# RIBF 制御系におけるオペレータインターフェース共有のためのプライベートクラウド実装の試み

# AN ATTEMPT TO IMPLEMENT PRIVATE CLOUD FOR SHARING OF OPERATOR INTERFACES IN RIBF CONTROL SYSTEM

内山暁仁#, A), 込山美咲 A), 西村誠 B)
Akito Uchiyama #, A), Misaki Komiyama A), Makoto Nishimura B)
A) RIKEN Nishina Center
B) SHI Accelerator Service, Ltd.

*Abstract*

Generally, EPICS-based control system utilizes GUI (graphical user interface) as an operator interface (OPI). Nowadays, CSS (Control System Studio)/BOY, which is an Eclipse-based OPI framework that use JAVA, is provided by the EPICS collaboration project, and CSS/BOY is widely used for construction of OPIs for the accelerator control system. When CSS/BOY is implemented for the RIBF control system, it is necessary to consider the method for sharing OPI files, because CSS cannot be multiple instances on a single PC. Since there is a large number of OPI file on the distributed client PCs that are installed CSS, the management will be complicated. Therefore, in order to utilize CSS/BOY in RIBF control system, we attempted to implement private cloud environment based on ownCloud. As a result, we confirmed that ownCloud-based private cloud is a useful method for sharing the OPI files. On the other hand, for a newly generated problem by using the attempt system, we also discuss another method using Soft IOC (Input/Output Controller) with Docker virtualization technology.

## 1. はじめに

RIBF 制御システムは主に EPICS (Experimental Physics and Industrial Control System) を用いた分散制御システムで構築されている[1]。 加速器制御システムでは、一般的にオペレータが加速器制御デバイスに命令を送るためのオペレータインターフェース (OPI) に GUI (Graphical User Interface)を用いる。EPICS では絵を描く感覚で簡便に OPI の構築を可能にするツールキットが提供されており、その有用性から多くの加速器施設で採用されている。従来 RIBF 制御システムにおける OPI 構築では MEDM[2]/EDM[3]を利用してきた。一方運用手法は、OPI ファイルをサーバ上で構築、起動させて GUI のみを X Window System 経由でそれぞれのクライアント端末に表示させる、といった一元管理で実現していた (Fig.1 参照)。近年では MEDM/EDM に代わり、統合開発環境である Eclipse ベースの OPI 構築ツールである CSS (Control System Studio) /BOY の運用が EPICS ユーザ間で広まっている[4]。RIBF 制御システムに CSS/BOY を実装するにあたり、構築者とオペレータ間で OPI 用ファイルの共有方法の検討が必要になる。なぜなら CSS は MEDM/EDM の様にサーバで複数起動させることは基本的にできず、クライアント一台一台にインストールさせる事で運用を行う。ファイル数が少なければそれも大きな問題にならないが膨大な数になると、インストール作業、ユーザ管理、そしてファイルのバックアップを含めたシステム管理といった、OPI 構築端末とオペレータの実行端末が異なる事による弊害が問題となる。一方、伝統的なファイル共有の手法として UNIX 系 OS の場合 NFS、Windows 系 OS の場合では Microsoft ネットワーク共有サービスが用いられている。しかし複数の構築者とユーザ間でファイル一つ一つに対して、細かくアクセスコントロールする事は複雑になり難しいという問題がある。また CSS/BOY を用いて構築した OPI ファイルを変更無しにウェブブラウザから制御可能にさせる WebOPI[5]というソフトウェアが提供されており、Web の利便性から積極的に利用したいと考えている。しかしウェブブラウザさえあればどこからでもアクセスできてしまう反面、出力を伴う操作は安全上問題となりうる可能性がある。加えて CSS/BOY、WebOPI 間でのシステム連携をどのようにデザインすれば良いか、といった課題もあった。運用する上での上記問題点を解決するため、我々は ownCloud を用いたプライベートクラウド環境を構築し、実装を試みた。

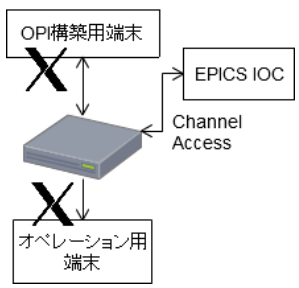


Figure 1: Implementation of OPIs using X Window for RIBF control system.

# a-uchi@riken.jp

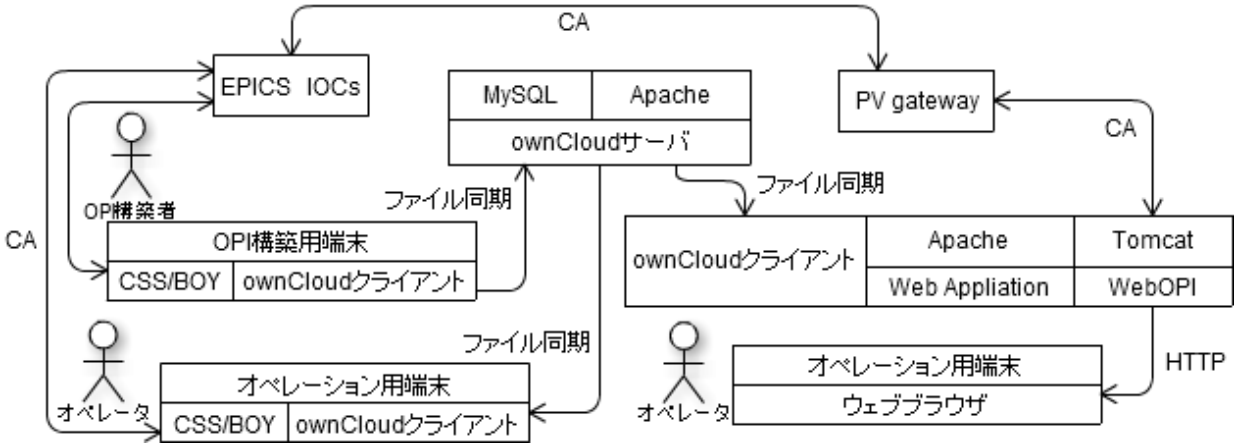


Figure 2: Outline of the attempted system.

## 2. ownCloud とは

ownCloud とはオンラインストレージを自前で構築することができるソフトウェアの事である[6]。オンラインストレージとは主に HTTP (hypertext transfer protocol) 経由のネットワークを使用して行うファイル共有サービスの事である。現在 Dropbox[7]や Google Drive[8]といったインターネットを経由したオンラインストレージサービスが一般的に多く利用されている。一方で RIBF 制御系のネットワークはインターネットに接続できないローカルな環境で構築されている為[9]それらを利用する事ができない。

ownCloud の特徴は、インターネットを経由せずともローカルネットワークのみでプライベートクラウド環境を構築可能、という事が挙げられる。ファイル同期させるクライアントは Microsoft Windows, Mac OS X, Linux だけでなく Android や iOS といったモバイル向けの OS でも利用する事が可能なのでシステム依存を防ぐ事ができる。また ownCloud は LDAP (Lightweight Directory Access Protocol) にも対応しており、ユーザとファイル毎に細かくアクセス制御を行う事も可能である。

## 3. システムデザイン

### 3.1 システム概要

システム全体の概要を Fig. 2 に示す。本システムは ownCloud サーバ、ownCloud クライアントと WebOPI 用サーバから成る。加速器オペレーション時は OPI 実行端末に同期された OPI ファイルを使用する。また構築した OPI は WebOPI 用サーバへも同期可能で、簡便な操作でウェブブラウザからもモニタ可能になる。

### 3.2 サーバサイドシステム

ownCloud v.6.0 の構築には PHP、リレーショナルデータベースと Web サーバが必要である。本システムでは OS に CentOS6.5 x86_64 を採用し、データベースは MySQL、Web サーバには Apache を選択した。これらのサーバは全て VMware 仮想サーバ上で構築された。

### 3.3 クライアントサイドシステム

クライアントサイドには CSS/BOY と ownCloud クライアントをインストールする必要がある。現在 RIBF 制御用端末に CentOS5.X を使用しているケースもあり、これは ownCloud で正式サポートされていない。よって、我々は CentOS5 用にクライアントをソースからビルドし、インストールを行った。

OPI 構築ユーザは CSS/BOY の Workspace ディレクトリを同期設定する。すると自動的に OPI ファイルが同期、ownCloud を構成している Web サーバにアップロードされる。その後 OPI 構築ユーザはウェブブラウザ上でどのファイルをどのユーザ、グループに同期させるかを設定する。操作作業中のスクリーンショットを Fig. 3 に示す。

オペレーション用端末では ownCloud クライアントをインストール時に予め用意されていたユーザを

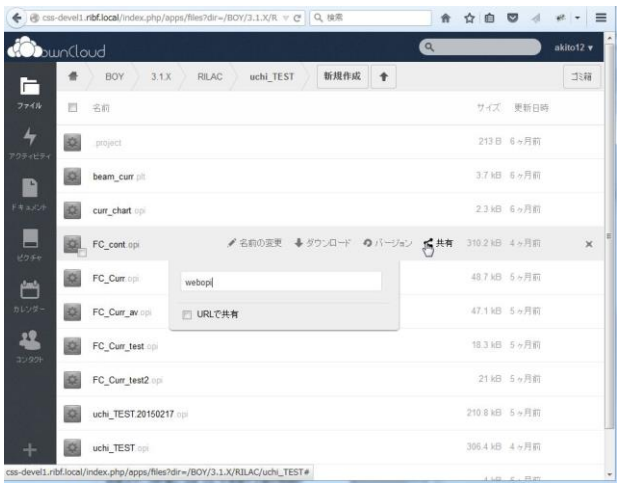

Figure 3: Screen shot of operation screen on ownCloud through Web browser. A synchronized file is shared to a particular user.

選択すると、そのユーザに許可された OPI ファイルがクライアント端末上にダウンロードされ、同期される。

### 3.3 WebOPI との連携

WebOPI を構築するためには JAVA サーブレットコンテナである Tomcat が必要である。Tomcat を走らせるサーバ上には ownCloud クライアントもインストールし、同期された OPI ファイルを Apache 上の Web アプリケーションで表示、WebOPI 経由で呼び出す仕組みをデザインした。この時 WebOPI 専用の ownCloud ユーザを作成、使用しているので、OPI 構築者は同期設定時にそのユーザをファイル共有の選択をするだけで構築した OPI を Web で配信することが可能になる。また WebOPI は出力を伴う制御を禁止設定した EPICS PV gateway[10]を経由して EPICS IOC (Input/Output Controller) から値を取得している。これにより WebOPI からの出力を伴う不正操作を防ぐことが可能になっている。CSS/BOY で構築した OPI を WebOPI で起動させる為の Web アプリケーションのスクリーンショットを Fig. 4 に示す。

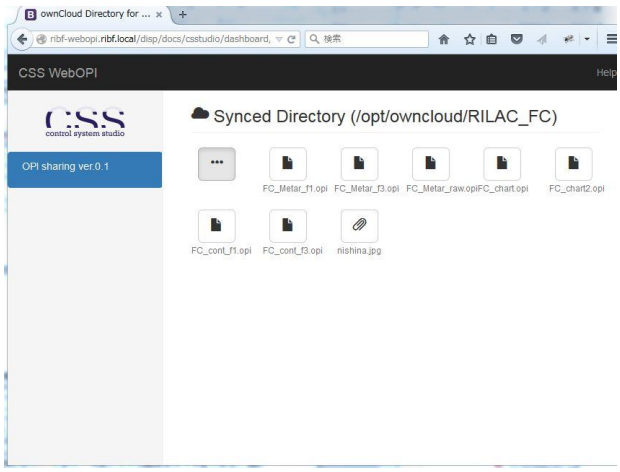


Figure 4: Web application for synchronized files and WebOPI. By clicking the file link, OPI file is opened through WebOPI.

## 4. 実装

本システムは RIBF の入射器である AVF と RILAC の制御系に一部インストールされ、使用されている。RIBF 制御系では、デバイス数の増減やオペレータからの要望に基づく GUI の変更といった事が頻繁に行われる為、柔軟なシステム更新が求められる。そのため OPI ファイルの更新も頻繁に行われる事が想定されるが、本システムを実装した結果、別端末で OPI を変更後、煩雑な手間が発生せずに、すぐに更新された OPI をオペレータが使用できる環境になった。

## 5. 検討事項

システム運用を進めていくうちに、以下の問題に気がついた。一般的に CSS/BOY で OPI を構築する時に、インストールされた CSS 毎で用意されるローカルな Process Variable (PV) を様々な用途で利用する事がある。一方、本システムで OPI ファイルを共有しても実際のオペレーションで使用するクライアント端末にインストールされた CSS/BOY で必ずしもその PV が準備されているわけではないので、そのような OPI については PV を用意しなくてはならない。PV を用意する手間を省くには、OPI 構築時にローカルな PV を使用せずに予め Soft Record を扱う EPICS IOC を別途用意すれば解決する、と考えられる。

しかし新規で Soft Record を追加、有効にするには IOC の再起動が必要になり、他のシステムにも影響が出てしまう。この問題は OPI 毎に IOC を準備すれば解決するが、一つの OS 上で複数 IOC を走らせると EPICS Channel Access (CA) のポートがデフォルトの 5064 ポートから変わってしまう為、PV の探索にブロードキャストを利用していない RIBF 制御系の環境では現実的でない。仮に VMware, Hyper-V といったソフトウェアを用いた仮想マシン (ハイパーバイザー方式) を IOC として利用[11]したとしても、仮想 OS が必要となる為に多くのハードウェアリソース (仮想化のために割り当てる CPU やメモリ等) が必要になってしまい有用ではない、と考えられる。以上の問題点を解決する為に、コンテナ型の仮想化を行う Docker[12]を利用した Soft IOC を提案する。

Docker とは、ホスト OS とは別な仮想ホスト上のアプリケーションを単体で走らせることを可能にさせるソフトウェアである。ホスト OS から見ると一つのプロセス (コンテナプロセス) 上に別システムのアプリケーションが起動している為、ゲスト OS を必要としない。VMware 等は、ハードウェアを仮想化している為 OS が必要になる分アプリケーションの起動に時間がかかり、かつアプリケーション以外のリソースも必要になる。一方 Docker はアプリケーション毎に仮想化を行う為、少ないリソースで効率的な運用が可能になると考えられる。

我々は CentOS6.5 x86_64 上の Docker (v1.3.2) で起動した Soft IOC (base-3.14.12.4) へ通常の IOC 同様、CA クライアントが接続できる事を確認した。この結果より、OPI 毎に Soft IOC を準備する事が現実的に可能になった。また Docker 上の仮想 IOC とホスト内の CA クライアントはホスト OS のネットワークアドレスとは別な仮想ネットワーク層を通じて接続される。つまり仮想 IOC への PV 探索にホスト OS からブロードキャストをしても、実ネットワークには影響がない。したがって、ホスト OS 上で PV gateway を起動し、仮想ネットワーク内へブロードキャストを用いて PV 探索をさせることで、既に走っているプロセスを落とさないまま、レコードを追加、適応でき、かつ OPI 側からは一つの Soft IOC としてふるまう事ができる、と考えている (Fig. 5

参照)。

## 6. まとめ

RIBF 制御系に CSS/BOY を導入するにあたり、OPI 構築者とオペレータ間で問題となるファイル共有の問題解決に ownCloud で構築したプライベートクラウド環境の適応が有効であるという事がわかった。OPI で利用するローカルな PV を提供する為に Docker 上で走らせた Soft IOC が効率的であると考えており、今後システム開発を行う予定である。具体的には、現状コマンドラインで行っている Docker 上での Soft IOC 構築、PV 追加を Web アプリケーションから行う事を可能にさせる。

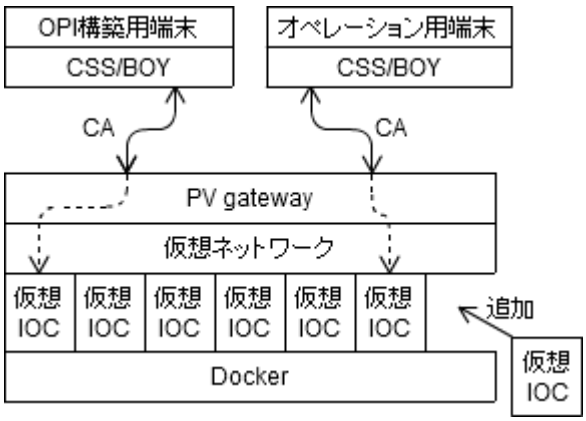


Figure 5: Concept diagram of Soft IOC using Docker. PV gateway behaves like single Soft IOC from a standpoint of OPI.

## 参考文献

[1] M. Komiyama, et al., Proc. ICALEPCS07, Knoxville, Tennessee, USA, 2007. P. 187-P. 189.
[2] Kenneth Evans Jr. Proc. ICALEPCS 1999 Trieste, Italy, P.466-P.468.
[3] R. Keitel. Proc. ICALEPCS05, Geneva, 2005.
[4] K. Yoshii, et al., Proc. 10 Annu Meet. Particle Accelerator Society of Japan, Nagoya, P. 728-P. 730.
[5] Kay-Uwe Kasemir, et al., Proc. ICALEPCS2011 Grenoble France, P. 1178–P. 1181.
[6] https://owncloud.org/
[7] https://www.dropbox.com/
[8] https://google.com
[9] A. Uchiyama, et al., Proc. ICALEPCS2011, Grenoble, France, P. 1161-P. 1164.
[10] Kenneth Evans and Martin Smith. Proc. PAC2005, P. 3621–P. 3623.
[11] L. R. Shen, et al., Proc. ICALEPCS2011, Grenoble, France, P.1138-P.1140.
[12] https://www.docker.com/