ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA184

MALCOLMJS: A BROWSER-BASED GRAPHICAL USER INTERFACE

I. J. Gillingham, T. Cobb
Diamond Light Source, Oxfordshire, UK

Abstract

A browser-based graphical user interface has been developed at Diamond. It is known as MalcolmJS as it communicates using Diamond's Malcolm Middleware protocol. The original goal was to communicate, via Websockets with a PandABox [1] in order to allow a user to examine and set attributes of numerous functional blocks within the instrument. With the continuing maturity of the Javascript language, in particular the release of ES6, along with the availability of off-the-shelf reactive open-source Javascript libraries, such as Facebook's React and Node.js, a rich set of tools and frameworks have entered the arena of user interface development suitable for control systems. This paper describes the design decisions based on these tools, experiences and lessons learned during and after the development process and the possibilities for future development as a generic, adaptable framework for instrument and control system user interfaces.

INTRODUCTION

In 2016/17, Diamond (in collaboration with SOLEIL) developed a digital signal level converter and position capture unit, named PandABox. PandABox consists of a number of static functional blocks defining functionality such as pulse stretching or position compare that can we wired together by an end user at run-time. An ASCII based TCP protocol allows control system access to the device, but a graphical interface was desired to allow end users to easily visualize the wiring of these functional blocks. It was also deemed necessary for an end user to be able to configure a PandABox without having to install any software on their computer. Having developed a tightly specified middle layer service named "Malcolm" [2], which allows high level configure/run control of control system components, the next stage was the design of a companion browser based user interface that has been named MalcolmJS.

REQUIREMENTS

Malcolm objects form a network of parent child relationships with duplicated nodes. For example, the same motor controller is likely to be used in many scans. A typical tree of objects might look like that shown in Fig. 1.

The bold items correspond to blocks, everything else is an attribute. The Palette items are generated by the User Interface(UI) giving a list of blocks from which the user can drag and drop (Fig. 2).

Outline UI Requirements

The external devices have diverse functionality; MalcolmJS must, therefore interrogate the attached device(s) for

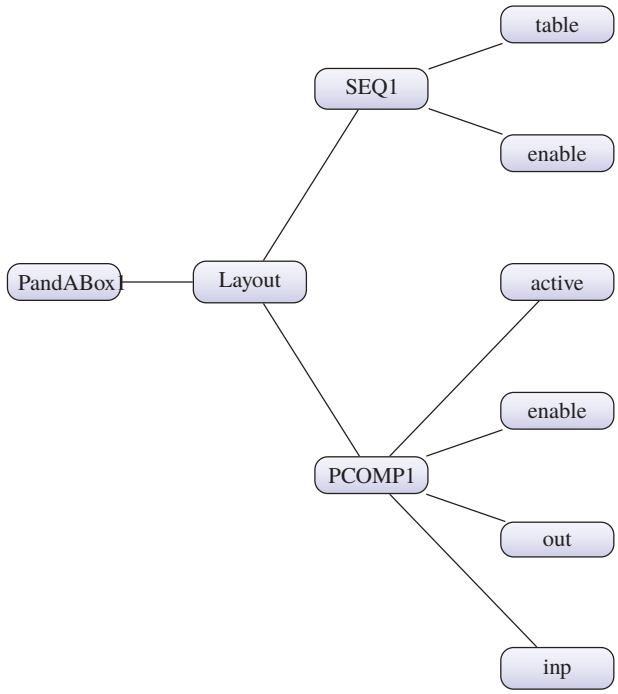

Figure 1: Example block and attributes tree structure.

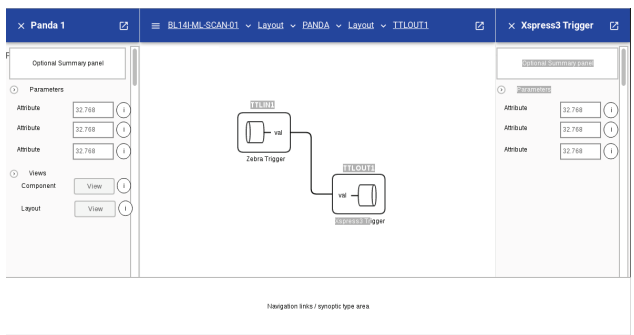

Figure 2: Proposed GUI.

information on all the available function blocks, attributes and associated meta-data. The collection of internal resources then being listed in a "palette" in the right-hand side-pane of the UI. A user will then have the option of selecting a block from the side-bar, dragging and dropping it onto the central canvas panel. A number of ports will be shown along the boundaries of each block; the ports have a type attributed to them and ports of compatible types can be connected together by selecting one port and dragging a connector "wire" to another port. When this is completed, the same connection information is transmitted to the device, where physical connections are made. This might, for example, be the "out" port of a Look Up Table(LUT),

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA184

connected to the "enable" port of an Output Encoder (OUTENC).

Usage needs to be intuitive and thus the user interface design must follow a standard approach, giving a familiar user experience (UX).

DESIGN DECISIONS

Target Platform

The user constructs the configuration in the central main instrument configuration area (main-pane) by placing blocks using drag-and-drop. The connections between the blocks are then made by selecting a defined port on one block and dragging a "wire" to a port on another block. After checks for port type compatibility, the new association will be sent to the instrument, where the connection is made in internal logic. With the demand of a high level of user interactivity, it became apparent that the user interface should be reactive and intuitive. Targeting standard HTML5 capable browsers was considered to be the most expedient route to being operating system agnostic and with this in mind, it was decided to create a prototype application, using a popular, off-the-shelf, open source framework.

React

React [3] was developed by Facebook and open sourced in 2013. It has been adopted as the Javascript framework of choice for MalcolmJS for the following reasons:

- Rendering in a virtual DOM (Document Object Model) – the react framework takes care of rendering only those items which have changed, so making the application very responsive.
- Clean Abstraction – UI components and information stores have clean, well defined interfaces, which reduces coupling and improves testability.
- Flux design pattern – Information flow is strictly in one direction only, with components being notified when data have changed, allowing them to update visual representation when required.
- Development tools – an impressive selection of open source tools.
- Long Term Support – React has already become a leading industry standard, having been adopted by large organisations such as Netflix, Facebook, Atlassian and many others. It has significant development momentum and a very large developer community.

Websockets

Historically, posting data from a browser application has been a straightforward; however receiving data asynchronously from an external source was problematic and always a compromise, utilising long-polling or AJAX techniques. Recently, Websockets [4] have become a mature and reliable method of delivering a persistent and bi-directional interface, facilitating an external entity to actively push information to a browser-based application at any time. Also the application can push information to an external entity, without the need to establish a full HTTP request.

Javascript Object Notation(JSON) Based Protocol

As JSON [5] is widely supported in most programming languages and a number of open source libraries are also available, the communications protocol between the browser application and external systems has been developed with JSON as the information exchange language. JSON is easily human readable, which brings an advantage to the development process.

Node Package Manager(NPM)

NPM [6] is the package manager for Javascript and the world's largest software registry. It was chosen for the wealth of tools and libraries it provides. The NPM repository offers approximately 475,000 open source tools and code modules. A sample of some of the NPM packages used within MalcolmJS is:

- interact.js – to manage user dragging of components on screen
- react-toolbox – a set of React components that implement Google's Material Design specification
- karma – a Javascript tool to facilitate Test Driven Development (TDD)

These are shown in Fig. 3.

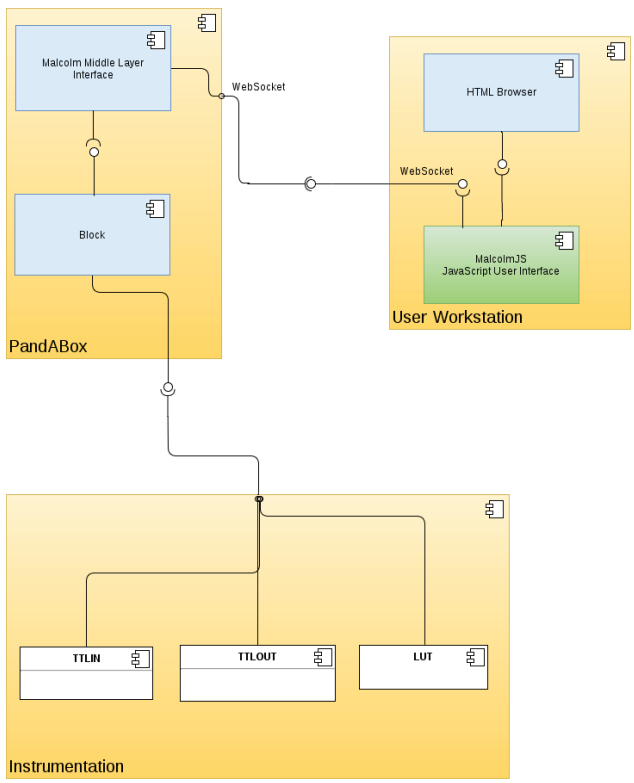

Figure 3: Component Context Model.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA184

Webpack

Webpack is a module bundler for modern Javascript applications, facilitating code to be designed, written and tested in a structured fashion, using the latest ECMAScript (ES) language edition, such as ES6. Most modern browsers fully support ES5, but not the full set of features of ES6 (and above) Javascript editions. To realise the benefits of developing with ES6, it has been necessary to transpile the ES6 code down to ES5 (or lower), as a bundled monolithic Javascript file. This is readily achievable using the Babel package along with Webpack.

User Interface Style

With the number of 'apps' available for mobile and desktop devices alike, growing at a seemingly exponential rate, users have come to expect an intuitive and responsive user interface and experience. This is typically achieved by adopting a standard, well documented design pattern, such as Google's Material-UI, Polymer and Semantic-UI. The UI pattern that was adopted for MalcolmJS is Material-UI [7], which has been achieved using the NPM packge: *react-toolbox*, which provides numerous components to facilitate a versatile user interface to be constructed quickly and built-in conformance with the Material-UI standard. Regardless of the platform used (mobile or desktop devices of various screen sizes), the user interface will automatically adapt appropriately for the device's format.

DEVELOPMENT PROCESS

Simulator

MalcolmJS is principally driven by the changes in external instrumentation. To facilitate efficient development and testing, a device simulator, communicating using Websockets and the Malcolm Protocol [8], was developed. This can provide further diagnostic information within the simulator in conjunction with a Javascript debugger on the client browser. As Websockets are the communication interface of choice, which operates seamlessly over a local connection within the client's host machine, or remotely over a network interface, the simulator provides an analogue which is close to the real thing.

Project Structure

The structure of the MalcolmJS development configuration has been given careful consideration. Javascript modules for views and stores are located in separate directories from the main project directory. Webpack is used as the central builder agent, whose project configuration is stated fully in webpack.config.js. The configuration file is itself, a Javascript module, which exports a Javascript object; it sets out rules as to what helper tools should be run to process the source code, how to handle style sheets and how to transpile the source into the bundled Javascript file, ready to send to the browser.

Debugging

Debugging the Javascript code is facilitated on the browser, using tools such as Web Developer Tools under Chrome or Firefox. Although the code which is running on the browser is a transpiled bundle from the original code, it is straightforward to perform a comprehensive debug, as the debugger can be pointed at souce code map file, generated by Webpack; so making available much needed functionality, such as single stepping and exception stack traces.

EXPERIENCES AND LESSONS LEARNED

Selecting Third Party Tools and Packages

The NPM packages can be readilly incorporated into the application under development, simply by 'npm install <package>'. However it is important to consider the following options and questions before employing a package:

- When was the package last updated and are there likely to be future releases to maintain compatibility with the ever evolving React platform, which has short release intervals?
- How popular is the package and how well is it rated among peer developers?
- On how many other packages is a particular package dependent? As this can lead to the bloating of the final application.

It is vital that libraries keep pace with developments in the React framework, or risk deprecation. This was the case with 'react-panels', which after a year of no maintenance by its author, became impossible to include in the local MalcolmJS project build. After some evaluation, react-toolbox was adopted, which is a very popular and well maintained UI library.

ECMAScript

Javascript is now a mature, highly versatile and efficient language, with an impressive code base of open source libraries. ES6 is now a well adopted standard and ES7 is also being deployed with many rich features fulfilling the expectation of a modern language. It is a robust tool for developing not just browser applications, but also stand-alone applications, running on various platforms.

Discard the Prototype

The prototype is invaluable as a vehicle for evaluating concepts and refining specifications, but be prepared to discard it. Putting effort into preserving code which has mostly evolved, is false economy; as it has been bolted together, new bits roughly glued in place, with less than optimal design coherence, maintenance of the code would be difficult and risky. This was the case with MalcolmJS, when it became apparent, especially as specifications evolved, that a re-write needed to be undertaken, whilst exploiting architecture and techniques from elements of the prototype.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA184

In hindsight, by undergoing a fundamental redesign for the production code base of MalcolmJS, it's likely that significant time and effort could have been avoided.

SUMMARY OF PROGRESS TO DATE

The Diamond Control Systems has now implemented a number of Malcolm compatible instruments across beamlines. Whilst these devices can be configured through other means, there is an increasing need to roll out an operational web interface - MalcolmJS. An alpha test release has been made internally and the feedback has been highly productive, with constructive changes to the original specification. It is likely that MalcolmJS will evolve with increasing versatility as user and instrument requirements change over time. Not only has the development of this application been beneficial to data acquisition and control of PandABox, but it potentially paves the way for adoption with accelerator control systems generally, due to the responsive nature of Websockets communication.

REFERENCES

[1] Zhang *et al.*, "PandABoxBox: A Multipurpose Platform For Multi-technique Scanning and Feedback Applications", TUAPL05, presented at ICALEPCS 2017, Barcelona, Spain, this conference.

[2] T. Cobb, *et al.*, "Malcolm: A Middlelayer Framework for Generic Continuous Scanning", TUPHA159, presented at ICALEPCS 2017, Barcelona, Spain, this conference.

[3] React: A Javascript library for building user interfaces, https://reactjs.org

[4] The WebSocket Protocol, https://www.websocket.org

[5] Introducing JSON, http://www.json.org

[6] npm: https://www.npmjs.com

[7] Material-UI: Google Inc., https://material.io

[8] Pymalcolm message documentation, http://pymalcolm.readthedocs.io/en/latest/reference/messages.html