Document title
The Arrowhead OPC-UA to REST Proof-of-concept — System Description

Document type
PUBLIC

Date
2019-03-01

Version
0.1

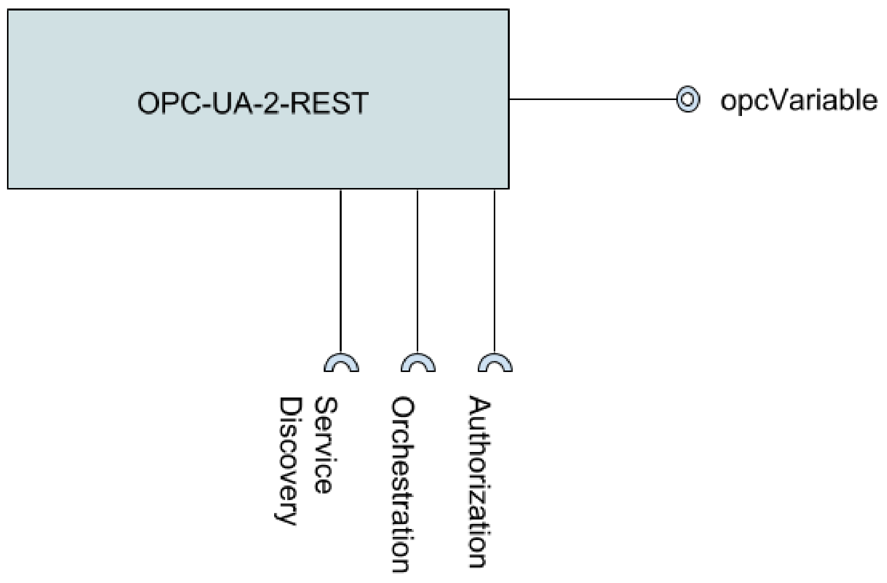
Author
Niklas Karvonen

Status
DRAFT

Contact
niklas.karvonen@ltu.se

Page
1 (8)

The Arrowhead OPC-UA to REST Proof-of-concept — System Description



Abstract

This document provides the description of a proof-of-concept OPC-UA Provider for the Arrowhead framework. When a Consumer makes a pull-request to the Provider, the Provider connects to an OPC-UA system and reads the requested variable which it then returns in a JSON format to the Consumer.

1 Basic operation

To start the Provider, run: "java -jar provider-4.0.jar -auth" after building the Maven module. The -auth flag is optional and will only add the Consumer found in the "tester" folder to the IntraCloud authorization so that it can consume the service. If you don’t need that, you can omit the -auth flag.

When the Provider is run, the first thing that it will do is to register itself with the Service Registry. This is done by sending a "Service Registry Entry" to the Service Registry core service. In more detail, this is achieved by sending a JSON object to URI:*serviceregistry/register*. An example of such a Service Registry Entry is found below:

```json
{
  "providedService" : {
    "serviceDefinition" : "OPCUARead",
    "interfaces" : [ "JSON" ],
    "serviceMetadata" : {
      "unit" : "unit"
    }
  },
  "provider" : {
    "systemName" : "InsecureOPCUA2RESTProvider",
    "address" : "127.0.0.1",
    "port" : 8460
  },
  "serviceUri" : "opcVariable",
  "version" : 1
}
```

The second thing that the Provider does is that it adds a Consumer called "client1" that is allowed to consume the service IntraCloud. This is done by sending an "IntraCloud Auth Entry" to the Authentication core service. In more detail, this is achieved by sending a JSON object to URI: *authorization/mgmt/intracloud*. An example of an IntraCloud Auth Entry can be found below:

```json
{
  "consumer" : {
    "systemName" : "client1",
    "address" : "localhost",
    "port" : 8080
  },
  "providerList" : [ {
    "systemName" : "InsecureOPCUA2RESTProvider",
    "address" : "127.0.0.1",
    "port" : 8460
  } ],
  "serviceList" : [ {
    "serviceDefinition" : "OPCUARead",
    "interfaces" : [ "JSON" ],
    "serviceMetadata" : {
      "unit" : "unit"
    }
  } ]
}
```

Once these registrations are complete, the Provider will listen for incoming requests from Consumers wanting to read OPC-UA variables. A Consumer can read an OPC-UA variable by making a HTTP GET request to the Provider sending it the OPC-UA endpoint, namespace, and variable it wishes to read. These values are sent as URL parameters in the following way: GET opcVariable/{address}/{namespace}/{variableName}. Here "address" is the IP and port to the OPC-UA endpoint, "namespace" is the OPC-UA namespace one wishes to use, and "variableName" is the name of an variable within that namespace wrapped in double quotation marks (you might need to URL encode these using %22). An example of such an GET request to read the variable xBG5 in namespace 3 can be found below:

HTTP GET that is sent to the Provider

```
{providerIP:port}/opcVariable/10.48.134.10:4840/3/%22xBG5%22
```

When the Provider has received a request, it will first see if there is already an open connection to that endpoint. If not it will connect to the endpoint and keep that connection until the Provider is shut down. Then, it will read the requested variable and return its value, type and timestamp in a JSON format.

2 Services

The system consists of a single Provider (but a test Consumer is included in the "tester" folder). This section briefly explains the single service offered by this Provider.

2.1 Produced Services

Service	IDD Document Reference
opcVariable	Arrowhead IDD OPC-UA-2-REST.pdf

2.2 Consumed Services

Service	IDD Document Reference
ServiceDiscovery	*(Not Available)*
Orchestration	*(Not Available)*
AuthorisationControl	*(Not Available)*

The system consumes the `ServiceDiscovery`, `Orchestration` and `AuthorizationControl` services in order to allow service lookup, determine what services to connect to, and regulate service access, respectively.

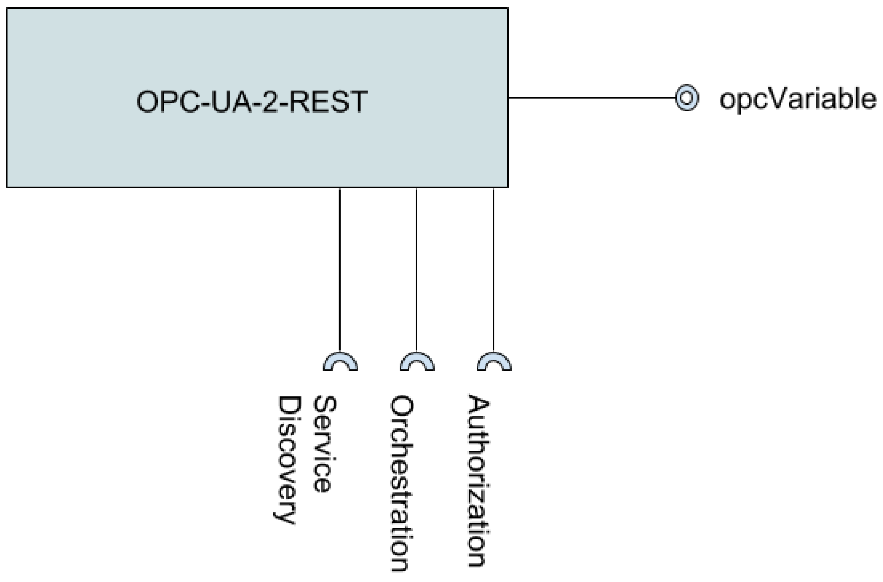


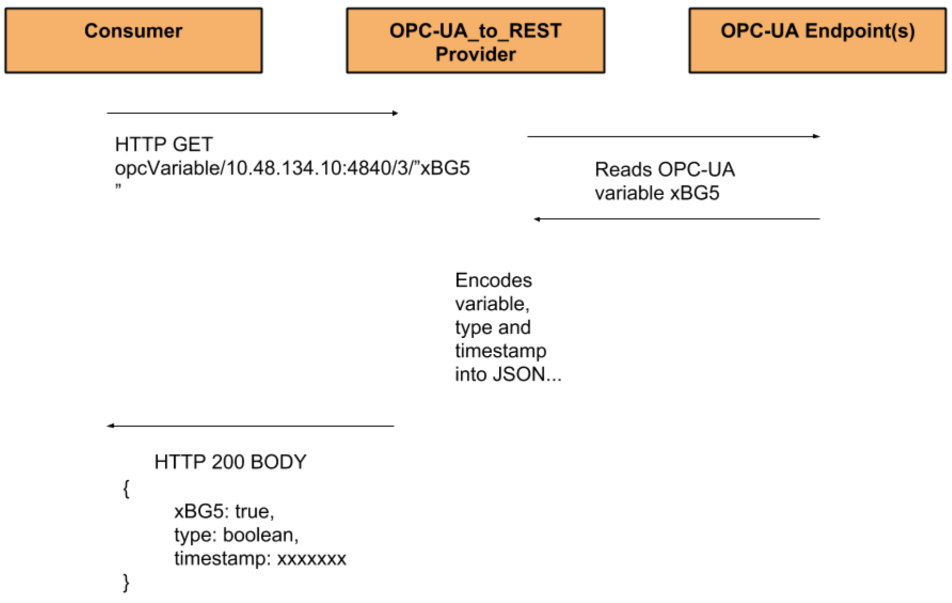
Figure 1: The system consumes the three basic Arrowhead core systems.

3 Use Cases

3.1 Monitoring an OPC-UA facility

This Provider can be used in order for a developer to monitor different variables in an OPC-UA facility. The "tester" Client (found in the tester module) can serve as a starting point for reading OPC-UA variables and can be extended by the developer to include logging, alarms, fault-detection and more.

4 Behavior Diagrams


Consumer
OPC-UA_to_REST Provider
OPC-UA Endpoint(s)
HTTP GET opcVariable/10.48.134.10:4840/3/"xBG5"
Reads OPC-UA variable xBG5
Encodes variable, type and timestamp into JSON...
HTTP 200 BODY
{
xBG5: true,
type: boolean,
timestamp: xxxxxxx
}

- Sequence diagram.png

Figure 2: Sequence diagram of a Client's interaction with OPC-2-REST Provider

5 Security

The OPC-UA connection does not use any security per default in order to provide a low-complexity example. It is up to the developer to take appropriate actions for securing both the Arrowhead Provider in a secure cloud, and to configure and implement the desired OPC-UA security if this is to be extended into a real implementation.

6 Unsolved Specification Issues

There are several issues that have not been addressed in this version. These issues include, but are not limited to, the following:

- **Disconnecting from the OPC-UA** In this version the only way the Provider will disconnect from an OPC-UA endpoint, is when the Provider is shut down. Any connection opened to serve a Client with an OPC-UA variable will remain persistent throughout the lifetime of the Provider.

7 References

8 Revision History

8.1 Amendments

No.	Date	Version	Subject of Amendments	Author
1				

8.2 Quality Assurance

No.	Date	Version	Approved by
1			