WEBRTC TECHNOLOGY AS A SOLUTION FOR A WEB-BASED DISTRIBUTED SIMULATION

Stepan Kartak[(a)], Antonin Kavicka[(b)]

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

[(a)]stepan.kartak@student.upce.cz, [(b)]antonin.kavicka@upce.cz

ABSTRACT
The modern web browser is a runtime environment which aspires to replace original (native or desktop) applications. This article describes a new HTML5 technology called WebRTC, which enables a direct connection between browsers and which enables to perform a peer-to-peer network connection, which is suitable for the creation of a distributed simulation model. We compare this new technology with original web-based distributed simulation solutions – implemented using applets – and present one of the possible approaches to distributed simulation model creation.

Keywords: web-based simulation, WebRTC, discrete-event simulation, HTML5

1. INTRODUCTION

Web browsers are part of daily life today. During the past few years, web browsers have grown enormously; standards have been unified – especially JavaScript – and the present-day web browser succeeds as a platform for a wide range of applications which used to be implemented as so-called desktop applications, which are bound to the operation system, processor architecture, etc. The web browser overcomes these dependencies and represents and ideal multi-platform runtime environment, which, together with extended HTML5 support, represents minimum restrictions to the deployment of applications which could only be implemented as desktop ones in the past.

Since the second half of 2013, major web browsers have supported WebRTC (included in HTML5), which enables to initiate a peer-to-peer network connection between browsers (clients), and thus to perform a smooth distributed and decentralized simulation. It is this type of simulation that we elaborate on.

The article also covers available competing technologies, the technology and use of WebRTC, and describes a practical implementation of distributed simulation models running in the web browser.

The aim of the solution is not to compete with the existing HLA solutions, but to present new possibilities opened up by modern web browsers, and point out their advantages (as well as disadvantages). This article is a follow-up to a previous work (Kartak 2013), where a web-based simulation was implemented using Java Applets (which was the only possible solution at that time). The article concludes with a simple comparison of the two solutions.

2. AVAILABLE TECHNOLOGIES

The aim of a web-based simulation is to create a browser ecosystem where the deployment of external elements (applets, extensions, etc.) is reduced to minimum.

Distributed compute nodes require a bi-directional communication between individual logical processes in the network. This purpose is best served using the peer-to-peer architecture (figure 1).

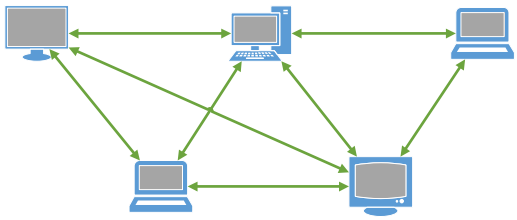

Figure 1: Peer-to-peer Network Topology

2.1. HTML Before Introducing HTML5

As mentioned in the previous paragraph, bi-directional communication is necessary. In the past, it was the absence of this functionality that did not allow for this type of connection among devices (typically, browser-browser or browser-server).

There were several solutions dealing with this issue. The simplest one, exclusively at the level of browsers, was accumulating client requests in a queue on a server, which was accessible to all clients involved in the communication, where the requests were waiting for the retrieval by the original target client (browser), see Figure 2.

This way, *defacto*, we pass from the peer-to-peer communication to the client-server one. It follows that this solution can be considered as an “emergency” one and thus not suitable in general. The use of a server within a distributed simulation may have its purpose; however, it is only complemental to the planned simulation topology. Using a server is purposeful for on-line gathering of information regarding the system behavior – statistics, animation output, etc.

Although it works, this solution brings several disadvantages (the major ones are listed below):

- It requires an efficient server (a network element which is virtually missing in the peer-to-peer network architecture).
- As a rule, it requires a low response rate of the client connection. In this case, requests are repeatedly sent to the server (requests repeated after several milliseconds). Most of the requests sent to the central server element are useless, yet they are necessary for the required low response rate of the whole system (in our case, a simulation). This considerably increases the communication traffic in the network.

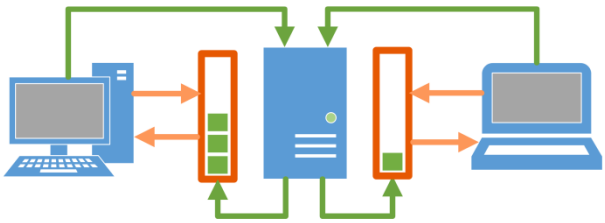

Figure 2: Message Queue For Clients On Server

2.2. The Use Of Applets

In the past, the only possibility to deal with the above-mentioned issue was the use of applets. In web browsers, it was the use of an external application (e.g. Java code running in Java Virtual Machine) in the context of a web page. (Byrne, Heavey and Byrne 2010)

All commonly available applets (Java Applet, Adobe Flash Player, Microsoft Silverlight) include features to perform the peer-to-peer connection; however, most of them (with the exception of Adobe Flash Player) are "process virtual machines", which usually have direct access to the host computer. This is very dangerous, as applets can be loaded from any web page, and thus allow an unauthorized user to access the computer. These applets deal with this issue by using so-called policy files – files which contain a security policy definition (i.e. a definition of enabled and disabled features or operations and network access). In these files, it is necessary to explicitly allow client call from a specific server/client, or more generally defined groups of servers/clients often called "domains". In addition, the above-mentioned must be allowed by the user. The communication is often blocked by a firewall or another security feature in the target computer or in the network.

The use of applets is easy due to the well-known languages (typically Java and Java Applet); however, in general it is not suitable because of a non-trivial use and network communication safety issues.

The most applicable out of the above-mentioned applet types is Java Applet; its safety policies enable a smooth bi-directional client-server communication (with the server from where the applet was loaded). This, of course, contradicts the notion of the peer-to-peer connection, and does not bring any vital benefit when compared to the above-mentioned solution based solely on HTML.

All these applet-based solutions had to deal with connection safety issues (connection blocked by firewalls, etc.) and incompatibility among browsers, platforms, or operating systems. (Martin, Rajagopalan and Rubin 2013)

2.3. New HTML(5) Possibilities

Together with new technologies (often called HTML5), HTML allows all necessary communication in the network:

- Download and upload data (standard browser features even before HTML5).
- The WebSocket technology allows for performing real bi-directional client-server communication.
- WebRTC technology allows bi-directional communication directly between browsers – real peer-to-peer communication.

This solves the issues related to network communication for the needs of distributed simulation (see Table 1 for the overview of the availability in web browsers). For a detailed description of HTML5 network technologies, refer to Chapter 3.

Table 1: The HTML5 Network Technology Availability In Major Web Browsers

Web Browser	WebSocket	WebRTC
Chrome	14	23
Firefox	6	22
Internet Explorer	10	-
Opera	12.10	22
Safari	6	-
Available since	May 2012	July 2013

HTML5 brings further useful technologies which find their use in a web-based simulation:

- *Canvas:* Allows for (mainly vector) 2D drawing using JavaScript. A crucial disadvantage of Canvas is the necessity to always redraw the whole scene (there are some mostly "caching" techniques to minimize the problem to a certain degree).
- *SVG:* Allows for drawing (and animation) using a declarative HTML-like approach.

3. HTML5 NETWORK TECHNOLOGIES

3.1. WebSocket

WebSocket technology enables us to perform a network connection correspoding to the standard behavior of desktop applications, familiar to us for years. Network sockets establish a bi-directional connection between two applications (client 1 can contact client 2 and vice

versa). In terms of web applications, it means that a web page (client 1) opens a connection with a server (responding to client 2) and this connection is bi-directional – there is no longer a queue (see Figure 2), the server sends messages directly to the client via the opened connection.

This solution, of course, requires a server. The server side does not have to meet any special requirements; it only needs to follow the WebSocket protocol. We also have at our disposal a wide range of ready-made open-source tools, from PHP (e.g. the Ratchet library), via Python (e.g. the Tornado framework) to Java (e.g. TooTallNate) or C# (the Alchemy WebSockets library).

3.2. WebRTC

At first it is crucial to mention that the development of this technology is still in progress. According to W3C, WebRTC is in the "Working Draft" phase (as of September 10, 2013, see References), and the behavior of some features in different browsers may not be 100% correct, or different browsers have implemented these features in different ways. Most of these issues can be solved using a JavaScript solution, which overcomes the existing browser-specific differences (at present only minor differences among browsers).

This technology crucially enhances web browser capabilities in terms of network communication. It is possible to perform the peer-to-peer connection without a server (a server is only required to initiate the connection, which is, of course, standard for peer-to-peer communication). To solve routing issues when communicating with a client in a local network using NAT, WebRTC implements directly the use of the ICE (Interactive Connectivity Establishment) protocol. More information is provided in the following chapter 4.2.

WebRTC transfers data in two ways:

- MediaStream – used for audio and video streaming.
- DataChannel – used for text message transfer – it is this type that we used for the communication in the simulation.

DataChannel (uses the SCTP protocol for the communication between clients; this protocol allows for optional reliability settings). The reliable variant ensures that the message is delivered to the addressed clients (reliability corresponds to TCP); in the opposite case, the delivery is not quarranteed (or, only a limited number of callbacks is quaranteed; this type is similar to the UDP protocol).

4. NETWORK COMMUNICATION ISSUES IN A PUBLIC NETWORK

In this section, we elaborate on the two major and restrictive issues which need to be considered when communicating in the network (not only in the web browser).

4.1. Same-Origin Policy

Web browsers require following the same-origin policy. In practice, a problem arises when JavaScript needs to call the source – usually to download data – from a server which is located in a different domain than the one from where the page was loaded.

This way, browsers prevent the cross-site request forgery (CSRF or XSRF) attack, where the local script (loaded on a currently viewed web page) might call a script (or data containing executable code) which is not under the control of the application author (is located in another domain), and thus may present a potential threat.

We are likely to encounter this issue if we want to incorporate a logical process (performed using a web page) into a web-based distributed simulation, where the simulation is located in a different domain than the domain where e.g. the initialization server is located.

There are several solutions which can perform this type of communication. The simplest and most dangerous solution is an explicit disabling of the cross-origin request blocked (CORB) in a web browser. The best and most straightforward solution is the use of a server script which runs in a domain from where the web page was loaded. The web page then calls the script with the respective request for loading the source from another domain. On request, the script loads the requested source and sends it to the web page.

4.2. Peer-to-peer Communication via NAT

In the peer-to-peer network architecture, a problem in the connection between clients may arise when the clients are located behind NAT (Network Address Translation) routers. In this case, clients are in a local network and communicate with a public network via NAT routers, which serve as a public network gateway. What causes a problem in this case is the addressing of the client behind the NAT router, because from the public network perspective, the client is not visible (only the NAT router is visible), and is thus unreachable. In this case, direct initiation of a peer-to-peer connection is not possible.

The client connection problem can be solved using the ICE protocol (RFC 5245, see References).

The solution requires the use of an initialization server, which provides the connection.

The following example of the ICE protocol function has been simplified to demonstrate the scope of the problem (the performing of this connection algorithm requires 2 computers (clients) A and B which are, from the public network perspective, located behind a single NAT router at maximum (not port-forwarding) – see Figure 3):

1. Client A contacts a STUN server via port X. STUN sends back the number of port Y, from where it was contacted by client A. Based on the response (the number of the communication port of the client STUN request and the number of the port in the

STUN server response is the same), if the client finds out that the computer is accessible from the public network (has a public IP address – in this case, port X = Y), there is no problem in the communication – the client can peform a connection and ICE protocol has finished.

2. If the client finds out, according to the STUN server response, that it is not publicly accessible (the port numbers of the request and of the response are not the same, i.e. $X \neq Y$), client A sends the public port number Y via the initialization server (the port which the STUN server was able to contact) to client B, with which it can then communicate.
3. Steps 1 and 2 are also performed by client B.
4. Clients A and B initiate a connection via public IPs and ports, which they exchanged via the initialization server.
5. If direct communication between clients A and B is not possible (is blocked by the NAT router, firewall, etc.), it is the TURN server which can function as a mediating element between the two clients. The TURN server exchanges messages from one client to another. This does not correspond to the peer-to-peer architecture but to the client-server-client network topology. The use of the TURN server is not an ideal solution as there is high latency and server load; however, in this case, it is the only solution to connect clients A and B. This is, however, a rather exceptional state.

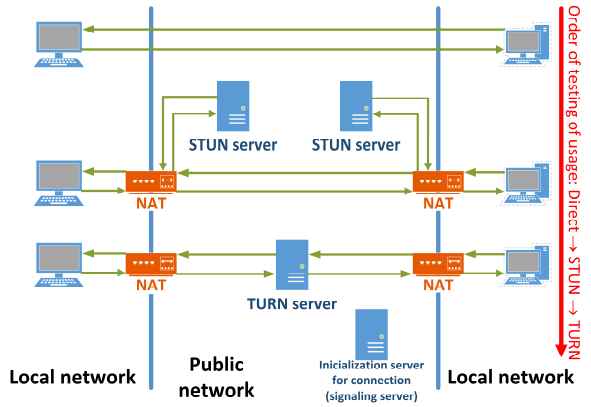

Figure 3: Visualisation Of ICE Protocol

From the above-mentioned algorithm it follows that the procedure is not rivial. Importantly, both the STUN and the TURN servers are available as open-source, and there are libraries available for the common programming languages (Java, C#, C++, etc.) which support the ICE protocol. WebRTC is no exception, and when the STUN (eventually TURN) server address is provided, the web-browser itself handles the operation without the need of human interference. Some publicly available STUN and TURN servers also exist and can be used, making the facilitation of direct communication a very easy operation.

5. WEBRTC PRACTICAL APPLICATIONS

To be used in practice, WebRTC has to meet yet another requirement. Each peer-to-peer connection requires an initialization server, which serves as an "intermediary" while initiating a connection between clients. WebRTC is no exception; in the context of WebRTC, the initialization server is called a signaling server. In practice, the situation is more complex; another (by no means less important) role is the role of the client, which performs a connection using the SDP and ICE protocol.

Signaling server is a server via which clients exchange SDP (Session Description Protocol) messages, where they provide the information about the network connection via which they are to communicate. (It follows that a signaling server is not required if the SDP message exchange can be facilitated using another option). The communication via a signaling server can be facilitated using a wide range of options:

- The simplest solution is sending requests to a server, or eventually sending the required SDP information (see Chapter 2.1.).
- A more efficient option is the usage of WebSockets.
- A wide range of other options.

If the initialization process is successful, a bi-directional communication channel between clients (web browsers) is created, with the help of which we can exchange messages (or other data types supported by WebRTC – especially audio and video streaming).

We also have at our disposal a number of open-source signaling servers, which we can use and only deal with the logical process modeling.

6. IMPLEMENTATION

For testing and application purposes, a collection of programming tools and protocols was introduced to create a simple ecosystem with focus on reusability, simplicity and development speed. We elaborate on the implementation in this chapter.

6.1. Concept

The central idea was to create a set of (relatively general) logical processes, which can be integrated into the distributed model. The sample implementation concerned traffic, where logical processes represented:

- A road,
- A road with a turnoff,
- A road with a crossroads.

The three logical processes can be used multiple times in a single distributed model in arbitrary combinations and individual instance configurations of the used logical processes. The three logical processes are sufficient for the purpose of simulations of e.g. trafic infrastructure of a small town.

We elaborate on a web-based simulation, i.e. individual logical processes were implemented as web pages programmed in JavaScript. The communication among the logical processes was performed using the peer-to-peer connection via the HTML5 WebRTC technology.

6.2. Used Synchronization Algorithm

To synchronize logical processes, we used a conservative synchronization technique of sending null messages with a lookahead (*Chandy-Misra-Bryant Distributed Discrete-Event Simulation Algorithm*, Fujimoto 2000). The algorithm was modified to a version where null messages are only sent on request. The lookahead ensures that the simulation calculation proceeds forward in time – there is no risk of a deadlock. In our logical process creation concept, the logical process lookahead is not dificult to define. The simplest method to define the lookahead limit is registering the most short-time activities which occur in the logical process.

The algorithm can be described in the following way (the example of simulators SC1 and SC2, where SC1 receives messages from SC2; LVT = Local Virtual Time, Calendar = the event queue).

1. SC1 has no planned activites from SC2.
2. SC1 sends the LBTS activity of the “Request” type, which is labelled as a “service”; it also sends its own SC1.LVT.
3. SC1 is waiting.
4. SC2 receives the LBTS event; as the LBTS activity is labelled as a “service”, SC2 does not queue it, but executes it immediately after the current activity has been finished.
5. SC2 performs the LBTS activity: it calculates SC2.LBTS = SC2.LVT + SC2. Lookahead.
6. If SC1.LVT > SC2.LBTS, SC1 does not accept such LBTS answer and the sending on the SC2.LBTS answer is scheduled in SC2.Calendar to the SC1.LVT time.
7. Otherwise, the LBTS event of the “Response” type with the SC2.LBTS time is sent to SC1 (it is no longer labelled as a “service”).
8. SC2 continues its activities.
9. SC1 queues the LBTS event which was sent by SC2.
10. By this time, SC1 knows the lower limit of the SC2 time and can execute the planned events before the received LBTS message.

6.3. Administration interface

To facilitate the administration of the distributed models, an administration interface was created which runs as a common web application. It allows for a simple addition of individual logical processes and their integration into the simulation.

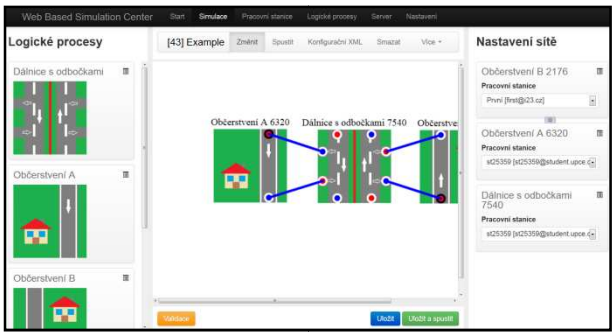

Figure 4: Administration Example

The simulation is created using the Drag & Drop method in a visual editor directly in an HTML page (implemented using the HTML canvas) and brings the editor a whole range of functions (basic features):

- A multiple use of individual logical processes (multiple instances of a single logical process type),
- An intuitive connection of logical process instances – logical processes have distinct input and output connection points. The connection distinguishes various types of received entities by logical processes.
- Individual logical process instances are configurable.
- Global configuration of the whole simulation is available.

6.4. A Software Library For Logical Process Implementation

To simplify the development, a software library (a set of classes and functions in JavaScript) was introduced which allows for a simple creation of logical processes and an implicit realization of logical process synchronization.

The library includes the implementation of:

- A simulation kernel,
- An activity prototype (a class from which activites can be created easily).
- A network connection using WebRTC,
- A synchronization mechanism (described in Chapter 6.2).
- Animation.

All was implemented using primarily the CoffeeScript language, which subsequently compiled to JavaScript.

CoffeeScript was used for faster and more transparent implementation than a comparable Java Script solution. Both types of source files are at the user’s disposal. The whole solution is opened; the encapsulation of functions and classes has been reduced to minimum to allow for prototyping (and especially inheritance). This does not represent a typical OOP approach; however, in its very nature, JavaScript was

designed differently. Although this solution may potentially be dangerous (the user can "rewrite" the code while the program is running), as a result it brings increased flexibility and a possibility to expand classes (especially as far as inheritance is concerned, some parts of the solutions are dependent on it).

As a result, the programmer is only required to implement activities which run on the basis of a logical process (eventually to create appropriate animation output). The rest is ready to use without any modifications or alterations.

6.5. Logical Process Communication Methods And Other Suggested Standards

If the logical process creator works with a library which has been designed specifically for this purpose (see the previous chapter), they do not have to solve the implementation of logical process communication or loading of configuration files at all.

It is of course possible to create a logical process quite independently of the above-mentioned software library. In this case, we have at our disposal a description of the communication between logical processes (*defacto* an internal communication protocol) and a description of configuration XML files using an XML schema.

The communication implements very easily. JavaScript implicitly solves a correct language coding of messages. Information is transfered in the JSON data format (a common means of data transfer via the Internet actively supported by JavaScript).

The configuration of the whole simulation is available in two XML files (can be processed using JavaScript):

- The logical process configuration, which describes especially the possibilities to connect with other logical processes and the required instance configuration. This file uses especially the administration for corresponding visual editor behavior.
- Configuration of the whole distributed simulation setup. The file contains a global simulation configuration and a configuration of all logical process instances and their interconnection.

6.6. A Sample Solution And Testing Logical Processes

The whole solution was tested on 3 logical processes representing:

- Highways with turnoffs,
- Two logical process types representing (highway) fastfood facilities.

The logical processes are accompanied with an animation, which provides visual information about the events within the logical process.

See an an example of an administration interface (Fig. 4) and used logical processes (Fig. 5).

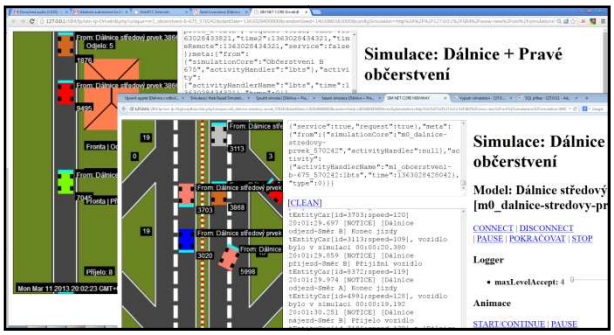

Figure 5: An Example Of Logical Processes Running In A Web Browser

As mentioned above, logical processes can be freely concatenated and allow for multiple usage. In the context of testing, the three logical processes were used in a distributed simulation which consisted of 20 logical processes. All worked smoothly and without issues.

6.7. Comparison Of Java Applet And JavaScript Solutions

As mentioned at the beginning, this work is a follow-up to a previous solution, where logical processes were also running in the web browser, yet using Java Applet. This allows us to compare the effort spent on both solutions as well as their results. The aim of this chapter is not to compare which of the two solutions, Java or JavaScript, is better (especially due to the fact that the two languages are quite different one from another); it is rather to point out which kinds of comparison the two solutions inspire.

As expected, the results for the same simulation scenarios in both realizations were equal.

A much more interesting comparison is that of the implementation of the same problem. Both source codes were written by the same author; we can thus claim that the style and the algorithm solution would be similar. The size of the JavaScript source code (realized solely by web technologies) represents approximately 70% of the size of the code in Java (used in the Java Applet). Provided that we do not take into account the used frameworks for the DOM manipulation and for the facilitation of canvas drawing (to a certain extent, both include the basic programming tool collection which is also provided by Java SDK), we realize that the JavaScript code is less than half the size of the Java code.

In slighly exaggerated terms, we can claim that a program which is half the size takes half the time to create and contains half of issues. Shorter code is faster to read and understand. Even from this perspective, the transfer from applets to web technologies is worth to consider, provided that it is feasible. And this article demonstrates that the transfer certainly is feasible.

The simulation speed was not measured, because in both cases, the animation was running in real time,

which itself decreased the speed so that the simulation could be observed by the user. When compared to the program itself, there was a significant time lag in the network communication. In the end, the speed of the code execution is not essential.

It has to be mentioned that JavaScript has numerous implementation drawbacks (as any other programming language, in fact); however, their description is beyond the scope of this article.

7. SUMMARY

The article introduced the reader into a web-based simulation and mainly into new HTML5 possibilites of web browsers, which provide opportunities for a web-based simulation using only a web browser, i.e. without complemental third-party software (applets, etc.). A web-based simulation provides the opporunity to create logical processes within the public network irrespective of the platform or processor architecture. A software platform, which is typically realized as a desktop application requiring installation, is to be replaced by the web browser, which is today widely available free of charge, and for almost any computer. The creation of a web-based distributed simulation faces only a limited number of issues (they usually concern network traffic safety policy); as opposed to desktop applications, the web browser brings a variety of ready-made features (in our case, the most important one is WebRTC), whose realization in common native applications is not trivial.

Using the web browser as a runtime environment, we can focus on the main aim, i.e. on the creation of distributed simulation models running in a web browser on any device connected to the network, from computers via cellular phones to e.g. smart TVs. This opens up an opportunity for a user-friendly, interactive simulation available for anybody at any time.

REFERENCES

Fujimoto, R.M., 2000. *Parallel and distribution simulation systems*. New York: Wiley.

Kuhl, F., Dahmann, J., Weatherly., R., 2000. *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*. Upper Saddle River, NJ: Prentice Hall.

Tropper, C., 2002. *Parallel and distributed discrete event simulation*. New York: Nova Science Pub Inc.

Bergkvist, A., Burnett, D.C., Jennings, C., Anant Narayanan, 2013. *WebRTC 1.0: Real-time Communication Between Browsers*. W3C. Available from: http://www.w3.org/TR/webrtc/ [accessed date July 2014]

Rosenberg, J., 2014. *Interactive Connectivity Establishment (ICE)*. IETF. Available from: http://tools.ietf.org/html/rfc5245 [accessed date May 2014]

Martin, D., Rajagopalan, S., Rubin, A., 2013. *Blocking Java Applets at the Firewall*. Available from: http://avirubin.com/block.java.pdf [accessed date June 2013]

Hridel, J., Kartak, S., 2013. Web-based simulation in teaching. *Proceedings of The European Simulation and Modelling Conference 2013*, pp. 109–113. September 23–25, Lancaster, UK.

Byrne, J., Heavey, C., Byrne, P.J., 2010. A review of Web-based simulation and supporting tools. *Simulation Modelling Practice and Theory* 18: 253–276

Javor, A., Fur, A., 2012. Simulation on the Web with distributed models and intelligent agents. Available from: http://sim.sagepub.com/content/88/9/1080 [accessed date June 2013]

Kartak, S., 2013. *The software tool for configuring distributed simulation model using a web simulation*. Thesis. University of Pardubice.