DEVELOPMENT AND IMPLEMENTATION OF EPICS CHANNEL ACCESS CLIENT WITH REAL-TIME WEB USING WEBSOCKET

Akito Uchiyama #,A), Kazuro FurukawaB), Yoshihide Higurashi c)
A) The Graduate University for Advanced Studies, School of High Energy Accelerator Science (Sokendai)
1-1 Oho, Tsukuba, Ibaraki, 305-0801
B) High Energy Accelerator Research Organization (KEK)
1-1 Oho, Tsukuba, Ibaraki, 305-0801
c) RIKEN Nishina Center
2-1 Hirosawa, Wako, Saitama, 351-0198

Abstract

In the large experimental facilities such as KEKB, RIBF, and J-PARC, the accelerators are operated by the remote control system based on EPICS (Experimental Physics and Industrial Control System). One of the advantages in EPICS-based system is the software reusability. Because it is available to develop client system by using Channel Access (CA) protocol without protocols with hardware dependencies, even if the system consists of the various kind controllers. As next-generation OPI (Operator Interface) using CA, we develop a server for the WebSocket, which is a new protocol provided by Internet Engineering Task Force (IETF), with combination of Node.js and the modules. As a result, we are able to use Web-based client system not only in the central control room but also with various types of equipment for accelerator operation.

WebSocket を用いた EPICS Channel Access Client の開発と実装

1. はじめに

EPICS を用いた制御システムの長所の一つとして、各コントローラに用いられる通信プロトコルは EPICS IOC (Input/Output Controller) によって CA プロトコルとして統一して扱われる、という事が挙げられる。よって OPI をはじめとしたクライアントシステムは CA プロトコルに基づいて開発を行えば良い事になる。

OPI 開発には C/C++、Python や Java 等でコーディングする手法や、MEDM/EDM・CSS (Control System Studio) /BOY [1] といったディスプレイマネージャを用いてコーディング無しに GUI を構築していく手法を取ること事が一般的である。一方 SPring-8 では実験用ビームライン制御用に新しい Web の規格である WebSocket を用いた OPI の開発手法を考案し、プロトタイプの実装が行われた[2]。SPring-8 の制御システムは独自に開発した MADOCA (Message And Database Oriented Control Architecture) をベースに構築されており、EPICS のそれとは異なる。WebSocket を用いた Web アプリケーションとシステムの利点を重要視し、我々は EPICS CA に対応する WebSocket 用サーバの開発とクライアントの実装を行った。

2. Web を用いたアプリケーションの利点

現在まで様々な Web を用いた加速器制御システムの開発が報告されている。加速器制御に Web 技術を用いる事の長所は次の通りである。

Web を用いる事で限定されたローカルなネットワークだけでなく、WAN を利用してシステムを提供する事が技術的には可能になる。エンドユーザはインストールする事無しにシステムを利用できるので X Window System 環境や専用ソフトウェアをインストールする必要は無くなる。

現在の社会一般において Web 及びそのネットワークは、すでにインフラの扱いであり、PC だけでなく携帯電話、テレビ等、様々な機器にブラウザが実装され Web を利用する事が可能である。よって Web に対応さえすれば将来的にはプラットフォームに限定されず、PC だけでなく様々な機器が制御端末として利用する事が可能になるはずである。専用端末が必要無くなるという事は、同時にシステムリプレース時のコストを大幅に抑える事も可能になるという事にもなる。

また、Web 技術を使用した場合におけるシステム開発の簡便さも挙げられる。一般的に HTML と JavaScript での構築は C/C++や Java のそれと比べた場合、容易でかつ開発速度が速い。例えばクライアント開発において、本質的な機能ではない GUI の開発は多くのリソースが必要になる工程である。ヒューマンリソースが多くない加速器施設にとっては大きなメリットになり得る。

3. 従来の EPICS における Web システム

ここで、比較のため、Web 技術を利用したこれまでの EPICS 向けオペレータ インターフェース機構について、概要を見てみる。

3.1 CAML

#a-uchi@riken.jp

CAML (Channel Access Markup Language) は XML ベースの言語で動作仕様を記述する事でブラウザから CA 通信を経由した制御を可能とするフレームワークである [3] 。プラグインとして Firefox, Google Chrome, Safari に組み込む事でブラウザが CA で通信を行う。直接ブラウザが CA 接続を行うのでネットワーク上にあるゲートウェイやファイヤーウォールを越える事は難しい。よってローカルな環境で運用する事が必須であると思われる。

3.2 PHP EPICS Module

PHP で書かれた Web アプリケーションにサーバサイドから EPICS CA をさせる為のモジュールである [4] 。PHP は Web アプリケーションを実装できるサーバサイド言語であるが、動的な Web アプリケーションを開発する為にはクライアント側に Ajax (Asynchronous JavaScript XML) で DOM (Document Object Model) 操作を行う JavaScript が必要になる。なお、最近のシステム・ソフトウェアに対応させた新版の案内は無い。

3.3 WebOPI

WebOPI とは CSS/BOY で開発した GUI パネルを Web で提供させるアプリケーションである [5] 。サーブレットコンテナに Tomcat7 が必要で、バックエンドに Eclipse の RAP (Rich Ajax Platform)[6]を利用している。CSS/BOY で開発したアプリケーションがソースに手を加えずにコンパチブルにブラウザ上で動作するので非常に有用である。動的な動きには Ajax を用いており、通信トラフィックを減らす為に JSON (JavaScript Object Notation) でデータを一括送信している。しかし変化がないレコードに関してもデータとして送信するので、その点には無駄がある。我々のテストではイントラネット環境において~100 msec の周期でデータ取得できる事を確認した。しかし 500 msec 遅延がある環境にある(日本から SNS のサーバに接続) WebOPI のデモページ[7]では Web 上の値の update 間隔が長くなり性能が落ちる。

4. WebSocket

WebSocket とはサーバ・ブラウザ間で双方向通信を実現する為のプロトコルである [8] 。当初 HTML5 の仕様の一部であったが、独立した仕様になりドラフト版を経て、2011 年 12 月に ITEF より RFC6455 として策定された。プロトコルの概要は以下である。

1. ブラウザが WebSocket の接続をする為にハンドシェイクの要求をサーバに送る。
2. サーバは承認後ブラウザにハンドシェイクのレスポンスを返す。
3. ハンドシェイク確立後プロトコルが WebSocket に切り替わり、以後サーバ・ブラウザ間で双方向通信を行う。

今まで Web を利用した CA クライアントが実装されてはきたが、ある値のモニタや速い応答が必要な い制御に限られてきた。何故なら従来の HTML 技術ではサーバ側とクライアント側の双方向通信ができない為ネイティブなアプリケーションの様なリアルタイム応答性の確保が難しかったからである。ここで言う”リアルタイム”とは人間が判断する上でディレイが無い、もしくは極めて小さい事を意味する。WebSocket を用いればサーバ・クライアントでコネクションを維持したまま双方向に通信する事が可能になるので上記問題は解決する。実際に WebSocket を用い 100 msec 周期程度で変化する値であれば MEDM/EDM と遜色ない動作を有する事を確認した。また Comet/Ajax の様に定期的に polling する必要がないのでネットワークトラフィックやコネクション数を減らす事が可能になる。問題点は全てのブラウザが対応しているわけではない点で主要ブラウザでは Internet Explorer (IE) 9 が未対応である。しかし次期バージョンである IE10 では WebSocket 対応が予定されている[9]ため時間が解決してくれる問題とも言える。

5. 開発手法

5.1 サーバサイド

WebSocket サーバの開発には Node.js [10] とそのライブラリである Socket.IO [11] を利用した。Node.js は Google V8 JavaScript Engine [12] を元に開発されたサーバサイド JavaScript の一つである。シングルスレッドベースな非同期処理の仕組みを持つ事が特徴として挙げられる。本開発において非同期な WebScoket サーバを実装するには多くのリソースが必要になると思われたが、Socket.IO が上記問題を解決した。現在同じように WebSocket 機能を提供するライブラリは C 言語では libwebsocket[13]]等がある。開発された WebSocket サーバはイントラネットだけでなく、将来的な WAN からのアクセスを考慮して WebSocket に対する SSL と認証の機能をオプションで持たせている。開発に使用したバージョンはステーブル版である Node.js 0.6.18、Sockt.IO 0.9.6 を用いた。

5.2 Channel Access for Node.js

CA と通信を行う事が可能な WebSocket サーバを Node.js で実装する為には Node.js から CA プロトコルが呼べなければならない。したがって CA をインターフェースする Node.js のアドオン、NodeCA を開発した。Node.js は C++で開発した関数をモジュールとして呼ぶ事が可能なので EPICS base で提供されている CA library を NodeCA では利用している。Node.js から CA を扱う為に用意した主な関数には、基本的な機能である caGet, caPut, caMonitor がある。イベントドリブンな動作をする caMonitor は何も考えずに実装するとシングルスレッドで走る Node.js においてブロッキングの原因となってしまう。これを防ぐ為に caMonitor のイベント待ちの関数を別スレッドにする事で対応した。概略を図 1 に示す。現

在 NodeCA は Array (ca_array_get, ca_array_put) に対応できていないので今後実装する予定である。

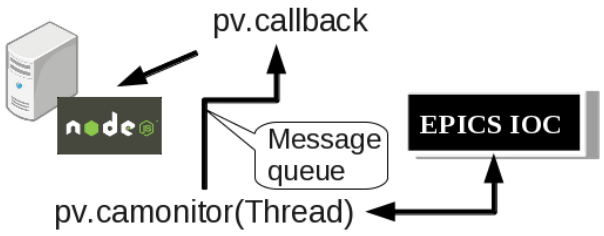

図１：NodeCA における caMonitor 関数の概略

5.3 クライアントサイド

テキストの単純な値の変更 (text update) は JavaScript で DOM 操作を記述する事で対応できる。その他、情報視覚化の為に jQuery[14]をベースとした JavaScript ライブラリである flot[15]と jsgauge[16]を利用した。上記以外にも JavaScript で利用できる情報視覚化のためのライブラリは有償・無償ともに選択肢は多く、通常であればコストがかかる工程を容易に行える事は大きなメリットである。

5.4 システム概要

全体のシステムチャートを図２、開発した NodeCA の概要を図３に示す。クライアントサイド JavaScript 内で Socket.IO のカスタムイベント(caGet, caPut, caMonitor, etc) を WebSocekt でサーバに投げる事で EPICS IOC からデータを取得する。本研究で開発された部分は灰色部分で表す。

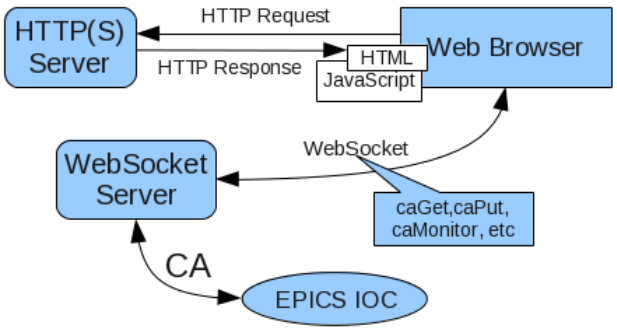

図２：全体のシステムチャート

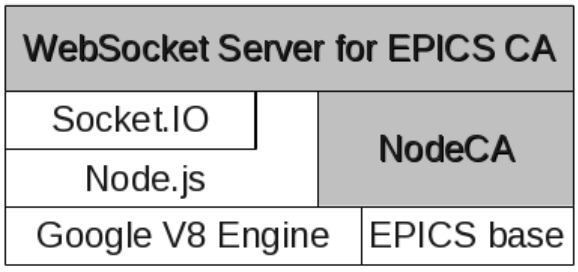

図３：開発した成果物の概要

6. 実装

ブラウザやシステムにおける対応状況を表１に示す。なお iPhone4S, Android 端末においても WebSocket 経由で CA から値を取得、表示できる事を確認した (図４)。iPhone を利用する場合、自ら開発したアプリケーションを Apple Store を通さずにインストールする事は簡単ではない。しかし Web アプリケーションであればその必要は無く、本成果物を利用する事で対応可能であると考えられる。

現在、理研 RIBF における 28GHz 超伝導 ECR イオン源[17]の制御でテスト的に実装を行っている。ガスバルブ・ロッド(試料)位置の駆動制御、 真空値のモニタに関して良好なレスポンスを確認しており、実運用に向けて開発が続けられている。

表１：2012/07 現在における本システムの対応を確認した主要ブラウザ状況

Windows/Linux	Google Chrome 20 Firefox 14 Opera 11
Android 4.0	Google Chrome 18 Firefox 14
iOS (iPhone)	Safari 5 Google Chrome 19 Mercury

図４：開発中の 28GECR イオン源制御用 GUI パネルを表示させている Android 端末(左)と iPhone 端末(右)

7. 考察

将来的には WAN からの加速器ネットワーク内への装置に対して WebSocket 制御を可能にさせる事で加速器運転支援システムという位置づけを目指して

いる。例えばビームオペレーションや実験中に、ある機器・装置に対して熟練者の極めてシビアなチューニングが必要になるケースがある。その時に担当者が海外出張や深夜で不在であれば、対応が遅れ最悪の場合実験自体円滑に進まないケースもある。以上を実現する為の考察を行った。

まず悪意ある外部からの侵入を防止する為には次の様に考える。ローカルなネットワーク内のサーバアクセスは VPN を用い、WebSocket については SSL と認証でセキュリティを確保する。また現場オペレータはどこから(どこのインターネットサービスプロバイダか) 誰がアクセスしてきたかを完全に把握しなければならない。よって加速器オペレータには WebSocket サーバのアクセスログを完全に公開する必要がある。

次にブラウザから WebSocket 経由での EPICS IOC への命令について考える。各機器の動作に関する命令である"caPut"については現場加速器オペレータが完全に把握する必要がある一方、単純な機器からの値の取得やモニタの為の命令である"caGet"、"caMonitor"についてはその必要はないと考える。何故ならビームチューニング中にもかかわらず、勝手に他の機器を制御されてしまうと、ビームへの影響がどのオペレーションによってなされたか現場加速器オペレータが混乱、判断不能に陥るからである。よって caPut に関しては現場オペレータと遠隔制御者の間で厳格なポリシーが必要である。ポリシーについて以下を提案する。

1. 遠隔制御者は電話もしくは E-mail で今から接続して遠隔制御を行いたい旨をオペレータに伝える。
2. 遠隔制御者は制御したい GUI の Web へ SSL で接続しユーザ ID、パスワードで認証を行う。認証後は WebSocket でモニタできる。
3. オペレータは遠隔制御者がどこから、どのユーザ ID で接続してきたかを確認する。
4. 遠隔制御者は WebSocket で caPut したい機器のレコードに対して現場オペレータへリクエストを送る。
5. オペレータは遠隔制御者からのリクエストに対して、現在それを制御してよいかどうかを判断し、可能であるならば許可をレスポンスとして返す。
6. 遠隔制御者は許可レスポンスが返ってきたらモニタだけでなく制御が可能になる。
7. オペレータが遠隔制御の中断を判断したら許可を取り消す。取り消すとオペレーションは行えなくなる。また、ある一定時間が過ぎるとタイムアウトで遠隔制御の許可は取り消される。

caPut 制御の可否における具体的な仕組みについては様々な手法が考えられるが、その内の一つは CA Gateway[18]の様なパケットフィルタリングを動的に行い対応させる手法がある。

8. まとめ

WebSocket で CA 接続を可能にさせる為のサーバを開発し、ブラウザからリアルタイムに加速器の情報を表示させる事が可能になった。Node.js から EPICS Channel Access を呼ぶ為のアドオン NodeCA を開発する事で実現した。現在、実運用に向けて最初のバージョンをリリース、テスト的に実装、制御を行っている。理研 28GHz SC-ECR イオン源の一部のコンポーネントについてネイティブなアプリケーションの様にモニタ、制御可能である事を確認した。将来の WAN からのアクセスを実現させる為のアクセスポリシーも検討中である。

9. 謝辞

本開発を進めるにあたり SPring-8 の古川行人氏には意見交換の機会と貴重な助言をいただきました。ここに感謝します。

10. 参考文献

[1] X. Chen, et al., Proceedings of 2011 Particle Accelerator Conference, New York, NY, USA, 2011, WEOBN3
[2] Y. Furukawa, et al., Proceedings of ICALPECS2011, Grenoble, France, 2011, WEMAU010
[3] T. Pelaia II, et al., Proceedings of ICALPECS2009, Kobe, Japan, 2009, FRA001
[4] A. Bertrand et al., Proceedings of 10th ICALEPCS, Geneva, Swiss, (2005), P3_087.
[5] K. U. Kasemir, et al., Proceedings of ICALPECS2011, Grenoble, France, 2011, THBHAUST01
[6] http://www.eclipse.org/rap/
[7] http://sourceforge.net/apps/trac/cs-studio/wiki/webopi
[8] I. Fette and A. Melnikov. The WebSocket Protocol, IETF HyBi Working Group. 2011.
[9] http://msdn.microsoft.com/en-us/library/ie/
[10] http://nodejs.org/
[11] http://socket.io/
[12] http://code.google.com/p/v8/
[13] http://git.warmcat.com/cgi-bin/cgit/libwebsockets/
[14] http://jquery.com/
[15] http://code.google.com/p/flot/
[16] http://code.google.com/p/jsgauge/
[17] Y. Higurashi et al., Rev. Sci Instrum. 83, 02A308 (2012)
[18] R. Lange, "CA Gateway Update", presented at the EPICS Collaboration Meeting at JLab, Newport News, Virginia, November 2002.