# EPICS クライアントアプリケーション開発のためのパイプラインツールの開発

# DEVELOPMENT OF A PIPELINE TOOL FOR EPICS CLIENT APPLICATION DEVELOPMENT

佐々木信哉#
Shinya Sasaki #
High Energy Accelerator Research Organization (KEK)

*Abstract*

Accelerator facilities that use EPICS for control system development employ a variety of EPICS client applications to fulfill diverse operational requirements. Some of these applications involve a process of data extraction, transformation, and transmission. To support the efficient and flexible development of such applications, CauliFlow has been developed. CauliFlow is a pipeline-oriented tool designed for EPICS client development. It decomposes application functionalities into modular components called Nodes, which can be interconnected to construct customized workflows. This paper presents the architectural design of CauliFlow and a its usage.

## 1. はじめに

加速器の制御システム構築のために EPICS[1]を採用する施設では、様々な用途の EPICS クライアントアプリケーションが利用される。これらのクライアントアプリケーションの中には、データの収集・変換・送信という工程に沿って動作するものがある。例えば、SuperKEKB では、EPICS PV の値を収集し、その値を Zabbix に定期的に送信するアプリケーションを開発して利用している[2]。

このような EPICS クライアントアプリケーションは、目的に応じて専用のアプリケーションとして開発されることが多い。しかし、収集・変換・送信という工程で動作するアプリケーションは、必要とされる機能や特性が共通する部分も多い。そのため、各工程での処理を分割し、それらの機能を繋ぎ合わせることでアプリケーションを実装できれば、汎用的で再利用性の高いシステムが実現できると考えた。そこで、Node という形に機能を分割し、それらを繋ぎ合わせることで目的に応じたアプリケーションを実現できるパイプラインツール CauliFlow を開発した。

本稿では、はじめに既存のパイプラインツールに関してまとめ、EPICS クライアントアプリケーションの構築に利用する際の課題をまとめる。続いて、CauliFlow の設計とその使用方法に関してまとめる。

## 2. 既存のパイプラインツール

パイプライン形式で利用可能かつ、EPICS と連携可能なワークフローシステムとして Lightflow[3]が挙げられる。Lightflow は ANSTO において開発された分散ワークフローシステムである。ワークフローで実行されるタスクは、利用者自身が定義し、Lightflow は各タスクの実行とタスク間のデータの受け渡しを行う。Python によって開発されており、タスクの定義や、実行するワークフローの構成も Python で記述する。EPICS extension という Python パッケージを導入することで、EPICS と連携するための事前定義済みのタスクが利用できる。Lightflow は分散環境での利用を想定している。そのため、ワークフローを実行するためのワーカーと呼ばれるサーバーを事前に起動しておく必要がある。また、タスク間の通信のために Redis を、ワークフロー全体のデータ共有のために MongoDB を利用する。

ログ収集ツールとして利用されることの多い Fluentd[4]や、Logstash[5]もパイプライン形式でデータ処理を行うシステムである。データの入出力や変換を行う機能がプラグイン形式になっており、目的に応じて機能を拡張することができる。そのため、EPICS と連携するためのプラグインを開発すれば、これらのツールで EPICS PV とのデータのやり取りが可能となる。しかし、これらのツールでやり取りされるデータは、パイプラインの上流から下流へ渡されるデータのみである。また、1 つのパイプラインにおいて複数の情報源からデータを取得することは難しい。

LightFlow は、EPICS との連携が可能であり、制御システムとの統合も行いやすい。しかし、LightFlow は分散環境での運用を前提として設計されており、外部のデータベースサーバーや、ワーカーの導入および運用が必要となる。また、タスクやワークフローの記述には Python が必須であり、タスクの共有にはコードのパッケージ化など追加の作業が求められる点も課題として挙げられる。

一方、Fluentd や Logstash といったデータ収集・処理ツールは、多数のプラグインを活用することで柔軟なデータ入出力が可能である。しかし、これらのツールは主にログ収集や単純なデータ変換を目的としており、複数の情報源からのデータを統合し、それに基づいた複雑な処理を行うには限界がある。

これらの課題を踏まえ、EPICS クライアントアプリケーションの構築に十分な柔軟性を持ちつつ、インストールおよび利用が容易なパイプラインツール CauliFlow を開発した。

## 3. CauliFlow の開発

### 3.1 アプリケーションの概要

EPICS クライアントアプリケーションをパイプライン形式の構成で実現するために、CauliFlow を開発した[6]。開

# shinya.sasaki@kek.jp

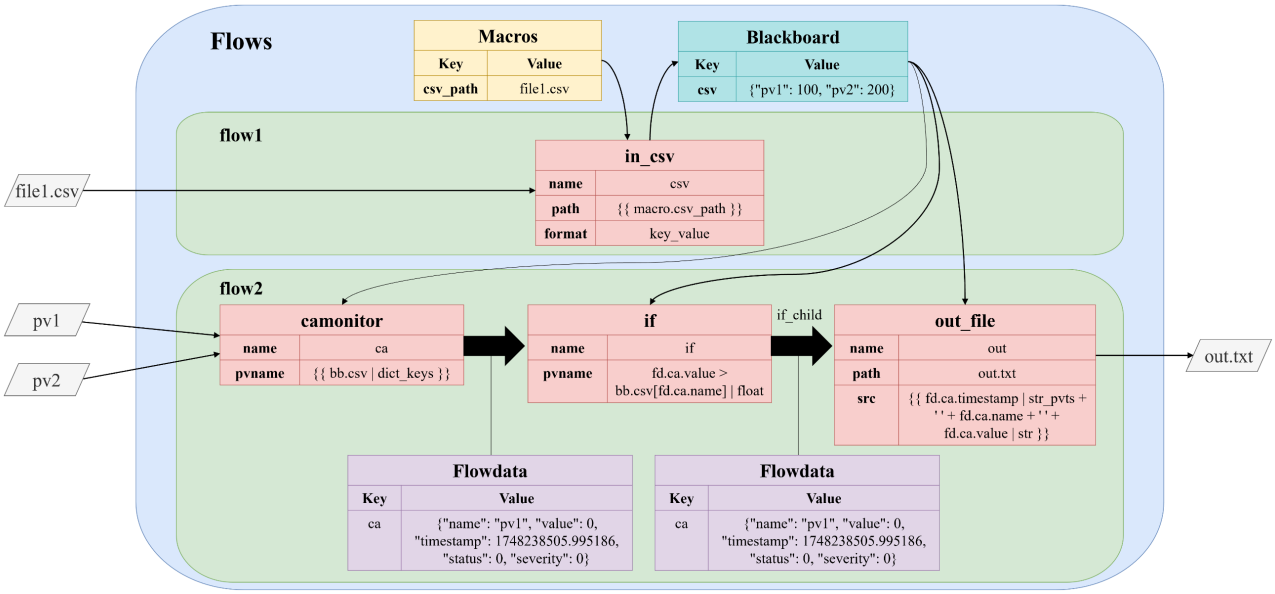


Figure 1: The diagram showing an example of workflows. These workflows run as follows: First, flow1 reads a CSV file and loads EPICS PV names along with their threshold values into the Blackboard. Then, flow2 monitors the specified PVs and passes the monitoring data downstream as Flowdata. Only when the value of PV exceeds the threshold, the relevant information is written to a file. The path to the CSV file can be specified at runtime using the macro.

発言語は Python である。

本アプリケーションの特徴を以下に記す。

- 単一のプロセスで動作。
- 種類に応じて異なるタスクを実行する Node を組み合わせて、目的に応じたワークフローを構成する。
- Node はプラグイン形式になっており、機能の拡張が可能。
- Flowdata・Blackboard・Macros という 3 つの変数を利用する事で、柔軟かつ再利用性の高いワークフローを実現できる。
- Node のパラメータに式を設定することで、変数の利用や変換、演算を行うことができる。
- ワークフローは YAML 形式のファイルで定義する。

以下では、CauliFlow の設計と機能、その利用方法について説明する。

### 3.2 ワークフローの動作

CauliFlow におけるワークフローの説明のため、Fig. 1 にワークフローの例を示す。CauliFlow のワークフローは 1 つ以上の Flow から構成される。各 Flow は、1 つ以上の Node によってツリー構造を構成する。Flow が複数ある場合、逐次的（sequential）に実行するか、並列的（concurrent）に実行するか選択できる。

Flow が実行されると、ツリーの上位に位置する Node から順に、その種類に応じたタスクを実行する。Node のタスクは Python の async/await の構文を用いたコルーチンとして定義されており、親 Node のタスクから子 Node のタスクを連鎖的に呼び出すことでパイプラインの動作を実現している。各 Node は、Flowdata・Blackboard・Macros という 3 つの変数を、目的に応じて利用する事が出来る。

Figure 1 のワークフローは次のように実行される。まず、flow1 が CSV ファイルを読み込み、EPICS の PV 名とそのしきい値を Blackboard に書き込む。この時、CSV ファイルへのパスは、ワークフロー実行時に Macros を使って指定することができる。次に、flow1 で読み込んだ情報を参照して、flow2 においてその PV を監視する。PV の値に変化があると、その値を Flowdata として下流に渡す。PV の値がしきい値を超えた場合にのみ、関連情報がファイルに書き込まれる。

### 3.3 変数

CauliFlow のワークフローは、Flowdata・Blackboard・Macros という 3 つのキー・バリュー形式の変数を利用できる。

#### 3.3.1 Flowdata

Flowdata は上流の Node から下流の Node へ受け渡される変数である。Node は自身の Node 名もしくは任意の文字列のキーの値を更新できる。ただし、Flowdata 内に既に存在するキーのデータを上書きすることはできない。

#### 3.3.2 Blackboard

Blackboard はワークフロー全体で共有される変数である。Flowdata と同様に、Node から値を更新できる。Blackboard は、既に存在するキーのデータを上書きすることができる。

#### 3.3.3 Macros

Macros はワークフロー実行時にその値を設定可能な変数である。ワークフローの定義において初期値を設定することもできる。

### 3.4 Node

Node は呼び出された際にタスクを実行する、Flow の基本的な構成要素である。Node はその種類に応じたパラメータを持つ。パラメータの設定によって Node の動作を変更することができる。

Node はその振る舞いによって、以下の 3 つのカテゴリーに分けられる。

- Trigger Node
- Process Node
- Control Node

以下からは各カテゴリーの Node の振る舞いに関して説明する。

#### 3.4.1 Trigger Node

Trigger Node は、自身が待ち受けるイベントが発生した際に自身の子 Node を呼び出す Node である。Trigger Node のタスクが呼び出されると、無限ループに入り、そのループ内でイベントの発生を待ち受ける。Table 1 に Trigger Node の例を挙げる。

例えば camonitor という Trigger Node は、監視している EPICS PV の値が変化すると、その PV の情報を Flowdata に書き加えた後に、自身の子 Node を呼び出す。

Table 1: Examples of Available Trigger Nodes

| Node 名 | 説明 |
|---|---|
| camonitor | EPICS PV を監視する。値に変化があるとその値を Flowdata に書き込む。 |
| interval | 一定時間ごとに子 Node を呼び出す。 |
| scheduler | cron 形式で指定したスケジュールで子 Node を呼び出す。 |

#### 3.4.2 Process Node

Process Node は、呼び出された際に即座に自身のタスクを実行する Node である。タスクが終了すると自身の子 Node を呼び出す。Table 2 に Process Node の例を挙げる。

例えば、in_csv という Process Node が呼び出されると、指定された CSV ファイルを読み込み、そのデータを Flowdata もしくは Blackboard に書き込む。その後、自身の子 Node を呼び出す。

Table 2: Examples of Available Process Nodes

| Node 名 | 説明 |
|---|---|
| caput | EPCIS PV に値を書き込む。 |
| http | http メソッドによるデータの取得や出力を行う。 |
| in_csv | CSV ファイルからデータを読み込む。 |
| out_file | テキストファイルにデータを書き込む |

#### 3.4.3 Control Node

Control Node はワークフローの制御を行う Node である。Control Node は複数の子 Node を持つことが可能であり、ワークフローの実行経路を制御できる。Table 3 に Control Node の例を挙げる。

例えば、if という Control Node は、chifld_if、child_else という特別な子 Node を持つことができる。if が呼び出されると、condition パラメータの式が評価される。式が真と評価されると child_if の子 Node が、偽と評価されると child_else の子 Node が呼び出される。

Table 3: Examples of Available Control Nodes

| Node 名 | 説明 |
|---|---|
| if | Flow の条件分岐を行う。 |
| foreach | リスト形式のデータを受け取って、その要素ごとに子 Node を呼び出す。 |
| dispatch | Flowdata を複数のターゲットノードに分配して呼び出す。 |
| buffer | データのバッファリングを行う。データが一定サイズを超えるか、一定時間を過ぎると、そのデータと共に子 Node を呼び出す。 |

### 3.5 式

Node のパラメータにおける文字列内で、2 重の中括弧（{{}}）で囲われた部分は式（Expression）として評価される。式の中では、Python で利用可能な値や演算子の一部、Flowdata などの変数、Filter が利用できる。変数は式の中で、それぞれ Table 4 の通りに参照できる。また、その値はドット区切りでのキー指定での参照、もしくは Python の辞書と同様の添え字による参照が可能である。

Table 4: Names Used to Reference the Variables within Expressions

| 変数の種類 | 式内での参照名 |
|---|---|
| Flowdata | fd |
| Blackboard | bb |
| Macors | macro |

### 3.6 Filter

Filter は値を受け取って、何らかの処理の後に値を返す関数である。値の型変換や変形に利用する。Filter は、それを適用したい変数や値の後に、バーティカルバー（|）を挟んでフィルター名を指定することで利用できる。フィルターの後に、更に続けてフィルターを指定することで、複数の Filter を連続的に適用することもできる。Table 5 に Filter の例を挙げる。

例えば、dict_keys という Filter は、受け取った辞書型のデータのキーのリストを返す。また、float や str といった Filter は、受け取ったデータを float や str に型変換する。

Table 5: Examples of Available Filters

| Filter 名 | 説明 |
|---|---|
| str_pvts | UNIX エポックからの秒数の値（float 型）を、整形した文字列に変換。 |
| dict_keys | 辞書型のデータからその全てのキーが入ったリストを取得する。 |
| dict_values | 辞書型のデータからその全ての値が入ったリストを取得する。 |
| join | リスト中の文字列を結合した文字列を返す。 |

### 3.7 ワークフローの定義

CauliFlow で実行するワークフローは YAML 形式のファイルに記述する。Figure 1 に示したワークフローを YAML 形式のファイルとして記述した例を Fig. 2 に示す。

ファイル内の記述は、ワークフローの構造を反映した構成になっている。Flows が複数の Flow を持ち、Flow は複数の Node によって構成される。

明示的に指定しない限り、Node は直前に定義された Node の子 Node となる。Node の parent パラメータに親 Node となる Node の name を指定すると、その Node の子 Node となる。また、Control Node のように、特別な子 Node をパラメータとして持つことが可能な Node に対しては、<name>.<parameter>のようにドット区切りでその指定が可能である。Figure 2 の例において、out は parent として if.child_if を明示的に指定している。これは、if の condition パラメータの評価が真の場合のみに out の Node を呼び出すためである。

```
---
sequential:
  flows:
    - name: "flow1"
      flow:
        - in_csv:
            name: "csv"
            path: "{{ macro.csv_path }}"
            format: "key_value"
            out_bb: yes
    - name: "flow2"
      flow:
        - camonitor:
            name: "ca"
            pvname: "{{ bb.csv | dict_keys }}"
        - if:
            name: "if"
            condition: "fd.ca.value > bb.csv[fd.ca.name] | float"
        - out_file:
            name: "out"
            path: "out.txt"
            src: "{{ fd.ca.timestamp | str_pvts + ' ' + fd.ca.name
+ ' ' + fd.ca.value | str }}"
            parent: "if.child_if"
macros:
  csv_path: "test.csv "
```

Figure 2: A workflow defined in YAML format.

### 3.8 ワークフローの実行

定義したワークフローは cauliflow run というコマンドで実行することが出来る。Macros の値の指定も、このコマンドの実行時に行える。Figure 2 のように定義されたワークフローを実行した例を Fig. 3 に示す。

```
$ cauliflow run -m csv_path ./file1.csv ./flows.yml
^C
Aborted!
$ cat out.txt
2025-05-26 16:28:02.30860 pv1 101.0
2025-05-26 16:28:03.30767 pv1 102.0
2025-05-26 16:28:10.30760 pv2 220.0
2025-05-26 16:28:11.30771 pv2 230.0
```

Figure 3: Running CauliFlow and its results.

## 4. まとめと課題

EPICS PV との連携が可能なパイプラインツール CauliFlow を開発した。CauliFlow を用いることで、EPICS クライアントアプリケーションの開発を簡便かつ柔軟に行えるようになることが期待される。

今後の課題として、Node のさらなる拡充、システム全体の負荷試験の実施、およびエラー処理機構の強化が挙げられる。これらの改善により、CauliFlow の実用性と信頼性がさらに向上すると期待される。

## 参考文献

[1] EPICS, https://epics-controls.org/

[2] S. Sasaki *et al.*, “Monitoring system with Zabbix at SuperKEKB”, Proc. 16th Annual Meeting of Particle Accelerator Society of Japan (PASJ2019), Kyoto, Japan, Jul.-Aug. 2019, pp. 596-599.

[3] A. Moll *et al.*, “Lightflow - a Lightweight, Distributed Workflow System”, in Proc. ICALEPCS'17, Barcelona, Spain, Oct. 2017, pp. 1457-1459. doi:10.18429/JACoW-ICALEPCS2017-THPHA043

[4] Fluentd, https://www.fluentd.org/

[5] Logstash, https://www.elastic.co/logstash

[6] CauliFlow, https://github.com/sasaki77/cauliflow