MASS-ACCESSIBLE CONTROLS DATA FOR WEB CONSUMERS

B. Copy, R. Niesler, F. Tilaro, M. Labrenz, CERN, Geneva, Switzerland

Abstract

The past few years in computing have seen the emergence of smart mobile devices, sporting multi-core embedded processors, powerful graphical processing units, and pervasive high-speed wireless network connections (supported by WIFI or EDGE/UMTS). The relatively limited capacity of these devices, when compared to desktop computers, requires relying on dedicated embedded operating systems (such as Android, or iOS), while their diverse form factors (from mobile phone screens to large tablet screens) require the adoption of programming techniques and technologies that are both resource-efficient and standards-based for better platform independence. We will consider the available options for hybrid desktop/mobile web development today, from native software development kits (Android, iOS) to platform-independent solutions (mobile Google Web toolkit, JQuery mobile, Apache Cordova, Opensocial). Through the authors' successive attempts at implementing a range of solutions for LHC-related data broadcasting, from data acquisition systems and LHC middleware such as DIP and CMW, on to the World Wide Web, we will investigate what are the valid choices to make and what pitfalls to avoid in today's web development landscape.

CONTROLS DATA ON THE INTERNET

Modern accelerator complexes such as CERN have been relying for many decades on industrial control systems. These off-the-shelf industrial components such as Programmable Logic Controllers (PLCs) have proven to be a cost-efficient and standard way to implement critical processes such as CERN's LHC Cryogenics, the Quench Protection System or CERN Experiments gas control systems (ATLAS, ALICE, CMS, LHCb, Cloud).

The dissemination of data issued from such business-critical systems is to this date certainly problematic : While certain critical infrastructure owners have no qualms about exposing their own equipment directly in the Internet, without any form of protection from even the most trivial network-based malicious users [1], even state-of-the-art robust computer servers must nowadays be placed under layers of protections and procedures before they can be safely exposed on the Internet. One very typical problem is the lack of resilience of these critical industrial process control equipments when faced with network traffic patterns : while such patterns would be harmless for a typical network device such as a router, a desktop computer or even a printer, a PLC on the other hand may refuse more than 5 concurrent network connections, or decide to delay certain parts of process control in order to honour complicated network requests. Another important aspect is that, no matter how robust these critical equipments are, being able to disrupt their operation remotely through simple denial-of-service attacks, is simply unacceptable for operation-minded assets such as CERN's LHC.

On the other hand of the spectrum, Web servers deployed in an era of social media and bandwidth-demanding mobile Internet access must be able to cope with sudden and significant bursts of interest, if only for limited periods of time. Cloud infrastructures such as the Amazon Elastic Computing cloud or Cloudstack can nowadays roll out tens or hundreds of virtual servers in a matter of minutes, exposing high throughput NoSQL in-memory databases and scalable web services to satisfy millions of individual visitors.

We will now expose emerging solutions that have been employed at CERN to bridge the gap between critical industrial control systems employed at CERN and massive numbers of world wide web originating visitors, without any compromise in the matter of operational availability and critical process integrity. We will first examine what makes today's industrial controls data unsuitable for mass-distribution and how to address this shortcoming. We will then consider what makes web-based data relevant in today's cloud and mobile IT market. We will conclude by inspecting in detail an implementation of these solutions, as applied to the CERN LHC infrastructure as a whole.

DEVELOPING FOR MASS-ACCESS

A critical process controller equipment such as a PLC is designed to operate in a sheltered environment : moderate and predictable network traffic to and from the equipment, a small number of concurrent network-based accesses (less than a dozen), stable processing cycle times in order to perform control with near real-time precision are all essential parameters to ensure a reliable behaviour from the controller's part.

Many such controllers are in operation inside the LHC complex. Monitoring their activity and gathering the data they expose requires the usage of industrial protocols, from OLE for Process Control (OPC) to the CERN Common Middleware (CMW) protocol, through STEP7 for SIEMENS PLCs or Modbus for Schneider equipment. Such protocols have also been designed for sheltered environments and peer-to-peer connections. Connecting more than a dozen consumers to these data sources is sometimes simply impossible, and most of the time too risky, without opening the door to operational mistakes and even malicious usages.

CERN for instance has made the choice of deploying a secure Technical Network, which acts as a sheltered production environment. Inside this CERN Technical Network, trusted sets of equipments have been identified to regroup network-capable devices that can speak to each

ISBN 978-3-95450-139-7

other. Such trusted sets are today limited to 10 members, and are essential in ensuring that stray network traffic (*e.g.* as observed at CERN : accidental multicasts, corrupt network traffic...) and operational mistakes (such as using the wrong IP address to query an equipment) do not disrupt the operation of the LHC.

Given this context, it is unthinkable to distribute process control data to hundreds or thousands of web-based viewers without an intermediate layer that exposes the following capabilities :

- **Requirement 1 :** Memory and network-efficient data distribution with a push-style mechanism, so as to cope with demand in a cost-efficient way.
- **Requirement 2 :** Scalability, replication and caching capabilities, so as to minimize the load placed upon critical process control devices.
- **Requirement 3 :** Support for efficient data format that are native to web consumers, so as to bring the amount of client-based data processing to a minimum.
- **Requirement 4 :** Adaptation to a large number of different data sources, for instance, support for multiple proprietary industrial protocols.

Such requirements are certainly not limited to the data distribution of industrial control systems: They have been under scrutiny for the last few years and have given rise to new web standards and implementation, starting in 2009, which have now matured into production quality platforms.

Requirements 1 and 2 for instance can be nowadays addressed with **Websocket**, a W3C standard web protocol [2] that replaces HTTP Polling and provides server-push capabilities. HTTP polling implies that a web client should out of its own accord regularly ask the web server for potentially new information. This in turn causes a significant waste of concurrent TCP connections (of which any network equipment only has a finite supply of), and has two catastrophic effects :

- In cases where the server has little to say, in small irregular bursts, a lot of empty meaningless traffic is generated (of which an English translation would be "Do you have something new for me ? No I don't. Okay, thank you.").
- In cases where the server has a lot to say in large regular amounts, the traffic is bunched into large information sets that collate, for instance, the entirety of the last five seconds of server activity - forcing the client to cope with it in one single go, instead of small increments.

Websockets [2] on the other hand offer web servers a way to contact web clients on demand, in an HTTP-friendly way (without forcing web server administrators to open holes in their firewall for instance). Only when the server has something to say does it need to open a connection to the client. When it does, it can efficiently replicate the information to all the web clients on its list of interested parties, allowing a memory and I/O efficient implementation.

Websocket, along with alternatives such as Server-sent Events (SSE), are protocols that have been gradually benefited from adoption since 2009, up to the point where 70.52 % of World Wide Web users (including mobile devices) can to date benefit from this technology out of the box [3].

For web clients that do not support websockets, it is of course possible to fall back on less efficient, but still tolerable approaches, such as HTTP polling and long-lived connections.

The **Atmosphere Framework** is a technology that implements memory-efficient, websocket-compatible data broadcast mechanisms. Atmosphere is being standardized and integrated as part of the Java Enterprise technology stack (JSR-356 [4]).

Requirement 3 is typically addressed with a combination of two new de-facto web standards, **Representational State Transfer** (REST) and **Javascript Object Notation** (JSON).

REST is a simple and intuitive way to map HTTP request onto business logic, so that web clients can access data in a familiar manner and without the use for any application layer (for instance, accessing the HTTP URL *http://myserver.com/customer/1*, would return the data associated with Customer ID 1).

Javascript Object Notation is a way to represent object-oriented data which any web client can also interpret out of the box in a computationally efficient way. Both REST and JSON have become part of the Java Enterprise technology stack and have spread (JSON is supported by 93.54% of World Wide Web users [3], while any HTTP-capable client supports REST from the ground up).

Requirement 4 is an implementation constraint that implies that capturing industrial control data and forwarding it to web clients must be done in a simple, platform-independent way. It is not today addressed by any standards, as even OPC-UA does not explicitly target web clients. A custom solution is therefore required to bridge this gap. Our prototype implementation provides a simple agent process that subscribes on demand to native data (OPC DA, DIP or any other control system specific protocol using the corresponding native libraries) and forwards the data through HTTP to the broadcasting web server. It is of course undesirable in most cases to collect raw data from a control equipment, but rather recommended to collect it from a SCADA data source, where data will have been refined, validated and pre-processed for widescale distrbution.

Figure 1 below provides a global summary of the broadcasting architecture, whereby multiple industrial control data sources can be exposed to web clients in a secure, efficient and scalable manner.

ISBN 978-3-95450-139-7

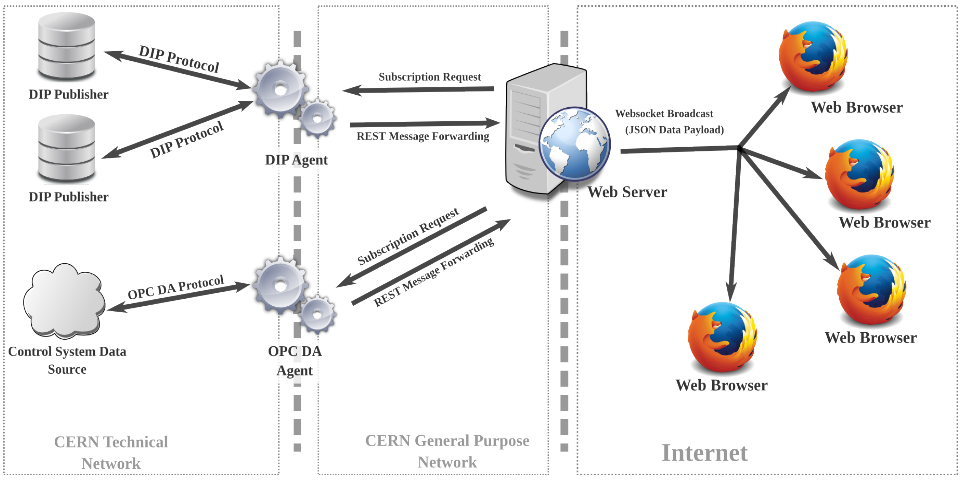


Figure 1: Broadcast framework architecture.

DEVELOPING FOR MOBILE DEVICES

Gartner, the influential IT market analysis firm, predicted in 2013 that mobile devices would overtake desktop-based ones as a medium to access the World-wide Web. And that by 2015, Internet tablet sales will overtake those of laptop computers. Mobile devices such as smartphones and tablets typically ship with powerful low-power CPU chips (some of them with quad cores) and even more capable graphic processing units (GPUs).

While each device brand will favour one dedicated mobile operating system (Apple's iOS, Google's Android or Microsoft's Windows 8 Mobile, to name but a few major players on a very fast-growing market), they all tend to support platform-independent web standards such as HTML5, Websockets and CSS3, which in turn provide potential for highly-interactive, visually appealing applications. Such Web standards-based applications can even store data locally, work offline and access advanced native device features such as GPS-based localization, access to personal data such as telephone contact lists or calendaring functions, access persistent local storage and file systems *etc...*

Despite advanced computational capabilities, certain resources remain scarce on such platforms : network and on-board memory usage are for instance severely limited. When it comes to network usage, developing a user-friendly mobile application implies working with small and swiftly delivered chunks of information. Even application logic and assets must fit within acceptable ranges for the applications to load within an acceptable amount of time (past 10 seconds of inactivity, an application is typically stopped by its user, on suspicion of it having stopped functioning).

And since mobile devices ship with limited memory and on-board persistent storage, developers cannot rely on a long initial loading-time that would download in one go all of the application's resources locally for later usage - past a certain size, these resources will not fit in memory or local storage and will be evicted from the device's web cache.

Distributing industrial controls data to mobile devices must therefore be done in manageable chunks, and strictly on-demand. Interactions with the web server must be numerous and fast, yet reduced to the utmost minimum.

While a 200 kilobytes-long web application may seem negligible in size to a traditional desktop-based web developer, a mobile web user accessing it over an EDGE data connection will experience severe delays before the application starts up.

Various mobile user interface development frameworks are available today to target mobile devices, such as mGWT (Mobile Google Web toolkit), Sencha Touch and JQuery Mobile. In the course of our development, we have eventually eliminated any framework that was not primarily written for mobile and low-end computing platforms.

- Both mGWT and Sencha Touch for instance are respectively desktop and internet tablet oriented UI frameworks, relying on the developer to ensure that their final application will be of a reasonable compiled size and make acceptable usage of network resources.
- JQuery mobile on the other hand focuses on lean mobile-phone capable applications, delegating all the rendering to the embedded GPU and the native HTML engine. Of course, should more ample resources be available, in the case of a desktop computer for instance, JQuery mobile transparently expands to make the most of a large screen, a sizeable browser cache or powerful graphical capabilities.

Finally, another widely-employed strategy to circumvent mobile device storage and network bandwidth limitations is the preparation and distribution of an “app” (a term familiar to iPhone and Android smartphone users). Presented to the end-user as a native application, it is simple, thanks to frameworks such as Apache Cordova or Adobe Phonegap, to pre-package web resources and install them locally, thereby granting them enhanced security privileges, local storage access rights and much faster application startup times.

ISBN 978-3-95450-139-7

DISTRIBUTING LHC DATA

Distributing data in provenance of the LHC to a wide-audience has been the goal of many. LHC status information is already widely available, from official sources such as the LHC PAGE 1 and other so-called VISTAR displays presented on television screens scattered across the various CERN sites, onto hobbyist attempts such as LHC PORTAL, an individual's initiative to regroup all CERN-sourced visual displays into a single location.

Two Case Studies at CERN

In 2009, the CERN EN department introduced a new way of visualizing LHC data in a graphical format called the "LHC Dashboard". Inspired by business intelligence and dashboard concepts that allow to synthesize a large amount of information into line charts, coloured maps or even animations. The LHC Dashboard regenerates every 15 seconds all of its data visualizations in the form of graphical files (such as PNG or JPG images). Web clients then download these images at regular intervals, to see a slowly updated evolution of the LHC status.

In parallel, on March 2010 for the official relaunch of the LHC, a web application called "LHC Status Display", implemented through a partnership between the CERN Press Office, a third-party Swiss consulting company (TechCare) and the EN department, presented an animated, live data coming from LHC operation. This data was collected via the DIP middleware protocol, transformed to XML then forwarded in 50 kilobytes chunks (roughly 20 seconds of highly summarized operational data) to an array of CERN Web Servers. An "Adobe Flash" application would then replay the data in graphical form, through animations of the LHC Tunnel and simulated fixed target displays, giving web visitors an animated feed (albeit 20-second delayed with regards to the live operational data) onto the LHC's current status, with respect to beam energy, luminosity, number of collisions observed by various LHC experiments *etc*...

This application met serious difficulties on the day of the launch : The array of web servers deployed by the CERN IT department, four production machines, proved unable to serve enough requests to cope with the demand. A mere 20 000 requests per minute were enough to render half the array inefficient due to the constant polling requests coming from web visitors.

Such experiences proved that polling-based HTTP data serving is inadequate for large numbers of visitors and that data formats must be tailored to the client platform, so as to maximize data bandwidth usage.

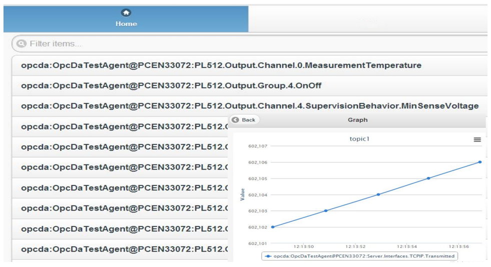
The two previously mentioned project could have greatly benefited from the mass-access enabling technologies identified earlier.

Efficient Data Distribution for the "World Wide Web"

Initiated in the spring 2012, a new version of the LHC Dashboard aims at identifying the best modern approach that would allow CERN to distribute copious amounts of control data to a large audience. More generally, data access should be possible from the simplest web-enabled mobile devices, yet take advantage to the fullest of their advanced capabilities for graphical data rendering.

This approach relies on an all-purpose Websocket-enabled data distribution platform called "Broadcast". On one hand, data collection agents are placed in specific locations on the CERN Technical Network. They forward the information to "broadcast topics", to which web clients can subscribe to receive raw JSON data using the most appropriate connection protocol at their disposal (websockets, server-sent events, HTTP 1.1 long polling or legacy HTTP polling).

Figure 2 shows screen captures of a lightweight JQuery mobile interface allowing live access to control systems data and interactive chart.



Filter items...

opcda:OpcDaTestAgent@PCEN33072:PL512.Output.Channel.0.MeasurementTemperature
opcda:OpcDaTestAgent@PCEN33072:PL512.Output.Group.4.OnOff
opcda:OpcDaTestAgent@PCEN33072:PL512.Output.Channel.4.SupervisionBehavior.MinSenseVoltage
opcda:OpcDaTestAgent@PCEN33072:PL512.
opcda:OpcDaTestAgent@PCEN33072:PL512.
opcda:OpcDaTestAgent@PCEN33072:PL512.
opcda:OpcDaTestAgent@PCEN33072:PL512.
opcda:OpcDaTestAgent@PCEN33072:PL512.
opcda:OpcDaTestAgent@PCEN33072:PL512.
opcda:OpcDaTestAgent@PCEN33072:PL512.

opcda:OpcDaTestAgent@PCEN33072:Server.Interfaces.TCPIP.Transmitted

Figure 2: JQuery mobile prototype UI.

To gracefully scale to a large number of visitors, the "Broadcast" web servers can be backed by a cloud database such as REDIS [5], which provides fail-over, automated scaling and clustering, data caching and disaster recovery. Frameworks like **Atmosphere** can readily leverage REDIS as a robust stateful session storage solely through configuration : Should any of the web servers fail, they can immediately be replaced by another instance, as any session-dependent data is safely kept in REDIS instead of the memory of the web server.

Data collection agents have to this day been written for OPC DA and DIP protocols as proofs of concept. The simplicity of the REST protocol and JSON data-encoding makes it trivial to provide more data collection agents.

REFERENCES

[1] R. O'Harrow Jr, "Cyber search engine Shodan exposes industrial control systems to new risks", The Washington Post, 06 June 2012; http://articles.washingtonpost.com/2012-06-03/news/35459595_1_computer-systems-desktop-computers-search-engine

[2] I. Hickson, "The WebSocket API", W3C Consortium, 20 September 2012, http://www.w3.org/TR/2012/CR-websockets-20120920/

[3] A. Deveria, "Can I Use... Websockets ?" , Consulted 25 July 2013 ; http://caniuse.com/websockets

[4] D. Coward et al, "Java API for WebSockets", Java Specification Request (JSR-356), 22 May 2013, http://jcp.org/en/jsr/detail?id=356

[5] S. Sanfilippo, P. Noordhuis "Redis performance benchmark", consulted on 20 August 2013 ; http://redis.io/topics/benchmarks

ISBN 978-3-95450-139-7