SNS ONLINE DISPLAY TECHNOLOGIES FOR EPICS*

K.U. Kasemir, X. Chen, J. Purcell, E. Danilova, ORNL, Oak Ridge, TN 37831, U.S.A.

Abstract

The ubiquitousness of web clients from personal computers to cell phones results in a growing demand for web-based access to control system data. At the Oak Ridge National Laboratory Spallation Neutron Source (SNS) we have investigated different technical approaches to provide read access to data in the Experimental Physics and Industrial Control System (EPICS) for a wide variety of web client devices. We compare them in terms of requirements, performance and ease of maintenance.

INTRODUCTION

Traditional control system operator interface tools were often standalone applications for a specific operating system [1]. Recent developments emphasized operating system portability, allowing users in their offices or even at home to run the same applications that are used in the control room [2]. These new tools are highly integrated, offering a workflow that previously required switching between multiple standalone programs [3]. While there is a continuing need for such feature-rich applications, there is at the same time a desire for web-based control system access. This online view of the control system might be read-only, at a limited update rate and restricted to a subset of the machine, but it should be accessible from any web browser, be it a desktop computer, laptop, netbook or cell phone, without need to install any additional software on the client device.

CONTROL SYSTEM DATA

The following examples refer to EPICS, but the fundamental ideas apply to any control system. The control system allows network access to data points called Process Variables (PVs). Certain PVs might change at a known rate, but the most common case is data that changes at arbitrary times. "Set point" PVs for example will stay constant until adjusted by an operator. PVs for temperature readings might vary slowly over time, while other machine parameters can change rapidly.

Polling vs. Event-Driven

A control system display tool should be event-driven, reflecting changes as they occur. If the display is polling the PVs for updates, resources are wasted polling data points that stay constant while at the same time missing updates that are faster than the poll rate.

BASIC WEB TECHNOLOGY

The Hypertext Transfer Protocol, HTTP [4], is the basis of all web technology. It typically functions as follows:

1) Web client establishes a network connection to the web server.
2) Web client requests a web page by sending GET /some_web_page.html HTTP/1.1
3) Web server returns the web page content.
4) Server and client close the network connection.

The request as well as the response can include additional parameters. Server and client may keep the network connection open for follow-up requests, but the protocol remains a request-response pattern.

There is no method in HTTP that is implemented by all web clients to support a "push" of updates from the server to the client. Every data transfer needs to start with a request sent by the client. This makes HTTP fundamentally unsuitable for event-driven updates as desired for control system displays! Users would have to manually initiate a request by pushing the "Reload" button on their web browser to update the display to the most recent data.

There are several commonly used approaches to work around this limitation in a way that is supported by the majority of web browsers.

Web Browser Plug-Ins

Most web browsers allow the installation of plug-ins to extend their functionality. The CAML project uses a WebCA plug-in to add native support for EPICS Channel Access to several web browsers [5]. This solution offers the best possible performance because the web client directly receives updates from the control system.

The approach is, however, specific to certain web browsers. The direct network traffic between the client and the control system is also often blocked by firewalls that only allow access to the standard web server port TCP 80, or generally restrict access to the control system to computers on the plant network.

Consequently, this approach cannot offer generic web access to control system displays from a wide range of devices, including cell phones.

Ajax

Ajax [6] uses JavaScript code running inside the web browser to create XMLHttpRequest (XHR) objects [7] that in turn perform requests to a web server. The JavaScript code then handles the response, typically by updating the web page with the received data. Ajax still uses the HTTP request-response pattern, but it no longer requires the user to *manually* initiate requests because the JavaScript code inside the web browser itself can initiate the requests.

Ajax can be used to implement a polling web client that periodically requests updates from the web server.

* SNS is managed by UT-Battelle, LLC, under contract DE-AC05-00OR22725 for the U.S. Department of Energy

Long Poll

A "Long Poll" [8] uses a polling Ajax web client as just described with the additional agreement between server and client that the server can delay its response until updates are available:

1) JavaScript in client starts XHR to request.
2) Web server waits until updates become available.
3) Once there are updates, web server returns the data.
4) Web client displays the data and immediately starts another XHR, i.e. repeat step 1.

Technically, this is still a polling request-response mechanism. In a true event mechanism the client would not have to re-issue any request. The server would simply send updates via a once established network connection.

Performance is lower than a true event mechanism because of the additional network traffic for sending the requests. The client or server might even close their network connection at the end of each transaction, requiring a new connection for each long poll.

To the end user, there is hardly a difference: The display can be updated as soon as the server sends new data.

SNS STATUS WEB SITE

The SNS Status web site is a Java Server Page (JSP) project running on a Tomcat web application server [9].

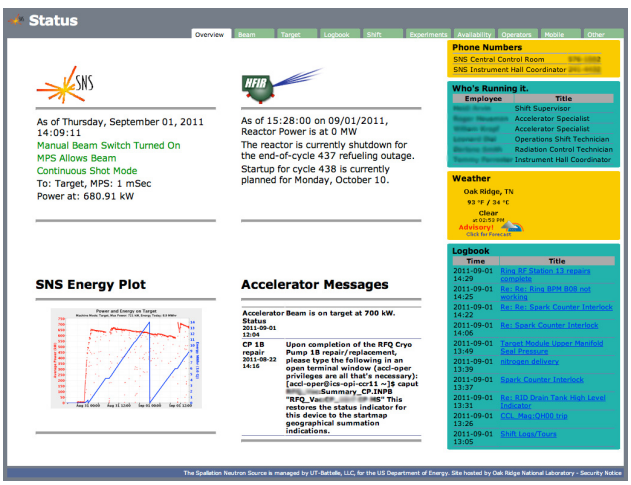

Figure 1: SNS Status Web Page.

It displays a fixed set of pages, arranged in "tabs", to summarize the recent history of SNS operation, including beam power, state of beam lines, availability statistics, recent logbook messages, and shift summaries.

Certain sections of these web pages are periodically updated via Ajax. The beam power in the upper left section of Fig. 1 for example is refreshed every 3 seconds, while other sections of the page are updated at intervals of 30 seconds or even minutes.

Java code on the web server, i.e. Tomcat, subscribes to a predefined, fixed list of PVs and maintains a table of their current values. Ajax requests from web clients invoke Tomcat servlets, which then read the current PV data and format it to for example generate the beam status information shown in Fig. 1.

The SNS Status Web site has been online since July 2010. It has been very stable. It can be accessed by any web client, and requires only a few CPU and memory resources. Sections of this web site are displayed on hallway monitors throughout the SNS facility. The beam history information from this web site is a key component of daily status meetings.

The majority of its content, however, is based on data read from a relational database, not online PVs. Only about 50 PVs are monitored by the Tomcat application, updating sections of the web page at comparably slow periodic rates as mentioned.

The web site offers a common denominator of data that is of interest to many users. The overall content and its layout are fixed. End users cannot customize it to meet their specific, individual needs.

SNS DASHBOARD

The SNS Dashboard was created to provide more flexibility. Users can choose from a list of widgets, some of which are customizable. Widgets can be opened and closed; they can be "dragged" to the desired position on the web page.

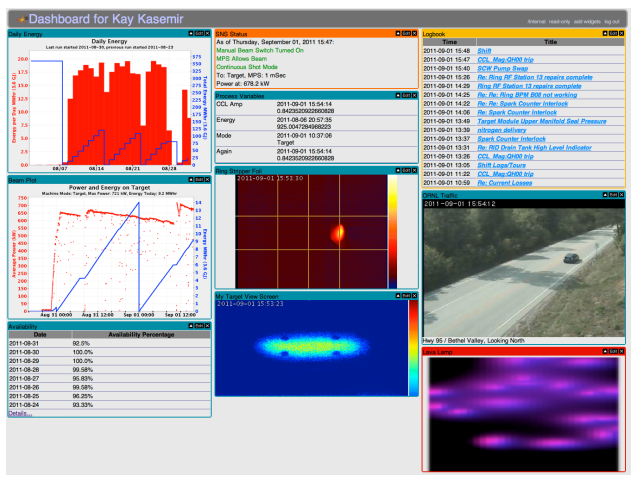

Figure 2: SNS Dashboard.

At this time there are about 20 widgets. Some duplicate information that is already offered on the SNS Status web site, but users can now consolidate the information of interest on their personal dashboard. There are also widgets for a user base that was too small to warrant inclusion in the SNS Status web site. Examples include a widget that lists the devices affected by scheduled power outages, or a widget with indicators for the status of certain groups of front-end computers.

Widgets that rely on PV data utilize a Long Poll, updating as events are received from the control system. The PV pool is not fixed as in the SNS Status web site. Some widgets allow the user to configure which PVs to display. The web client will invoke a subscription servlet

for these PVs. The web server subscribes to the control system PVs and returns updates as they arrive from the control system in the next Long Poll.

In our initial implementation, each widget that depends on PV updates would issue a Long Poll, but we learned that most web browsers limit the number of parallel XHR requests to 6 or even 2. To adjust for this limitation, our dashboard software performs only one Long Poll for the whole dashboard page of a user. The data returned by the web server contains tags, which are used by the client to dispatch the data to the widgets that requested it.

The achievable Long Poll update rate varies. 10 Hz are often possible, but we throttle it to 1 Hz because it was considered fast enough for a generic online display. The web server tracks the PV subscriptions of each web client. If a PV changed, it is marked for update. Adding a 1 second delay to the Long Poll means that the web server can often return the accumulated updates for multiple PVs within one Long Poll, reducing the network traffic.

The Dashboard software is more complex than the SNS Status web page. The dashboard configuration for each user is persisted in a relational database. It is read by the web server when a user logs on, then duplicated in the web client. This duplication in the web client allows the user to efficiently edit the configuration inside the web browser. When the user relocates a widget inside the web browser or adds a PV name to a widget, the web client does not need to communicate each mouse click or key press to the web server. The web client only sends the end result to the web server, for example the new widget location or the added PV name, thereby minimizing the network traffic and optimizing the responsiveness of the web page.

Keeping the dashboard data consistent between the web server and client requires attention to detail. To complicate the implementation, this has to be done in two very different programming environments: Java on the server, but JavaScript, a Document Object Model (DOM) and Style Sheets in the web client. The JQuery scripting library [10] simplified the client-side implementation, especially the Ajax handling for different web browsers.

The Dashboard was published in June 2011 and is gaining acceptance. Users can create multiple setups, for example one for the desktop and one for the cell phone. We have been adding new widgets based on user requests. At this time, however, users can not create their own widgets or contribute them to the list of available widgets.

CSS WEB OPI

Most of our recent developments for control room and office computer user interfaces were based on the Eclipse Rich Client Platform (RCP) [2, 3]. RCP is a powerful Java framework that allows the implementation of applications, which then run on different operating systems like Microsoft Windows, Apple Mac OS X and Linux. The RCP Standard Widget Toolkit (SWT) library implements the actual user interface components like windows, buttons and other graphical elements that the end user sees on her computer screen.

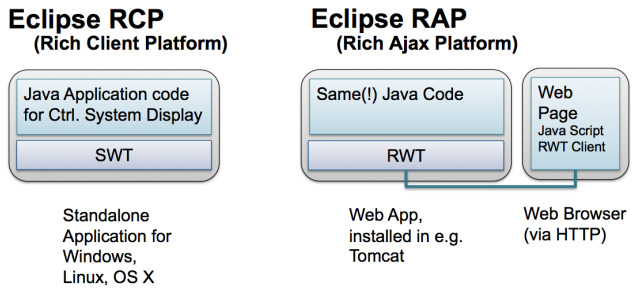

Figure 3: Comparison of RCP and RAP.

The Eclipse Rich Ajax Platform (RAP) project offers an RWT library that replaces SWT [11]. Ideally, the same Java application code that was originally developed for SWT will also run with RWT, but instead of creating a display on the local computer screen, as was the case for RCP, the RAP application is acting as a web server. Web clients that connect to this web server will see a display in their web browser that mimics the original SWT display.

The RAP application is typically installed in a web container like Tomcat. The RWT library creates HTML and Java Script code for the web client, using a Long Poll mechanism to send updates to the web client. An important point, however, is that the application developer can almost completely ignore the details of Tomcat, HTML, JavaScript, Ajax, Long Poll, because this is handled by RAP.

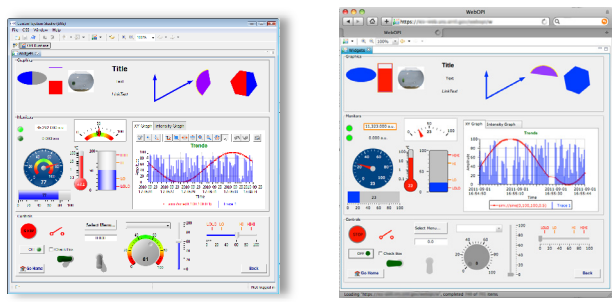

Figure 4: CSS 'BOY' display (left) and corresponding Web OPI representation.

The CSS operator interface BOY [12] was modified to execute not only with RCP but also RAP. As shown in Fig. 4, existing display files that were created with BOY can now also be executed in a web browser, providing an almost identical look and behavior. Users can develop any type of BOY display and then not only use that in the control room or their office computer but also at home and on portable device, including smart phones.

To port the existing RCP version of BOY to RAP, code for reading display files, connecting to EPICS network channels and handling control system events remained the same. This code is now executed within the web server. Its performance is naturally identical to the same code running in the RCP version.

Calls into SWT, which originally updated a physical display, now go into RWT, which in turn generates JavaScript that is queued for delivery to the web client via Long Polls. The JavaScript generated by RWT causes the web client to update the display inside the web browser.

The direct display update as performed by RCP is naturally more efficient than the RAP implementation necessitated by the shortcomings of HTTP. On a computer where an RCP display for 100 text widgets updating at 10 Hz uses about 12% of the CPU and 50MB of RAM, the combination of RAP under Tomcat with Firefox as a web browser used about 40% CPU and 80MB of memory.

When increasing the number of text widgets to 1000, the RCP display would continue to update at 10Hz, while the RAP version would degrade to 3Hz.

In an operational setup, the CPU load is of course spread across different computers. The web server can be a high-performing server machine, while the clients can be an office PC as well as a much simpler smart phone. The server handles most of the logic except for the actual display. As long as there are not too many events from the control system that require display updates, small devices like smart phones can display operator screens of considerable complexity.

The RAP design aims to allow "single sourcing". The same source code should seamlessly compile for both RCP and RAP. The main difference between the two is the fact that RCP code targets a single display, while RAP needs to handle multiple (virtual) displays. Since the SWT programming interface associates fonts, colors and other user interface resources with a display, the BOY code needed to be extended to track such resources for multiple (virtual) displays.

The network traffic between web server and client adds a certain delay to RAP that is not present in an RCP application, including the extreme case that the web client can be closed without directly notifying the web server. RAP can only detect this via time-outs.

While the BOY RCP code and the Web OPI RAP code are currently not fully single-sourced, the majority of the source code is shared by both approaches. Most important, the Web OPI offers all the features of the RCP BOY display without having to perform any coding on the HTML or JavaScript level.

We presented the web OPI to SNS users in August 2011. Its current implementation relies on certain pre-release snapshots of RAP code because RAP is still maturing technology, but our initial experience has been very good.

SUMMARY

The core web technology, HTTP, is less than ideal for online control system displays. Appropriate use of Ajax, especially the Long Poll paradigm, can alleviate fundamental HTTP limitations.

The SNS Status web uses basic Ajax technology to generate generic displays for a wide audience. The Dashboard uses Long Poll and more client-side JavaScript to offer more customization and faster updates for users that need specialized displays. The Web OPI uses RAP for web access to any BOY display, offering utmost flexibility because users can create their own BOY displays in CSS.

These three approaches complement each other. Users can access generic status displays with zero effort, invest time in creating their fully customized displays for the Web OPI, or use the Dashboard as an intermediate solution.

The Web OPI explores the limits of HTTP-based control system displays. A true event mechanism that can transfer binary data could encode the display updates more efficiently. In the future, WebSockets [13] might be available in all web clients, allowing for more efficient control system displays.

REFERENCES

[1] "MEDM", "StripTool", "ALH" and other network client tools for EPICS, http://aps.anl.gov/epics/extensions
[2] K. Kasemir, "New User Interface Capabilities for Control Systems", PAC 2009, Vancouver, Canada
[3] K. Kasemir, "The Best Ever Alarm System Toolkit", ICALEPCS 2009, Kobe, Japan
[4] HTTP - Hypertext_Transfer_Protocol Overview, http://www.w3.org/Protocols/
[5] Th. Pelaia, "CAML and Web CA Status", EPICS Collaboration Meeting, Legnaro, Italy, 2008
[6] Ajax (programming), August 2011, http://en.wikipedia.org/wiki/Ajax_(programming)
[7] XMLHttpRequest Specification Candidate Recommendation 3, August 2010, http://www.w3.org/TR/XMLHttpRequest
[8] Comet (programming), August 2011, http://en.wikipedia.org/wiki/Comet_(programming)
[9] Tomcat web application server, http://tomcat.apache.org
[10] JQuery Scripting Library, August 2010, http://jquery.com
[11] Eclipse Rich Ajax Platform, August 2010, http://www.eclipse.org/rap
[12] X. Chen, K. Kasemir, "BOY, a Modern Graphical Operator Interface Editor and Runtime". PAC 2011, New York, NY
[13] The WebSocket API, Draft 25 August 2011, http://dev.w3.org/html5/websockets