WEB BASED EVENT DISPLAY SERVER FOR MPD/NICA EXPERIMENT

A. Krylov[1,3,a], O. Rogachevsky[1,2], V.Krylov[1], A. Bychkov[1]

[1] *Joint Institute for Nuclear Research, 6 Joliot-Curie, 141980, Dubna, Moscow region, Russia,*

[2] *Dubna State University, 19 Universitetskaya, 141982, Dubna Moscow region, Russia*

[3] *Moscow State University, 1-2 Leninskiye Gory, 119991, Moscow, Russia*

E-mail: [a] avkrylov@jinr.ru

Modern experiments in nuclear physics last for years and require enormous human and energy resources. There are various methods for monitoring engineering, network and computer systems of the experiment. As a rule, they have a common name for all - the Event Display and include a whole range of monitoring and control systems. The operator must receive comprehensive information about the course of the experiment in an understandable and intuitive form. The ability to obtain information in three-dimensional form that most fully reflects real physical processes is one of the current trends in experimental nuclear physics. In this work, a prototype of the Event Display was created using a high-level language - JavaScript, built into any standard Internet browser. Modern technologies make it possible not to take into account the platform and type of operating system on which the Internet browser is launched. A program written in JavaScript will execute in the same way on every platform, including mobile devices with Internet access. The work was carried out for the MPD (Multi-Purpose Detector) detector at the NICA (Nuclotron-based Ion Collider fAcility) accelerator complex at the Joint Institute for Nuclear Research (JINR, Dubna, Russia).

Keywords: Event Display, WebGL, MPD NICA, web-server

Alexander Krylov, Oleg Rogachevsky, Victor Krylov, Alexander Bychkov

Copyright © 2021 for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR-WS.org/Vol-3041/562-567-paper-104.pdf

1. Introduction

The Event Display is designed to visualize experimental data in high energy physics, showing the structure of the detectors and information about the event that has been registered. This tool is used for a variety of purposes:

- Engineering services use it in the control room of the experiment to control the operation of individual parts of the detector;
- Physicists can use it to improve reconstruction algorithms and physical analysis. With this system, physicists can quickly understand the structure of the detector and observe events of interest;
- The Event Display can be used to present special events as well as for popularization of high energy physics among schools and universities.

Until now other experiments had udes different approaches to create the Event Display. For example, the ALICE and JUNO experiments use the OpenGL (Open Graphics Library) used by the ROOTEVE library [1]. JUNO had a different Event Display based on Unity [2]. The CMS experiment has web-based event display called i-Spy using WebGL (Web Graphics Library) [3].

Here we will discuss principles and technologies that were used in the EDS (Event Display Server) for the MPD/NICA experiment.

2. Principles

With the development of HTML, developers have been able to create more complex web applications. In its early days, HTML offered only the ability to display static content, but with the addition of JavaScript support, it became possible to implement more complex element interactions and display dynamic content.

JavaScript is a dynamic single-threaded programming language that is mostly used for web development. It was developed by Brendan Eich in 1995 while working for Netscape Communication, and began to develop rapidly after Google presented its JavaScript Engine V8 in 2010. Thanks to the V8 engine, the execution speed of a web application became comparable to a similar application working natively, so it became possible to create complex applications like the Event Display.

To simplify the creation of user interfaces, many libraries have been created for the JS language. To achieve the goal of this work, the React library was chosen, due to its simplicity and popularity in the world.

React is a JavaScript library for building user interfaces. This library greatly facilitates the creation of interfaces by fragmenting each HTML page into small components. The other reason for choosing React was the presence of such a feature as the Virtual DOM.

DOM (Document Object Model) is a way of representing a structured document using objects. Web browsers handle the DOM components, and we can interact with them using JavaScript and CSS. We can work with document nodes, modify their data, delete and insert new nodes.

In React, instead of interacting with the DOM directly, we work with a lightweight copy of it. We can make changes to the copy based on our needs, and then apply the changes to the real DOM. React compares the DOM tree with its virtual copy, determines the difference, and starts redrawing what has been changed [fig. 1]. This approach is faster because it does not include all the unchanged parts of the real DOM.

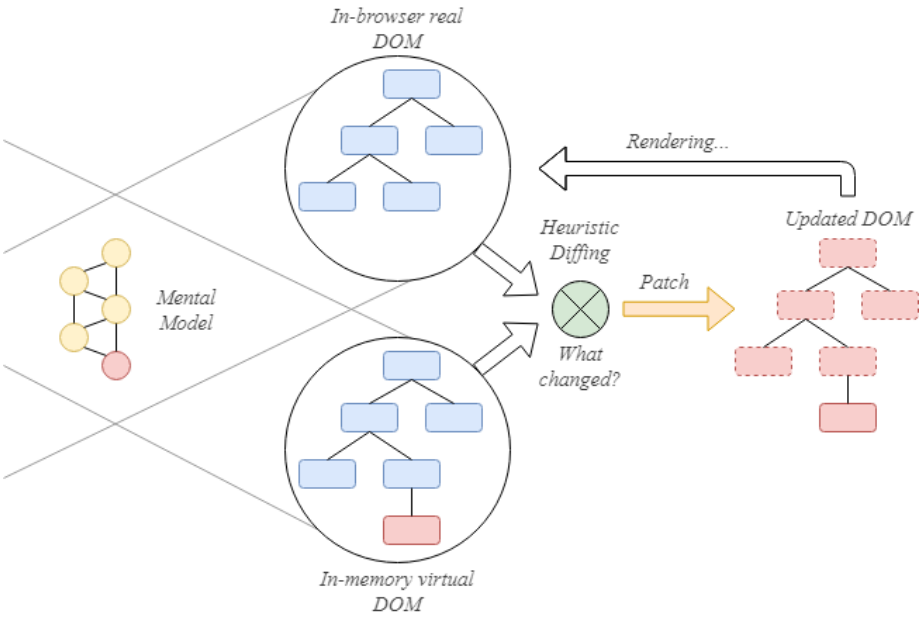

Figure 1. Virtual DOM feature in React framework [4]

Also, to implement the visual part of the Event Display, a graphics package that allows the use of 3D graphics in browsers is needed. The chosen library is WebGL by the Khronos Group. The creation of the WebGL technology made it possible to display and manipulate 3D graphics on web pages using JavaScript. Despite its impressive capabilities, WebGL differs from other technologies in its accessibility and ease of use, which contributes to its rapid distribution.

Working with WebGL and shaders is a rather laborious process. During the development process, it is necessary to describe each point, line, face, etc. To visualize all this, we need to write a fairly huge piece of code. That is why the ThreeJS library is used.

ThreeJS is a JavaScript library containing a set of pre-built primitives for creating and displaying interactive 3D graphics in WebGL. When using ThreeJS, there is no need to write shaders, since it becomes possible to operate with familiar concepts such as scene, light, camera, objects and their materials [fig. 2].

Next, it was necessary to choose the environment in which the web server would work. A suitable environment for this task is NodeJS, based on the V8 Engine. NodeJS adds to JavaScript the ability to interact with I/O devices and connect other external libraries written in different languages, providing calls to them from JavaScript code. Thanks to this, it became possible to launch the execution of JS code on a local machine or server as a separate application.

3. MPD Geometry transformation

The geometry for the Event display was created by converting the geometry from the MpdRoot software to the VRML[1] format using the Geant4 VGM[2] utility [5]. Then the geometry was converted to the glTF[3] format (the "native" WebGL format of The Khronos Group Inc.), using free open source 3D creation suite "Blender" [6] for the conversion [fig. 2].

1 VRML - Virtual Reality Modeling Language
2 VGM – Virtual Geometry Model
3 glTF – Graphics Library Transmission Format

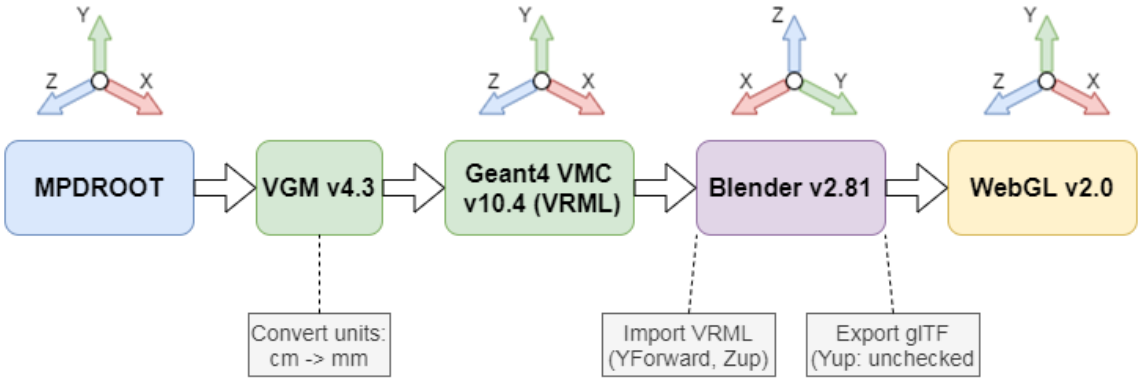

Figure 2. MPD geometry transformation scheme

VRML – a file format for representing 3-dimentional (3D) interactive vector graphics. In MPD Event Display, the VRML format was used just as intermediate between Geant4 and Blender software.

glTF – a royalty-free specification for the efficient transmission and loading of 3D scenes and models by web applications. It minimizes both the size of 3D assets, and the runtime processing needed to unpack and use those assets. It defines an extensible, common publishing format for 3D content tools and services that simplifies creation workflows and enables interoperable use of content.

Using that principle, the detector geometries were transformed to the glTF format and added to the EDS.

4. Event data visualization

The event data is stored in the ROOT library format file. The file contains two TTree structures (data structure of the ROOT library): one contains the simulated data; the other contains the responses of the detectors.

Unpacking and reading event files occur on the server side. An add-on (extension) for the main program is created using the Node-API (NAPI) programming interface. This add-on is written in C++ and is used to implement the interface between JavaScript NodeJS and C/C++ libraries, which are used to read the event data file.

The first connection between a client and a web server is provided via the HTTP protocol [fig. 3]. As a result of the connection, the client receives a full package of front-end APIs for managing the web application, as well as detector geometry files. Event Display front-end works via server-side rendering method. In order to work with events, at the request of the client, a WebSocket connection is opened. After the user selects an event file from, for example, the local file system or database, this file is read using the NAPI add-on and after that, a single event in the JSON[4] format is sent through the message broker to the client.

The visualization of the detectors geometries and read events is performed on the client side using its graphic resources.

4 JSON – JavaScript Object Notation

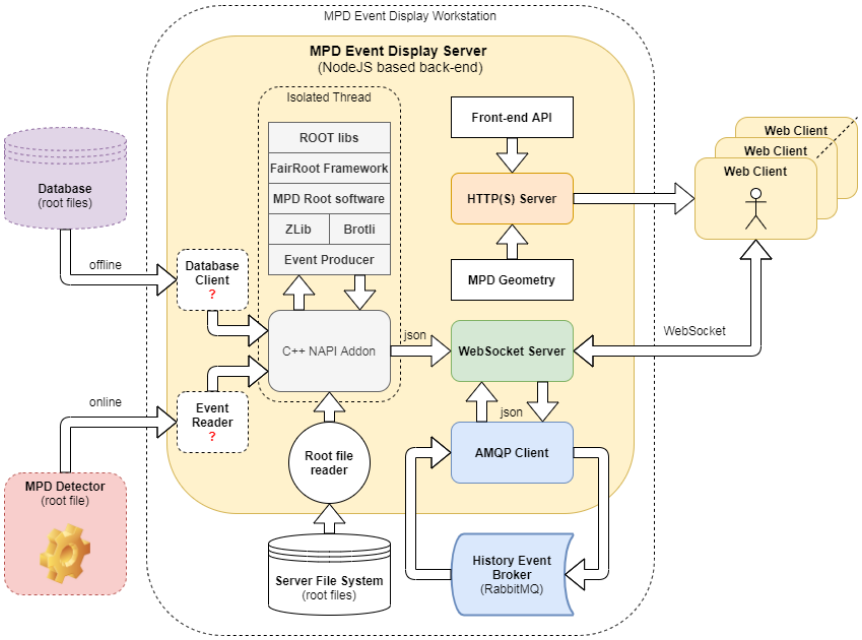

Figure 3. Event Display Server data flow

5. Conclusion

The 1-st stable prototype of the Web interactive Event Display for the MPD detector is released and it has been tested on all major platforms and browsers, including mobile devices https://mpd-edsrv.jinr.ru/ [fig. 4].

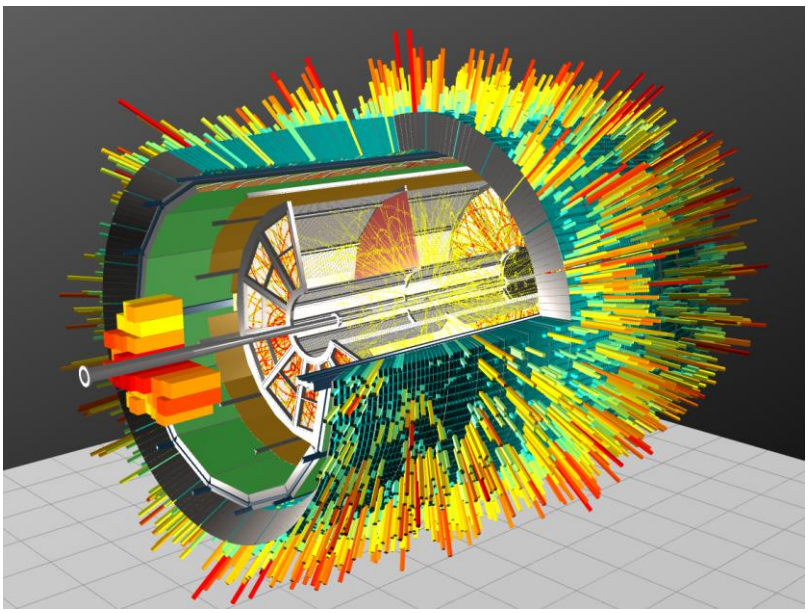

Figure 4. Event Display Server data visualization

The interactive graphics part based on WebGL allows showing more than 120K active sensors from different detectors in one scene without undue and annoying delays even on mobile devices.

C++ Node API add-on makes it possible to read ROOT data files on the back-end side of the Event Display directly in native code without any cost-sensitive transformation. In order to minimize the time for event data transfer, we use online compress/decompress in parallel working threads on the server and client-side.

6. Acknowledgement

This work was supported by RFBR grant №18-02-40102.

References

[1] Matevž Tadel Overview of EVE - the Event Visualization Environment of ROOT Journal of Physics: Conference Series 219 (2010) 042055.

[2] Jiang Zhu Event Display in the JUNO Experiment /Zhengyun You, Yumei Zhang // Journal of Physics: Conf. Series 1085 – 2018 – 032038.

[3] T.McCauley A browser-based event display for the CMS Experiment at the LHC using WebGL Conf. Series: Journal of Physics: Conf. Series 898 (2017) 072030.

[4] Mark Tielens Thomas React in Action 2018 ISBN 978-1617293856

[5] VGM – geometry conversation tool between Geant4 VMC and ROOT TGeo geometry models. Available at: https://github.com/vmc-project/vgm

[6] Blender – free 3D creation software. Supports plugin for exporting glTF format. Available at: https://www.blender.org