C 2016). This solution is based on these features. All these new functions are commonly denoted as HTML5.

In spite of the new functions mentioned above, web browsers are still an imperfect platform for demanding applications. The limiting factors essential for this work are as follows:

JavaScript is a programming language which web browsers use to add dynamic behaviour to web pages. It is a weakly (loosely) typed, prototype language, which is compiled at runtime of the script. JavaScript also contains several functions, which complicate optimisation of the compiled code, although this condition is constantly improving with new versions of web browsers. Libraries which compile C/C++ code into a well-optimised JavaScript code also exist. However, a second, greater problem persists – the “single-threaded” (this term is only partially accurate, but covers the essence of the problem, which is why the term is used throughout the text) approach.

A single threaded event-based system of processing user code is a critical problem for a distributed simulation application. These characteristics of JavaScript practically mean that all operations are executed synchronously in a single thread. There is no concurrent multithreaded solution available to users. (Processes the user cannot influence, such as output data rendering, network communication etc. are done asynchronously by the browser.) This does not present a problem to many algorithms that the browser usually realises. But in general, this single-threaded situation makes creating algorithms more difficult, because some problems require parallel operation (in terms of both multi-threaded and multi-processor processing), due to effectivity or time requirements of individual tasks. Specific reasons why this fact presents a serious problem for the implementation distributed simulation are defined in chapter 4.

# 4. IMPLEMENTATION OF A DISTRIBUTED SIMULATION IN A WEB BROWSER

The previous chapter covers JavaScript and its functionality. This chapter focuses on a specific description of the simulator and simulation topology, in relation to the previously mentioned problems, and explains how the synchronisation was done.

## 4.1. Single-threaded web application

The simulator must process four independent critical tasks that are constantly running: (i) simulation core executing discrete events, (ii) network communication (accepting and sending messages) with other members of simulation, (iii) user input processing and (iv) animation output.

These four tasks are usually realised as parallel tasks in classic desktop applications. This is not possible in JavaScript. Because of this fact, the simulator is realised as a series of cyclically repeated instructions (only essential steps are listed):

1. Evaluation of incoming messages.
2. User input processing.
3. Execution of available (especially in relation to the logical process synchronisation) discrete events.
4. Animation output calculation:
   - entities' position,
   - collisions and other interactions between entities,
5. The information about state changes are sent to all other logical processes.
6. Situation rendering onto the animation output.
7. Cycle repeats from step 1.

The algorithm described hints at a problem, which originates in the event-based nature of JavaScript. The event-based approach means that any sequence of commands may be interrupted by another task (user simulation input, accepting messages, etc.). An event-based approach is not a problem in itself, as after an event is processed, the interrupted "main" algorithm continues. The problem is, that these "unexpected" events naturally take some time to be processed, and are thus slowing down the computation, which negatively influences the synchronisation of logical processes, and the animation fluidity as well.

## 4.2. Logical process interconnection topology

Considering the above stated problems with computational capacity of web browsers, realisation takes place as a purely peer-to-peer simulation. Communication between processes takes place directly, without a server. This implementation was chosen in order to: (i) minimise infrastructure expenses and (ii) lower communication load (i.e. latency in communication between individual browsers). The server is used only for setting up the connection and "bookkeeping" of the simulation state.

A one-on-one connection is realised between every logical processes (with the WebRTC technology). During the initialisation process of the simulation, a connection is made between every logical process. The connections are realised, above all, to establish and ensure a global memory state. All changes of the state of a logical process are sent to every other logical process (in fact, changes are broadcasted), and it is up to each individual logical process whether or not the accepted data will be processed, and how. This technique is inspired by the DIS standard.

These general broadcasts are also utilised for synchronisation purposes.

## 4.3. Logical process synchronisation algorithm

The optimistic methods of synchronisation are usually more suitable for interactive simulation, as they do not require strict logical process runtime synchronisation. This synchronisation in turn, makes the calculation (and, by the same rule, animation as well) smoother (i.e. there is no waiting for "slower" logical processes). To ensure a fluent performance (with animation fluidity in mind), the conservative approach is not effective, as it requires a short lookout (look-ahead) to ensure a fluent animation, which increases communication load.

The synchronisation solution was designed with interactivity (the user influences the immediate state displayed in the form of an animation – see chapter 2) of the application in mind.

In the end, a "two-level" synchronisation method was chosen:

1. For basic synchronisation, the Conservative synchronization technique of sending null messages with a look-ahead (Chandy-Misra-Bryant Distributed Discrete-Event Simulation Algorithm, Fujimoto 2000) is used – a specific implementation is defined in the previous work (Kartak 2015). This method is applied, above all, to start the simulation, and to intercept greater-than-average fluctuations (delays) in network communication. Look-ahead of 120 ms is used.

2. For precise synchronisation, time readings of state messages sent periodically (every 40 ms) by every logical process to all other processes in simulation (broadcast message sending). These messages are labelled *broadcast*.
   - The animation output displays a continuous animation of the logical process state. The animation is scaled in terms of time, with 1 second of animation corresponding to X seconds of real time. Considering the user experience and possibilities, a 1:1 ratio is used, where one second of animation corresponds to one second of real time.
   - Considering point one, synchronisation of animation outputs between individual processes is not ensured (look-ahead 120 ms).
   - Information from broadcast messages are used for precise synchronisation (chapter 4.2), which are realised as discrete activities (they are part of the basic synchronisation).

The application allows two ways of synchronisation (depends on the nature of the simulation), which differ primarily in the animation – simulation core relationship (executes discrete activities and synchronisation):

1. **Fully conservative synchronisation:** Time order of activities is strictly upheld, i.e. the simulator awaits null messages if necessary. Animation is executed only in the safe time between discrete activities. This way of synchronisation is completely safe from the time perspective, and allows for acceptable framerates of animation, considering the network latency when transferring the null message request and response averages between 10 – 15 ms on a local area network, and 25 FPS gives 40 ms between each rendering of the animation scene. A disadvantage is FPS limitation and slight “slacking” of the simulation, caused by necessary waiting for synchronisation (during this wait, the animation may not proceed – the simulation time is not advancing).
2. **Minimal synchronisation:** The animation output is not fixed on animation activities. The simulation core works with regard to the time of the animation output, i.e. activities corresponding to the time of the animation output. In this case, the conservative synchronisation method is used primarily for preliminary synchronisation. This method of synchronisation is still in development, but provides more computing power to simulations with focus on interactive behaviour instead of on perfect synchronisation (there is no waiting for synchronisation). As an example (see chapter 6 - use case) the achieved framerate is between 70 and 80 FPS (the minimalistic scene - around 50 animation activities), three times more than the first method. The user can then observe a perfectly continuous animation, eventually there is more space for demanding calculations.

Due to the above stated facts, use only on local networks is assumed, where the latency of WebRTC messages varies between 10 and 20 ms, which is a state allowing fluent operation, assuming the extent of the application.

## 5. BASIC DESCRIPTION OF THE WEB APPLICATION

The web application is divided into five parts. The administrative interface is made up of four parts: (i) model configuration, (ii) simulation management, (iii) visualisation centre, and (iv) an initialisation (signaling WebRTC server, W3C 2016) server. The fifth part is represented by a software library for logical processes. The individual parts are described in the following sub-chapters.

### 5.1. Software library for logical process implementation

The simulation model is made up of logical processes, which are hereby represented by web pages. The logic of these processes is implemented by users. A JavaScript library is available, which provides:

- Connection of the logical process to the administration interface.
- Synchronisation of a running simulation.
- Basic functional support for animation output.
- Auxiliary classes and functions extending the standard JavaScript functions and commonly available JS libraries with practical classes (working with time) and data structures (priority queues etc.)

#### 5.1.1. Basic structure of logical processes

A logical process is made up of 6 parts (for an UML diagram see Image 2, for graphic illustration of relations see Image 3):

1. Simulation core: operates simulation activities, ensures synchronisation. Includes:
   - Calendar: priority queue for simulation activity planning.
   - Environment: contains environment and state information related to the simulation (primarily activity handler).
   - Modules: any named data structure, usually auxiliary, available to all dependent parts (usually activity handler).

Used, among others, for the text report of simulation states.

2. Simulation activity: specified the type of activity, time of execution and any other additional information
3. Activity Handler: execution of given activity type
4. ConnectionRegister: logical process communication realisation layer
5. Animation Activity: Described a graphic element for animation rendering. One of the modules of the simulation core.
6. AnimationManager: renders a scene based on the animation activities

Other program parts that are not critical for the execution of a logical process:

7. SettingsManager: contains a description of the simulation configuration.
8. EntityManager: contains information about entity types and individual entities.
9. ActionManager: describes interactions and eventual reactions of individual entity types.

The solution as a whole works under several basic premises:

- All simulation and animation activities can be serialised.
- All simulation and animation activities can be interrupted at any time (removed from the queue or scene).

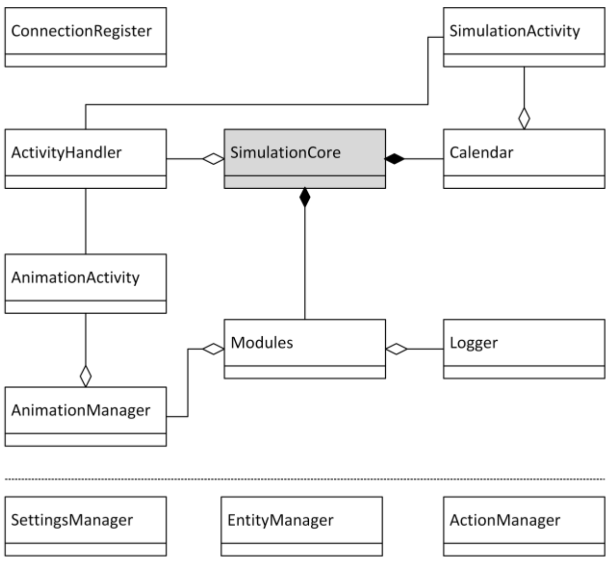


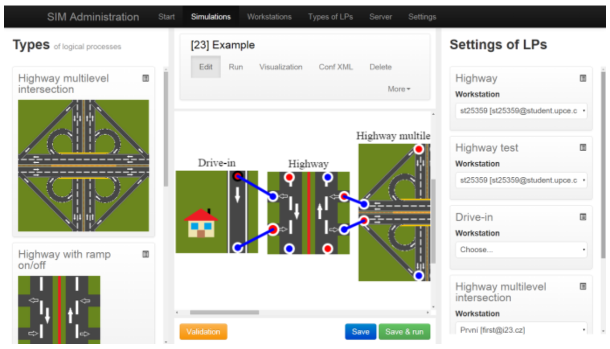
Figure 2: Basic UML schema of logical process (the simulator itself)

## 5.2. Model configuration

This administrative interface serves to register individual types of logical processes, and allows their subsequent use when building the distributed simulation model.

The basic approach to create a distributed simulation is as follows:

1. A user registers a custom logical process type.
2. The visual editor provides a 2D space (the distributed simulation space). Figure 3.
3. In this area, the chosen logical process types are placed (the logical process types are reusable in the frame of a simulation model, i.e. one type of a logical process can be used several times in a single distributed simulation).
   The user configures logical processes, and eventually the simulation itself (i.e. configuration of the 2D shared space) – the specification is done by XML for testing purposes at the moment.
4. Simulation is ready to run. Every logical process has a unique link, which can be sent to a user, who will then operate it (dispatcher, production operator, player, etc.).



Highway
Workstation: st25359 [st25359@student.upce.c

Highway test
Workstation: st25359 [st25359@student.upce.c

Drive-in
Workstation: Choose...

Highway multilevel intersection
Workstation: První [first@i23.cz]

Figure 3: Administration web interface, visual editor; blue lines are network connections between logical processes

## 5.3. Simulation management

To simplify the organisation and launch of a distributed simulation, a module for central management of all logical processes is available (realised as a standalone application).

### 5.3.1. Application characteristic

JavaScript Remote control (JSRC) is an application generally designed to control web pages by text commands in bulk. In this case, it serves to control logical processes. Primarily, it is used to load a logical process, create individual WebRTC connections between all logical processes and launch of all logical processes. It is also used for simulation state accounting. A detailed description of this application is published in previous paper (Kartak, 2015). A brief description follows.

Web pages are identified as *workstations* within JSRC. Workstations are defined by:

- name,
- initialization page (page that will be controlled).

These workstations are grouped into categories of workstations (*workstation groups*). And just within these groups it is possible to enter mass commands (nevertheless it is possible to specify a workstation subset that will receive the command).
Really the workstation corresponds to the logical process and one group of workstations corresponds to the whole model.
The commands are entered by text form (figure 4, Kartak 2015), moreover there is an option to create advance prepared command sets at "one click" (figure 5, Kartak 2015).

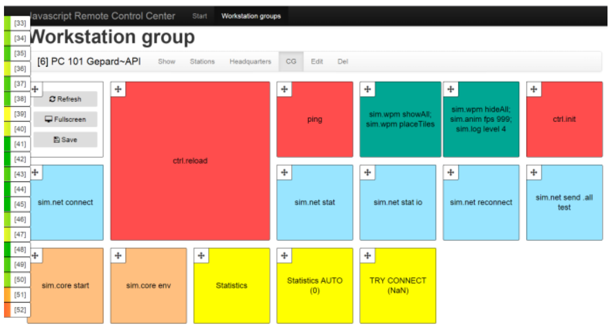


Figure 4: JSRC: Prepared command set, one square is user-defined command (or commands), prepared for touch-devices

The commands are user-defined JS functions of any kind. The use is virtually unlimited. Commands are easily and quickly extensible.

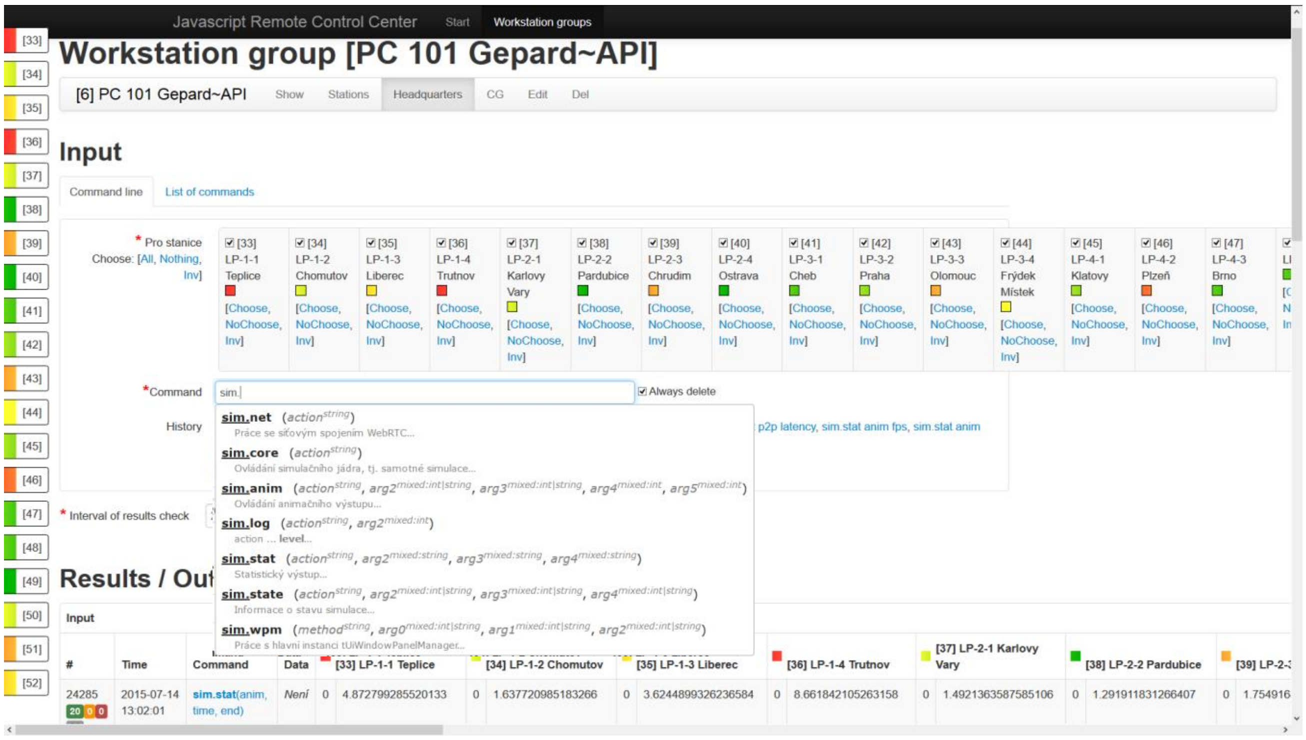
## 5.3.2. Incorporation in the application, API description

In reality, a workstation corresponds to a logical process, and a whole model corresponds to a workstation group.

As stated before, JSRC is an independent application on an independent server, with other applications (here: administrative interface, logical processes, central visualisation) communicating through API, which allows:

- add groups of stations (WG_ADD),
- add stations to groups (WE_ADD),
- add commands (COMMAND_ADD),
- remove commands for processing (COMMAND_FETCH),
- send responses (results) of commands (COMMAND_RESULT),
- get information about commands (primary results, COMMAND_GET),
- get information about station groups (WG_INFO),
- get information about very stations (WE_INFO).

An usage scheme in the simulation is displayed at figure 6, Kartak 2015.


Javascript Remote Control Center | Start | Workstation groups

[33] [34] [35] [36] [37] [38] [39] [40] [41] [42] [43] [44] [45] [46] [47] [48] [49] [50] [51] [52]

**Workstation group [PC 101 Gepard~API]**

[6] PC 101 Gepard~API | Show | Stations | Headquarters | CG | Edit | Del

**Input**

Command line | List of commands

* Pro stanice
Choose: [All, Nothing, Inv]

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ☑ [33] LP-1-1 Teplice 🟥 [Choose, NoChoose, Inv] | ☑ [34] LP-1-2 Chomutov 🟨 [Choose, NoChoose, Inv] | ☑ [35] LP-1-3 Liberec 🟨 [Choose, NoChoose, Inv] | ☑ [36] LP-1-4 Trutnov 🟥 [Choose, NoChoose, Inv] | ☑ [37] LP-2-1 Karlovy Vary 🟨 [Choose, NoChoose, Inv] | ☑ [38] LP-2-2 Pardubice 🟩 [Choose, NoChoose, Inv] | ☑ [39] LP-2-3 Chrudim 🟧 [Choose, NoChoose, Inv] | ☑ [40] LP-2-4 Ostrava 🟩 [Choose, NoChoose, Inv] | ☑ [41] LP-3-1 Cheb 🟩 [Choose, NoChoose, Inv] | ☑ [42] LP-3-2 Praha 🟩 [Choose, NoChoose, Inv] | ☑ [43] LP-3-3 Olomouc 🟧 [Choose, NoChoose, Inv] | ☑ [44] LP-3-4 Frýdek Místek 🟨 [Choose, NoChoose, Inv] | ☑ [45] LP-4-1 Klatovy 🟩 [Choose, NoChoose, Inv] | ☑ [46] LP-4-2 Plzeň 🟧 [Choose, NoChoose, Inv] | ☑ [47] LP-4-3 Brno 🟩 [Choose, NoChoose, Inv] |

*Command: sim.| ☑ Always delete

History: p2p latency, sim.stat anim fps, sim.stat anim

* Interval of results check

```
sim.net   (action^string)
   Práce se síťovým spojením WebRTC...
sim.core  (action^string)
   Ovládání simulačního jádra, tj. samotné simulace...
sim.anim  (action^string, arg2^mixed:int|string, arg3^mixed:int|string, arg4^mixed:int, arg5^mixed:int)
   Ovládání animačního výstupu...
sim.log   (action^string, arg2^mixed:int)
   action ... level...
sim.stat  (action^string, arg2^mixed:string, arg3^mixed:string, arg4^mixed:string)
   Statistický výstup...
sim.state (action^string, arg2^mixed:int|string, arg3^mixed:int|string, arg4^mixed:int|string)
   Informace o stavu simulace...
sim.wpm   (method^string, arg0^mixed:int|string, arg1^mixed:int|string, arg2^mixed:int|string)
   Práce s hlavní instancí tUiWindowPanelManager...
```

**Results / Out**

Input

| # | Time | Command | Data | | 🟥 [33] LP-1-1 Teplice | | 🟨 [34] LP-1-2 Chomutov | | 🟨 [35] LP-1-3 Liberec | | 🟥 [36] LP-1-4 Trutnov | | 🟨 [37] LP-2-1 Karlovy Vary | | 🟩 [38] LP-2-2 Pardubice | | 🟧 [39] LP-2-3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 24285 20 0 0 | 2015-07-14 13:02:01 | sim.stat(anim, time, end) | Není | 0 | 4.872799285520133 | 0 | 1.637720985183266 | 0 | 3.6244899326236584 | 0 | 8.661842105263158 | 0 | 1.4921363587585106 | 0 | 1.291911831266407 | 0 | 1.754916 |

Figure 5: Remote Control web interface

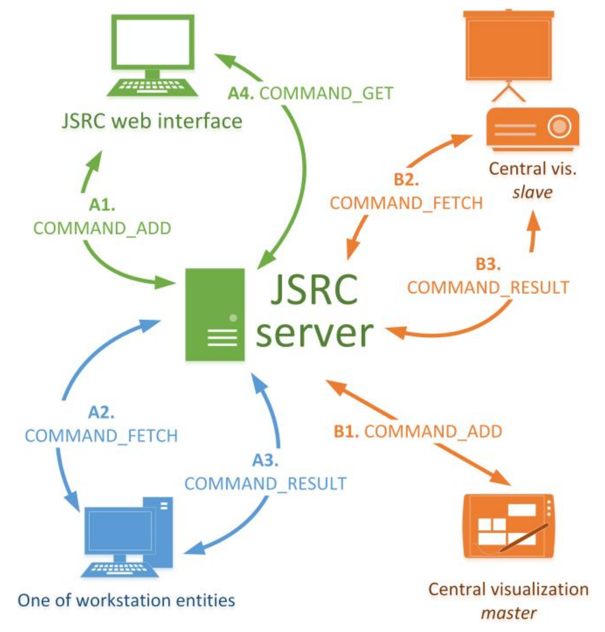


Figure 6: The scheme of usage JSRC API in the simulation

A JS library is available, which implements API calls, which then in turn executes specified actions on the client side (e.g. „Launch logical process“) or on the server side (e.g. „Add Workstation“).

### 5.4. Central visualisation

The administration interface is extended by the central visualisation (CV). This module allows (i) recording of animation output (Image 7) and (ii) capture screenshots for static preview of logical process state (Image 8).

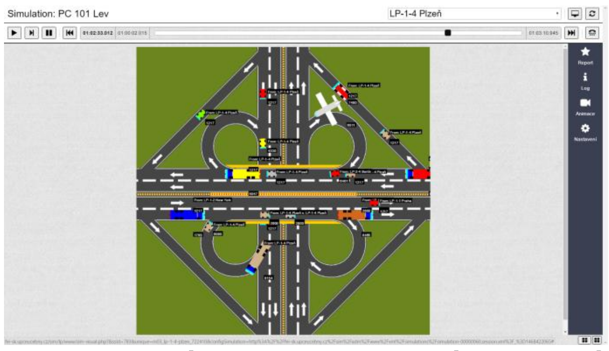


Figure 7: Logical process animation replay in central visualisation

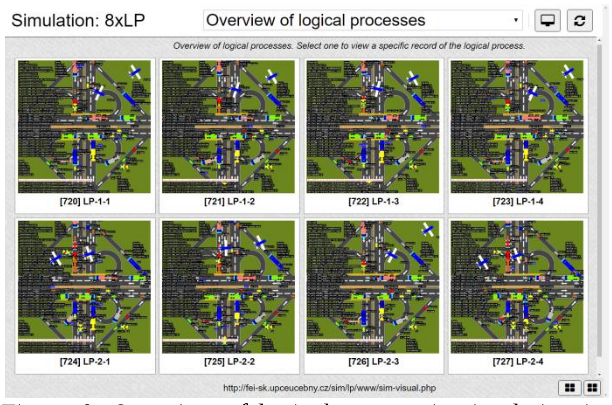

Simulation: 8xLP
Overview of logical processes
Overview of logical processes. Select one to view a specific record of the logical process.
[720] LP-1-1
[721] LP-1-2
[722] LP-1-3
[723] LP-1-4
[724] LP-2-1
[725] LP-2-2
[726] LP-2-3
[727] LP-2-4
http://fei-sk.upceucebny.cz/sim/lp/www/sim-visual.php

Figure 8: Overview of logical process in simulation in central visualisation

Animation output recording is realised as a recording of the animation activities. These activities are distributed in batch into a player (see Image 7). Due to time delays lower server load with increased number of logical processes, it is possible to:

- Observe the animation output of a chosen logical process "on-line" with a 0 to 2 second delay (batch update of the player). Due to conservative method of synchronisation and used look-ahead, it is possible to observe the same scene as the workplace operator (under optimal conditions). Appropriate for e.g. supervising worker during dispatcher training etc.
- Run an animation recording at any time.

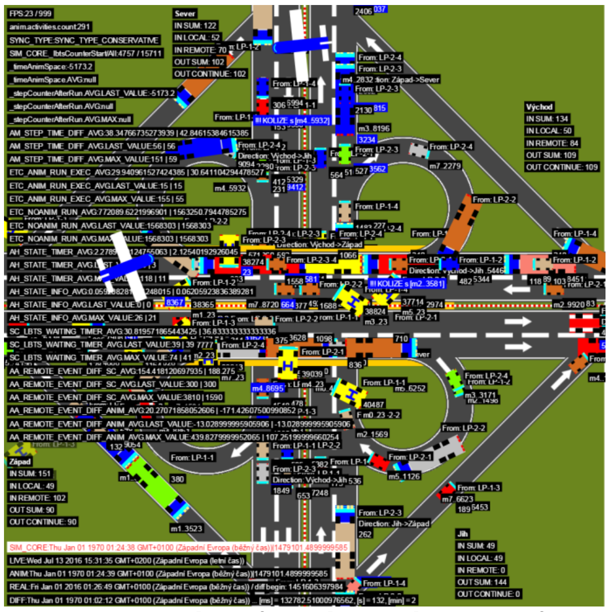
Detailed description is stated in previous paper (Kartak 2015).

### 5.5. Initialisation server

The initialisation (signaling) server is designed to establish a peer-to-peer connection between all logical processes. It is a simple web application, which allows monitoring of connection state (unconnected, negotiation, connected). Meaningful only before a simulation starts.

## 6. TESTING AND USE CASE

For testing, a scenario from previous developing activities was chosen (Kartak 2015). For a logical process scene (screenshot) see Image 9. It is a typical two-level cloverleaf highway interchange).



```
FPS:23 / 999
anim.activities.count:291
SYNC_TYPE:SYNC_TYPE_CONSERVATIVE
SIM_CORE_lbtsCounterStart/All:4757 / 15711
_timeAnimSpace:-5173.2
_timeAnimSpace.AVG:null
_stepCounterAfterRun.AVG.LAST_VALUE:-5173.2
_stepCounterAfterRun.AVG:null
_stepCounterAfterRun.AVG.MAX:null
AM_STEP_TIME_DIFF_AVG:38.34766735273939 | 42.84615384615385
AM_STEP_TIME_DIFF_AVG.LAST_VALUE:56 | 56
AM_STEP_TIME_DIFF_AVG.MAX_VALUE:151 | 59
ETC_ANIM_RUN_EXEC_AVG:29.940961527424385 | 30.641104294478527
ETC_ANIM_RUN_EXEC_AVG.LAST_VALUE:15 | 15
ETC_ANIM_RUN_EXEC_AVG.MAX_VALUE:155 | 55
ETC_NOANIM_RUN_AVG:772089.6221996901 | 1563250.7944785275
ETC_NOANIM_RUN_AVG.LAST_VALUE:1568303 | 1568303
ETC_NOANIM_RUN_AVG.MA[illegible]ALUE:1568303 | 1568303
AH_STATE_TIMER_AVG:2.2789[illegible]124755063 | 2.12540192926045
AH_STATE_TIMER_AVG.L[illegible] | 3
AH_STATE_TIMER_AVG.M[illegible]118 | 11
AH_STATE_INFO_AVG:0.059[illegible]248015 | 0.06205923836389281
AH_STATE_INFO_AVG.LAST_VALUE:0 | 0
AH_STATE_INFO_AVG.MAX_VALUE:26 | 21
SC_LBTS_WAITING_TIMER_AVG:30.81957186544325 | 36.833333333333336
SC_LBTS_WAITING_TIMER_AVG.LAST_VALUE:39 | 39
SC_LBTS_WAITING_TIMER_AVG.MAX_VALUE:74 | 41
AA_REMOTE_EVENT_DIFF_SC_AVG:154.418120697935 | 188.275
AA_REMOTE_EVENT_DIFF_SC_AVG.LAST_VALUE:300 | 300
AA_REMOTE_EVENT_DIFF_SC_AVG.MAX_VALUE:3810 | 1590
AA_REMOTE_EVENT_DIFF_ANIM_AVG:20.27071858052606 | -171.42607500990852
AA_REMOTE_EVENT_DIFF_ANIM_AVG.LAST_VALUE:-13.028999995905906 | -13.028999995905906
AA_REMOTE_EVENT_DIFF_ANIM_AVG.MAX_VALUE:439.8279999952065 | 107.25199999660254

Sever
IN SUM: 122
IN LOCAL: 52
IN REMOTE: 70
OUT SUM: 102
OUT CONTINUE: 102

Východ
IN SUM: 134
IN LOCAL: 50
IN REMOTE: 84
OUT SUM: 109
OUT CONTINUE: 109

Západ
IN SUM: 151
IN LOCAL: 49
IN REMOTE: 102
OUT SUM: 90
OUT CONTINUE: 90

Jih
IN SUM: 49
IN LOCAL: 49
IN REMOTE: 0
OUT SUM: 144
OUT CONTINUE: 0

SIM_CORE:Thu Jan 01 1970 01:24:38 GMT+0100 (Západní Evropa (běžný čas))|1479101.4899999585
LIVE:Wed Jul 13 2016 15:31:35 GMT+0200 (Západní Evropa (letní čas))
ANIM:Thu Jan 01 1970 01:24:39 GMT+0100 (Západní Evropa (běžný čas))|1479101.4899999585
REAL:Fri Jan 01 2016 01:26:49 GMT+0100 (Západní Evropa (běžný čas)) / diff begin: 1451606397984
DIFF:Thu Jan 01 1970 01:02:12 GMT+0100 (Západní Evropa (běžný čas)) ... [ms] = 132782.51000976562, [s] = 132, [min] = 2
```

Figure 9: Screenshot of simulation, the scene of two-level cloverleaf highway interchange is consist from work of 8 logical processes, debug level

The scene is shared between all logical processes, with every logical process generating „its own" vehicles at driveways. All vehicles are displayed in all other logical processes. There are several vehicle types, differentiated by colour, speed and appearance. All of the aforementioned elements are used to create an environment that generates activities and tasks for processing, but are still considered to be „static", non-interactive parts (the user cannot influence them).

The interactive element is represented by a single vehicle, which reacts to user input (keyboard – cursor arrows – changing directions, spacebar – stopping the car, mouse click – sets a specific destination). All changes are transferred to all logical processes, and all logical processes display the shared scene. To further test the dynamic behaviour and possible interactions between individual entities, mouse-based controls check for eventual collisions with other entities at the given destination. The solution was tested on a configuration of 8, 12 and 24 logical processes (each on an individual PC) on a company local area network. The critical computing power pointers are stated in table 1. Measurements took place on identically configured PCs (Intel® Core™ i3-3240 CPU @ 3.40 GHz, 4 GB RAM, Windows 7 64bit, only one application running – Google Chrome browser, version 51).

Table 1: Results of tested use case, lookead 120 ms, critical data for simulation run with focusing on animation output and interactive approach

| Test | #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 | #9 | #10 | #11 | #12 | #13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LP count | 8 | 8 | 8 | 8 | 12 | 12 | 12 | 12 | 20 | 20 | 20 | 20 | 20 |
| Sync method | CONS | CONS | TIME | TIME | CONS | CONS | TIME | TIME | CONS | CONS | TIME | TIME | TIME |
| User interaction | NO | YES | NO | YES | NO | YES | NO | YES | NO | YES | NO | NO | YES |
| Animation FPS | 23 | 23 | 29 | 27 | 24 | 21 | 27 | 25 | 6 | 8 | 8 | 13 | 21 |
| Sync request count | 1256 | 675 | 186 | 142 | 161 | 321 | 137 | 44 | 6123 | 4820 | 108 | 119 | 22 |
| Sync waiting time [ms] | 9 | 8 | 8 | 8 | 8 | 9 | 8 | 8 | 9 | 9 | 8 | 8 | 10 |
| Animation 1 frame draw time (AVG) [ms] | 31 | 42 | 34 | 36 | 38 | 40 | 40 | 45 | 182 | 195 | 119 | 54 | 49 |
| Animation 1 frame draw time (MAX) [ms] | 60 | 62 | 37 | 51 | 55 | 62 | 70 | 68 | 383 | 360 | 150 | 71 | 56 |
| Animation activity count in animation scene | 266 | 292 | 215 | 245 | 270 | 245 | 278 | 266 | 1050 | 1120 | 758 | 480 | 300 |
| FPS per animation activity | 0,08 | 0,08 | 0,13 | 0,11 | 0,08 | 0,08 | 0,08 | 0,07 | 0,005 | 0,007 | 0,1 | 0,03 | 0,07 |

**Result notes (table 1):**

- Two types of synchronization were tested (see chapter 4.3) - marked as CONS and TIME.
- User interaction (mouse click) was programmatically generated. Calculated by uniform distribution (min 400 ms, max 3000ms) between simulated interactions.
- One animation activity is consists from 10 graphics base elements / primitives in average.
- Results were collected after 5 minuts (real time) run.

**Summary of results:**

- *Sync waiting time [ms]:* The time required to send a request for synchronization and recieve all required data is almost constant.
- User interaction generated broadcast messages and this is reason of decreasing number of synchronization requests. Negative is lower FPS, becouse every interaction must be handled.
- Critical for fluent performance (with animation fluidity in mind) is count of animation activities. FPS is directly dependent on their number (row *FPS per animation activity*).

## 7. CONCLUSION

The presented solution is still being developed, but the class of applications, for which it will be beneficial, can already be defined. Specifically, interactive distributed simulations, which do not require calculation-heavy operations (e.g. real-life object representing entity movement dynamic calculations, expansive animation scenes). Due to the single-thread nature of JavaScript, simulations with large amounts of user input in short time intervals (generally tens in a second) are not suitable, because JavaScript will be busy with dispatching these events, and will not have time left for calculation of other critical parts (activity calculation, animation).
Considering the aforementioned facts, the primary motivation for use of web-based simulation is the availability of the runtime environment – web browser – on any computer or modern device connected to a computer network. JavaScript is very well supported by modern-day browsers, and is extensible and well known. This comfort of availability and simplicity is not without a cost – when compared to native applications, the scripts are slow.
The solution as a whole is still not finished, Lots of optimisations could be done on the administrative interface, and integration of the individual parts of the solution, which are at this time resolved by external applications (JSRC, RTCC, VC), primarily because of ongoing development. Also, if the JavaScript runtime seems too slow, the logical process software library can always be optimised at source code level.

## REFERENCES

Fujimoto, Richard M. Parallel and distribution simulation systems. New York: Wiley, 2000. Print.

Kartak, Stepan, and Antonin Kavicka. “WebRTC Technology as a Solution for a Web-Based Distributed Simulation”. Proceedings of the European Modeling and Simulation Symposium 2014. Genova: Università di Genova, 2014, s. 343-349.

Kartak, Stepan. “Web Simulation as a Platform for Training Software Application”. Proceedings of the European Modeling and Simulation Symposium 2015. Genova: Università di Genova, 2014, s. 70-78.

Kuhl, Frederick, Judith Dahmann, and Richard Weatherly. Creating computer simulation system: an introduction to the high level architecture. Upper Saddle River, NJ: Prentice Hall PTR, 2000. Print.

Tropper, Carl. Parallel and distributed discrete event simulation. New York: Nova Science, 2002. Print.

Hridel, Jan, and Stepan Kartak. “Web-based simulation in teaching”. The European Simulation and Modelling Conference 2013. EUROSIS-ETI, 2013. Print.

The Institute Of Electrical And Electronics Engineers, Inc, 2012, 1278.1-2012: IEEE Standard for Distributed Interactive Simulation - Application Protocols. New York; IEEE. 2012.

W3C, 2016. WebRTC 1.0: Real-time Communication Between Browsers. Available from: https://www.w3.org/TR/webrtc/ [Apr 2016].