Assignment 2

ASE-9476 Factory Information Systems

Juha Suntila
Md Aman Khan
Jarno Nieminen

Assignment objectives

The assignment objectives were to help understanding the features of the OPCUA and experience the usage of hypermedia in web applications. Target was the usage of OPC UA protocol for communicating with simulated water tank. Server-program was provided to be interacted with our self-made client-program.

Code description

First the required libraries were imported. In this case those were node-opcua and async.

Then there were made the coerceId, which uses the method of opcua class to access the nodeId:s of the water tank.

then the client was created. The endpoint was defined so that it will point to the address where the server is running. In this case it was on the same computer where client was running.

After that a couple of variables were made for later use.

```
var opcua = require("node-opcua");
var async = require("async");

var coerceId = opcua.coerceNodeId;
var client = new opcua.OPCUAClient();
var endpointUrl = "opc.tcp://" + require("os").hostname() +
":3003/MyLittleServer";


var tank1, the_subscription;
```

Then the async series was created. In first part the client connects to the endpoint and it prints a comment whether connection was successful or not.

```
// step 1 : connect to
function(callback)  {
    client.connect(endpointUrl,function (err) {
        if(err) {
            console.log(" cannot connect to endpoint :" , endpointUrl );
        } else {
            console.log("connected !");
        }
        callback(err);
    });
},
```

After that a session is created. For session the tank1 variable was used. session creating is one method of opcua class and tank1 is sent there as a parameter.

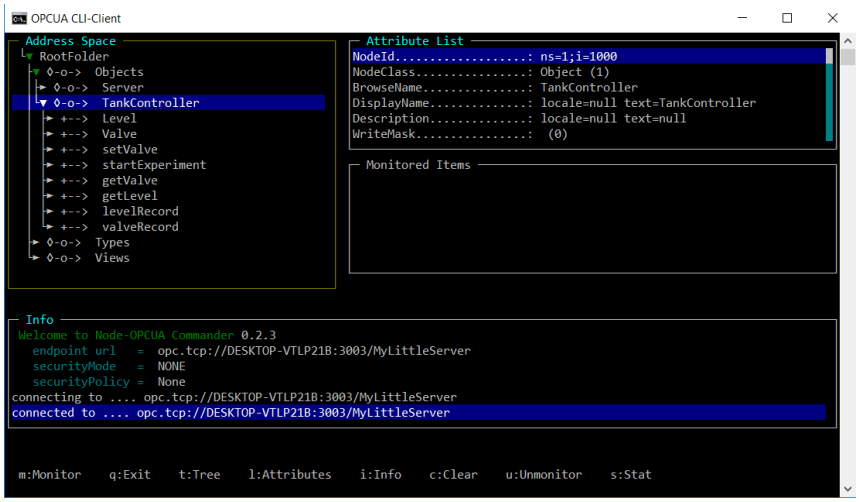
```
// step 2 : createSession
function(callback) {
    client.createSession( function(err,session) {
        if(!err) {
            tank1 = session;
        }
        callback(err);
    });
},
```

After this the server devices were browsed. these results were linked to tank1 variable so later the methods of the water tank class could be accessed.

```
// step 3 : browse
function(callback) {
    tank1.browse("RootFolder", function(err,browseResult){
        if(!err) {
            browseResult.references.forEach(function(reference) {
                console.log( reference.browseName.toString());
            });
        }
        callback(err);
    });
},
```

After it was possible to access properly the water tank server and to start up the experiment.

For starting up the tank, a call method was used. it was done by including it inside a callback function. First an array of methods to be called was made. (var payload). In this case it was only one method, startExperiment, that was called. So array included just one instance. To this instance the tankController nodeId was given and also the methodId of the startExperiment method. These Id's were found from opcua commander interface (picture 1).



Picture 1 OPCUA commander

For startExperiment method no input arguments were required, so as input arguments an empty array was sent. Then the method was called by tank1.call and the payload variable was sent in with function.

```
// trigger start experiment to server

        function(callback) {
            var payload = [];
            payload.push({
                objectId: coerceId('ns=1;i=1000'),
                methodId: coerceId("ns=1;i=1003"),
                inputArguments: []
            });
            tank1.call(payload,function (err, result) {
                console.log(err)
                console.log(result)
            });
            callback();
        },
```

After the ecperiment was started, the subscription request was created.

```
// step 5: install a subscription and install a monitored item
        function(callback) {
            the_subscription=new opcua.ClientSubscription(tank1,{
                requestedPublishingInterval: 10,
                requestedLifetimeCount: 10,
                requestedMaxKeepAliveCount: 2,
                maxNotificationsPerPublish: 100,
                publishingEnabled: true,
                priority: 10
            });
```

Inside subscription function all the things were included that needed continuous monitoring.

First the water level was requested from server. The level was requested as level value so it could be used for P- type controller.

By using the level value the output valve was controlled as follows:

If water level was below 6, the valve was set to be closed without any controlling. Also if the water level was above 7, the valve was set to be fully open without controlling.

Between 6 and 7, the P-controller starts to work. First the setpoint was defined. Also there was a multiplier variable defined to act as P-control multiplier.

After that the level value was compared to setpoint and difference was written to variable as an absolute value.

Then it was checked if the level value was above or below setpoint. If the variable was below the setpoint, the difference variable was multiplied with multiplier and valve value was set so that from 50% opening of the valve, the multiplied difference was reduced. Then it was set that the valve cannot be opened more than 100% or closed under 0%.

After that the valve opening was sent to server in similar method like startExperiment. Only difference was that now methodId was selected from tankController.setvalve path. Also now the input parameters were needed and there the valve opening was sent as a value to the server. Below codes of control.

```
monitoredItem.on("changed",function(dataValue){
    var actualValue = dataValue.value.value;
    var setpoint = 6.5; //point where level is supposed to set
    var diff = Math.abs(setpoint - actualValue);
    var valveOpening;
    var controllerMultiplier = 5.905; //P-controller multiplier
```

The level value was asked from server

```
//this part works when water level is less than setpoint in P-control area
console.log("waterLevel", actualValue);
if ((actualValue >= 6 && actualValue <= 7) && (actualValue <= setpoint)){
    valveOpening = 0.5 - controllerMultiplier * diff;
    if(valveOpening > 1){
        valveOpening = 1;
    };
    if(valveOpening < 0){
        valveOpening = 0;
    };
    var payload = [];
    payload.push({
        objectId: coerceId('ns=1;i=1000'),
        methodId: coerceId("ns=1;i=1001"),
        inputArguments: [{
            dataType: opcua.DataType.Float,
            arrayType: opcua.VariantArrayType.Scalar,
            value: valveOpening}
        ]
    });
    tank1.call(payload,function (err, result) {
        //console.log(err)
        //console.log(result)
    });
};
```

If water level is below setpoint but equal or greater than six.

```
//this part workswhen water level is over the setpoint in P-control area
if ((actualValue >= 6 && actualValue <= 7) && (actualValue >= setpoint)){
    valveOpening = 0.5 + controllerMultiplier * diff;

    if(valveOpening > 1){
        valveOpening = 1;
    };
    if(valveOpening < 0){
        valveOpening = 0;
    };
    var payload = [];
    payload.push({
        objectId: coerceId('ns=1;i=1000'),
        methodId: coerceId("ns=1;i=1001"),
        inputArguments: [{
            dataType: opcua.DataType.Float,
            arrayType: opcua.VariantArrayType.Scalar,
            value: valveOpening}
        ]
    });
    tank1.call(payload,function (err, result) {
        //console.log(err)
        //console.log(result)
    });
};
```

If water level is above setpoint but less or equal to seven.

```
//if level below 6, the P-control is put off and valve is set to be closed
if(actualValue < 6){
    valveOpening = 0;
    var payload = [];
    payload.push({
        objectId: coerceId('ns=1;i=1000'),
        methodId: coerceId("ns=1;i=1001"),
        inputArguments: [{
            dataType: opcua.DataType.Float,
            arrayType: opcua.VariantArrayType.Scalar,
            value: valveOpening}
        ]
    });
    tank1.call(payload,function (err, result) {
        //console.log(err)
        //console.log(result)
    });
}
```

if level is less than six

```
// if level over 7 the P-control is put off and valve is set to be full open
if(actualValue > 7) {
    valveOpening = 1;
    var payload = [];
    payload.push({
        objectId: coerceId('ns=1;i=1000'),
        methodId: coerceId("ns=1;i=1001"),
        inputArguments: [{
            dataType: opcua.DataType.Float,
            arrayType: opcua.VariantArrayType.Scalar,
            value: valveOpening
        }]
    });
    tank1.call(payload,function (err, result) {
        //console.log(err)
        //console.log(result)
    });
}
```

if level is greater than seven.

Then the values of valve value, valve trend and water level trend were requested from server for visualization uses.

```
            // valve value requesting from server

            var monitoredItem  = the_subscription.monitor({
                    nodeId: opcua.resolveNodeId('ns=1;s=valv'),
                    attributeId: opcua.AttributeIds.Value
                },
                {
                    samplingInterval: 10,
                    discardOldest: true,
                    queueSize: 10
                },
                opcua.read_service.TimestampsToReturn.Both
            );
            monitoredItem.on("changed",function(dataValue){
                var valveValue = dataValue.value.value;
                console.log("valvepos ", valveValue);
```

```
            });

//level trend requesting from server

            var monitoredItem  = the_subscription.monitor({
                    nodeId: opcua.resolveNodeId('ns=1;i=1008'),
                    attributeId: opcua.AttributeIds.Value
                },
                {
                    samplingInterval: 10,
                    discardOldest: true,
                    queueSize: 10
                },
                opcua.read_service.TimestampsToReturn.Both
            );
            monitoredItem.on("changed",function(dataValue){
                var levelTrend = dataValue.value.value;
                console.log("levelTrend ", levelTrend);
            });

//try to get valve trend

            var monitoredItem  = the_subscription.monitor({
                    nodeId: opcua.resolveNodeId('ns=1;i=1011'),
                    attributeId: opcua.AttributeIds.Value
                },
                {
                    samplingInterval: 10,
                    discardOldest: true,
                    queueSize: 10
                },
                opcua.read_service.TimestampsToReturn.Both
            );
            monitoredItem.on("changed",function(dataValue){
                var valveTrend = dataValue.value.value;
                console.log("valveTrend ", valveTrend);
            });
        },
    ],
```

Challenges and limitations

Challenges were mostly related to nodeJS and OPC UA syntax and programming language. Most of our time went for figuring out the correct way to call the server program from our client program. When that problem was solved the rest of the client program was easier to continue.

Questions

Question 1: Explain briefly what is the Address Space, what is its importance?

OPC UA Address Space is composed of Nodes and References between them. It provides a standard way for servers to represent objects to clients. It defines Objects in terms of Variables and Methods. It also allows relationships to other objects to be expressed.

Different address space and services are integrated and standardized by OPC-UA and gives the faciality to use single interface for navigation. The hierarchical structure helps to interoperate between clients and server, also all the node can be assessed by exploiting this structure. [2]

Question 2: What is the difference between Object and ObjectType?

Object structures the address space and group variables, methods or other objects. It can even be an event notifier. The main role of the objects is to expose a selected, well-defined part of the information of the underlying process.

ObjectType defines a hierarchy with inheritance relations. Children that are marked mandatory are automatically instantiated together with the parent object. It allows to define custom types and can be nested.

Question 3: Mention the transport mechanisms for OPC UA. Tell in which situations you would use them.

All OPC UA messages are delivered over a TCP/IP connection. There are two protocols which are used depending of the requirements. One is binary (UA TCP), which is fast, high performance optimized TCP protocol small overhead and requires minimum resources. That is used when fast and quick inputs and outputs are needed Small package sizes are quick to deliver and fast to execute. Used usually in lower levels of pyramid of automation. Second is webservice based (SOAP/HTTP(S)) protocol what can be used for larger package transports and where the real time requirements are not so high. Used usually in enterprise levels.

Question 4: Can new transport mechanism be added in the future?

New transport mechanisms are possible and may be added in the future if necessary. Standard only defines previously mentioned mechanisms but doesn't rule out future additions.

Question 5: Mention tasks that you can perform with OPC UA services.

Services can be used to:

- read and write data
- find servers
- manage connections between servers and clients
- find information in the Address Space
- subscribe for data changes and Events
- access history of data and Events
- modify the structure of the server in Address Space

References

[1] "OPC Unified Architecture; Interoperability for Industrie 4.0 and the Internet of Things," [Online]. Available: https://opcfoundation.org/wp-content/uploads/2016/05/OPC-UA-Interoperability-For-Industrie4-and-IoT-EN-v5.pdf.

[2] M. Damm, S.-H. Leitner ja W. Mahnke., OPC Unified Architecture, Springer-Verlag, 2009.

[3] W. M. Luis Gonzalez Moctezuma, *Class Lecture of ASE-9476 Factory Information Systems (S'18), TUT,* Tampere, 2018.