Adding Control to Fledge

Introduction

One area that has been discussed on numerous occasions is the addition of a degree of control, or ability to write to devices through Fledge. This ability to modify the devices or settings of the devices attached to Fledge is not “control” in the sense of a PLC could be considered to control a real time process, but it does allow machine behaviours to be modified by altering the operating parameters of a machine. This is known as “Set Point Control” generally. It also allows specific operations to be implemented in a south service that can be triggered by other components of the Fledge system. These operations could be things such as calibration operations, mode changes or shutdown instructions.

Fledge as a platform is not designed or well suited to be a real time process controller, it does not offer strict guarantees on the processes of messages or events within the system that a device such as a PLC would.

Control Operations

Two distinct control functions exist;

- Value updates - this is a control operation that alters a configuration or other variable in the device.
- Operations - some service wishes to perform an operation on a device. A request is sent that will be executed in the south plugin for the device.

Southbound Flow

Currently data flows from the south service via filter chains to the north, control or update information would naturally flow into the south and out via the plugin to the device that is being updated or controlled.

Data is currently read using a map or other plugin specific specification and translated into generic JSON encoded readings, with data points, timestamps and other common information. In the control environment the data would come into the south service in a generic structure and be sent via the plugin to the device. The plugin would be responsible for converting that generic format into something device specific. In the same way that a modbus map in the modbus south plugin converts between one or more modbus registers and a data point within an asset, a write map would be used to convert from a generic parameter that needed to be set to a set of modbus registers.

South plugins would be augmented to have a new write entry point for taking data that needs to be sent to the device in the south. An operation entry point will also be added to trigger an operation.

Data going northwards is sent by the south service to the service interface of the receiving micro-service, the storage service, in the case of flow to the south then it would be the job of the controlling service to send the data regarding the items to be set to the service interface of the south service. The south service would then invoke the write entry point of the south plugin.

South Service Interface

Currently the south microservice does not have a service interface, it merely sends data to the other services within Fledge and offers a management service common with the other microservices. In order to facilitate writing data to the devices the south microservice will implement a service interface that will accept key/value pairs for the data to be written to the device. These key/value pairs will form a JSON payload to the write interface that will allow a set of values to be set in a single call.

The service interface will be implemented as a REST interface and will require authentication to be performed before any write operations can be accepted. The authentication will use tokens managed by the Fledge core and secured by the core. This will prevent applications outside of Fledge from injecting write commands into the devices via Fledge.

All write operations performed by the south service will be logged in the audit log.

Write API

Write a set of values to the device connected to this south service.

PUT /fledge/south/setpoint

Request Payload

Name	Type	Description	Example
values	JSON Object	An object containing name/value pairs to write to the south service	
values.*name*	string	The name and value pair to be written.	“speed” : “15000”

Example

Assume we have a device attached to the south service that rotates at a given speed for a given time as part of the process it executes. We change that speed and duration based on the current conditions. To do this we execute the following REST API call against the south service interface.

PUT /fledge/south/setpoint

With the PayLoad

```
{
        "values" : {
                        "speed"    : "12500",
                        "duration" : "90"
                }
}
```

This would set the speed to 12500 RPM and the duration at that speed to 90 seconds. The plugin would receive these values and convert them to whatever commands are required to be sent to the machine itself.

Operation API

PUT /fledge/south/operation

Request Payload

Name	**Type**	**Description**	**Example**
operation	string	The name of an operation to execute on the device.	calibrate
parameters	JSON Object	An optional object containing name/value pairs of the parameters to the operation.	
parameters.*name*	string	The name of the parameter	
parameters.value	string	The value of a parameter to pass the operation.	

Response Payload

The response payload is a JSON structure.

Name	**Type**	**Description**	**Example**
status	string	The status of the operation. This may be one of “Running”, “Complete” or “Failed”.	
id	numeric	An id for this operation. This can be used to access the status of the operation.	

Example

Suppose we wish to run a calibration operation on a device. This operation requires no parameters.

```
PUT /fledge/south/operation
```

```
with payload
```

```
{
    "operation" : "calibrate"
}
```

Alternatively, if we wish to run an operation that takes parameters, say a calibration on a single axis

```
PUT /fledge/south/operation
```

```
with payload
```

```
{
    "operation" : "calibrate",
    "parameters" : {
                         "axis" : "x"
                   }
}
```

Status Query

GET /fledge/south/operation/{id}

Return the status of a previously started operation.

Response Payload

The response payload is a JSON structure.

Name	Type	Description	Example
status	string	The status of the operation. This may be one of “Running”, “Complete” or “Failed”.	
id	numeric	An id for this operation. This can be used to access the status of the operation.	

Plugin Write Entry Point

South plugins that support the ability to write data to a sensor or device will encode this by setting the flag PLUGIN_WRITABLE in the plugin_info return structure. If this flag is set then the plugin must also support the plugin_write entry point.

The plugin write entry point is defined as follows

```
     bool plugin_write(PLUGIN_HANDLE *handle, string name, string value)
```

Where the parameters are;

- handle - the handle of the plugin instance
- name - the name of the item to be changed
- value - a string presentation of the new value to assign top the item

The return value defines if the write was successful or not. True is returned for a successful write.

Plugin Operation Entry Point

The plugin will support an operation entry point. This will execute the given operation synchronously, it is expected that this operation entry point will be called using a separate thread, therefore the plugin should implement operations in a thread safe environment.

The plugin write operation entry point is defined as follows

```
    bool plugin_operation(PLUGIN_HANDLE *handle, string operation, int parameterCount, PLUGIN_PARAMETER parameters
[])
```

Users of the South Service Interface

The south service interface would not be exposed outside of the Fledge system, but rather consumed by other microservices within Fledge, such as the notification service in order to allow notifications to trigger updates of machines, from the new dispatcher or by a service that provided an interface to other systems such as an MES system.

We could offer access to this API via the public REST API of Fledge itself, however we should probably limit it to authenticated users only and we also would need to consider having user permissions within that API andthe associated GUI as general access to alter the behavior of the machine is a radical departure from the current Fledge/Fledge offering.

North interface access to write operations is something to be considered. It makes more sense for some north services but not all. OPCU-UA north would be an obvious candidate, as would GCP or other cloud IoT core based systems, however it should consider running that as a true north microservice rather than a task. PI systems, and data stores such as InfluxDB or HarperDB make less sense having a control route.

Control Dispatcher Service

Rather than every end point knowing about every other end point and dispatching control updates and operations between service endpoints a new microservice, the control dispatcher will be introduced. This service will be responsible for dispatching control messages between the services, it will use rules to perform this function and these rules will be extensible.

Dispatching Rules

The operations and control updates will have a number of attributes that will be used by the dispatcher to route the control messages to the correct end points.

The simplest of these rules will be name based, the control message names the service to which it is destined.

A rule will also allow an operation or write event to be dispatched to an asset source. Asset tracker data will be used to determine the service that created the named asset and the control operation will be forwarded to that service by the dispatcher.

However more complex rules will be added, these will include the ability to dispatch a single control request to multiple destinations with operational rewriting of requests as they are dispatched. As with other aspects of Fledge the dispatcher rules will be added via plugins, allowing the user/developer to tune the the rules that will be used to determine how to route control messages both for set point calls and for operations. These dispatcher plugins will be available to be written in both C/C++ and Python.

The dispatcher will be able to route a single request to multiple destinations.

North Plugin Changes

Another source of control is the north plugin. In this case an external system with which Fledge is communicating requires an operation or a value to be written to a device via Fledge. The north plugin interface will be updated to allow north plugins to direct control messages to the dispatcher service and for the north plugin to receive notification of the status of those operations.

Control Callbacks

A north plugin will dispatch control requests via a callback into the north service. The north plugin will request, via the flags in the plugin information structure that it requires these callbacks to be registered with the plugin. The callbacks take the form of C function pointers in a C plugin and a Python call for a Python based plugin.

Write Callback

The write callback is a function pointer with the signature

```
    bool write(char *name, char *value, ControlDestination destination, ...)
```

Where name is the name of the value to write, and value is the string representation of the value to write. The destination argument controls where the request will be routed and uses additional argument to qualify that routing.

Operation Callback

The operation callback is a function pointer with the signature

```
    int operation(char *operation, int paramCount, char *names[], char *parameters[], ControlDestination
destination, ...)
```

Where name is the operation is the name of the operation to be executed, and paramCount is the number of parameters to be passed to that operation names is an array of the parameter names and parameters is an array of string values that are the parameters themselves. The length of the names and parameters arrays should both be paramCount. The destination argument controls where the request will be routed and uses additional argument to qualify that routing.

Control Destination

The control destination is an enumerated type that is used to encode the dispatcher rules that will determine the destination of this operation. A number of arguments may follow the destination in order to further qualify the destination.

A write operation to a named service, say the service called “Trans1” would look as follows

```
    write(“test”, “value”, DestServiceName, “Trans1”);
```

If the destination was a asset source, say the service that created the asset “Transformer1Temperature”, then the call would be

```
    write(“test”, “value”, DestAssetSource, “Transformer1Temperature”);
```

Callback Initialisation

A north plugin that implements control flow must signal to the north service that is loading the plugin that it implements the control flow and provide an entry point that will be used by the north service to pass the callback pointers to the north plugin. Note, north plugins that are run by a north task rather than as a service will not be able to use the control flow.

A north task signals that it supports the control flow by adding the flag SP_CONTROL to the flags field of the information structure that the plugin_info call returns.

When the north service detects that the plugin supports the control flow it will make a call to pass the callback function pointers to the plugin. This call is made after the call to plugin_init has completed.

```
    void plugin_register(PLUGIN_HANDLE handle, bool ( *write)(char *name, char value, ControlDestination
destination, ...),
                                     int (* operation)(char *operation, int paramCount, char *parameters[],
ControlDestination destination, ...))
```

The plugin should retain these two function pointers for use when it wishes to invoke control operations.

Security

The advent of control increases the need for strict security mechanisms within the system. However end to end security can not be achieved by simply adopting the security mechanism of the control message producer with that of the consumer, they may have different security models and a single producer may be communicating with multiple consumers, each with a different security model.

Micro Service Authentication

It is important that messages can be authenticated as originating from a particular microservice, to this end we will use the microservice authentication scheme which was previously outlined. In this scheme when a service is started it will be given a one time use token that it must pass in its registration request to the core. The core will then issue a bearer token to the microservice that is tied to the microservice name, type, address and service port. This bearer token must then be included in all messages that are secured sent to any other micro service within the Fledge system. If there is no bearer token then the message will be unauthenticated and may be rejected by the recipient. In is a policy decision of the recipient as to whether to reject unauthenticated messages or not.

The receiver of this message can then authenticate the bearer token with the core, requesting that it matches any or all of the criteria encoded within the token. This allows communication between the microservices to be authenticated.

South services will use this mechanism to authenticate that a control request has come form a valid microservice and can decide if it trusts that the given microservice based on the ability of that microservice to support the security protocol with the external system.

Service Security Configuration

All micro services within Fledge will have a new subcategory of the configuration of the micro service called <service name>Security, this category will contain a number of elements that can be used to control the access to the functionality of that service.

Authenticated Caller

A boolean option. If set the calls to the service interface of the service must be from an authenticated service within Fledge. Any calls from an unauthenticated source will be rejected.

Service Access Control List

An access control list that can be used to restrict the authenticated services which are able to call the service interface of this service. If empty then the ACL is not checked, if it is populated then the caller will be checked against the list to determine if it can use the service.

Entries in the access control list may reference particular service names or service types. For example a south service may set a service access control list that states access is only granted to north services and notification services or it may state that access is only granted to the service called “OPC/UA North”.

URL Access Control Lists

A set of access control lists tagged with an operation name that can be used to further restrict access to particular URL’s of the service interface. An example might be that the /fledge/south/operation URL is restricted to just the service called “IEC-104 North”.

Implementation

A new configuration item type, called ACL will be introduced, internally it will save the ACL data as a JSON structure, however custom GUI code will display this JSON structure in a user friendly interface.

The JSON ACL structure will be as follows

```
{
        "service" : [
                        {
                                "name" : "IEC-104 North"
                        },
                        {
                                "type" : "Notification"
                        }
                ],
        "URL" : [
                        {
                                "URL" : "/fledge/south/operation",
                                "ACL" : [
                                                {
                                                        "type" : "Notification"
                                                }
                                        ]
                        }
                ]
}
```