Assignment II – NODE OPC UA deployment

Dharmendra Sharma – 272534
Junaid Iqbal - 272538
Ronal Bejarano – 272544

Objective

Implement a web-based client application by using an OPC UA (Object Linking and Embedding for Process Control, Unified Architecture) client over Node.JS framework, able to discover, subscribe, control and display data from a predefined OPC UA server.

Code description

The code was developed in Javascript and HTML. Backend is based on *node-opcua*, an implementation of a OPC UA stack fully written in Javascript and NodeJs [1].

The architecture of the solution is shown on Figure 1.

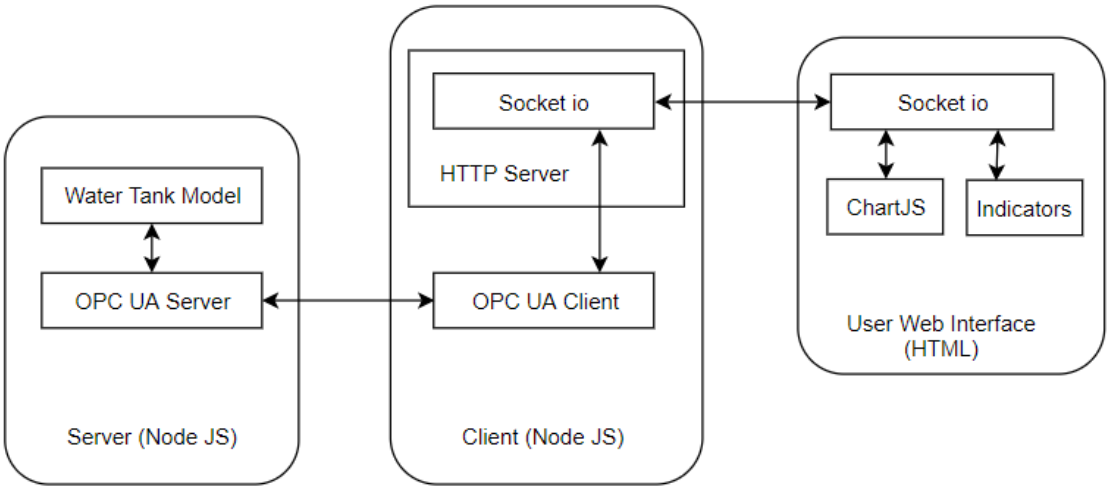

Figure 1 Solution architecture

Initially, two approaches were explored, iterative invocation of methods and subscription to variables. Before the UI development started, it was decided to continue with the subscription approach, since it is an event driven methodology that can be more efficient on network resources than time driven polls.

Data exchanged with OPC UA has object / array format, so is possible to pick specific content of it by calling attributes or array slots. The variable list is described on Table 1.

Type	Name / Tag	Tag	Raw value range	Display range and unit	Comment
Analog Output	setValve	setvlv	0-1	0-1	Valve command
Analog Input	Valve	vlv	0-1	0-100%	Valve status
	Level	lvl	0-10	0-100%	Level sensor
Internal	SetPoint	sp	0-10	0-10	Desired level
	Gain	kp	0-1	0-1	Proportional control gain

Table 1 Variable list

Control methodology implemented follows a continuous time proportional structure. Equation 1 describes the formula implemented.

$$error = vlv = kp\left(1 - \frac{sp - lvl + 10}{20}\right)$$

Equation 1 Proportional control formula

For the extra points activity, the development of a user interface was done in HTML using a line chart template form ChartJs [2]. Instantaneous data can be visualized on numbers and historic data can be visualized in a trend chart. This implementation does not use the historic data module of OPC UA. Figure 2 shows the final design of the web interface, accessible on http://localhost:3700 from a web browser (tested on Chrome V 65.0.3325.181 and Mozilla Firefox 58.0.2).

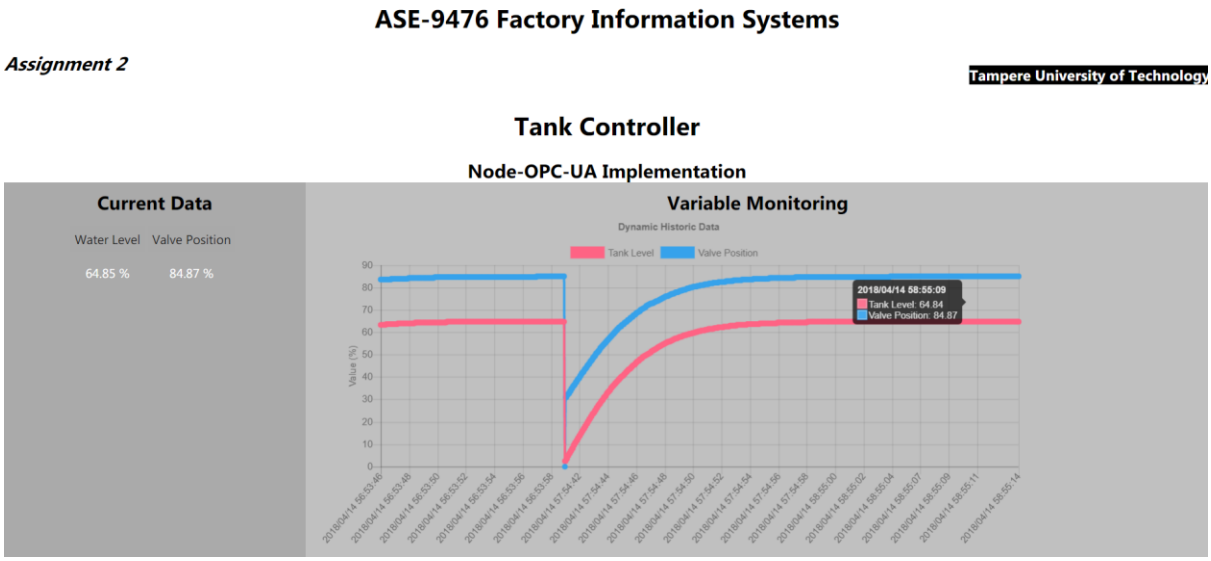

Figure 2 Web Interface design

Code structure:

1. Connect to OPC Server
2. Create Session
3. Browse node ID of TankController
4. Create subscription object and its responses to be logged in console
5. List variables and methods available on TankController
6. Browse node ID for methods and variables to be used in control
7. Start experiment (call method)
8. Start HTTP Server - UI and control function
9. Setup HTTP Server Link HTML to get request
   - 9.1. Link HTML and set port
   - 9.2. Setup socket.io (backend - frontend data connection)
   - 9.3. Create object to subscribe for Level
   - 9.4. Subscription handler, capture variable "Valve" every change event
   - 9.5. Subscription handler, capture variable "Level" every change event, executes control and set valve

Code implemented for backend (Javascript):

```
/*******************************************************/
/*Attaching the required modules and create variables */
/*******************************************************/
var express = require("express");
var opcua = require("node-opcua");
var async = require("async");
```

```
var port = 3700;
var tgtlvl= 6.5;
var kp = 1.7; //Tuned at 1.7
var getL=0;
var getV=0;
var setV = 0;
var client = new opcua.OPCUAClient({
        requestedSessionTimeout: 3000000 //milliseconds (default was 3000x1000
milliseconds )
});
var endpointUrl = "opc.tcp://RON-LAPTOP:3003/MyLittleServer";
var the_session,the_subscription,LevelNodeId, TankCtrlNodeId, startExpeNodeId,
getLevelNodeId, ValveNodeId, setValveNodeId=0;
/*****************************************************/
/* Sequential execution of OPC UA by "series"        */
/*****************************************************/
async.series([
        /*****************************************************/
        /* Step 1 : Connect to OPC Server                    */
        /*****************************************************/
        function(callback)  {
            client.connect(endpointUrl, function (err) {
                if(err) {
                    console.log("Cannot connect to server :" , endpointUrl );
                } else {
                    console.log("Connected to OPC UA Server at: "+ endpointUrl);
                }
                callback(err);
            });
        },
        /*****************************************************/
        // Step 2 : Create Session                           */
        /*****************************************************/
        function(callback1) {
            client.createSession( function(err, session) {
                if(!err) {
                    the_session = session;
                    console.log("session created");
                    console.log("the timeout value set by the server is ",
session.timeout ," ms");
                }
                else {
                    console.log("cannot create a session");
                }
                callback1(err);
            });
        },
        /*****************************************************/
        /* Step 3: Browse node ID of TankController          */
        /*****************************************************/
        function (callback) {
            var path  =
opcua.makeBrowsePath("RootFolder","/Objects/TankController");
            the_session.translateBrowsePath(path,function(err,results){
                if (!err) {
                    if (results.targets.length > 0){
                        TankCtrlNodeId = results.targets[0].targetId;
                        console.log("TankController -> nodeID is:
"+TankCtrlNodeId.value);
                    } else {
                        console.log("Cannot find TankController -
",results.toString());
                        err = new Error("cannot find TankController");
                    }
                }
                else {
                    console.log("Error:",err.toString());
                }
```

```
                callback(err);
            })
        },

/*******************************************************************************
********/
        /*Step 4 : Create subscription object and its responses to be logged in
console           */

/*******************************************************************************
********/
        function(callback) {
            the_subscription=new opcua.ClientSubscription(the_session,{
                requestedPublishingInterval: 100,
                requestedMaxKeepAliveCount:  500,
                requestedLifetimeCount:      600,
                maxNotificationsPerPublish:  1,
                publishingEnabled: true,
                priority: 10
            });
            the_subscription.on("started",function(){
                console.log("Subscription created");
                callback();
            }).on("keepalive",function(){
                console.log("Keepalive");
            }).on("terminated",function(){
                console.log(" TERMINATED ------------------------------>")
            });
        },
        /*************************************************************************/
        /*Step 5: List variables and methods available on TankController      */
        /*************************************************************************/
        function(callback) {
            the_session.browse(TankCtrlNodeId, function(err, browseResult) {
                if(!err) {
                    browseResult.references.forEach( function(reference) {
                        console.log('Element:' + reference.browseName + ', is a:' +
reference.nodeClass.key);
                    });
                }
                callback(err);
            });
        },

/*************************************************************************/
        /*Step 6: Browse node ID for methods and variables to be used in control
*/

/*************************************************************************/
        function (callback) {
            var pathSE  = opcua.makeBrowsePath(TankCtrlNodeId,".startExperiment");
            var pathGL  = opcua.makeBrowsePath(TankCtrlNodeId,".getLevel");
            var pathSV  = opcua.makeBrowsePath(TankCtrlNodeId,".setValve");
            var pathV   = opcua.makeBrowsePath(TankCtrlNodeId,".Valve");
            var pathL   = opcua.makeBrowsePath(TankCtrlNodeId,".Level");
            var errors;
            the_session.translateBrowsePath(pathSE,function(err2,results2) {
                if (!err2) {
                    if (results2.targets.length > 0) {
                        startExpeNodeId = results2.targets[0].targetId;
                        console.log("\nstartExperiment -> nodeID is:
"+startExpeNodeId);
                    } else {
                        console.log("cannot find MethodIO", results2.toString());
                        err2 = new Error(" cannot find MethodIO");
                        errors=errors+err2+", ";
                    }
                }
```

```
    });
    the_session.translateBrowsePath(pathGL,function(err3,results3) {
        if (!err3) {
            if (results3.targets.length > 0) {
                getLevelNodeId = results3.targets[0].targetId;
                console.log("getLevel -> nodeID is:   "+getLevelNodeId);
            } else {
                console.log("cannot find MethodIO", results3.toString());
                err3 = new Error(" cannot find MethodIO");
                errors=errors+err3+", ";
            }
        }
    });
    the_session.translateBrowsePath(pathSV,function(err4,results4) {
        if (!err4) {
            if (results4.targets.length > 0) {
                setValveNodeId = results4.targets[0].targetId;
                console.log("setValve -> nodeID is:    "+setValveNodeId);
            } else {
                // cannot find objectWithMethodNodeId
                console.log("cannot find MethodIO", results4.toString());
                err4 = new Error(" cannot find MethodIO");
                errors=errors+err4+", ";
            }
        }
    });
    the_session.translateBrowsePath(pathV,function(err5,results5) {
        if (!err5) {
            if (results5.targets.length > 0) {
                ValveNodeId = results5.targets[0].targetId;
                console.log("setValve -> nodeID is:    "+ValveNodeId);
            } else {
                // cannot find objectWithMethodNodeId
                console.log("cannot find MethodIO", results5.toString());
                err5 = new Error(" cannot find MethodIO");
                errors=errors+err5+", ";
            }
        }
    });
    the_session.translateBrowsePath(pathL,function(err6,results6) {
        if (!err6) {
            if (results6.targets.length > 0) {
                LevelNodeId = results6.targets[0].targetId;
                console.log("Level -> nodeID is:   "+LevelNodeId);
            } else {
                console.log("cannot find VariableIO", results6.toString());
                err6 = new Error(" cannot find VariableIO");
                errors=errors+err6+", ";
            }
        }
        callback(errors);
    });

},
/**************************************************************************/
/* Step 7 : Start experiment (call method)                                */
/**************************************************************************/
function(callback) {
    var methodToCalls = [{
        objectId: TankCtrlNodeId,
        methodId: startExpeNodeId
    }];
    the_session.call(methodToCalls, function (err, results) {
        if (!err) {
            console.log("\nExperiment started successfully");
        }
        callback(err);
    })
```

```
        }],
        /*************************************************************************/
        /* Step 8 : Start HTTP Server - UI and control function                  */
        /*************************************************************************/
        function(err) {
        if (!err) {
            startHTTPServer();
        } else {
            console.log(err);}
        });

/*******************************************************/
/* Step 9 : Setup HTTP Server                          */
/*******************************************************/
function startHTTPServer() {

    /* Step 9.1 : Link HTML to get request             */

    var app = express();
    app.get("/", function(req, res){
        res.sendFile(__dirname + "/" + "index.html");
    });
    app.use(express.static(__dirname + '/'));

    /* Step 9.2 : Setup socket.io (backend - frontend data connection) */

    var io = require('socket.io').listen(app.listen(port));
    console.log("UI on port " + port);
    io.sockets.on('connection', function (socket) {
        console.log("UI connected");
    });

    /* Step 9.3 : Create object to subscribe for Level */

    var monitoredItemLvl = the_subscription.monitor({
            nodeId: opcua.resolveNodeId(LevelNodeId),
            attributeId: opcua.AttributeIds.Value
        },
        {
            samplingInterval: 100,
            discardOldest: true,
            queueSize: 1
        }
    );

    /* Step 9.4 : Create object to subscribe for Valve */

    var monitoredItemVlv = the_subscription.monitor({
            nodeId: opcua.resolveNodeId(ValveNodeId),
            attributeId: opcua.AttributeIds.Value
        },
        {
            samplingInterval: 100,
            discardOldest: true,
            queueSize: 1
        }
    );
    console.log("-------------------------------------");

    /* Step 9.4 : Subscription handler, capture variable "Valve" every change event
*/

    monitoredItemVlv.on("changed", function (dataValueVlv) {

        /* Step 9.4.1 : Get "Valve" value */

        getV=dataValueVlv.value.value;
```

```
        /* Step 9.4.2 : Forward data to UI */

        io.sockets.emit('event', {waterL: getL,ValveL: getV});

    });

    /* Step 9.5 : Subscription handler, capture variable "Level" every change
event, executes control and set valve */

    monitoredItemLvl.on("changed", function (dataValueLvl) {

        /* Step 9.5.1 : Get "Level" value */

        getL=dataValueLvl.value.value;
        console.log("Tank level = ", (getL*10).toFixed(2), " %");

        /* Step 9.5.2 : Proportional control calculation */

        setV=kp*(1-(((tgtlvl-getL)+10)/20));

        /* Step 9.5.3 : Set new value for "Valve" calling a method */

        var methodToCalls = [{
            objectId: TankCtrlNodeId,
            methodId: setValveNodeId,
            inputArguments:[
                {
                    dataType: opcua.DataType.Float,
                    value: 0
                }
            ]
        }];
        methodToCalls[0].inputArguments[0].value=setV;
        the_session.call(methodToCalls, function (err, results) {
        });

        /* Step 9.5.4 : Forward data to UI */

        io.sockets.emit('event', {waterL: getL,ValveL: getV});

    });
}
```

Code implemented for frontend (HTML):

```
<!DOCTYPE html>
<meta charset="utf-8">
<hr>
<head>
    <title> FIS A2 </title>
    <script src="http://code.jquery.com/jquery-1.6.2.min.js"></script>
    <script src="javascript/jquery.min.js"></script>
    <link href="main.css" rel="stylesheet" type="text/css" />
    <script src='/socket.io/socket.io.js'></script>
    <script src="javascript/Chart.bundle.js"></script>
    <script src="javascript/utils.js"></script>
    <style>
        canvas {
            -moz-user-select: none;
            -webkit-user-select: none;
            -ms-user-select: none;
        }
        .chart-container {
            width: 500px;
            margin-left: 40px;
            margin-right: 40px;
            margin-bottom: 40px;
```

```
        }
        .container {
            display: flex;
            flex-direction: row;
            flex-wrap: wrap;
            justify-content: center;
        }
    </style>
</head>

<body>
    <header>
        <h1><b> ASE-9476 Factory Information Systems </b></h1>
        <h2 style="color:black"><em> Assignment 2 </em></h2>
        <h3> Tampere University of Technology </h3>
    </header> <br>
    <div>
        <h1> Tank Controller </h1>
        <h2> Node-OPC-UA Implementation </h2>
    </div>


    <script>
        var config = {
            type: 'line',
            data: {
                datasets: [{
                    label: 'Tank Level',
                    backgroundColor: window.chartColors.red,
                    borderColor: window.chartColors.red,
                    data: [ ],
                    fill: false
                }, {
                    label: 'Valve Position',
                    fill: false,
                    backgroundColor: window.chartColors.blue,
                    borderColor: window.chartColors.blue,
                    data: [ ]
                }]
            },
            options: {
                responsive: true,
                title: {
                    display: true,
                    text: 'Dynamic Historic Data'
                },
                tooltips: {
                    mode: 'index',
                    intersect: false,
                },
                hover: {
                    mode: 'nearest',
                    intersect: true
                },
                scales: {
                    xAxes: [{
                        display: true,
                        scaleLabel: {
                            display: true,
                            labelString: 'Time Stamp (UTC+3)'
                        }
                    }],
                    yAxes: [{
                        display: true,
                        scaleLabel: {
                            display: true,
                            labelString: 'Value (%)'
                        }
```

```
                    }]
                }
            }
        };

        window.onload = function() {
            var ctx = document.getElementById('canvas').getContext('2d');
            window.myLine = new Chart(ctx, config);
        };

        var socket = io.connect('http://localhost:3700');
        socket.on("connect", function() {
            //socket.emit("some event", {some: "data"});
        });
        socket.on("event", function(data1) {
            var TS = new Date();
            var year = TS.getUTCFullYear();
            var month = ((TS.getUTCMonth()+1)<10?'0':'') + (TS.getUTCMonth()+1);
            var day = ((TS.getUTCDay()+8)<10?'0':'') + (TS.getUTCDay()+8);
            var hour = ((TS.getUTCMinutes()+3)<10?'0':'') + (TS.getUTCMinutes()+3);
            var minutes = (TS.getUTCMinutes()<10?'0':'') + TS.getUTCMinutes();
            var seconds =  (TS.getUTCSeconds()<10?'0':'') + TS.getUTCSeconds();
        var CTS = year + "/" + month + "/" + day + " " + hour + ":" + minutes+":"+
seconds;
            var WL = (10*data1.waterL).toFixed(2);
            var VL = (100*data1.ValveL).toFixed(2);
            document.getElementById('waterLevel').innerHTML = WL.toString() + " %";
            document.getElementById('valvePosition').innerHTML = VL.toString() + "
%";
            if (config.data.datasets.length > 0) {
                //var month = MONTHS[config.data.labels.length % MONTHS.length];
                config.data.labels.push(CTS);
                config.data.datasets[0].data.push(WL);
                config.data.datasets[1].data.push(VL);
                window.myLine.update();
            }
        });
    </script>

    <div>
        <div class="column left" style="background-color:#aaa;">
            <h2>Current Data</h2><br>
                <div class="GeneralDiv">
                    <div class="Horiz_Div">
                        <p> Water Level  </p>
                        <div id="waterLevel">0</div>
                    </div>
                     <div class="Horiz_Div">
                        <p> Valve Position </p>
                        <div id="valvePosition">1.0</div>
                     </div>
                </div>
        </div>
        <div class="column right" style="background-color:silver;">
            <h2>Variable Monitoring</h2><br>
            <div style="width:80%;text-align: center;">
                <canvas id="canvas"></canvas>
            </div>
        </div>
    </div>
    <br><br>
    <footer>
        Developed by: Group 6 <br>
        2018
    </footer>
</body>
```

Challenges and limitations

Even the syntaxes of the OPC-UA module for Node Js is very intuitive, the object structure for data exchange was not clear until it is explored by using an external tool as OPC UA Commander or Red-Node.

Executing a controller function relying exclusively on data exchange through OPC UA variable subscription, add certain vulnerability to the system since it will depend on complex systems with multiple additional factors to fail. If the network or any of the systems suffer any lag or malfunction, control should be taken by safety functions implemented physically on the OPC UA server side, to avoid risky conditions on industrial environments (as tank overflow, leaks and other fluid containment accidents in this case).

Dependence of Node JS libraries on system behavior may affect the execution time of different cycles which may lead instability for time-critical control system.

Questions

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

The primary objective of the OPC UA address space is to provide a standard way for servers to represent objects to the client and stablish relationships between them. Object model has been designed to be defined in terms of variables and methods (similar to classes in object-oriented programming). In addition, objects can by typed, i.e. OPC UA provides a way to define and expose object types (classes with member variables and member methods) and object instances.

However, it is not necessary to use these object-oriented approaches. A simple address space model like in classic OPC Data Access can be built using folder objects and variables. Availability of enhanced object-oriented features improves the representation of object-oriented systems with OPC UA [3]. Object space is specified on the part III of IEC 62541 standards.

2. What is the difference between Object and ObjectType?

Both are node classes and have base node class attributes (NodeId, NodeClass, BrowseName, DisplayName, Description, WriteMask and UserWritemask), but Object is used to represent systems, system components, real-world objects and software objects, while ObjectType is used represents a type node for objects in the server address space. ObjectTypes are similar to classes in object-oriented languages.

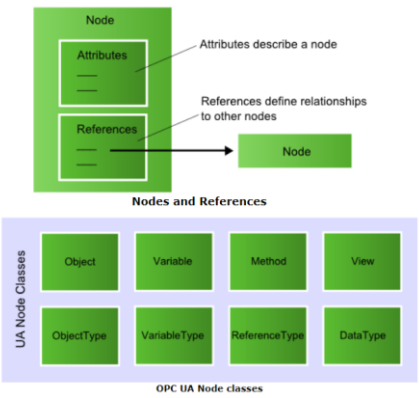

Figure 3 Object and ObjectType as node classes

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

   The transport defines different mechanisms optimized for different use cases. The first version of OPC UA is defining an optimized binary TCP protocol for high performance intranet communication as well as a mapping to accepted internet standards like Web Services, XML, and HTTP for firewall-friendly internet communication. Both transports are using the same message-based security model known from Web Services. The abstract communication model does not depend on a specific protocol mapping and allows adding new protocols in the future [3].

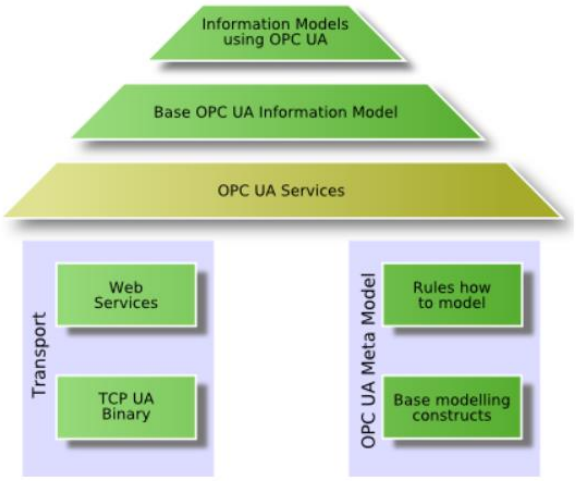

Figure 4 OPC UA foundation - Transport as a basis [3]

4. Can new transport mechanisms be added in the future?

   Yes, since the abstract communication model does not depend on a specific protocol mapping and allows adding new protocols in the future [3]. As seen on Figure 5, transport is a whole independent block, then it can be replaced.

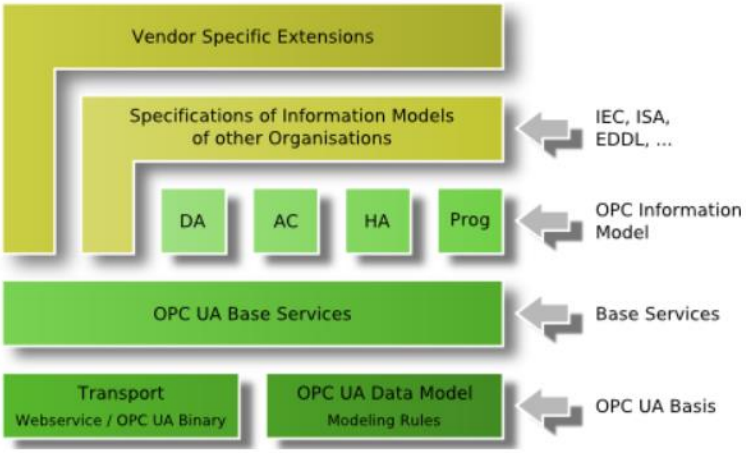

Figure 5 OPC UA Architecture [3]

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

    - Data exchange from plant floor to control level and vertical integration from control level to layers above (MES, ERP).
    - Integration of standard mobile devices to industrial applications. (HMI on handheld)
    - Distributed control
    - Improve interoperability of existent systems
    - Enable a client to access the smallest pieces of data without the need to understand the whole model exposed by complex systems.

Conclusions

- OPC UA demonstrated to be a multiplatform flexible framework to develop data exchange between IT systems, that can also be applied for basic OT operations.
- Successful integration of industrial applications to web services has reached a level where user interface applications are stable and reliable. Even the application developed was a simple example, the

References

[1] GitHub, "Node-opcua," [Online]. Available: https://github.com/node-opcua/node-opcua. [Accessed 13 April 2018].

[2] ChartJs, "ChartJs Open sourche HTML5 Charts," [Online]. Available: https://www.chartjs.org/. [Accessed 13 April 2018].

[3] Unified automation, "Embedded OPC UA Stack 1.0.0.125," [Online]. Available: http://documentation.unified-automation.com/uasdkhp/1.0.0/html/index.html. [Accessed 13 April 2018].