WEB SIMULATION WITH SUPPORT OF MOBILE AGENTS

Jan Voracek[(a)]

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

[(a)]jan.voracek@student.upce.cz

ABSTRACT
This paper solves the problem of mobile agents in distributed simulation. Its main focus is on the method of implementation of transferring mobile agents in the JavaScript language, which does not support marshalling. The secondary focus is on the general suitability of agent-oriented architectures in JavaScript. All of this with the use of WebRTC, a peer-to-peer communications technology.

Keywords: web simulation, mobile agent, WebRTC

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 many benefits, e.g., the simulation calculation can scale easily or, in some cases, more cost-effective resources can be used, etc. The more complex the problem is, the more advantageous it is to distribute it to more computing nodes (Fujimoto, 1999). Another frequent reason for introducing distributed solution is using multiple operators.

Some of the problems, which have to be solved with running distributed simulation, are (of course among implementation of the simulation model itself) the setup of environment which the simulation will be running in and the interconnection of computing nodes for communication. This interconnection becomes nontrivial when the computing nodes are not on the same computer network.

For the simulation of more complex systems, it is also advantageous to use the agent-oriented architecture. This architecture allows to divide the complex system into several individual logical units which complexity is substantially lower (Macal 2005).

2. THE INTERNET AS A PLATFORM FOR COMMUNICATION
As told in the introduction, the distributed simulation works by distributing the calculation to a cluster of computing nodes. For the needs of distributed simulation, it is often sufficient to communicate in a private network. However, we can be in a situation where we need to run the simulation between two or more dislocated departments which are not connected to a private network. Alternatively, we want to link individual computing stations around the world, where the building of a private network is not possible for economic or pragmatic reasons. In this case, it is convenient to use the Internet, the global computer network.

This approach, however, has some drawbacks. When communicating using the Internet it cannot be guaranteed which way the data will be transferred or who will be able to access it. Therefore as a security precaution, it is preferred to encrypt all the communication (Mitrovic 2014).

Another difficulty is a problematic direct communication of computing nodes. The stations performing the simulation are typically connected to a private network behind a router and do not have assigned a public IP address.

3. THE WEB AS A RUNTIME ENVIRONMENT
In the introduction, there was also told that one of the problems of running distributed simulation was putting a runtime environment into operation. It is necessary to install this environment on all the stations involved in the simulation calculation. A self-executable simulation model gives us an alternative way to run the simulation. However, both of these approaches are more or less dependent on the hardware (processor architecture) or software (operation system) platform.

Proposed solution uses, as its runtime environment, a web browser, which is available not only for personal computers but also for tablets, smartphones and smart TVs (Mitrovic 2014). The fact that web browsers are (unlike the majority of specialized environments) free may be also considered as a benefit.

A solution using a web browser is not so much dependent on a particular platform as usual approach, but in combination with various types of communication it may be dependent on specific browsers or their versions (Kartak 2014).
After mentioning the communication, it would be appropriate to outline what are the possibilities of communication on the web. It is possible to divide the communication options into two groups:

- native – used from JavaScript,

- third-party products – Java Applets, Flash, Silverlight (Kartak 2014).

Solutions that use third-party products were evaluated as unsuitable as they used to be unwieldy and brings a lot of their own problems – especially in terms of safety (Kartak 2014).

As for the native forms of communication, there are several technologies to use:

- XmlHttpRequest known as AJAX,
- WebSockets,
- WebRTC.

The AJAX is available in all commonly used web browsers while WebSockets and WebRTC come as part of an HTML5 specification and their availability in popular web browsers is listed in Table 1.

Table 1: Availability of HTML5 communication technologies in popular web browsers by their version and its release date

Web Browser	WebSockets	WebRTC
Chrome	14 (09-2011)	23 (11-2012)
Firefox	6 (08-2011)	22 (06-2013)
Internet Explorer	10 (08-2012)	–
Opera	12.10 (11-2012)	22 (06-2014)
Safari	6 (07-2012)	–

Adapted from Kartak 2014

3.1. AJAX
It is a standard client-server communication based on HTTP requests. The communication is always initiated by the client (the web browser), and the server only responds to the requests. It cannot initiate the connection (Zehe 2013). For the needs of distributed simulation, it is not ideal because the communication is centralized, and the server can easily become a bottleneck of the entire simulation calculation.

3.2. WebSockets
WebSockets also offers a client-server communication, however, compared to AJAX, the connection remains open, and the data can flow in both directions. Therefore, the server can initiate a data transfer and send a message to a client at any time (Green 2012). However, the communication is still centralized.

3.3. WebRTC
WebRTC significantly extends the capabilities of web browsers in the field of communication. The main benefit of the distributed simulation is the support of peer-to-peer communication, which does not require any central element, through which all the data flow (Barnes, 2014). In addition, the WebRTC implements an ICE protocol (Interactive Connectivity Establishment), which allows connecting two stations in separate private networks without requiring any configuration of network elements. The proposed solution uses WebRTC despite the limited support in web browsers.

WebRTC Security
As told, the encryption of communication is one the most important aspects of communicating in a public network. Also because of that, the WebRTC standard specifies that all its implementations have to support the DTLS-SRTP (Data Transport Layer Security – Secure Real-time Transport Protocol). This protocol works like HTTPS only it is based on the UDP communication protocol instead of TCP (Barnes 2014).

4. AGENT-ORIENTED SIMULATION
The agent-oriented simulation consists in dividing of a comprehensive system to autonomous logical units capable of interaction (a.k.a. agents). Combined actions of individual agents then simulate the behavior of the modeled system.

Agents can be classified into different groups. For example, in terms of complexity of their autonomous behavior, they can be classified as (Nwana 1996):

- intelligent – with the ability to meet their goals using logical deduction,
- reactive – with the ability to respond to stimuli,
- deliberative – with the ability to plan their actions and to affect their environment to gain an advantage,
- cognitive – with the ability to deduce logical conclusions from observing their environment, they can learn and create a knowledge base,
- rational – they have all of the characteristics.

In a distributed system, the agents can be classified in terms of variability of place of execution of their code as:

- stationary – the agent is allocated on a single computation node, where its code is executed until its termination or until the end of simulation,
- mobile (migrating) – agent may move its code and state to another computation node by itself (without human intervention), where it continues with performing its task (Pinsdorf 2002, Zehe 2013).

4.1. Mobile agents in web environment
The main focus of proposed solution is an implementation of mobile agents in JavaScript. Unfortunately, this language does not support marshalling (the serialization of data structures representing the agent into a form suitable for transmission over the network), which is an essential prerequisite for moving the agent to another computation node.

5. JAVASCRIPT SERIALIZATION

As mentioned above, unlike most currently used languages, JavaScript does not support a complete transform of objects and functions into a form suitable for transmission over the network, or for storing. It supports only conversion into JSON, which, unfortunately, supports only these data types:

- *Number,*
- *String,*
- *Boolean,*
- *Array,*
- *Object,*
- *Null,*

where the *Object* data type is only a map and can only contain string keys and values with data types listed above (see RFC 7159 and ECMA-404).

5.1. Serialization libraries

Besides the built-in functionality for converting objects to JSON, there are also libraries, which tries the full serialization. Each of them has a greater or lesser restrictions. According to search results, libraries *GSerializer* and *JASON* are the most used. Both project, however, are no longer maintained, and they do not work with the current version of JavaScript. Also, for example, the *GSerializer* does not support serialization of parametric methods.

5.2. Proposed solution of serialization

The proposed solution is slightly inspired in the Dart language and the way this language solves conversion to JSON and back without losing information. Dart allows each class to define methods *toJSON* and *fromJSON* whose implementation is completely up to the developer. Support of a similar approach can also be found in other languages. E.g. in Java, there are methods *writeObject* and *readObject* in C# method *GetObjectData* and a special private constructor. The main taught is that there are two methods. One method can extract the object state to a simpler data structure (e.g. those which can already be converted into JSON or XML) and the second one is able to restore the object to its original state based on the data. These methods are called *getState* and *setState* in the proposed solution.

For example, assume an instance of the class *Car*, which has properties *color* and *topSpeed* and several methods representing the behaviour of this object. The methods are the same for all objects. What differs is the current state of individual objects represented by values of their properties. To create any instance of the *Car* class, we therefore, need a prescription of this class and mentioned values of its properties.

The prescription of a class is represented by its source code, and the state is obtained using method *getState*, as shown in Figure 1.

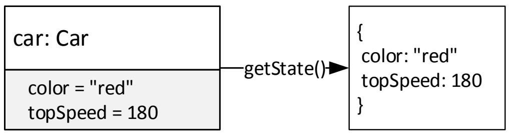


Figure 1: Obtaining the state of an object

To restore the original object just create any (presumably “empty”) instance and use method *setState*, as shown in Figure 2.

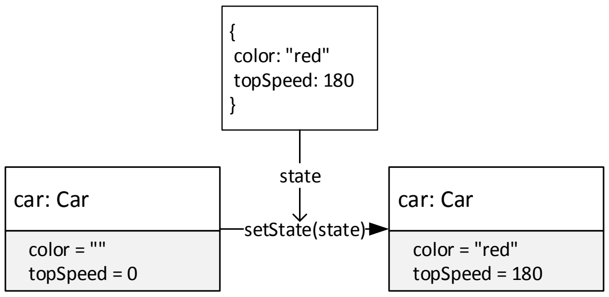


Figure 2: Restoring the object

The problem will occur when a part of the state is an object shared with other serialized object (shown in Figure 3). This may be the subject of further research.

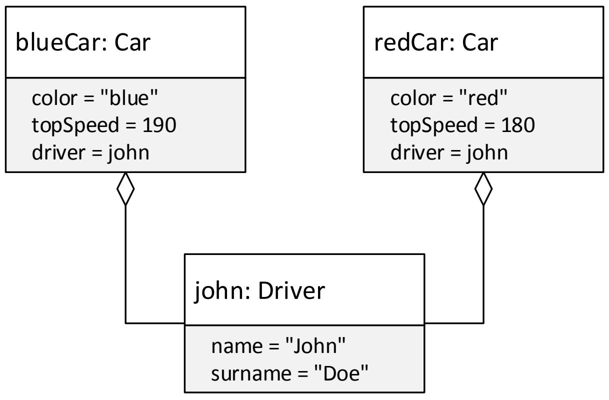


Figure 3: Two objects sharing third one

6. CASE STUDY

As a case study was chosen distributed strategic game codenamed “devconquest”. The target is to conquer the opponent’s castle while protecting their own. Each player builds a castle from available material and programs his military units to defend his castle and conquer the opponent’s one.

The player can choose from several different materials on the construction of fortifications, where more solid are of course more expensive. He can also build loopholes up on the walls into which can be placed archers.

Regarding the military units, the game offers several basic types:

- soldier – offensive / defensive unit with a sword,
- archer – offensive / defensive unit with a bow,
- scout – reconnaissance unit a dagger.

Each unit has a basic offensive and defensive number (its attack and defence capabilities), visibility range and maximum velocity of movement. The player can modify these abilities, but only so that their sum dos not exceed a fixed maximum. Each unit is needed to program its behaviour so that player has reached the victory.

During the game, the military units turn in performing their action – moving over the game field, attacking an enemy unit in range, exploring the condition of walls, communicating with other units, etc. First, all units of one player perform their turns, and then continue the second one's and so on.

At the beginning of the game the player has covered the opponent's part of the map and gradually reveals it as his units move over the opponent's field. Opposing units are visible only if some player's unit has them in sight. For a better imagination, there is a game in progress shown in Figure 4.

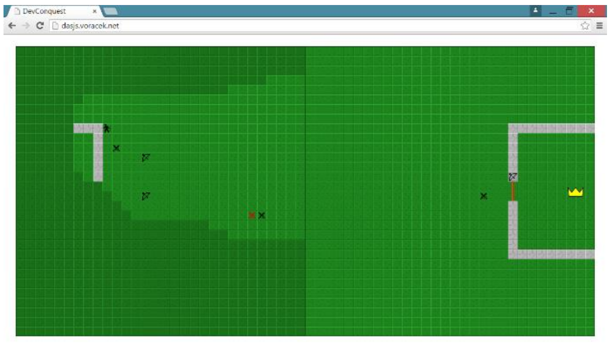
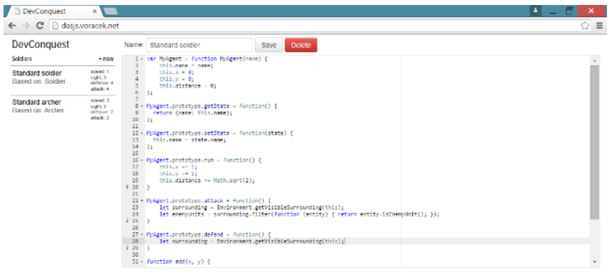


Figure 4: Game in progress

6.1. Utilization of mobile agents

Every military unit represents one agent who can travel between the connected stations. It uses the services provided by the hosting system for communication with its environment. The agent can, for example, send messages to the other agents, explore the game field or interact with it (attack enemy units, walls).

The agents' logic can be implemented directly in the browser. The application contains simple code editor with highlighting, as you can see in Figure 5.



Name: Standard soldier

```
var MyAgent = function MyAgent(name) {
    this.name = name;
    this.x = 0;
    this.y = 0;
    this.distance = 0;
};

MyAgent.prototype.getState = function() {
  return {name: this.name};
};

MyAgent.prototype.setState = function(state) {
  this.name = state.name;
};

MyAgent.prototype.run = function() {
    this.x += 1;
    this.y += 1;
    this.distance += Math.sqrt(2);
}

MyAgent.prototype.attack = function() {
    let surrounding = Environment.getVisibleSurrounding(this);
    let enemyUnits = surrounding.filter(function (entity) { return entity.isEnemyUnit(); });
}

MyAgent.prototype.defend = function() {
    let surrounding = Environment.getVisibleSurrounding(this);
}

function add(x, y) {
```

Figure 5: Agent editor

7. CONCLUSION

The article introduces the reader to the possibilities of implementing a distributed simulation on the Web, especially using the WebRTC technology. WebRTC brings the possibility of creating distributed simulation without central node (server). The article extends the distributed simulation with a support of mobile agents. To implement mobile agents has to be solved the problem of missing serialization of objects in JavaScript, which eventually became a major problem that this article addresses. The proposed solution is verified through a case study represented by a strategic, conquering game, where mobile agents represent different military units. The proposed solution provides space for further research – especially in the sharing object between multiple agents.

REFERENCES

Barnes R. L., Thomson M., 2014. Browser-to-Browser Security Assurances for WebRTC. Internet Computing, IEEE 18 pp. 11–17.

Fujimoto R. M., 1999. Parallel and Distribution Simulation Systems. John Wiley & Sons, Inc.

Green I., 2012, Web Workers: Multithreaded Programs in JavaScript. O'Reilly Media.

Kartak S., Kavicka A., 2014. WebRTC Technology as a Solution for a Web-Based Distributed Simulation. In Proceedings of the European Modeling and Simulation Symposium, pp. 343–349, Genova: Università di Genova.

Macal C. M., North M. J., 2005. Tutorial on agent-based modeling and simulation. In Proceedings of the 37th conference on Winter simulation, pp. 2–15, Orlando, Florida.

Mitrovic D., Ivanovic M., Budimac Z., Vidakovic M., 2014. Radigost: Interoperable web-based multi-agent platform. Journal of Systems and Software 90 pp. 167–178.

Nwana H. S., 1996. Software agents: an overview. The Knowledge Engineering Review 11 pp 205-244.

Pinsdorf, U., Roth, V., 2002. Mobile agent interoperability patterns and practice. In Proceedings of the 9th IEEE International Conference on Engineering of Computer-based Systems, pp. 238–244.

Zehe D., Aydt H., Lees M., Knoll A., 2013. JavaScript Distributed Agent Based Discrete Event Simulation. In Proceedings of the 2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications (DS-RT '13). IEEE Computer Society, Washington, DC, USA, 21-29.