A Web-based Platform for OPC UA integration in IIoT environment

Salvatore Cavalieri, Damiano Di Stefano, Marco Giuseppe Salafia, Marco Stefano Scroppo
University of Catania, Department of Electrical Electronic and Computer Engineering
Viale A.Doria, 6 – 95125 Catania (ITALY)
salvatore.cavalieri@unict.it, damiano.di.stefano@gmail.com, marco.salafia@gmail.com , marcostefano.scroppo@dieei.unict.it

***Abstract*—The paper presents a web-based platform able to offer access to OPC UA Servers. The proposed platform may be used by web-users to exchange data with OPC UA Server without any knowledge of the standard. The software solution described in the paper is available on GitHub.**

Keywords—IIoT; Web-based application; OPC UA; interoperability

I. INTRODUCTION

A common factory automation scenario features different levels of controls. At the lowest level, a large number of field devices (e.g., sensors and PLC) collect data coming from the industrial process. Upper levels use this data to improve the process and to make intelligent business decisions. In order to provide interoperable data exchange inside and between these levels, communication standards have been defined; among them, there is the OPC UA (i.e., IEC 62541) [1].

During the last decade, software development has seen a shift towards (mobile) web applications. These applications do not feature particular requirements except that to use web-based technologies for the communication. In the following, the term "web user" will be used to identify the entity running these application; it may be represented, for instance, by a human connected by a web browser, or a smart sensor/actuator connected through web-based technologies. Web-based applications have shown many advantages over native applications, including straightforward cross platform support, ease of development, faster development cycles and improved portability [2]. Web-based applications are increasingly hosted in computing clouds, which offer higher scalability, reliability and flexibility compared to hosting the applications on dedicated servers. All in all, the developments of web-based industrial applications in cloud computing are paving the road for the highly anticipated Industrial Internet of Things (IIoT).

Taking into account that OPC UA currently represents one of the most used communication standards into the factory automation, integration of web-based industrial applications with OPC UA may represent an important contribution to realise horizontal and vertical integration between control levels of factory automation. OPC UA is a very powerful communication standard, but it is featured by a great software complexity which makes very simple devices unable to use it. On the other hand, simple devices (e.g., smart sensors) may easily support basic web-based technologies.

Offering communication services of an OPC UA Server through web-based technologies, means to allow every "web user" to retrieve information from an OPC UA Server without any knowledge of the OPC UA communication standard. This is a key concept on which the paper is based. The paper presents a web-based platform able to offer to a generic web-user the access to one or more OPC UA Servers. The proposed platform allows web users to exchange data with OPC UA Server without any knowledge of the standard; the only requirements is the support of basic web-based technologies.

Current literature presents solutions that provide OPC UA connectivity from a web device; they may be classified into three main categories: pre-rendered user interfaces [3][4], web APIs [5][6], and native stacks [7][8]. The proposal here presented may be classified as a web API solution, as it will be shown in the remainder of the paper.

The main differences between the proposal and the solutions available in literature is that all these solutions require, on the front-end side, the knowledge of the communication services and the data modelling provided by OPC UA standard. In all of the three categories of solutions just presented, each request performed by the user at the front-end must be done respecting the OPC UA specifications. In our solution, user at the front-end may have no knowledge of OPC UA standard, as the information flow with the proposed web-based platform does not contain any command and/or parameter strictly linked to the OPC UA standard. User is requested only to have knowledge of general concepts concerning industrial automation systems.

The web-based platform presented in the paper has been implemented and the relevant code is available at [9].

II. OPC UA OVERVIEW

OPC UA allows applications to exchange information using a client/server model. Inside an OPC UA Server, OPC UA Nodes are used to represent any kind of information, including variable instances and data types. The set of Nodes inside an OPC UA Server is called AddressSpace. Each Node belongs to different NodeClasses among which: Variable (representing physical values of the process under control), Object (representing real-world entities like system components, hardware and software components). Moreover, Nodes modelling types are also present in the AddressSpace, among which: VariableType (used to provide type definition for variables), DataType (used to provide type definitions for the values of variables), ReferenceType (used to define references,

978-1-5090-6505-9/17/$31.00 ©2017 IEEE

which will be described in the following) and ObjectType (holding type definition for Objects).

In particular, DataType may be Built-in or Structured. Built-in DataTypes provide base types like Int32, Boolean, Double; OPC UA specification defines standard data types specifying the encoding/decoding rules corresponding to the Built-in DataTypes. Structured DataTypes allow definition of vendor-specific data types; they are generally made up by a structure that may contain other Structured and/or Built-in DataTypes. For each Structured DataType, different encodings/decodings may be available; the default one is the DefaultBinary encoding. The description of each encoding/decoding for a particular Structured DataType is given inside a dictionary in the OPC UA AddressSpace. The dictionary is made up by several entries called StructuredTypes, that define the encoding/decoding for the Structured DataTypes defined in the OPC UA Server. Each StructuredType contains several Field elements. Each Field refers to a component of the Structured DataType. In particular, it features a TypeName attribute that describes the encoding/decoding of the component.

Particular relationships may be defined from one Node to another; they are called References. They may be classified in different ways, among which: hierarchical and non-hierarchical. The first type is used to realise a hierarchy between Objects and Variables Nodes (e.g., a folder containing Variables).

OPC UA AddressSpace is made up by different subsets of Nodes, called Information Models, each of which features an integer index. Information Model relevant to index 0 is mandatory and is relevant to native OPC UA Information Model. Each Node inside an Information Model is univocally identified by a couple (NodeId, NamespaceIndex), where NodeId is a Node identifier and NamespaceIndex is the index of the Information Model.

To promote interoperability of Clients and Servers, the OPC UA specification defines particular Nodes which may be considered as the entry points of the AddressSpace; starting from one of these Nodes, a client may navigate among all the other Nodes following the References between Nodes, jumping from one Information Model to another one. In particular, *Identifiers.Objects* is the entry point for the entire set of Objects and Variables. *Identifiers.Types* is the entry point of the Nodes defying the types; in particular, it allows to reach the definition of the Structured DataTypes, i.e. vendor-defined ones.

Nodes are accessible by OPC UA Clients using OPC UA Services inside a Session that must be previously created [1]. The simplest way for a Client and Server to exchange data is using the Read and Write OPC UA services. Subscriptions and MonitoredItems [1] allow a client to receive cyclic updates of variable values, object attributes and events.

A Subscription is the context needed to realise this cyclic exchange. MonitoredItems can be created in a Subscription by the OPC UA Client. Different types of MonitoredItems may be defined; among them, there is a type used to subscribe for data changes of Variable Values. MonitoredItems have common settings among which there are the *Sampling interval* and the *Threshold value*. The Sampling interval defines the rate at which the server checks for changes in the Variable values, for example. In this case, the Threshold is used for detecting changes in the Variable values. Each MonitoredItem produces particular message, called Notifications, at the frequency given by the Sampling interval; these messages are put in a queue inside each MonitoredItem. A Subscription features a Publish Interval, which defines the interval at which the server clears the MonitoredItem queues and conveys their contents (i.e., Notifications) into a NotificationMessage to be sent to the OPC UA Client. Transmission of NotificationMessages by OPC UA Server is triggered by Publish requests sent by client [1][11]. The server enqueues the Publish requests until a NotificationMessage is ready (according to the Publish Interval, as said before). When this occurs, the NotificationMessage is sent back to the client through a Publish response. For each Publish request sent by the OPC UA Client, exactly one NotificationMessage is transmitted by the OPC UA Server through a Publish response.

OPC UA gives a lot of importance to the secure communication between OPC UA Client and Server. Sessions must be created on the top of a Secure Channel, which may be featured by different levels of security, through the choice of the so-called endpoints [1][10].

III. WEB-BASED TECHNOLOGIES

The aim of this section is to point out the web-based technologies adopted in the proposal here presented.

The proposed platform offers Web Service based on *REpresentational State Transfer* or REST (called RESTful Web Service Services) [12].

Token based authentication has been adopted. A web user accessing services offered by the proposed platform provides a token in each HTTP request header. Secure transmission of the token is achieved by the HTTPS [13].

Asynchronous communication may be realised between the web user and the platform; it has been achieved by the Publisher Subscriber Pattern [14][15], featuring the presence of three entities: Publishers and Subscribers and a Broker. This last accepts the messages published by Publishers and forward event notifications about it to the right Subscribers.

IV. OPC UA WEB PLATFORM ARCHITECTURE

Figure 1 shows the OPC UA Web Platform, made up by two main modules: Web Service Interface module and a Middleware module. The platform may use one or more Publisher/Subscriber Brokers, needed to handle asynchronous services described in the following.

The Web User in Figure 1 is a client of the OPC UA Web Platform. From the OPC UA Servers point of view, the OPC UA Web Platform is an OPC UA Client. In particular, the Middleware is made up at least by an OPC UA Client connected to the available OPC UA Servers. This OPC UA client is used for the access to the OPC UA Servers on behalf of each web user. For each HTTP request coming from an user to the Web Service Interface module, the Middleware performs the operations needed to fulfil user's request; operations are done through the OPC UA Client. The Web User will receive a HTTP response containing the result of the operation requested.

Among HTTP requests made by the Web User, those requesting asynchronous services has been foreseen; in this case, a mechanism based on Publisher Subscriber Pattern will be used by the OPC UA Web Platform, in order to fulfil user's request.

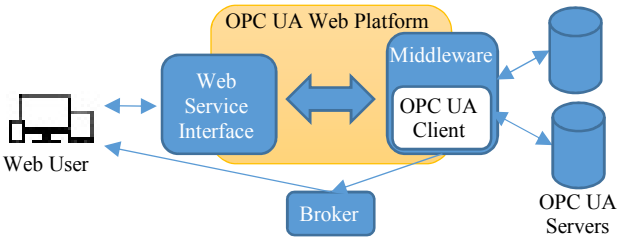

Fig.1. OPC UA Web Platform

The following assumptions have been made.

Web User must use the web-based software technologies described in the previous section.

The OPC UA Web Platform has been conceived to allow Web Users to retrieve the value attribute of each OPC UA Variable maintained by an OPC UA Server. OPC UA Variables are often linked to OPC UA Objects in the AddressSpace of a generic OPC UA Server (e.g., folder objects organising OPC UA Variables, data types defining the type of the value attribute of the OPC UA Variable). For this reason, platform also allows the web user to retrieve general information about OPC UA Objects linked to OPC UA Variables.

Each user needing to use the OPC UA Web Platform must be previously registered. During registration, user credentials in terms of username and password must be stored in the OPC UA Web Platform.

The Platform must have the grants to access the OPC UA Servers on behalf of web users. Typically grants means holding a couple of username/password, or a X509 certificate recognised by an OPC UA Server.

Secure communication is realised between the OPC UA Web Platform and each OPC UA Server, through the use of OPC UA secure mechanisms; the OPC UA Web Platform will try to realise with OPC UA Server the most secure communication as possible.

V. OPC UA Web Platform Services

The authors identified the following basic services offered by the OPC UA Web Platform.

A. *SecureAccess Service*

In order to guarantee security between OPC UA Web Platform and Web User an ad-hoc service, named *SecureAccess*, has been defined. Web User uses the *SecureAccess* request to be authenticated, specifying his credentials stored in the platform during registration phase. The OPC UA Platform validates the user credentials and generates a signed token for the user, which is returned to the user through the *SecureAccess* response. An encryption mechanism based on HTTPS allows a secure transmission of the token. The Web User will use the signed token received, in each next service request issued to the OPC UA Web Platform (including it in the HTTP header).

B. *GetDataSources Service*

GetDataSource service allows a Web User to request information about available data sources (i.e., the OPC UA Servers). It has been assumed to identify each OPC UA Server by a *ServerURL*, made up by a string containing the URL of the server.

For each *GetDataSources* request made by the user, the OPC UA Web Platform will invoke the OPC UA FindServers service [1] to achieve the list of the available OPC UA Servers. The Web User will receive a list of data sources identified by a *ServerURL*. In each following service request, the user will indicate the *ServerURL*, allowing to perform the requested operations on the relevant AddressSpace.

C. *ReadInfo Service*

ReadInfo service gives to the user a description of a set of OPC UA Objects and Variables. It has been assumed to identify each OPC UA Node by an *ID*, made up by a string containing the concatenation between the OPC UA NodeId and NamespaceIndex.

ReadInfo service request features the following parameters:

- *ServerURL:* it is the identifier of the data source.
- *StartingNode:* it is the *ID* related to the OPC UA Node from which information should be retrieved to the user. It is optional.
- *Reserialize:* it is a boolean allowing the user to request the automatic decoding procedure of Structured DataType, as described in the remainder of the paper.

ReadInfo service can be used into two different way, according to its parameters. If *ReadInfo* service request is called without *StartingNode* parameter, the OPC UA Web Platform accesses the OPC UA AddressSpace of the requested OPC UA Server starting from the Identifiers.Object root node. Otherwise, the OPC UA Web Platform accesses the OPC UA AddressSpace starting from the OPC UA Node relevant to the *StartingNode* parameter. In both cases, when the user invokes the *ReadInfo* service request, the OPC UA Web Platform builds a temporary graph through a cut of the OPC UA AddressSpace with depth one starting from the starting node (i.e., the Identifiers.Object root node or the node related to the *StartingNode* parameter), as shown by Figure 2.

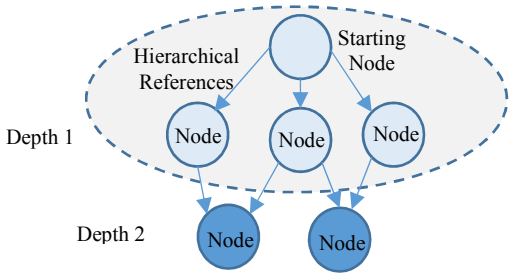

Fig.2. Graph realised cutting OPC UA AddressSpace

As the OPC UA Web Platform was mainly aimed to allow user to retrieve information about Objects and Variables, the graph depicted by Figure 2 is realised including all the OPC UA

Nodes connected to the starting node by the hierarchical references, as they are only used for OPC UA Objects and Variables. Information relevant to OPC UA Nodes contained in this graph are passed to the user.

In particular, for each OPC UA Variable contained in the graph, information passed to the user by the platform is limited to the following attributes:

- *ID*: it is the ID of the OPC UA Node in the OPC UA Web Platform.
- *Name*: it is a string containing the DisplayName attribute of the OPC UA Node (it is a localised text specifying the name of the Node [1][16]).
- *Value*: it contains the current value attribute of the OPC UA Variable [1][16].
- *ValueTypeSchema*: it contains the JSON-Schema [17] defining the structure of the *Value*, explaining the Web User how the Value must be read.
- *Monitorable:* a Boolean attribute that indicates if the OPC UA Variable can be monitored for changes in value.
- *MinSamplingInterval*: this attribute represents the minimum sampling interval for the monitoring of the OPC UA Variable on value changes.

In the case of OPC UA Object, only *ID* and *Name* are given to the User.

Information about Nodes contained in the graph are passed to the user as a JSON string, inside the *ReadInfo* response. It has been assumed that the JSON string must be compliant to the JSON-Schema shown Figure 3; as shown, it is made up by two properties named *StartingNode* and *LinkedNodes*.

```
{
 "$schema": "http://json-schema.org/draft-04/schema#",
 "type": "object",
 "properties": {
     "StartingNode": {
          "type": "object",
          "properties": {
               "ID": {"type": "string"},
                "Name": {"type": "string"},
                "Value": {"type": "object"},
                "ValueTypeSchema": {"type": "string"},
                "Monitorable": {"type": "boolean"},
                "MinSamplingInterval": {"type": "number"}
          },
          "required": ["ID", "Name"],
          "additionalProperties": false
     },
     "LinkedNodes": {
          "type": "object",
          "properties": {
               "ID": {"type": "string"},
               "Name": {"type": "string"}
          },
          "additionalProperties": false
     }
 },
 "required": ["StartingNode", "LinkedNodes"],
 "additionalProperties": false
}
```

Fig.3. JSON-Schema for node description

StartingNode object contains the representation of the OPC UA starting node of the graph; it is made up by *ID*, *Name, Value*, *ValueTypeSchema*, *Monitorable*, *MinSamplingInterval* in the case of OPC UA Variable and by *ID* and *Name* in the case of OPC UA Object. The *LinkedNodes* object contains a list of representations of each OPC UA Node connected to the starting node by a hierarchical reference; for each node only *ID* and *Name* is given.

In the case that OPC UA Variable value belongs to a Built-in DataType, OPC UA Web Platform must find the JSON base type [19] corresponding to the standard data types relevant to the Built-in DataType. It has been verified that for each OPC UA Built-in data type a JSON base type may be found. In this case, the JSON string sent inside the *Value* property contains the real value of the OPC UA Variable; it has been assumed to write this string according to the JSON Schema shown in Figure 4. As said before, this schema is inserted into the *ValueTypeSchema*. The value "XXX" is replaced by the JSON base type corresponding to the OPC Built-in DataType.

```
{
 "$schema": "http://json-schema.org/draft-04/schema#",
 "type": "XXX"
}
```

Fig.4. JSON-schema for Built-in DataType Value description

It may happens that an OPC UA Server exposes Variables with value attribute belonging to OPC UA Structured DataTypes [18]. As said in Section II, it is required that an OPC UA Server maintains the definition of each Structured DataType inside the AddressSpace, together with the information about how to encode and decode it. This last information is maintained in a dictionary.

As it has been assumed that user has no knowledge about OPC UA, he cannot decode OPC UA Structured DataTypes; for this reason, a solution for the transmission of this kind of data to Web User was defined. For each Structured DataType found in an OPC UA Server, the OPC UA Web Platform uses a parser mechanism that allows its decoding in order to present complex data to the user in an understandable format, based on JSON [19]. In any case, it has been foreseen the possibility that an user may implement the parser functions inside. In this case, the OPC UA Web Platform provides a JavaScript file to the user, containing the procedures which can be used by user to decode the stream of complex data. On submission of the *ReadInfo* request, user will specify to the OPC UA Web Platform which of the two choices he prefers; this is realised through the *Reserialize* parameter present in the *ReadInfo* request. In the following, only the parsing procedure realised by the OPC UA Platform will be described.

In case of Structured DataType, the OPC UA Web Platform uses the relevant information contained in the dictionary to build the two JSON strings used to fill the *Value* and *ValueTypeSchema* properties of the JSON Schema in Figure 3, respectivelly.

The *Value* property will be a JSON object containing several properties, one for each Field elements of the StructuredType entry in the dictionary. If the Field refers to a Built-in DataType, the property is achieved decoding the encoded stream received by the OPC UA Server, according to the standard data types relevant to the Built-in DataType. If the Field refers to a

StructuredType a nested JSON object is inserted, recursively applying the above described procedure.

The *ValueTypeSchema* property will be a JSON-Schema describing how the *Value* property is made. Figure 5 shows an example of the JSON-Schema corresponding to a StructuredType. It contains several properties, one for each Field elements of the StructuredType entry, as explained above. For example, the "AAA" value corresponds to the Name of a Field whose TypeName refers to a standard data types. In this case, the value "XXX" is filled by the corresponding JSON base type. In the case of a Field whose TypeName refers to a StructuredType, a nested JSON Object representing it is inserted. The JSON Object shown in Figure 5 presents a property "BBB" representing the name of a StructureType Field. As shown, the property is structured as another JSON Object is present modelling the StructuredType in term of Fields.

```
{
 "$schema": "http://json-schema.org/draft-04/schema#",
 "type": "object",
 "properties": {
     "AAA": {"type": "XXX"},
     ……
     "BBB": {
             "type": "object",
             "properties": {
                     "CCC": {"type": "XXX"},
                     ……
             }
     }
 }
}
```

Fig.5. JSON-schema for Structured DataType Value description

The *ReadInfo* is realised by the OPC UA Web Platform by OPC UA Browse and Read Services [1][10], as shown by Figure 6. The OPC UA Web Platform checks if an OPC UA Client instance exists. If this does not occur, the instance is made and it will be used for all the Web Users requesting services from the platform. Then, the platform checks for the existence of a Session for the ServerURL specified by the Web User. If the OPC UA Session does not exist, it is created. In particular, the OPC UA Web Platform creates a SecureChannel adopting the best security options allowable, using the OPC UA GetEndpoint and OPC UA CreateSecureChannel services [11], as requested by OPC UA specification. Then, an OPC UA Session is created and activated through the OPC UA CreateSession and ActivateSession services [11].

Coming back to the *ReadInfo* Service, after the check about existence of OPC UA Client and OPC UA Session has been completed, the OPC UA Web Platform will invoke the OPC UA services shown in Figure 5, i.e. Read and Browse, in order to build the graph shown by Figure 2.

OPC UA Read request allows the platform to retrieve the full set of information about the (default or explicit) *StartingNode* received in the ReadInfo request. OPC UA Browse request allows to retrieve the description of the references contained in the *StartingNode*. For each hierarchical reference, the Browse service allows also to retrieve the *ID* and Name of the target node pointed by the *StartingNode*.

Information collected by OPC UA Read and Browse services are used to create the JSON string according to the schema shown by Figure 3.

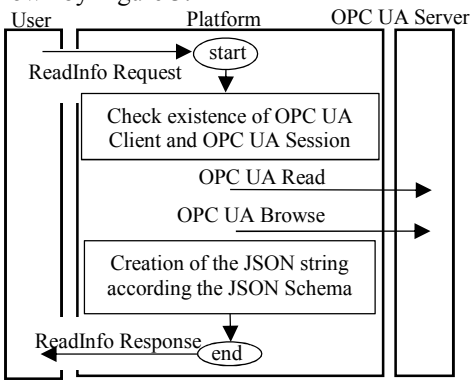

Fig.6. ReadInfo Services

D. WriteInfo Service

The *WriteInfo* Service needs the following parameters: *ServerURL*, *ID*, *ValueToWrite* (i.e., the value to write in the node specified by *ID*). After the check of existence of a OPC UA Client and the OPC UA Session (as seen before for the *ReadInfo* request), the platform performs the OPC UA Write Service using the parameters provided in the *WriteInfo* service request.

E. Monitor Service

Web User may be interested to automatically monitor the changes on the value *attribute* of one or more OPC UA Variables. To this aim, he calls a Monitor service, providing for:

- *ServerURL*: it allows to identify the OPC UA Server.
- *List of:*
  - *ID*: it specifies the variables to monitor.
  - *SamplingInterval*: for each variable, it specifies the desired sampling period
  - *ThresholdValue*: for each variable, it allows to specify the value of the threshold used for the monitoring.
  - *Reserialize:* as defined for the *ReadInfo*.
- *Topic*: it allows to specify the topic that user has created inside the Broker to be used for the publication of information produced by through the monitoring of the list of nodes specified in the request.

Choice of the OPC UA Nodes to monitor is made by the Web User on the basis of information received through the *ReadInfo* service; only Variable featuring *Monitorable* parameter set to true can be monitored and passed to the *Monitor* service request.

The *Monitor* service is based on the use of the OPC UA CreateSubscription and CreateMonitoredItems services [1][11], according to the procedure shown by Figure 7.

After having checked the existence of OPC UA Client and OPC UA Session, the OPC UA Web Platform analyses all the current Subscriptions created for that Session, with the aim to find a Subscription associated to the specified topic. If no Subscription with this feature exists, it is created using the OPC UA Create Subscription request. Inside the Subscription, a list of OPC UA MonitoredItems is created using the OPC UA Create MonitoredItem requests. Each OPC UA MonitoredItem will be associated to the OPC UA Node relevant to the *ID* passed by the

user. Furthermore, the *SamplingInterval* and *ThresholdValue* values passed through the Monitor service request, are assigned to this MonitoredItem. Once MonitoredItems have been created, Monitor service response is sent to the User. Choice of the PublishInterval of the Subscription is made according to the SamplingInterval values (e.g., choosing the lowest value). If Subscription associated to the topic exists, the MonitoredItems are added to it.

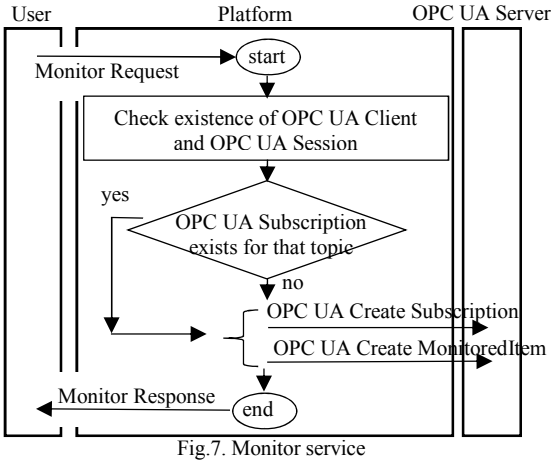

Fig.7. Monitor service

Section II explained that each monitored value produced by a MonitoredItem is put into a NotificationMessage, which is received by an OPC UA Client using the OPC UA Publish service. Figure 8 shows the synchronous exchange of Publish service between the OPC UA Web Platform and the OPC UA Server. For each OPC UA Publish response (containing a NotificationMessage) received from the OPC UA Server by the OPC UA Web Platform, a JSON-based string made up by the *ID* of the monitored variable and the relevant value is sent to the Broker, to which user has previously subscribed, on the topic chosen by the user. User will be notified by the Broker about the information just received, on the same topic.

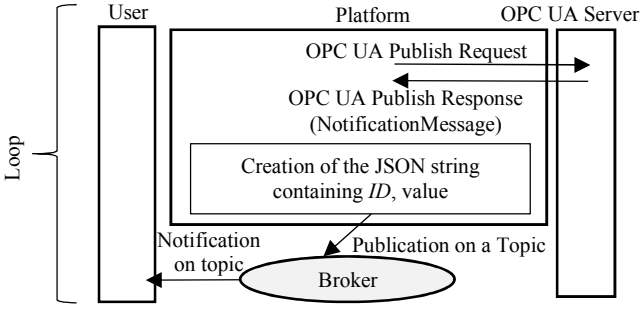

Fig.8. Publishing Procedure

Each value transmitted inside the JSON string may undergo a decoding procedure in the case the value monitored belongs to a Structured DataType maintained by the OPC UA Server and the user has specified that this decoding must be performed by the platform (through the *Reserialize* parameter).

VI. Final Remarks

The OPC UA Web Platform described in the paper has been developed and it is available on GitHub [9]. The authors chose RESTFul Web Service to implement Web Service Interface. It is made up by a Microsoft cross platform .NetCore WebAPI [20]. For the authentication and authorization, JSON Web Tokens (JWT) standard was the solution adopted [21]. The JWT mechanisms was developed through the use of libraries provided by Microsoft in the .NetCore Framework. Encryption was realised through HTTPS protocol [13]. Implementation of OPC UA Client was based on the OPC UA cross platform .NETCore stack available on GitHub [22]. The Publisher Subscriber technologies used for the asynchronous communication through the Broker shown by Figures 1,8 are both Microsoft SignalR and MOSCA MQTT [23][24].

References

[1] W.Mahnke, S.H.Leitner, M.Damm, "*OPC Unified Architecture",* Springer Verlag, 2009

[2] H. Heitkötter, S. Hanschke, T. Majchrzak, "Comparing cross-platform development approaches for mobile applications", In WEBIST 2012 - Proceedings of the 8th International Conference on Web Information Systems and Technologies, Porto, Portugal, 18 – 21 April, 2012 (2012), pp. 299-311.

[3] Groov, 2014. http://groov.com/

[4] Atvise - HMI and SCADA in pure web technology, 2014. http://www.atvise.com/

[5] HyperUA - Projexsys, 2014. http://projexsys.com/hyperua/

[6] Tatu Paronen, Aalto University, School of Science, Finland "A web-based monitoring system for the Industrial Internet" Espoo April 13, 2015.

[7] S.Hennig , A.Braune, M.Damm, "JasUA: A JavaScript stack enabling web browsers to support OPC Unified Architecture's binary mapping natively", Proceeding of ETFA, 2010, pp.1-4.

[8] E.Rossignon, "Node OPC UA", 2014, http://github.com/erossignon/node-opcua

[9] OPC UA Web Platform, 2017. https://github.com/OPCUAUniCT/OPCUAWebPlatformUniCT

[10] OPCFoundation, OPC UA Part 2: Security Model Specification, release 1.03, 2015.

[11] OPCFoundation, OPC UA Part 4: Services Specification, release 1.03, 2015.

[12] Roy Thomas Fielding, "Architectural Styles and the Design of Network-based Software Architectures", Dissertation for Ph.D., 2000, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[13] E.Rescorla, "RFC2818: HTTP Over TLS", The Internet Engineering Task Force https://tools.ietf.org/html/rfc2818, May 2000.

[14] R.Baldoni, M. Contenti and A. Virgillito, "The Evolution of Publish/Subscribe Communication Systems", Future Directions of Distributed Computing, Springer Verlag LNCS Vol. 2584, 2003.

[15] MQTT Version 3.1.1. Edited by Andrew Banks and Rahul Gupta. 29 September 2014. OASIS Standard. Latest version: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.

[16] OPCFoundation, OPC UA Part 3: Address Space Model Specification, release 1.03, 2015.

[17] JSON Schema: A Media Type for Describing JSON Documents, https://tools.ietf.org/html/draft-wright-json-schema-01

[18] OPCFoundation, OPC UA Part 5: Information Model Specification, release 1.03, 2015.

[19] JSON Data Interchange Format, RFC 7159, https://tools.ietf.org/html/rfc7159

[20] C.Nagel, "Professional C# 6 and .NET Core 1.0", Wrox, 2016

[21] JSON Web Token, RFC 7519, https://tools.ietf.org/html/rfc7519

[22] OPC UA stack mantained at GitHub, https://github.com/OPCFoundation/UA-.NETStandardLibrary

[23] K. Nayyeri, D. White, "Pro ASP.NET SignalR -Real-Time Communication in .NET with SignalR 2.1", Apress, 2014.

[24] MOSCA MQTT source code mantained at GitHub, https://github.com/mcollina/mosca