RevMexAA (Serie de Conferencias), **45**, 75–78 (2014)

III Workshop on Robotic Autonomous Observatories (Torremolinos, Malaga, Spain, 7-11 October 2013)
Editors: J. C. Tello, A. Riva, D. Hiriart & A. J. Castro-Tirado

NEW WEB TECHNOLOGIES FOR ASTRONOMY

P-G. Sprimont,[1] D. Ricci,[2] and L. Nicastro [1]

RESUMEN

Gracias a las nuevas capacidades de HTML5, y las grandes mejoras del lenguaje JavaScript es posible diseñar interfaces web muy complejas e interactivas. Además, los servidores que eran una vez monolíticos y orientados a servir archivos, están evolucionando a aplicaciones de servidor fácilmente programables, capaces de lidiar con interacciones complejas gracias a la nueva generación de navegadores. Nosotros creemos que la comunidad de astrónomos profesionales y aficionados entera puede beneficiarse del potencial de estas nuevas tecnologías. Nuevas interfaces web pueden ser diseñadas para proveer al usuario con herramientas mucho más intuitivas e interactivas. Acceder a archivos de datos astronómicos, controlar y monitorear observatorios y en particular telescopios robóticos, supervisar pipelines de reducción de datos, son todas capacidades que pueden ser implementadas en una aplicación web JavaScript. Describimos el paquete `Sadira` que estamos implementando exactamente con este propósito.

ABSTRACT

Thanks to the new HTML5 capabilities and the huge improvements of the JavaScript language, it is now possible to design very complex and interactive web user interfaces. On top of that, the once monolithic and file-server oriented web servers are evolving into easily programmable server applications capable to cope with the complex interactions made possible by the new generation of browsers. We believe that the whole community of amateur and professionals astronomers can benefit from the potential of these new technologies. New web interfaces can be designed to provide the user with a large deal of much more intuitive and interactive tools. Accessing astronomical data archives, schedule, control and monitor observatories, and in particular robotic telescopes, supervising data reduction pipelines, all are capabilities that can now be implemented in a JavaScript web application. In this paper we describe the `Sadira` package we are implementing exactly to this aim.

Key Words: methods: data analysis — astronomical databases: miscellaneous — astronomical databases: virtual observatory tools

1. GENERAL

Being cross-platform and easy to learn, the HTML[3] language, invented by physicists at Geneva's CERN in the late 1980's[4], has always been a good choice to build simple network oriented user interfaces. Being used by billions of devices around the world, the modernization of web related languages and protocols takes a very long time to make their way down to the final user. Today, after many years of sedimentation and refinements, most of the new HTML5 specifications are finally implemented in everyone's web browser. These new powerful tools will allow web developers to deeply reshape the WWW we know today.

Along with the development of HTML and HTTP[5], JavaScript[6], the browser-side programming language has become a lot more powerful. At the beginning, the JavaScript engines were slow interpreters, but nowdays they are much faster and effective, making use of *just in time compilation*[7] techniques like the open source V8[8] engine. Developed for the chrome/chromium[9] browser, it is now also used by other independent large projects such as *Node.js*[10] or MongoDB[11].

`Sadira`[12] is a web browser and *Node.js* exper-

[1]Istituto di Astrofisica Spaziale e Fisica Cosmica di Bologna, via Piero Gobetti, 101, 40129 Bologna, Italy.
[2]Instituto de Astronoma, UNAM, Campus Ensenada, 22860 Ensenada, B.C., Mexico (indy@astrosen.unam.mx).

[3]Hyper Text Markup Language

[4]See `http://home.web.cern.ch/topics/birth-web`

[5]HTTP, the Hypertext Transfer Protocol, `http://www.w3.org/Protocols/rfc2616/rfc2616.html`

[6]JavaScript, `http://en.wikipedia.org/wiki/JavaScript`

[7]JIT, `http://en.wikipedia.org/wiki/Just-in-time_compilation`

[8]V8, `https://code.google.com/p/v8/`

[9]`http://en.wikipedia.org/wiki/Chromium_(web_browser)`

[10]*Node.js*, `http://nodejs.org/`

[11]MongoDB, `http://www.mongodb.org/`

[12]The `Sadira` experimental framework, `http://sadira.iasfbo.inaf.it/`

III Workshop on Robotic Autonomous Observatories (Torremolinos, Malaga, Spain, 7-11 October 2013)
Editors: J. C. Tello, A. Riva, D. Hiriart & A. J. Castro-Tirado

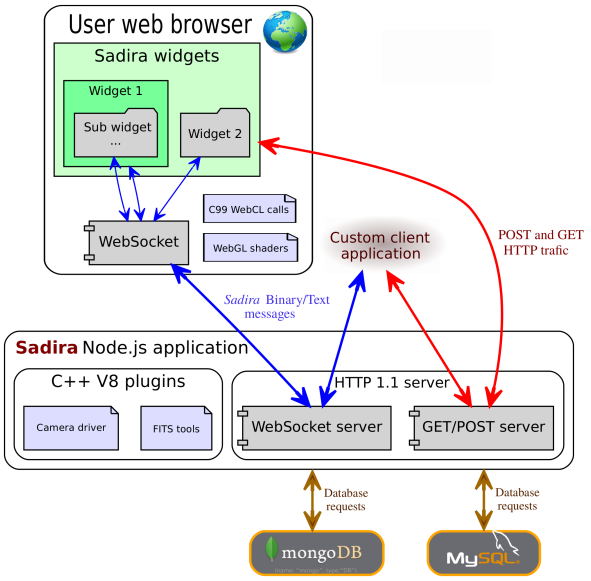

Fig. 1. Overview of the main `Sadira` components.

imental framework used for testing new HTML technologies, database systems and data processing tools. Developed within the EU/FP7 funded GLORIA project, `http://gloria-project.eu/`, it is a tool of interest for the astronomical community at large. Figure 1 shows its main components.

2. `Sadira` FRAMEWORK

The HTML DOM[13] displayed in the user's browser is populated by a dynamically constructed JavaScript object tree made of `Sadira` widgets. The browser widgets interact with the *Node.js* server handler functions through a message based protocol built on top of the WebSocket[14] or WebRTC[15] API. When widgets send messages, their "coordinates" in the widget tree are passed to the server so that incoming reply messages can be routed back to them. A `Sadira` web page can contain hundreds of widgets, all able to receive custom messages from the server, allowing for real-time updates of single objects of the page and a fully interactive user interface.

2.1. *TCP socket based data exchange*

In the HTTP 1.1 specifications, requests and submissions of data are sent to the server by the user's browser as GET and POST methods respectively. Unfortunately, there is no way for the server to contact the client web browser on its own, even if some non-standard use of the GET method could provide some sort of solution[16]. Today, thanks to the new WebSocket protocol (built as an "extension" of HTTP 1.1), a new full-duplex TCP/IP based socket API is available for the communication between browsers and web servers.

Most of the data exchange between the `Sadira` *Node.js* server and the browser widgets is done through a single WebSocket connection. The TCP connection is established only once per browser process, so there is no overload due to multiple TCP/IP connection initiations, nor there is the need to transfer HTTP headers. However the WebSocket API is limited to browser-to-server connections only: there is no "listen" function available browser-side. This limitation can be overcome by the use of the new, and now quite mature, WebRTC[17] API, opening the path to any kind of peer to peer (P2P) data applications between browsers without the need of an intermediary server.

2.2. *Message protocol*

The WebSocket protocol is very minimalist. Data is sent through the socket as a sequence of binary *frames* constituting the fragments of the *message*. There is no limitation on the message or fragment sizes, although browsers may impose their own arbitrary limit. Messages can be made of a single frame. The data payload of the messages can be either UTF-8[18] character string or arbitrary binary data.

To benefit of the asynchronous capabilities of JavaScript and to be able to handle multiplexed data streams coming from different widgets at the same time through a single WebSocket, the messages need to be fragmented in reasonable size pieces. To achieve this, both client and server JavaScript must provide methods to construct and reassemble the data fragments.

In order to provide the user with a simple communication API, the `Sadira` framework defines its own message protocol on top of the low-level one provided by WebSocket. As mentioned, there are two kinds of `Sadira` messages: UTF-8 and binary. The UTF-8 encoded is simply an encapsulation of the JSON[19] text format. On the other hand, the binary messages are more complex but allow for a more powerful use. Each binary message is made of

[13] Document Object Model, `http://www.w3.org/TR/DOM-Level-2-HTML/`

[14] `https://tools.ietf.org/html/rfc6455`

[15] WebRTC, Web Real-Time Communication, `http://www.webrtc.org/`

[16] See, *comet* to the programmer, `http://en.wikipedia.org/wiki/Comet_(programming)`

[17] http://www.webrtc.org/

[18] UTF-8 definition, see `http://tools.ietf.org/html/rfc3629`

[19] ECMA-404, better known as JSON, `http://www.json.org/`

III Workshop on Robotic Autonomous Observatories (Torremolinos, Malaga, Spain, 7-11 October 2013)
Editors: J. C. Tello, A. Riva, D. Hiriart & A. J. Castro-Tirado

a header describing the size, byte-offsets and type of the binary payloads following the header in the message data stream. These payloads are either BSON[20] data blocks or arbitrary binary data, like files, vector or image data bytes.

Binary data manipulation in JavaScript is not completely standardized yet and the performance of the code varies among the different JavaScript engines. Nevertheless, transferring binary data instead of text can make a huge difference in terms of bandwidth usage efficiency and, on the client side, data loading elapsed time. The server load is significantly reduced too.

2.3. Node.js *C++ add-ons*

JavaScript is a great language to write complex object oriented code, but in some situations it lacks the performance of compiled object code. This makes it inefficient to accomplish some types of tasks, like executing long loops containing massive floating point operations.

Thanks to V8, the *Node.js* interpreter being written in C++, it is possible to extend the JavaScript engine built-in functions and objects with new add-on objects and functions written in C++, dynamically linked to the *Node.js* interpreter when needed. This feature solves, at least server side, the limitations of JavaScript when executing computing intensive tasks. We will see in the next section how we can deal with the same problem client side, in the browser.

The `Sadira` server will make use of these C++ add-ons to provide JavaScript interfaces toward intensive computing tasks. In fact it allows the user to build front-ends to existing C/C++ libraries or to write applications that need low-level device access, like CCD camera drivers[21].

2.4. *Data visualization*

The complexity and amount of the available astronomical data call for new, "modern" data visualization tools. Data analysis pipelines, powerful graphical database manager/browser and multi-dimensional visualization tools can finally be combined in a single environment we all are familiar with: a web browser (Ricci & Nicastro 2013).

SVG[22], already supported in HTML4, allows for rich and complex vector graphics to be rendered in the browser. High level JavaScript libraries like D3.js[23] (Bostock et al. 2011) help the web developer in the creation of rich and dynamic SVG vector graphics (Ricci et al. 2013).

HTML5 offers new possibilities for raster graphic display with the introduction of the `<canvas>` tag, allowing the programmer to access the pixel bytes of a rectangular area, the *canvas*. The new HTML specifications also introduces WebGL[24], an OpenGL ES 2.0 JavaScript interface able to render accelerated 3D graphics in a HTML `<canvas>`.

Although WebGL allows for compiled "shader objects" to be fed directly into the GPU[25], hence providing accelerated graphics, the JavaScript limitations make some computing tasks, like texture pixels or vertex computations, too slow to be usable. The situation will change quickly with the launch of an OpenCL[26] JavaScript front-end in the browser: WebCL[27]. It will allow high performance parallel computations to take place within the browser, replacing slow JavaScript functions with fast object-code whenever needed.

2.5. *Databases: MongoDB and MySQL*

The `Sadira` framework makes intensive use of the non-relational MongoDB database system. The BSON data storage model used by MongoDB allows complex data structures to be created and managed on the fly; something very difficult to achieve with classical relational databases.

The absence of pre-defined data structures, and hence of a static "relational data model", is a nice feature of NoSQL databases. However that extreme freedom needs to be properly handled, or even restricted by the programmer in order for the database system to be usable. To answer this problem, `Sadira` introduces user defined *templates* which describe the expected *minimal* content of the MongoDB documents within a given collection. Templates are simple JSON structures containing 1. the key name, 2. type and 3. access rights of the member objects expected to be found in a document belonging to a given MongoDB collection.

`Sadira` also uses the relational database

[20] BSON, binary version of JSON, see `http://bsonspec.org/`

[21] See the `Sadira` *node-fits* and *node-sbig* C++ add-ons, `https://github.com/GLORIA-project/node-fits`

[22] SVG, Scalable Vector Graphics, `http://www.w3.org/Graphics/SVG`

[23] D3, Data-Driven Documents JavaScript library, `http://d3js.org`

[24] WebGL, OpenGL ES 2.0 for the Web, `http://www.khronos.org/webgl/`

[25] GPU, Graphics processing unit, `http://en.wikipedia.org/wiki/Graphics_processing_unit`

[26] OpenCL, Open Computing Language, `http://www.khronos.org/opencl/`

[27] WebCL, Heterogeneous parallel computing in HTML5 web browsers, `http://www.khronos.org/webcl/`

III Workshop on Robotic Autonomous Observatories (Torremolinos, Malaga, Spain, 7-11 October 2013)
Editors: J. C. Tello, A. Riva, D. Hiriart & A. J. Castro-Tirado

MySQL[28], in particular when the data structure is fixed, so that it can provide optimized and faster queries than MongoDB.

2.6. *Astronomical images database*

2.6.1. *FITS format based instrument data*

Most of the instruments used by astronomers store their data in the FITS file format[29]. FITS is a versatile format made of a sequence of data segments called *header data units* (HDU). These HDUs can be of three kinds: ASCII text buffers, binary data-cubes or data tables; in any number and order. Furthermore, each of these HDUs are preceeded by an ASCII header made of a list of key/value pairs, the metadata.

The *great* freedom given to the user by the FITS specifications explains both why it is still in wide use in the astronomical community and also why it is impossible to handle FITS files relying solely on their key/value information, without further knowledge of its content. The solution we propose is again to use a non-relational database, MongoDB in particular, to store FITS-based instruments data and metadata[30]. MySQL can still be used in parallel, as we do.

`Sadira` includes a *Node.js* C++ add-on `node-fits` providing interfaces to JSON or the JavaScript representation of the FITS data, leaving to the higher level JavaScript the sorting task and the MongoDB communication task.

2.6.2. *MongoDB data structure*

Generic `Sadira` "root" templates must be defined for every category of data we need to manage. These templates are linked to a given MongoDB collection in a database and describe the minimum keywords content excpected for the "documents" belonging to that collection. This way the document structure becomes constrained, though in a quite loosy way.

In the case of astronomical data, and CCD imagery in particular, we need at least two collections: `instruments` and `observations`. These collections need to be constrained by their respective root templates. More specialized instruments will be represented by new templates inheriting all the properties of a root template, but extending its characteristics with new fields. Although the various instruments will have different fields, they can be recorded into the same `instrument` collection because there is no rule in MongoDB regarding the field content. However, the constraint for each instrument to own at least the root template fields will ensure that we can manage them all in an aggregate way.

FITS-based instruments will have to provide a special JSON structure containing a list of keyword/-value pairs, translating some of the FITS keywords into (mandatory or not) database keywords. This extra knowledge provided by the instrument definition itself will allow us to automatically and simultaneously handle data coming from various "known" instruments.

A web graphical interface based on specific `Sadira` widgets is beeing written to perform the MongoDB management: the user will be able to edit/create new templates, create new collections for new instruments or experiments, and associate a base template to them.

3. CONCLUSIONS

Computer science is in constant evolution and mutation, and the astronomical community can surely benefit from its new capabilities. In particular the Internet and its associated technologies are evolving at a fast pace as they have become the heart of modern communication systems.

In this paper we briefly described some of these new web technologies from an astronomer's perspective and presented the experimental `Sadira` framework we are developing within the GLORIA project with the aim of testing and, in the near future, providing new modern software libraries and tools to the astronomical community.

Acknowledgements GLObal Robotic telescopes Intelligent Array for e-Science (GLORIA) is a project funded by the European Union Seventh Framework Programme (FP7/2007-2012) under grant agreement number 283783.

REFERENCES

Bostock, M., Ogievetsky, V., & Heer, J. 2011, IEEE Trans., 17, 2301

Ricci, D. & Nicastro, L. 2013, in EAS Publications Series, Vol. 61, EAS Pub. Ser.(Castro-Tirado, A. J. and Gorosabel, J. and Park, I. H.), 263

Ricci, D., Nicastro, L., & Pio, M. A. 2013, in INTED2013 Proceedings, 7th International Technology, Education and Development Conference (IATED), 2998

[28]MySQL database, `http://www.mysql.com/`

[29]`http://fits.gsfc.nasa.gov/`, Astronomical instruments can even output more than one FITS file per exposure, for example one for each CCD of a multi-CCD camera.

[30]GridFS, `http://docs.mongodb.org/manual/core/gridfs`