Designing and developing portable large-scale JavaScript web applications within the Experiment Dashboard framework

J Andreeva, I Dzhunov, E Karavakis, L Kokoszkiewicz, M Nowotka, P Saiz, D Tuckett

CERN, European Organization for Nuclear Research, CH-1211 Genève 23, Switzerland

E-mail: david.tuckett@cern.ch

Abstract. Improvements in web browser performance and web standards compliance, as well as the availability of comprehensive JavaScript libraries, provides an opportunity to develop functionally rich yet intuitive web applications that allow users to access, render and analyse data in novel ways. However, the development of such large-scale JavaScript web applications presents new challenges, in particular with regard to code sustainability and team-based work. We present an approach that meets the challenges of large-scale JavaScript web application design and development, including client-side model-view-controller architecture, design patterns, and JavaScript libraries. Furthermore, we show how the approach leads naturally to the encapsulation of the data source as a web API, allowing applications to be easily ported to new data sources. The Experiment Dashboard framework is used for the development of applications for monitoring the distributed computing activities of virtual organisations on the Worldwide LHC Computing Grid. We demonstrate the benefits of the approach for large-scale JavaScript web applications in this context by examining the design of several Experiment Dashboard applications for data processing, data transfer and site status monitoring, and by showing how they have been ported for different virtual organisations and technologies.

1. Introduction

We present an approach to designing and developing large-scale JavaScript web applications that achieves the goals of rich functionality, code sustainability, and data source portability. The approach is presented in the context of the Experiment Dashboard [1] framework, which is used to develop applications for monitoring the distributed computing activities of virtual organisations on the Worldwide LHC Computing Grid (WLCG) [2], however it can be applied to web application development in general.

In section 2, we look at traditional and JavaScript web user interfaces (UI) with examples from past and present Experiment Dashboard applications. We consider traditional web UIs, highlighting the core concepts, as well as their strengths and weaknesses. In a similar way, we consider JavaScript web UIs, showing how they overcome the weaknesses of traditional web UIs but also raise new issues of their own. We conclude this section by looking at how improvements in web browser performance, web standards compliance and JavaScript libraries have already resolved some of the issues raised by JavaScript UIs and by asserting that the remaining issues can be resolved through careful software design.

In section 3, we present an approach to developing JavaScript web UIs, which implements a client-side model-view-controller (MVC) framework and leverages several open-source JavaScript projects to realise the full benefits of JavaScript UIs whilst avoiding the common pitfalls. We highlight that, in this approach, the UI is treated as an external application and show that this naturally leads to decoupling of the data source behind a web API.

In section 4, we introduce the concept of a configurable UI, which, based on the presented approach, mandates data source decoupling and supplements the client-side MVC framework with pre-defined view plug-ins, in order to enable very rapid development of functionally rich UIs for new data sources and use cases. We present two configurable UIs: hbrowse [3] and xbrowse [4], which are suited to presenting hierarchical and matrix structured data respectively. We explore more deeply the concept of configurable UIs by examining the implementation of xbrowse and its application to the development of a current Experiment Dashboard application.

We conclude the paper by considering what has been and can be achieved with the presented approach to web application development, including configurable UIs, whilst identifying areas that could be improved in future work.

2. Web user interfaces

In this section, we look at the strengths and weaknesses of traditional web UIs. We see that JavaScript web UIs address these weaknesses but raise their own issues, which, we assert, can be addressed within the paradigm.

2.1. Traditional web user interfaces

The very first web pages were hypertext based [5], where links in one page will load a different related page. Traditional web UIs follow this paradigm but the content is dynamically generated based on the parameters attached to a given link. Key characteristics of such UIs are multi-page interface, full-page load for each user interaction, and server-side view generation. Figure 1 shows a typical user interaction with a traditional web UI, highlighting these characteristics (the technologies referenced in the figure are those used within Experiment Dashboard applications in particular).

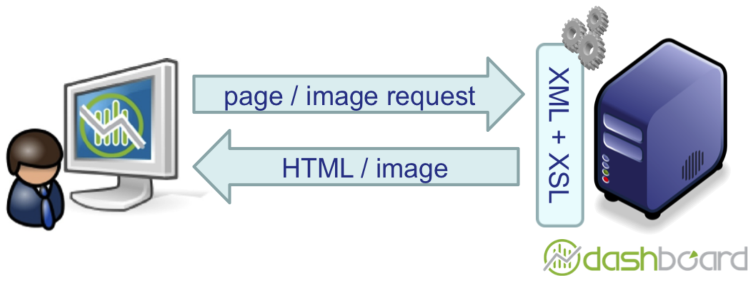
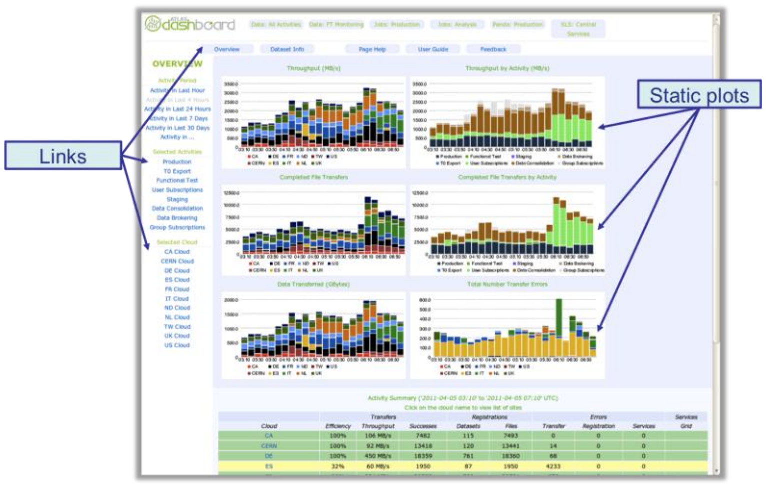


Figure 1. Typical user interaction with a traditional web UI.

A representative example of a traditional web UI is that of the ATLAS DDM Dashboard 1.0 [6]. This application was used until 2011 by ATLAS distributed computing shifters, site administrators and management to monitor data transfers over the WLCG. Figure 2 shows a screenshot from the application. The links reload the full page with different content based on the link parameters. The plots are static images generated on the server-side providing no interactivity beyond linking to another page or plot.



Activity Summary ('2011-04-05 03:10' to '2011-04-05 07:10' UTC)
Click on the cloud name to view list of sites

	Transfers			Registrations		Errors			Services
Cloud	Efficiency	Throughput	Successes	Datasets	Files	Transfer	Registration	Services	Grid
CA	100%	106 MB/s	7482	115	7493	0	0	0	
CERN	100%	92 MB/s	13418	120	13441	14	0	0	
DE	100%	450 MB/s	18359	761	18360	68	0	0	
ES	32%	60 MB/s	1950	87	1950	4233	0	0	

Figure 2. Annotated screenshot from ATLAS DDM Dashboard 1.0 as a representative example of a traditional web UI.

The key strengths of traditional web UIs include:

- Cross-browser compatibility
- Modularity for team working (each page naturally corresponds to a feature)
- Bookmarkable URLs
- Search-engine friendly
- Accessibility friendly

The strengths are mostly due to the UI being essentially a set of linked documents rather being highly interactive. The flip side of these strengths is the following weaknesses:

- Low interactivity
- Slow loading

Low interactivity prevents users from easily customising data visualisation. Slow loading is due to a full-page load for each user interaction and the need for a server-side call even for a new view of the same data.

The weaknesses of traditional web UIs are not simply inconvenient but, in a real way, hinder novel analysis of data. We note that these weaknesses are largely inherent and cannot be fully addressed within the paradigm of traditional web UIs, which brings us to consider JavaScript web UIs.

2.2. JavaScript web user interfaces

Modern web UIs can be highly interactive and provide a user experience akin to a desktop application. Such UIs may be implemented with a number of different technologies but we focus our attention on web UIs built using JavaScript and AJAX. Key characteristics of such UIs are single-page interface, data loaded on-demand via AJAX, and client-side view generation. Figure 3 shows a typical user interaction with a JavaScript web UI, highlighting these characteristics (the technologies referenced in the figure are those used within Experiment Dashboard applications in particular).

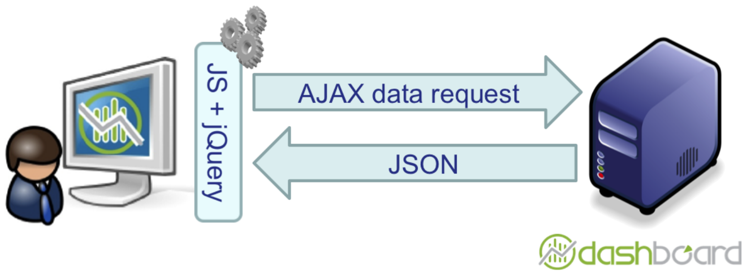


Figure 3. Typical user interaction with a JavaScript web UI.

A representative example of a JavaScript web UI is that of the ATLAS DDM Dashboard 2.0 [7]. This application was developed to replace version 1.0, discussed in the previous subsection. As such, it provides a much richer more interactive visualisation of the data seen in version 1.0. Figure 4 shows a screenshot from the application. Elements typical of a desktop application, such as an accordion, movable panels and sliders, are present. User interactions cause data to be loaded on demand and only relevant parts of the interface are updated. The plots are generated on the client-side and provide interactive features such as hover tips, zooming, styling, types (column, area, pie), etc. without the need for any server-side calls.

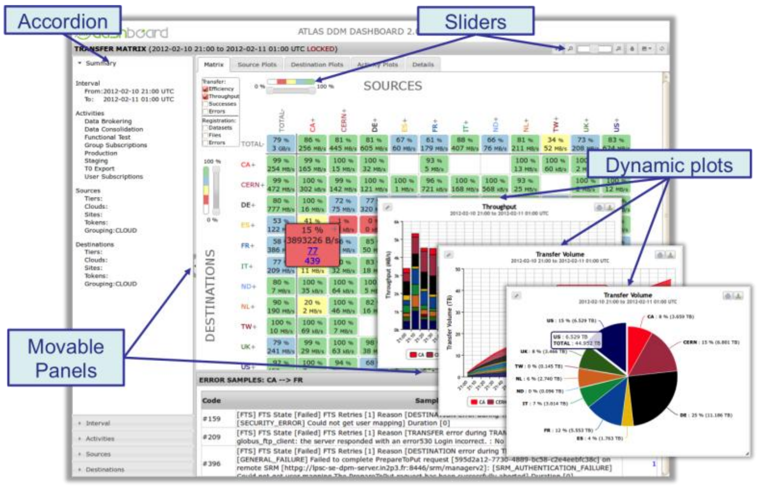


Figure 4. Annotated screenshot from ATLAS DDM Dashboard 2.0 as a representative example of a JavaScript web UI.

The key strengths of JavaScript web UIs, with respect to traditional web UIs include:

- High interactivity.
- Fast-loading.

High interactivity allows the user to easily customise data visualisation. Fast loading is due to data being loaded on demand and the ability to generated new views of the same data on the client-side avoiding a server-side call. The flip side of these strengths is the following weaknesses:

- Browser incompatibilities
- Slow client-side rendering
- Lack of modularity for team working (single page contains all features)
- Non-bookmarkable URLs
- Not search-engine friendly
- Not accessibility friendly

Comparing the above strengths and weaknesses with those of traditional web UIs, it is immediately apparent that they are the converse of each other. However, just as the corresponding weaknesses of traditional web UIs are not simply inconvenient, the strengths of JavaScript web UIs are not simply convenient but, in a real way, facilitate novel analysis of the data. It is therefore worth attempting to address the weaknesses of JavaScript web UIs, and we show in the next subsection that, unlike those of traditional web UIs, they can be addressed within the paradigm.

2.3. Addressing JavaScript web user interface issues

Several of the issues of JavaScript web UIs, identified in the previous subsection, have already been resolved by improvements in web browsers and JavaScript libraries. Browser incompatibilities have largely been eliminated by improved standards compliance by the current releases of the major web browsers. This is attested by the results of the Web Standards Project Acid3 [8] tests that check web browser compliance with elements of various web standards, particularly the Document Object Model (DOM) and JavaScript. Furthermore, JavaScript libraries, such as jQuery [9], provide abstraction layers that cover most of the remaining corner case browser incompatibilities. Faster hardware and browsers mean that slow client-side rendering is no longer a major issue and can often be resolved with basic tuning of UI code.

Resolving the remaining issues requires explicit effort on the part of the UI developer. Lack of modularity, for team working or sustainable development, can be resolved by using a client-side MVC framework and a view object per a feature. URLs can be made bookmarkable by explicit URL hash management using JavaScript library plugins. Search-engine optimisation and accessibility can be addressed in similar ways.

Although some explicit effort is required to resolve the issues of JavaScript UIs, all the issues can be resolved within the paradigm. An approach that can be used to achieve this is presented in the next section.

3. An approach to developing JavaScript web user interfaces

In this section we present an approach to developing JavaScript web UIs, which realises the full benefits of JavaScript UIs whilst avoiding the common pitfalls. Furthermore, we show that this approach naturally leads to decoupling of the data source behind a web API

3.1. False start

Like many web UI developers, our first attempts at building JavaScript web UIs were not sustainable. We did not use a client-side MVC framework, so the code was not sufficiently modular and the session state and application data were stored in form inputs and the DOM. We had not selected a single JavaScript library for all projects, and use of third-party JavaScript plugins was limited (partly due to lack of availability).

Although rich interactive UIs were developed in this way, the code was inevitably non-modular and highly coupled making it difficult to maintain or extend. The experience gained on these early projects proved invaluable in elaborating the approach presented in the next subsection.

3.2. The approach

This approach realises the benefits of JavaScript UIs such as high interactivity and fast loading whilst avoiding the common pitfalls such as non-modular code and non-bookmarkable URLs.

A key design feature is that the UI is treated as an external application, so that apart from the initial page request to load the UI, all interaction with the server consists of HTTP requests, specifying session state, and JSON responses, containing application data. Therefore the UI is neatly decoupled from the server, which becomes a pure data source and is only required to expose a well-defined restful web API.

At the core of the UI code is a client-side model-view-controller (MVC) framework with event-based communication between components. The model contains the session state, i.e. user selected filters, and application data, i.e. JSON data loaded from the server. Views are responsible for updating the model state in response to user interaction, reflecting changes in the model, and updating the model with data loaded from the server. Views are notified of changes in the model via events, so that more than one view can represent the same data in different ways. A typical UI will contain many view objects, with a set of views corresponding to a feature, so it becomes easy and natural to write modular code. The controller manages the URL hash keeping it synchronised with the model state, so that URLs are bookmarkable and browser history works as the user would expect. Figure 5 shows the various components involved and their relationships to each other and the web API.

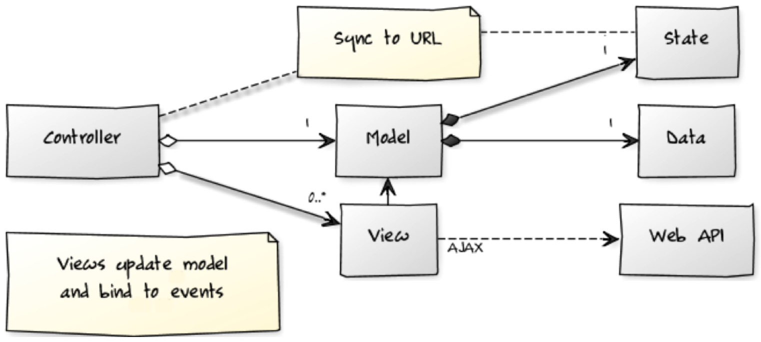


Figure 5. Client-side MVC components and their relationships to each other and the web API.

The data flow following a typical user interaction is shown in figure 6. The user makes an input, say selecting a time range from a dropdown menu. The view updates the model state with the user selected time range. The controller synchronises the model change to the URL. The controller in turn notifies the model of the URL change, which also allows changes in the URL to drive changes in the UI; this is essential to enable the UI to respond correctly to bookmarked URLs and browser history back / forward buttons. Any interested (i.e. bound) view receives an event notifying it of changes in the model state. In reaction to the model state change, the view may refresh its DOM element or load data asynchronously from the web API into the model data. Again any interested (i.e. bound) view receives an event notifying it of changes in the model data and it may refresh its DOM element accordingly.

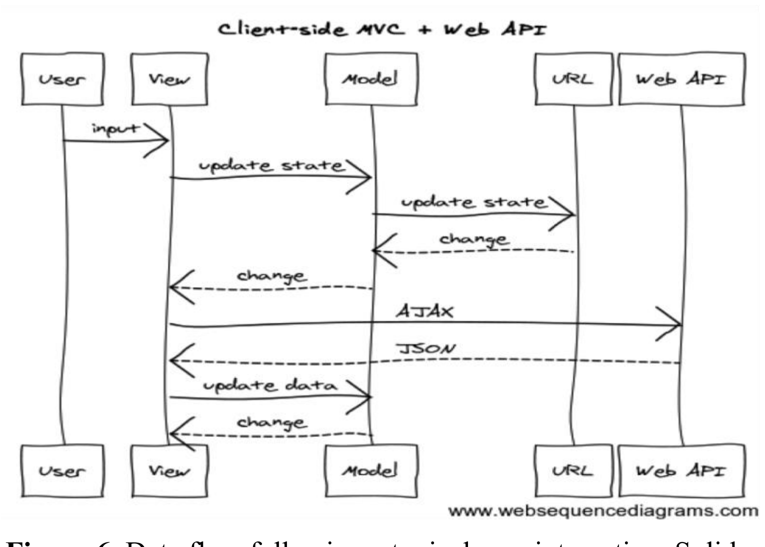


Figure 6. Data flow following a typical user interaction. Solid arrows are method calls. Dashed arrows are event notifications.

The MVC framework and UIs rely on a number of third-party JavaScript libraries and plug-ins. In particular, the highly popular jQuery core and UI libraries are used. In addition, a number of plugins are used: BBQ [10] (URL hash management), Handlebars [11] (templating), Highcharts [12] (plotting), DataTables [13] (tables), Underscore [14] (utilities), etc.

Following this approach, using the client-side MVC framework and leveraging JavaScript libraries and plugins, it is straightforward to build highly interactive, responsive yet modular and bookmarkable JavaScript web UIs. The approach therefore addresses all of the issues identified in section 2.2 with the exception of search engine optimisation and accessibility. Furthermore, by taking full advantage of the data source decoupling that comes from treating the UI as an external application, it is possible to extend this approach to the concept of configurable UIs that enable very rapid development of UIs for new applications. Configurable UIs are described in more detail in the next section.

4. Configurable user interfaces

In this section we introduce the concept of configurable UIs that, by supplementing the approach presented in the previous section with pre-defined view plug-ins and by taking advantage of data source decoupling, provide a means to rapidly develop functionally rich UIs for new data sources and use cases. We also look at two examples of configurable UIs that were used to develop the web UIs of several Experiment Dashboard applications.

4.1. Concepts

Configurable UIs can be thought of as providing a skeleton UI that can be adapted to set of use cases. Key design features include:

- Client-side MVC framework.
- Server interaction via well-defined web API.
- Pre-defined view plug-ins.

The client-side MVC framework should provide a configurable model and a plug-in architecture for views. Mandating that all server interactions pass via a well-defined web API ensures data source decoupling and hence portability to new data sources. The pre-defined view plug-ins should be sufficiently rich to be usable with minimal adaptation but sufficiently generic to be applicable to different use cases.

If these key design features are satisfied then adapting a configurable UI for a new use case, where the data source already provides a web API, consists of the following two steps:

- Define model state default values.
- Combine and adapt pre-defined view plugins.

In this way, functionally rich UIs can be set up for new applications in a few dozen lines of code, as we demonstrate in the next subsection.

4.2. Xbrowse

Xbrowse is a configurable UI that includes the client-side MVC framework presented in section 3, along with a number of pre-defined view plug-ins. The currently available view plug-ins are particularly suited to building web UIs for matrix structured data, such as transfer monitoring where each raw event has a source and a destination. However, with the addition of more view plug-ins, it can be adapted to other structures of data. It has already been adapted for use in the ATLAS DDM Dashboard and the WLCG Transfers Dashboard [15] for transfer monitoring and for an internal Experiment Dashboard application doing consistency monitoring between the two aforementioned applications.

The pre-defined view plug-ins include: sidebar, controls, tabs, matrix, plots, toolbars, etc. Figure 7 shows how some of these view plug-ins are represented in the ATLAS DDM Dashboard.

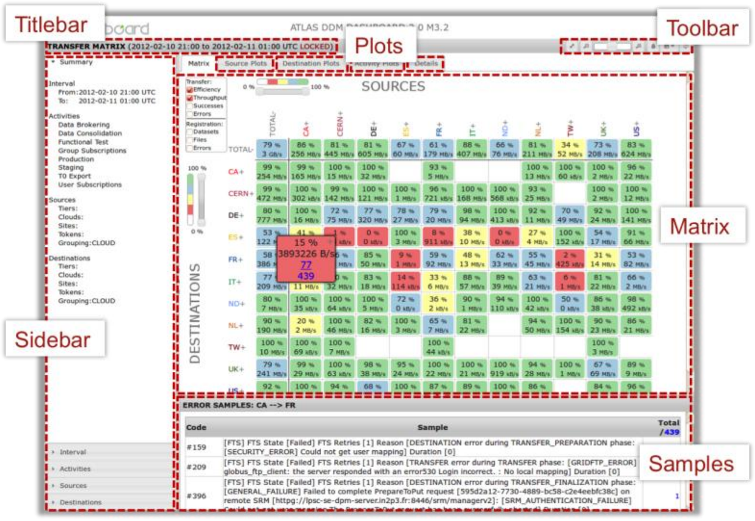


Figure 7. Annotated screenshot showing xbrowse view plug-ins in the ATLAS DDM Dashboard.

The only required adaptation of the model is to specify the default state values. Model adaptation can also include type information, to control how state values are serialised to the URL hash, and state change side effects. Figure 8 shows how to define default state values and types and also how to set up the side effect that date range changes force the plot bin size to reset.

```
/* Model Adaptor */
xmvc.adaptor = $.extend(true, xmvc.adaptor, {
  model: {
    post_init: function(model) {
      // define state defaults
      model.state.opts.defaults = {
        date:{interval:240, from:'', to:''},
        vo:['atlas', 'cms', 'lhcb'],
        ...
      };
      // define state types for automatic coercion
      model.state.opts.types = {
        date:{interval:'number', from:'date', to:'date'},
        ...
      };
      // define state change side-effects
      model.state.on('change:date', function(e) {
        this.set({plot:{bin:null}}); //reset plot.bin
      });
    }
  }
});
```

Figure 8. Xbrowse model adaptor code.

View plug-ins are adapted in many ways that may be specific to the view. Figure 9 shows how the matrix view is plugged into xbrowse and how the AJAX interaction can be adapted for a different web API. Typically the template may also be changed to provide a different representation.

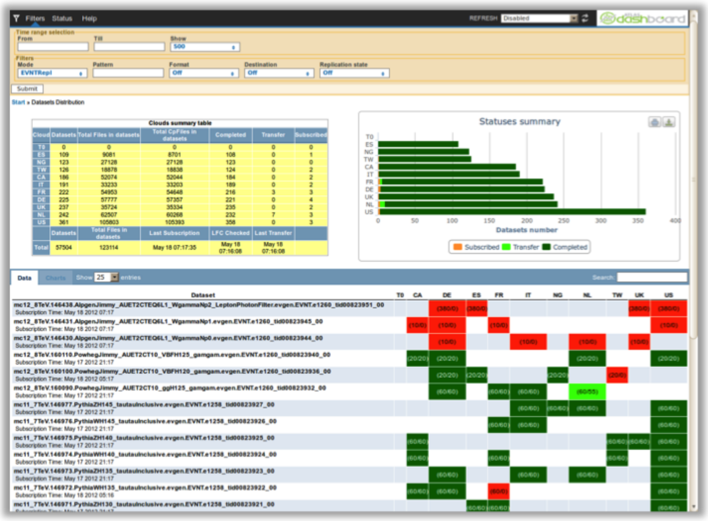
```
/* View Adaptor */
xmvc.adaptor = $.extend(true, xmvc.adaptor, {
  view:{
    create: function() { // define view plugins
      return [
        new ViewDates(), view_matrix, ...
      ];
    }
  }
});
var view_matrix = new ViewMatrix({
    ctx:'#t_tab_matrix', // DOM ID
    ajax:{
      url:'/dashboard/request.py/transfer-matrix', // AJAX URL
      params:function(state, data, opts) { // AJAX parameters
        return {
          from_date:state.date.from,
          to_date:state.date.to,
          vo:state.vo,
          ...
        };
    },
    template: ... // template (optional)
  }
});
(new xmvc.Controller()).init(); // initialize controller
```

Figure 9. Xbrowse view adaptor and matrix plug-in code.

Although only samples of code were presented here, the entire code to adapt xbrowse for the application seen in figure 7 consists of just a few dozen lines.

4.3. *Hbrowse*

Hbrowse is another example of a configurable UI. Like xbrowse it includes a client-side MVC framework, along with a number of pre-defined view plug-ins. However, it is specifically designed for the visualisation of hierarchical data with an unlimited number of levels. It has already been adapted to for use in several Experiment Dashboard applications including ATLAS Task Analysis [16], CMS Interactive View [17] and ATLAS Dataset Distribution [18]. Figure 10 shows a screenshot from ATLAS Dataset Distribution monitoring, which is based on hbrowse.



Clouds summary table

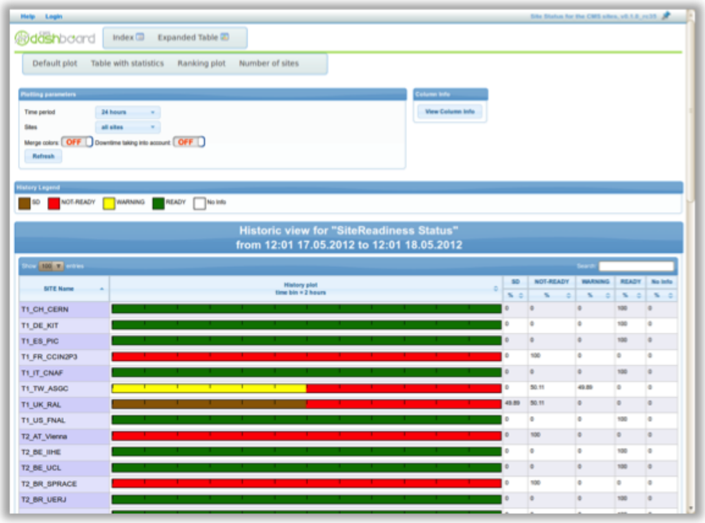
Cloud	Datasets	Total Files in datasets	Total CpFiles in datasets	Completed	Transfer	Subscribed
T0	0	0	0	0	0	0
ES	109	9081	8701	108	0	1
NG	123	27128	27128	123	0	0
TW	126	18878	18838	124	0	2
CA	186	52074	52044	184	0	2
IT	191	33233	33203	189	0	2
FR	222	54953	54648	216	3	3
DE	225	57777	57357	221	0	4
UK	237	35724	35334	235	0	2
NL	242	62507	60268	232	7	3
US	361	105803	105393	358	0	3
	Datasets	Total Files in datasets	Last Subscription	LFC Checked	Last Transfer	
Total	57504	123114	May 18 07:17:35	May 18 07:16:08	May 18 07:16:08	

Dataset

mc12_8TeV.146438.AlpgenJimmy_AUET2CTEQ6L1_WgammaNp2_LeptonPhotonFilter.evgen.EVNT.e1260_tid00823951_00
Subscription Time: May 18 2012 07:17
mc12_8TeV.146431.AlpgenJimmy_AUET2CTEQ6L1_WgammaNp1.evgen.EVNT.e1260_tid00823945_00
Subscription Time: May 18 2012 07:17
mc12_8TeV.146430.AlpgenJimmy_AUET2CTEQ6L1_WgammaNp0.evgen.EVNT.e1260_tid00823944_00
Subscription Time: May 18 2012 07:17
mc12_8TeV.160110.PowhegJimmy_AUET2CT10_VBFH125_gamgam.evgen.EVNT.e1260_tid00823940_00
Subscription Time: May 17 2012 21:17
mc12_8TeV.160100.PowhegJimmy_AUET2CT10_VBFH120_gamgam.evgen.EVNT.e1260_tid00823936_00
Subscription Time: May 18 2012 05:17
mc12_8TeV.160090.PowhegJimmy_AUET2CT10_ggH125_gamgam.evgen.EVNT.e1260_tid00823932_00
Subscription Time: May 17 2012 21:17
mc11_7TeV.146977.PythiaZH145_tautauInclusive.evgen.EVNT.e1258_tid00823927_00
Subscription Time: May 17 2012 21:17
mc11_7TeV.146976.PythiaWH145_tautauInclusive.evgen.EVNT.e1258_tid00823926_00
Subscription Time: May 17 2012 21:17
mc11_7TeV.146975.PythiaZH140_tautauInclusive.evgen.EVNT.e1258_tid00823925_00
Subscription Time: May 17 2012 21:17
mc11_7TeV.146974.PythiaWH140_tautauInclusive.evgen.EVNT.e1258_tid00823924_00
Subscription Time: May 17 2012 21:17
mc11_7TeV.146973.PythiaZH135_tautauInclusive.evgen.EVNT.e1258_tid00823923_00
Subscription Time: May 17 2012 21:17
mc11_7TeV.146972.PythiaWH135_tautauInclusive.evgen.EVNT.e1258_tid00823922_00
Subscription Time: May 18 2012 05:16
mc11_7TeV.146971.PythiaZH130_tautauInclusive.evgen.EVNT.e1258_tid00823921_00

Figure 10. Screenshot from ATLAS Dataset Distribution monitoring, which is based on hbrowse.

4.4. Site Status Board
Site Status Board [19] is not a configurable UI but a specific application. It is used for monitoring WLCG site and service status. It is deployed for the four main LHC [20] experiments and heavily used by CMS and ATLAS. It is mentioned here because it uses the approach presented in section 3 with a third-party client-side MVC framework called Backbone [21]. Standardising on a single client-side MVC framework is one of the future improvements proposed in the conclusion. Figure 11 shows a couple of screenshots from the Site Status Board.



Time period: 24 hours
Sites: all sites
Merge colors: OFF
Downtime taking into account: OFF

🟫 SD 🟥 NOT-READY 🟨 WARNING 🟩 READY ⬜ No info

Historic view for "SiteReadiness Status"
from 12:01 17.05.2012 to 12:01 18.05.2012

Show 100 entries

SITE Name	History plot time bin = 2 hours	SD %	NOT-READY %	WARNING %	READY %	No info %
T1_CH_CERN	🟩	0	0	0	100	0
T1_DE_KIT	🟩	0	0	0	100	0
T1_ES_PIC	🟩	0	0	0	100	0
T1_FR_CCIN2P3	🟥	0	100	0	0	0
T1_IT_CNAF	🟩	0	0	0	100	0
T1_TW_ASGC	🟨🟥	0	50.11	49.89	0	0
T1_UK_RAL	🟫🟥	49.89	50.11	0	0	0
T1_US_FNAL	🟩	0	0	0	100	0
T2_AT_Vienna	🟥	0	100	0	0	0
T2_BE_IIHE	🟩	0	0	0	100	0
T2_BE_UCL	🟩	0	0	0	100	0
T2_BR_SPRACE	🟥	0	100	0	0	0
T2_BR_UERJ	🟩	0	0	0	100	0

Figure 11. Screenshot from Site Status Board, which uses Backbone as its client-side MVC framework.

5. Conclusion
We have seen that whilst JavaScript web UIs can provide a much more interactive and responsive user experience than traditional web UIs, they raise their own issues. Some of these issues, such as browser incompatibilities and slow client-side rendering, are largely resolved by recent improvements in browsers and JavaScript libraries. Other issues, such as lack of modularity and non-bookmarkable URLs, have to be explicitly addressed by the developer. We presented an approach, based on a client-side MVC framework and decoupling the data source behind a well-defined web API, that addresses these issues and so facilitates the rapid development of functionally rich yet sustainable web UIs. We showed how the approach can be extended by supplementing the MVC framework with pre-defined view plug-ins to create configurable UIs. Examples were presented showing how these configurable UIs have been successfully adapted to many data sources and use cases, which proves the effectiveness of the approach presented in this paper.

Despite the proven success of the approach and configurable UIs, a number of areas may be improved in future work including standardisation on a common MVC framework, search engine and accessibility support, automated testing, documentation and optimisation.

References
[1] Andreeva J et al 2010 Experiment Dashboard for Monitoring Computing Activities of the LHC Virtual Organizations *J. Grid Comp.* 8 pp 323-339
[2] Shiers J 2006 The Worldwide LHC Computing Grid (worldwide LCG) *Proc. Of the Conference*

on Computational Physics (CCP06) Computer Physics communications 177 pp 219-223

[3] L Kokoszkiewicz et al 2012 hBrowse - Generic framework for hierarchical data visualization *Proc. of EGI Community Forum 2012 / EMI Second Technical Conference* PoS(EGICF12-EMITC2)062 (http://www.hbrowse.net/ retrieved 2012-05-18)

[4] Xbrowse (http://twiki.cern.ch/twiki/bin/view/ArdaGrid/XbrowseFramework retrieved 2012-05-18)

[5] Berners-Lee T, Cailliau R, 1990 WorldWideWeb: Proposal for a HyperText Project (http://www.w3.org/Proposal.html retrieved 2012-06-22)

[6] ATLAS DDM Dashboard 1.0 (http://dashb-atlas-data.cern.ch/dashboard/request.py/site retrieved 2012-05-18)

[7] ATLAS DDM Dashboard 2.0 (http://dashb-atlas-data.cern.ch/ddm2/ retrieved 2012-05-18)

[8] Web Standards Project Acid3 tests (http://www.webstandards.org/action/acid3/ retrieved 2012-06-22), (http://acid3.acidtests.org/ retrieved 2012-05-18)

[9] jQuery (http://jquery.com/ retrieved 2012-05-18)

[10] BBQ (http://benalman.com/projects/jquery-bbq-plugin/ retrieved 2012-05-18)

[11] Handlebars (http://handlebarsjs.com/ retrieved 2012-05-18)

[12] Highcharts (http://www.highcharts.com/ retrieved 2012-05-18)

[13] DataTables (http://datatables.net/ retrieved 2012-05-18)

[14] Underscore (http://documentcloud.github.com/underscore/ retrieved 2012-05-18)

[15] J Andreeva et al 2012 Providing global WLCG transfer monitoring *Proc. of CHEP12 New York, USA* (http://dashb-wlcg-transfers.cern.ch/ui/ retrieved 2012-05-18)

[16] ATLAS Task Analysis (https://dashb-atlas-prodsys-prototype.cern.ch/templates/task-analysis/ retrieved 2012-05-18)

[17] CMS Interactive View (http://dashb-cms-job.cern.ch/dashboard/templates/web-job2/ retrieved 2012-05-18)

[18] ATLAS Dataset Distribution (http://dashb-atlas-task.cern.ch/templates/pandadatasetdist/ retrievd 2012-05-18)

[19] J Andreeva et al 2012 Experiment Dashboard - a generic, scalable solution for monitoring of the LHC computing activities, distributed sites and services *Proc. of CHEP12 New York, USA* (http://dashb-ssb.cern.ch/, http://dashb-atlas-ssb.cern.ch/ retrieved 2012-05-18)

[20] Evans L and Bryant P 2008 LHC machine *J. Instr.* 3:S08001 doi:10.1088/1748-0221/3/08/S08001

[21] Backbone (http://documentcloud.github.com/backbone/ retrieved 2012-05-18)