Proceedings of the ASME 2019 International Manufacturing Science and Engineering Conference
MSEC2019
June 10 – June 14, 2019, Erie, Pennsylvania, USA

MSEC2019-2748

TESTING OF THE MTCONNECT – OPC-UA COMPANION SPECIFICATION

Ryan Fisher
Department of Aerospace Engineering,
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061 U.S.A.

Guodong Shao
Systems Integration Division
National Institute of Standards and Technology
Gaithersburg, Maryland 20899 U.S.A.

KEYWORDS
Companion Specification; Interoperability; MTConnect; OPC-UA; Testing.

ABSTRACT
Smart Manufacturing (SM) is the future of the manufacturing industry. Seamless, accurate, and fast connection and communications among devices are critical for SM. By leveraging information technologies, devices can dynamically communicate with each other to increase factory production, while decreasing engineering costs. MTConnect and Open Platform Communications - Unified Architecture (OPC-UA) standards facilitate such communication. MTConnect is a manufacturing interoperability standard that provides a semantic vocabulary for manufacturing equipment to provide structured contextualized data with no proprietary format. The OPC-UA is a platform-independent standard through which various systems and devices can communicate by sending messages between clients and servers over various networks. OPC-UA enables syntactic interoperability between clients and servers. The MTConnect - OPC-UA Companion Specification integrates the two standards to provide manufacturers more efficient and powerful interoperability capabilities. In this paper, we report the test of version 1.02 of this companion specification. This specification sets a standard means of communication between MTConnect devices and OPC-UA Clients/Servers based on Extensible Markup Language (XML) structures. To test the standard, the following components have been developed: an OPC-UA Server, an OPC-UA Client, a probe that translates data structures in MTConnect XML format to MTConnect OPC-UA Companion XML format that can be recognized by the server, a MTConnect XML data parser, and a MTConnect device simulator. The activities of the standard testing include passing varying data structures and objects through the server and confirming the information is received accurately by the client. The findings of the standard testing will be provided to the standard developing organizations for improving the future versions of the standard.

INTRODUCTION
By increasing the availability of information, the processes of product design, manufacturing, and quality control can all be improved. Now, more than ever, there is a demand for larger amounts of real-time information on manufacturing floors of every scale. The dynamic information enables manufacturers to have better awareness and control over their machines and processes. The information is generated and communicated through the connection and integration of machines with other devices, systems, and applications (e.g., controllers, simulators, or Graphical User Interface (GUI)).

MTConnect [1] and the Open Platform Communications (OPC) are two standards that help achieve these goals in the manufacturing industry. MTConnect provides semantic vocabularies for manufacturing capabilities including, but not limited to, device monitoring, automation, and process analytics. The OPC Foundation has created the OPC-Unified Architecture (UA) [2] standard that supports object-oriented implementations that can handle information such as alarms and events, commands, real-time data, and complex data in Extensible Markup Language (XML). MTConnect and the OPC Foundation have partnered to develop the MTConnect - OPC-UA Companion Specification [3] to set a standard means of communication (a gateway) between MTConnect devices, servers, or agents, and OPC-UA servers and clients. Testing the MTConnect - OPC-UA Companion Specification could demonstrate the usability and feasibility of the standard for the manufacturing industry while identifying issues in the current version of the specification can provide feedback to the two standard developing organizations (SDOs), i.e., MTConnect and OPC Foundation, for improvements in future versions. Previous research conducted on manufacturing data integration claims

that the integration of many devices with the companion standard in a factory-wide network is easier compared to small scale implementations; small scales were more efficient with the sole use of MTConnect [4]. The testing in this study may confirm such claims, however this is not a formal requirements-driven verification and validation (V&V) activity. It is a preliminary specification conformance testing by comparing the outputs of MTConect machine and the OPC-UA client. Conducting such a test requires a cross-platform testing environment that allows a variety of test cases to be performed. The cross-platform capability is essential to the OPC-UA components. The test cases defined will represent a variety of data types and data items to highlight key aspects of the specification including Data Access, Historical Access, and Event and Alarm Notifiers.

This paper describes the development of the testing environment and the testing components including an OPC-UA server, an OPC-UA client, an MTConnect probe, an MTConnect data parser, a MTConnect device simulator, and test cases. These components can also be reused for testing future version of companion specification. The test cases are typically applicable to all versions, as the MTConnect data model will not change.
The organization of this paper is as follows. The relevant standards, i.e., MTConnect, OPC-UA, and the MTConnect - OPC-UA companion specification, are introduced first. The next Section explains the methodology of testing the companion specification and discusses how the testing environment and its components were established and challenges that were encountered during the testing. The three key features of OPC-UA are also discussed. Then, the following Section identifies test cases and provides a justification for such test cases. After that, the next Section presents and discusses the testing results of the companion specification. The final Section provides a conclusion and discusses the future work.

RELEVEANT STANDARDS

MTConnect and OPC-UA standards are two interoperability standards that facilitate communication among smart devices in manufacturing domain. Integrating the two standards provides manufacturing companies powerful interoperability capabilities. Each of the standards is briefly discussed as follows.

The MTConnect standard provides a semantic vocabulary for manufacturing equipment to generate structured, contextualized data with no proprietary format. With uniform data, users can focus on manufacturing applications rather than translation. MTConnect data sources include production equipment, sensor packages, and other factory floor hardware. MTConnect's Extensible Markup Language (XML) data format provides both human and machine-readable features. MTConnect is extensible and can be integrated with other standards by design, which facilitate the integration with OPC-UA [1].

OPC-UA is a platform-independent service-oriented architecture that integrates all the functionality of the individual OPC Classic specifications into one extensible framework. It provides the equivalent functions of the original OPC Classic, while extending the object-oriented capabilities to complex and multi-level structures, but with more functionality: on-demand data access, data subscriptions, event notifiers, method executions, and server discovery. Through OPC-UA, various systems and devices can communicate by sending messages between clients and servers over various networks. OPC-UA enables syntactic interoperability between clients and servers. The communication is provided on a safe and secure network by 128-bit or 256-bit encryption levels, message signing, user/system auditing, and authentication [2]. OPC-UA uses a predefined semantic vocabulary represented in XML to provide a descriptive schema of how items are in general mapped to the object-oriented model.

The combination of MTConnect and OPC-UA into a companion specification provides a powerful means of connecting MTConnect-enabled machining tools to OPC-UA servers and clients. As well, the inverse can be stated; the companion specification gateway allows for integration between OPC-UA servers and MTConnect applications. We will connect MTConnect machine tools to OPC-UA servers and clients, as shown in Figure 1 from Companion Specification Version 1.02 [3]. A device such as a milling machine is linked directly to an MTConnect server. Using the MTConnect to OPC-UA Gateway, a connection can be established between the OPC-UA client and the MTConnect server, allowing for information transfer over the network.

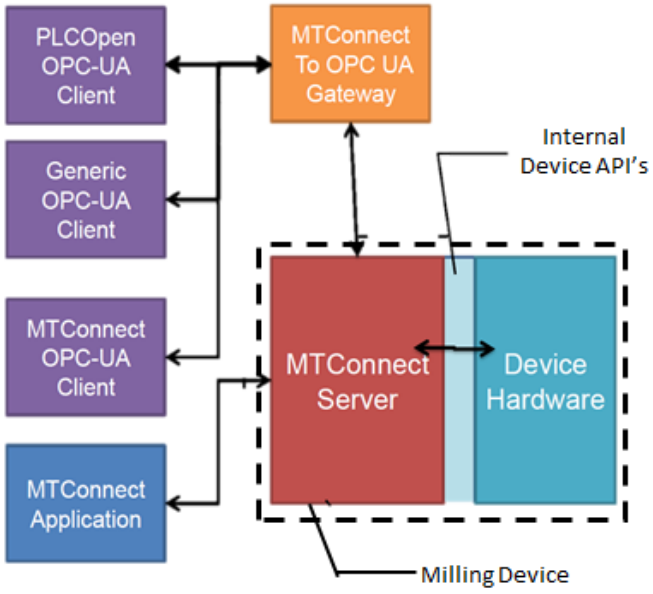

Figure 1. MTConnect to OPC-UA Gateway Implementation [3]

TESTING ENVIRONMENT IMPLEMENTATION

To begin testing the MTConnect - OPC-UA Companion Specification, a testing environment including a server/client setup must be established. Figure 2 shows an information flow among the key components in the testing environment. MTConnect data is sent to the OPC-UA server via an XML parser and probe. The OPC-UA client retrieves the data from the server and sends the data to a GUI for visualization. The XML parser and probe also provides the data to the GUI for visualization and data comparison with the client output.

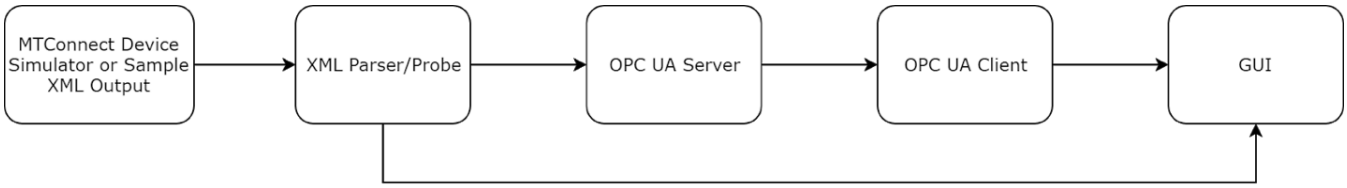

Figure 2. Companion Specification Implementation

MTConnect Device Simulator

There are several options for obtaining MTConnect data for a testing environment. For example, it can be real-time data from the MTConnect devices, data from online agent simulators, or data from software driven simulation agents. We have used the online agent simulator to retrieve device data. The online agent simulator is provided by the National Institute of Standard and Technology (NIST) and MemexOEE [5]. These sites provide the standard MTConnect XML files that contain a schema, a probe, real-time streams, and samples. The schema and probe have different XML structures compared to the stream file: the schema or probe file is required for the XML probe to generates nodes, and the stream is required for the parser to obtain data.

MTConnect Data Parser

Establishing MTConnect devices within an OPC-UA server requires the server to identify the structure of the device. For example, the server must know the component names such as Axes, and how many subcomponents each component contains, e.g., the Axes may contain Linear X Axis, Linear Z Axis, and Rotary C Axis. These objects have a parent-child relationship. Each component or subcomponent will have a child called "DataItem" at the end of its line of children components. Each of these components translates to an object in OPC-UA, while the DataItems translate to either variables or properties, all of which must be accounted for when generating nodes since each item becomes a node.

The data from the MTConnect device is in the form of an XML file, but it is being taken from the internet via a Uniform Resource Locator (URL). Using a URL to XML module [6], the MTConnect stream file is imported and filtered through the ElementTree XML API [7]. The ElementTree module allows for an XML file to be imported and its children can be analyzed from the root down, by exploring either the attributes, the tags, or the values. The DataItem tags are filtered via their attributes to locate the proper value. All values in XML are strings, which provides an ease of exportation to text files with a naming convention that uses a combination of component names and attributes to properly label data. The timestamp associated with each value is exported as a separate text file with identical component names and attributes but with a time extension added. It provides any other programs the ability to locate the data value along with its corresponding timestamp. For the purpose of visualization, timestamps are converted to a decimal version: 03:15 becomes 03.25.

MTConnect Data Probe

To search the XML MTConnect file for its components and DataItems, an XML probe is developed using Python and the ElementTree module. Using ElementTree, the probe generates an XML "companion file" to define a node structure for all devices in the MTConnect file, the XML companion file conforms to the OPC-UA "MTConnectModel.xml," found in the model compiler stack. The MTConnectModel.xml file represents basic MTConnect devices, components, and DataItems in the OPC-UA format. The .NETStandard server takes in only .uanodes files, which are binary file representations of the XML predefined nodes. The XML companion file needs to be converted to this binary format via a model compiler, which is provided by the OPC Foundation [8]. Conveniently, the .uanodes file, along with other files created by the model compiler, are automatically stored to the directory of the OPC-UA server. By simply setting a file path and executing the probe, the server can be started, and the nodes will automatically appear in the address space. A sample address space of the NIST Test Bed, obtained using UaExpert, is shown in Figure 3. Each node is automatically generated using either the MTConnect name or id attribute.

OPC-UA Server

The OPC-UA server in the testing environment is developed using the UA.NET Standard provided by the OPC Foundation [9]. There are a couple of options for the server development. We selected this option because of the code stacks are free and well-documented. In the package, QuickStart applications are provided for quickly generating a server using .NET in C#. For

example, a boiler server conducts simulations of boilers, and the boiler objects in the address space are created using the OPC-UA model of object-typing and object instantiation. Once instantiation is completed, the simulation locates the nodes and establishes the values. Similarly, device nodes can be instantiated if the server reads in a file type of ".uanodes". Note that nodes can either be predefined or generated within the server configuration, however, the devices will have predefined nodes that are generated using the model compiler. Once the predefined nodes are established, objects with a specified type (e.g., MTDeviceType) are created within the server and variables with a specified data type are also created. Untyped nodes are converted to typed nodes that can be manipulated in the server, allowing data to be uploaded to these nodes. Any predefined nodes that remain untyped are replaced with their typed versions. This method of instantiation is extrapolated to devices to establish nodes for devices. For MTConnect devices, ideally the data should be streamed directly from the device to reduce the points of failure; however, in this study, due to time constraints, the data was streamed from a text file, which is generated by the MTConnect XML parser. Once the nodes are instantiated, a client can communicate with the server to access the information.

OPC-UA Client

Similar to the requirements for the server, the tool for the OPC-UA client development was selected based on its price and available documentation. Additional consideration included the programming language used, as clients come in many forms; some use GUIs while others are based in the programming environments of languages such as MATLab [10] or Python [11]. We used UaExpert by Unified Automation [12] because it facilitates the GUI usage, is free, and well documented. UaExpert enables the connection with the server and supports the testing plan because it has features, such as Data Access view, Data Logger view, and Event view. The comma-separated values (CSV) export feature in the Data Logger view enables data visualization through the GUI.

Graphical User Interface

To confirm proper transfer of data from the MTConnect device to the OPC-UA server/client, a GUI is developed to take in real-time data from the parser and client, to compare the two data sets, and display the plots. The GUI is developed using Dash by Plotly [13], which helps display data in real time and allows for expandable data sets and graphs that are particularly useful when dealing with multiple components or DataItems. The implementation uses a Dash module in Python, which requires data to be imported. The simplest method of doing so is through the text files. This method gives the user the option to select which graphs are visible based on the text files imported with a graph description given by the component and attribute names. The selected data set and its corresponding timestamp are imported and plotted in real-time as the text file continually grows with streaming data. Simplifications have been made to the GUI to reduce the memory usage on the system while promoting faster graph response time for recording since Dash GUIs are web-based interfaces displayed using porting on the local machine that can be accessed via a browser.

The GUI is developed for the visualization of highly dynamic data since UaExpert only displays a node's value, which can be changing constantly, sometimes faster than human eyes can detect, depending on the sampling rate. Attempting to compare rapidly changing position values of two data sets by eye does not uffice for validating a standard. The GUI is only needed for highly dynamic values since less dynamic values, such as events or conditions, are easy to assess visually; an Emergency Stop is either armed, triggered, or unavailable, and an oil temperature is either normal, unavailable, or in the warning or fault zone. These types of DataItems tend not to change repeatedly and therefore can be validated via observation, leveraging UaExpert's Data Access view since the value of a node is shown directly on its interface.

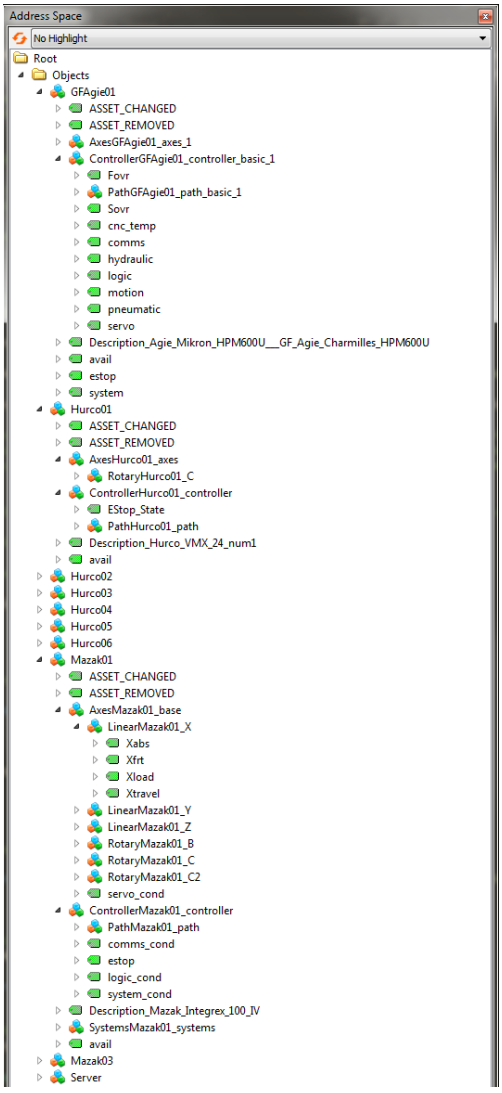

Figure 3. A Sample Partial NIST Test Bed Address Space Depicted in UaExpert

TEST CASES

A variety of test cases must be defined to test the information mapping between the MTConnect agent and OPC server/client properly. Before defining test cases, we fist discuss the key features of OPC-UA and then briefly discuss the types of MTConnect data.

Key Features of UA

While OPC-UA has many features, Data Access (DA), Historical Access (HA), and Event and Alarm Notifiers (EA), are three typical ones that can be used to begin testing the MTConnect to OPC-UA gateway [3].

- The Data Access feature provides instant machine/process monitoring and control capabilities by allowing clients or applications to directly stream real-time data from the server.
- The Historical Access feature is essentially an extension of the Data Access feature, but it keeps record of previous values for specified nodes. Cases for using this feature include data analysis, machine failure prevention, modeling, and simulation.
- The Event and Alarm Notifiers component provides OPC-UA clients with the ability to detect a change in an event or condition. It triggers an alarm if such a change occurs for a pre-existing node. An example for effective use of this feature would be the initiation of an alarm for the MTConnect predefined EmergencyStop event that has three possible values, one of which is "Triggered," meaning an emergency stop has occurred. Setting an alarm for this type of event would notify applications and interfaces by relaying this important information. Alarms can be selected for specific events since machine operators may not desire to have alarms continually activating for each node defined as an event or condition.

These three core features on a specified test case are sufficient for determining if a proper mapping of information occurs.

Defining DataItem Types

There are three types of DataItems that can exist in an MTConnect model [14]:

(1) Samples: samples must be numeric values from a stream. Examples include Rotary Speed, Angle, and Position.
(2) Events: events can be a variety of data types and generally have a predefined controlled vocabulary for specific components; however, there are cases where only a character string representing data is returned by the device.
(3) Conditions: conditions are another type of DataItem that exist in the MTConnect model. Each component may have more than one condition active at a specific instance where the conditions are defined by the string type, but each condition can be in one of four states: Normal, Warning, Fault, or Unavailable.

From these DataItems as shown in Figure 4, test cases can be generated.

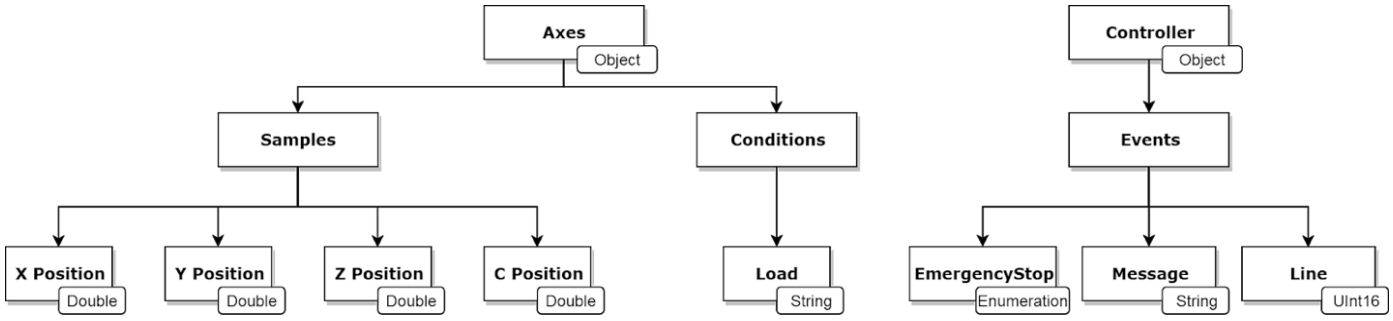

Figure 4. Types of DataItems

Defining Test Cases

The samples selected for test cases include the X, Y, Z, and C positions under the Axes component. A multitude of these DataItems were selected to ensure the server could handle multiple highly dynamic streams of data. For each of these four positions, the MTConnect schema or probe, given by the simulator, may display multiple DataItems such as an actual position, commanded position, and loaded position; the actual position is used for the mapping to a variable. The original MTConnect schema breaks down the Axes component into Linear Axis and Rotary Axis for better accuracy of data representation, however, a simplification is made by eliminating this extra component and placing the variables directly under the

Axes component. If the Linear Axis and Rotary Axis were to be implemented in the testing, it would only be an additional object in the hierarchy of the node tree, viewed in the address space. This would make no difference in the testing of the mapping. Custom types (e.g., MTComponentTypes) could be created to map a device with a custom structure (e.g., Axes with only Rotary components), which allows for expansibility to other devices. The testing of custom types requires only a syntactic change in node generation executed by the probe.

The condition test case selected is the Load, which gives the load condition for the specified axis in the MTConnect schema (in our case the X Axis). This condition is selected since it pre-exists under the Axes component. This avoids the need to create another component such as "Coolant" or "Electric" to the address space. Note that only one condition was selected since conditions can only be a string type, and their output is one of four values.

The controller component is selected to host the event test cases since it provides a variety of data types under one object. The string and integer are standard data types, however, the enumeration requires an integer be passed to the server to access a string from the enumeration. To account for this, the parser must translate the XML string (e.g., Unavailable) to the corresponding integer in the enumeration definition. The enumeration definition for this variable could be defined in the specification or could be custom, depending on the device; prior knowledge of the enumeration list must be obtained. For all test cases, an assumption was made stating that each event DataItem would return its declared data type to define the rigidity of the data being returned by the machine. If an alternative data type is returned, it will not be processed, and the previous node value is kept.

Applying Test Cases

Each of the identified test cases and their corresponding DataItem is to be explicitly declared in the MTConnect files being passed into the XML probe and parser. The server accesses them via the .uanodes and stream data files generated respectively. A secure connection between the server and client enables the data to be properly transmitted. If the test case contains highly dynamic data, the GUI is used to access the parser and client for displaying the test cases and their data for I/O analysis. If the test case contains less dynamic data, then the initial XML file, Data Access view, and Event view (if applicable) are used to observe changes in data. Validation of the standard is performed via the I/O analysis.

TESTING RESULTS

Data Access

Using the simulator provided by NIST and MemexOEE, after probing the device structure for the test cases, the nodes were properly generated in the address space of the server for all components and DataItems specified on all the devices. This confirms the MTConnect XML probe operated correctly. The streaming of data from these devices into the server was completed successfully, as the client reported changing values with the Data Access feature. Using the Data Logger, the machine data for the X-Position was recorded and plotted directly from the machine simulator and the client via the GUI. The results shown in Figure 5 display the X-Position data over an approximate twenty-minute period. While there is no delay implemented into the parser, a one-hundred millisecond delay was established on the server.

Comparing the I/O of X-Position data, a conclusion can be made that the MTConnect data was properly transferred from the agent to the OPC-UA server and client. While the overall data mapping is executed correctly, there were small errors in parts of the stream that were foreseen. Due to the XML parser outputting the streamed data to a text file and the server extracting the data from the text file simultaneously, occasionally the server would be incapable of opening the file. To allow the server to continue operating, try-catch statements were implemented, which returned the previous value of the node as the current instantiation.

Issues found: The samples for the positions of the Axes component in the companion specification are currently defined as a string data type when being mapped to the server, even though MTConnect specifically defines their position data as numerical values only. This indicates a design issue in the specification. This definition discrepancy created an issue when attempting to use the Data Logging tool to save data for the GUI's real-time plotting since the Data Logging tool needs to receive a double data type, not a string. For practical purposes, having numerical values stored as doubles or floats is justifiable while a representation in strings is not.

Alternate solution: To temporarily correct this definition error (assuming it will be permanently fixed in an updated version), the string data type was simply altered to a double in the predefined nodes, and the streamed value was also changed to a double; this change allows the use of Data Logging to transmit data for the visual representation in the GUI. All test cases assessed passed as compliant with the Data Access feature after data type definitions were corrected.

Historical Access

Issues found: Another issue occurred when executing the Historical Access feature on the identified test cases. Using UaExpert and its History view, the historical data was unable to be retrieved from the nodes. Attempts have been made to the nodes within the server to fix this error: the access levels were altered to allow history reading, history writing, and a current reading. The "Historizing" attribute was also enabled. The UaExpert still reported the node history to be empty. With the generation of this error, it is expected that nodes were not storing

their historic values; the process of enabling the node storage functionality could not be determined.

Alternate solution: Ideally this should be engaged automatically in the generation of the node structure to eliminate the need for altering individual node attributes within the server. After using a C# .NET client to confirm this error, the Historical Access feature worked to a small extent by recording and storing values only after the client was started (similar to the Data Logger). This limited functionality, however, is attributed to the client's capabilities and therefore is not sufficient for justifying full usage of the Historical Access feature. Once the nodes are capable of storing their historic values, the History Access functionality is expected to be operable. To modify the nodes for this capability, the standard developers must investigate it further and address the issue accordingly.

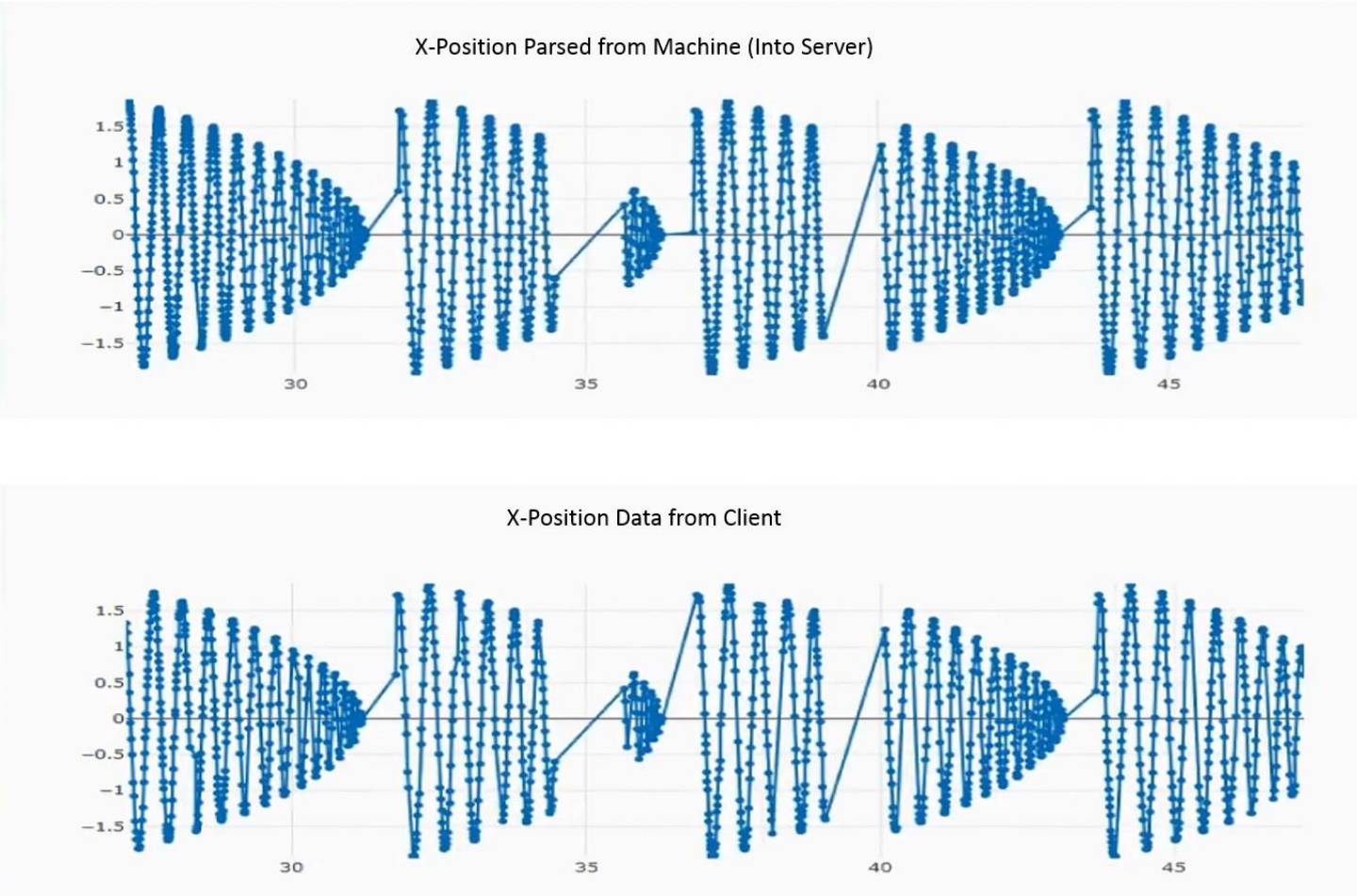

Figure 5. X-Position Data from the Machine and the OPC-UA Client

Event and Alarm Notifiers

Issues found: Another key issue that occurred during the testing was the use of the Event and Alarm Notifiers feature. From the defined test cases, the condition and event nodes were unable to respond properly to the feature. Although we tried different data types and used an alternative client, the feature continued to fail. We also modified the primary object's (device's) "EventNotifier" attribute by establishing a "SubscribeToEvents" setting. This revision did not allow the Event and Alarm Notifiers feature to recognize any changes in the event test cases. The same modification was also applied to the sub-objects (sub-components) of the primary object, and no changes occurred. The absence of the feature may be due to improper configuration of the server, an improper mapping of the companion specification, and/or a lack of description in the specification; more investigation should be performed.

CONCLSUIONS

In this paper, we discussed an initial implementation and testing of the MTConnect - OPC-UA Companion Specification

Version 1.02. Our initial testing results show some issues in the specification that need to be addressed in future versions of the standard. These improvements will help the companion specification to better integrate semantics of MTConnect with the syntactic representation of OPC-UA. By conducting an expanded investigation of this and future versions of the standard, the specification could eventually be established as a viable use in the smart manufacturing setting.

The following recommendations regarding data types, Event and Alarm Notifiers, and enumerations, can be made to the SDOs based on the conducted testing.

- Data type definitions should be improved to properly map data for practical uses, i.e., the positions being mapped as doubles or floats compared to strings.
- The enablement of Event and Alarm Notifiers should be explicitly declared in newer versions of the specification, as currently there is no method stating how objects are enabled as events. Whether the enablement of nodes for this feature occurs within the server or within the model generation, the companion specification should state one or both options. A simple solution to be proposed is enabling Event and Alarm Notifiers upon generating nodes, allowing the OPC-UA server to establish a clear distinction between imported DataItems. If enablement of notifiers is to be completed within the server, it will require a tedious task for users with complex device structures.
- Regarding enumerations, for data to properly be retrieved by the server/client, an integer representing position of the data type in the enumeration had to be imported into the server. While this justifies the functionality of the companion specification, it must be noted that the data taken from the stream (string retrieved such as "ARMED") had to be converted to an integer representing the proper position in the enumeration. An example of this is the streamed value of "ARMED" being translated to the integer "1", which represents the second position in the enumeration list declared in the MTConnectModel.xml. The integer is then used by the server to locate the proper string value. This requirement means that a converter must be made for all enumerations to allow the server to properly access data using the current method. While this may or may not be a correction that needs to be accounted for in future versions, it is an aspect of the companion specification that should be brought to the SDOs' attention.

Future work for better validating the standard includes increasing the number of test cases, performing a larger variety of test cases, if possible, in which further examination of data types is needed, as other errors are bound to exist, similar to the position data type error. To correct for the minor errors that were occurring in the data streaming process, the parser should be implemented using C# so that it can execute directly within the server. By having the server access the agent directly, compared to indirectly through text files, the chances of data-access errors occurring will be reduced or eliminated since files will not be opened and closed simultaneously. Finally, we should work closely with the standard developers to figure out the issues with the testing of the Event and Alarm Notifiers feature.

NOMENCLATURE

CSV	Comma-Separated Values
DA	Data Access
DX	Data Exchange
EA	Event and Alarm notifiers
GUI	Graphical User Interface
HA	Historical Access
NIST	National Institute of Standard and Technology
OPC	Object linking and embedding for Process Control
SDO	Standard Developing Organizations
SM	Smart Manufacturing
UA	Unified Architecture
URL	Uniform Resource Locator
V&V	Verification and Validation
XML	Extensible Markup Language

ACKNOWLEDGMENTS

The authors would like to thank the Summer Undergraduate Research Fellowship (SURF) program for supporting the project and thank Moneer Helu, Will Sobel, and Randy Armstrong for their valuable discussion and support.

DISCLAIMERS

No approval or endorsement of any commercial product by the National Institute of Standards and Technology (NIST) is intended or implied. Certain commercial software systems are identified in this paper to facilitate understanding. Such identification does not imply that these software systems are necessarily the best available for the purpose.

REFERENCES

[1] MTConnect (2018) A free, open standard for the factory. http://www.mtconnect.org
[2] OPCUA (2018) Unified Architecture. https://opcfoundation.org/about/opc-technologies/opc-ua/.
[3] MTConnect (2013) MTConnect - OPC UA Companion Specification Release Candidate Version 1.02.
[4] Hirvonen, Markus. "Streamlining Manufacturing Data Integration." Tampere University of Technology, Tampere, Finland. 2017. https://core.ac.uk/download/pdf/144141231.pdf
[5] Memex (2018) Driving efficiency and productivity from the shop floor to the top floor. http://www.memexoee.com/.
[6] Urllib (2018) Open arbitrary resources by URL. https://docs.python.org/2/library/urllib.html.

[7] ElementTree (2018) The ElementTree XML API. https://docs.python.org/2/library/xml.etree.elementtree.html.
[8] UA-ModelCompiler (2018) Model compiler converts XML files into C# and ANSI C. https://github.com/OPCFoundation/UA-ModelCompiler.
[9] OPCUA .NET (2016) Build OPC UA .NET applications using .NET Standard Library. http://opcfoundation.github.io/UA-.NETStandard/.
[10] MathWorks (2018) MATLAB. https://www.mathworks.com/products/matlab.html.
[11] Python (2018) A programming language that lets you work quickly and integrate systems more effectively. https://www.python.org.
[12] UaExpert (2018) A full-featured OPC UA Client. https://www.unified-automation.com/products/development-tools/uaexpert.html.
[13] Dash Plotly (2018) Build beautiful web-based interfaces in Python. https://plot.ly/products/dash/.
[14] MTConnect (2014) MTConnect Specification and Materials. http://www.mtconnect.org/docs/streams/.