MTConnect-OPC UA Companion Specification

FINAL

Version 1.0

November 8, 2012

CONTENTS

1 Introduction 5
1.1 Background 5
1.2 MTConnect-OPC UA Goals 5
1.3 Who Will Find Benefit from this Specification? 6
1.4 References 6
2 Use Cases 8
2.1 Overview 8
2.2 Device Maker 8
2.3 Independent Software Vendor 9
2.4 End-User Engineer 10
3 MTConnect 11
3.1 What is MTConnect? 11
3.2 Basics of MTConnect 11
4 OPC UA 13
4.1 What is OPC UA? 13
4.2 Basics of OPC UA 14
4.3 Information Modeling in OPC UA 14
4.3.1 Concepts 14
4.3.2 Namespaces 17
4.3.3 Companion Specifications 18
4.3.4 OPC UA for Devices (DI) 19
5 MTConnect in OPC UA 20
5.1 Devices 20
5.2 Component 20
5.3 Device 22
5.4 DataItems 23
5.4.1 General 23
5.4.2 Sample DataItems 24
5.4.3 Event DataItems 25
5.4.4 Conditions 25
5.5 Streams 27

Figure 1 – The Device Maker Use Case 9
Figure 2 – The Independent Software Vendor (ISV) Use Case 10
Figure 3 – The End User Engineer Use Case 11
Figure 4 – MTConnect Overview 12
Figure 5 – The Scope of OPC UA within an Enterprise 14
Figure 6 – A Basic Object in an OPC UA Address Space 15
Figure 7 – The Relationship between Type Definitions and Instances 16
Figure 8 – Examples of References between Objects 16
Figure 9 – The OPC UA Information Model Notation 17
Figure 10 – A Visual Representation of the Sample ObjectType 19
Figure 11 – The OPC UA for Devices (DI) Model 19
Figure 12 – MTConnect Devices in a UA Address Space 20
Figure 13 – MTConnect DataItems in the UA Address Space 24
Figure 14 – MTConnect Conditions in the UA Address Space 27

Table 1 – Example *ObjectType* Definition 18
Table 2 – *MTComponentType* Definition 21
Table 3 – *MTDeviceType* Definition 22
Table 4 – *MTDataItemType* Definition 23
Table 6 – *MTEventDataItemType* Definition 25
Table 7 – *MTConditionType* Definition 26
Table 8 – *MTConditionClassType* Definition 27
Table 9 – An Example Condition Event 29

1 Introduction

1.1 Background

In September 2010, the OPC Foundation and the MTConnect Institute signed a memorandum of understanding to provide a mechanism for OPC and MTConnect to collaborate to extend the reach of the existing manufacturing data exchange standards and implementation technologies in order to:

- Evolve the existing standards for each organization to provide complete manufacturing technology interoperability.
- Provide the mechanism for continuous improvement of standards and specifications overseen by each body.
- Work directly with the end users and suppliers of technology and manufacturing.
- Provide a coordinating function to exchange insights, identify overlaps, and harmonize work where appropriate.
- Facilitate clear communication and education for users and others concerning possible overlaps and the ways the standards and specifications can be used.

Provide a solid foundation to develop and deliver specifications, technology and processes to facilitate adoption of the technology into real products.

The outcome of that agreement is this companion specification called MTConnect-OPC UA. MTConnect-OPC UA is a set of companion specifications to ensure interoperability and consistency between MTConnect specifications and the OPC Unified Architecture (UA) specifications, as well as the manufacturing technology equipment, devices, software or other products that implement those standards.

1.2 MTConnect-OPC UA Goals

MTConnect-OPC UA is designed with all of the following first-class goals in mind, in the interest of wide and rapid adoption by vendors of equipment and software:

- *Incremental adoption*—the technical barrier to MTConnect-OPC UA enablement will be greatly reduced with this companion specification and the source code and binaries available in the MTConnect-OPC UA reference port

- *Evolution*—MTConnect-OPC UA can incrementally evolve without jeopardizing backwards compatibility of previous MTConnect-OPC UA versions.
- *Customizability*—MTConnect-OPC UA's extensibility makes it easy to create value-added software and tools that are machine-specific or installation-specific, without jeopardizing compatibility with other equipment or software.
- *Non-proprietary*—built on open standards, backed by both the OPC Foundation and the MTConnect Institute which represents hundreds of companies, individuals, government organizations and non-profits all working toward the goal of increased productivity in the manufacturing arena.

1.3 Who Will Find Benefit from this Specification?

To adopt the MTConnect-OPC UA one will need to have a clear understanding of both MTConnect and OPC. From the technical side, we will discuss MTConnect-OPC UA from:

- The backend or OPC UA Server and MTConnect agent/adapter architecture.
- The client or software application side, we will discuss how one develops an application that is MTConnect-OPC UA enabled.

From the business side, we will reference a companion business MTConnect-OPC UA white paper that addresses the concerns from the owners and top management of the business as well as the operations and engineering management. It is the objective of this white paper to provide information primarily to those MTConnect and OPC UA software developers. We do not make assumptions about the level of programming expertise beyond what would be considered to be "reasonable" level of expertise. It is for this reason that we include enough details about both MTConnect and OPC UA to provide the ability to implement this companion specification without having references back to other documents. However, the OPC and MTConnect standards are critical and become much more meaningful with the appropriate overview from this document.

1.4 References

1.4.1 OPC Foundation

The following specifications from the OPC foundation are referenced by this specification. The OPC Foundation also provides addition specification for other related topics such as historical data. For these additional specifications see http://www.opcfounadtion.org .

[UA Part 1] OPC UA Specification: Part 1 – Concepts

http://www.opcfoundation.org/UA/Part1/

[UA Part 2] OPC UA Specification: Part 2 – Security Model

http://www.opcfoundation.org/UA/Part2/

[UA Part 3] OPC UA Specification: Part 3 – Address Space Model

http://www.opcfoundation.org/UA/Part3/

[UA Part 4] OPC UA Specification: Part 4 – Services

http://www.opcfoundation.org/UA/Part4/

[UA Part 5] OPC UA Specification: Part 5 – Information Model

http://www.opcfoundation.org/UA/Part5/

[UA Part 6] OPC UA Specification: Part 6 – Mappings

http://www.opcfoundation.org/UA/Part6/

[UA Part 7] OPC UA Specification: Part 7 – Profiles

http://www.opcfoundation.org/UA/Part7/

[UA Part 8] OPC UA Specification: Part 8 – Data Access

http://www.opcfoundation.org/UA/Part8/

[UA Part 9] OPC UA Specification: Part 9 – Alarms and Conditions

http://www.opcfoundation.org/UA/Part9/

[UA DI] OPC UA: Devices (DI) Companion Specification.

http://www.opcfoundation.org/UA/DI/

1.4.2 MTConnect Institute

[MT Part 1] MTConnect® Standard: Part 1 – Overview, Version 1.1

http://mtconnect.org/media/7571/mtc_part_1_overview_1.1.0.pdf

[MT Part 2] MTConnect® Standard: Part 2 – Components and DataItems, Version 1.1

http://mtconnect.org/media/7574/mtc_part_2_components_1.1.0.pdf

[MT Part 3] MTConnect® Standard: Part 3 – Streams, Version 1.1

http://mtconnect.org/media/7577/mtc_part_3_streams_1.1.0.pdf

1.5 Abreviations

The following abbreviation are used in this document

- ERP – Enterprise Resource Planning
- HMI – Human Machine Interface
- Http – Hyper Text Transport Protocol
- MES – Management Execution Systems
- PLC – Programmable Logic Controller
- PMS - Production Management Systems
- SCADA -
- TCP/IP -
- XML - eXtensible Mark-up Language

2 Use Cases

2.1 Overview

Before delving into the details of the specification it is useful to identify some of the key use cases for the technology. The use cases defined here are not an exhaustive list; however, they should help demonstrate how this specification is expected to be used.

2.2 Device Maker

The use case shown in Figure 1 centers on the maker of a piece of equipment or device that needs to provide connectivity to other systems. In some cases, the device maker will be targeting markets other than equipment (Machine Tool) and would benefit from a more generic specification like OPC UA. On the other hand, the standardized semantics of MTConnect are extremely important to interoperability within the Machine Tools space. The MTConnect-OPC UA specification allows the device makers to standardize on OPC UA as the network interface while making their information accessible to MTConnect aware applications. Figure 1 shows several clients developed for different purposes that can access information produced by the device via OPC UA.

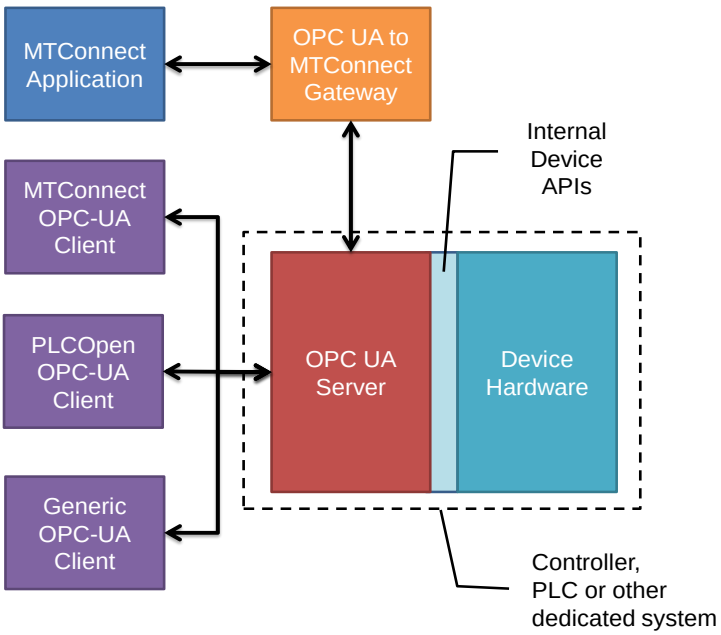

Figure 1 – The Device Maker Use Case

2.3 Independent Software Vendor

The use case shown in Figure 2 centers on an Independent Software Vendor (ISV) that wishes to sell products to users of equipment such as Machine Tools. An ISV will typically want to provide gateways that convert information between MTConnect and OPC UA as well as adding numerous features that add value to the semantics defined in the MTConnect standards. The MTConnect-OPC UA specification allows the ISV to extend the MTConnect-OPC UA information model with application specific constructs which can be easily accessed via any standard OPC UA client product. These added features will exist in parallel to the standard MTConnect interfaces. Figure 2 shows an ISV product that consumes data from MTConnect and OPC UA enabled devices and then makes it available via MTConnect and OPC UA.

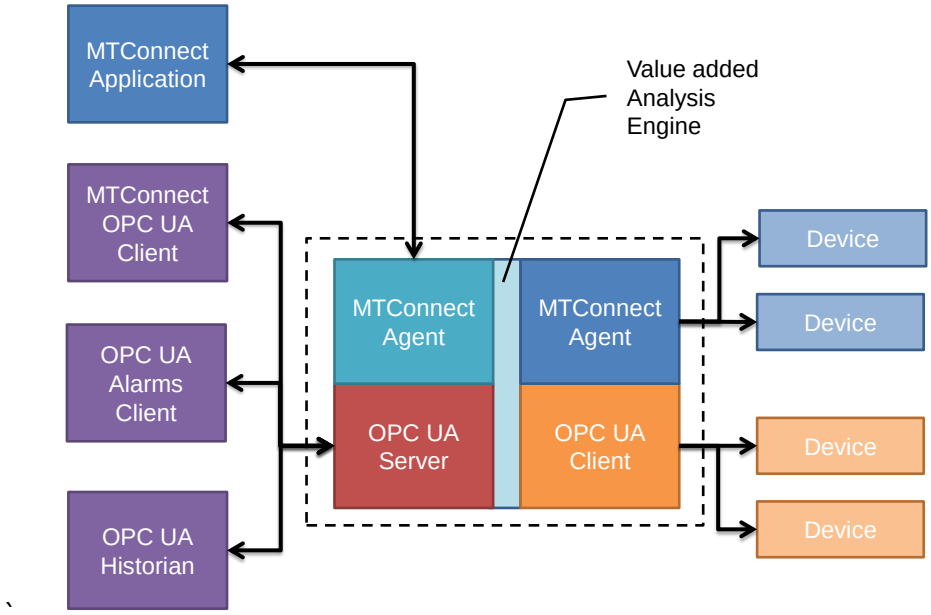

Figure 2 – The Independent Software Vendor (ISV) Use Case

2.4 End-User Engineer

This use case shown in Figure 3 centers on an Engineer or Systems Integrator responsible for setting up and configuring an MTConnect enabled system for a user of Machine Tools. The Engineer is typically familiar with the MTConnect specification but wishes to configure generic OPC UA client applications. The MTConnect-OPC UA specification allows the Engineer to understand how MTConnect concepts are represented in OPC UA and determine what they need to do to configure their OPC UA Applications. Without this specification, an Engineer interested in OPC based data would have had to rely on vendor documentation and a laborious process of manually mapping tags to MTConnect concepts. This specification eliminates the need for that by providing a standard mapping. Figure 3 shows how the common Information Model defined by this specification gives the End User Engineer choices when it comes to accessing device data.

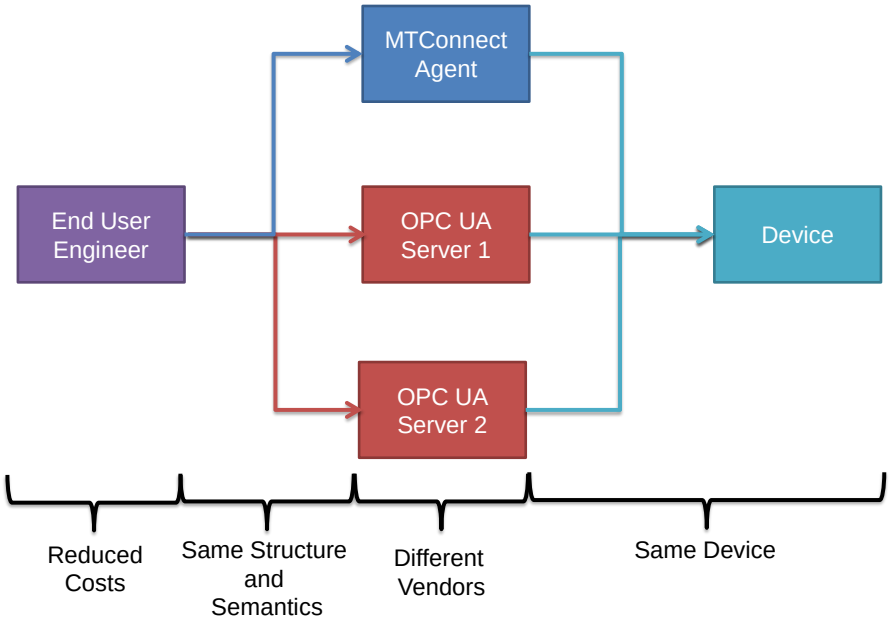

Figure 3 – The End User Engineer Use Case

3 MTConnect

3.1 What is MTConnect?

MTConnect is an open and royalty-free set of standards designed as a universal factory floor communications protocol.

MTConnect is intended specifically for the shop floor environment. While there are numerous communication solutions available, MTConnect defines a "dictionary" for manufacturing data. This means that all data is provided with full context – name, definition, scaling, etc.

With most communication networks, all data is defined at the point of use – the application. With MTConnect, the data is defined at the source – the device or Machine Tool.

MTConnect devices process information locally and then provide that data in a consistent format to any client application requesting data - ERP, MES, Production Management Systems, Maintenance Systems or a standard Browser, for examples.

3.2 Basics of MTConnect

MTConnect is based on standard Internet technologies – HTTP, Ethernet, and XML (Extensible Mark-Up Language – the underlying language of most web sites).

As an Extensible Standard, MTConnect cannot address every conceivable data need on the shop floor. MTConnect provides a clearly defined method for adding additional data types which

can be exchanged between equipment, devices, controllers and applications; providing the flexibility to meet the demands of varying environments.

MTConnect is made up of five fundamental components (see Figure 4 below):

Device – A type of equipment (Machine Tool) or data source.

Adapter – An optional piece of software (and sometimes hardware) that provides a link or conversion from the data source and data definition in the device to the MTConnect Data definition. This can be thought of as a translator. The Adapter is not needed for devices that use MTConnect as their native language.

Agent – A piece of software that collects, arranges, and stores data from the device. It receives requests for data from applications, processes those requests, and then transmits the required data.

Network - The physical connection between a data source (device) and the data consumer (application). Normally, this is an Ethernet network. The communication on the network normally uses standard internet communications methods – http:// protocol. It should be noted that the MTConnect Structure is adaptable and can be implemented in conjunction with other networking solutions other than Ethernet and Internet protocols.

Client – A Client initiates all requests for MTConnect data. A Client resides in an application or device. The Client is a software function in the application or device that actually requests data from the Agent.

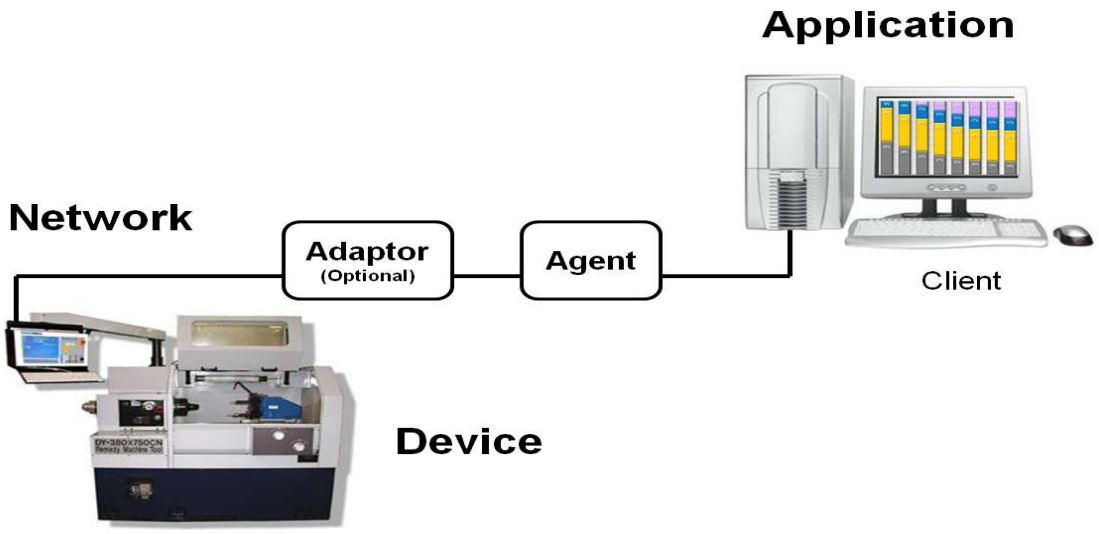

Figure 4 – MTConnect Overview

The MTConnect Standard does not restrict the physical implementation of how the MTConnect system is designed.

- The Network may be a physical implementation, like an Ethernet network. It can also be implemented using wireless or other technologies.
- The Internet Protocol (http) does not mean that your machine is automatically connected outside your plant to the Internet. This is a communications method only. Protection of your data is controlled by your networking standards.
- There is no specific requirement for where the Adaptor and Agent function is located. These typically are located at the device. However, they can be placed anywhere in the networking architecture. Also, they do not need to be located together. It is totally MTConnect compliant to have the Adapter installed at the device and the Agent installed along with the Client. The location of these functions should be considered when implementing MTConnect since they will impact the level of data flow on different segments of your network.

4 OPC UA

4.1 What is OPC UA?

OPC UA is an open and royalty free set of standards designed as a universal factory floor communications protocol.

OPC UA is designed specifically for the factory environment. While there are numerous communication solutions available, OPC UA combines a state of art security model (see [UA Part 2]), a fault tolerant communication protocol and an information modeling framework that allows application developers to represent their data the in a way that makes sense to them.

OPC UA has a broad scope which delivers for economies of scale for application developers. This means that a larger number of high quality applications at a reasonable cost are available to factory owners. When combined with powerful semantic models such as MTConnect, OPC UA makes it easier for factory owners to access data via generic commercial application.

The OPC UA model is scalable from small devices to ERP systems. OPC UA devices process information locally and then provide that data in a consistent format to any application requesting data - ERP, MES, PMS, Maintenance Systems, HMI, Smartphone or a standard Browser, for examples. For a more complete overview see [UA Part 1].

4.2 Basics of OPC UA

As an Open Standard, OPC UA is based on standard Internet technologies – TCP/IP, HTTP, Ethernet, and XML.

As an Extensible Standard, OPC UA provides a set of services (see [UA Part 4]) and a basic information model framework. This framework provides an easy manner for creating and exposing vendor defined information in a standard way. More importantly all OPC UA Clients are expected to be able to discover and use vendor defined information. This means OPC UA users can benefit from the economies of scale that come with generic visualization and historian applications. This specification is an example of an OPC UA Information Model designed to meet the needs of Machine Tool developers and users.

OPC UA Clients can be any consumer of factory data from another device on the network to browser base thin clients and ERP systems. The full scope of OPC UA applications are shown in Figure 5.

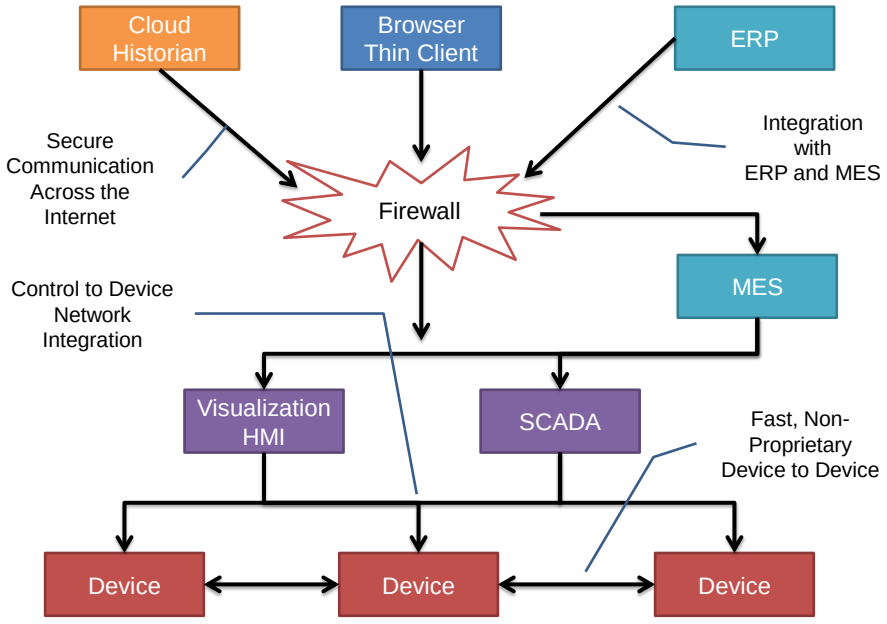

Figure 5 – The Scope of OPC UA within an Enterprise

4.3 Information Modeling in OPC UA

4.3.1 Concepts

OPC UA provides a framework that can be used to represent complex information as Objects in an address space which can be accessed with standard web services. These Objects consist of Nodes connected by References. Different classes of Nodes convey different semantics. For

example a Variable Node represents a value that can be read or written. The Variable Node has an associated DataType that can define the actual value, such as a string, float, structure etc. It can also describe the variable value as a variant. A Method Node represents a function that can be called. Every Node has a number of Attributes including a unique identifier called a NodeId and non-localized name called as BrowseName. An Object representing a 'Reservation' is shown in Figure 6.

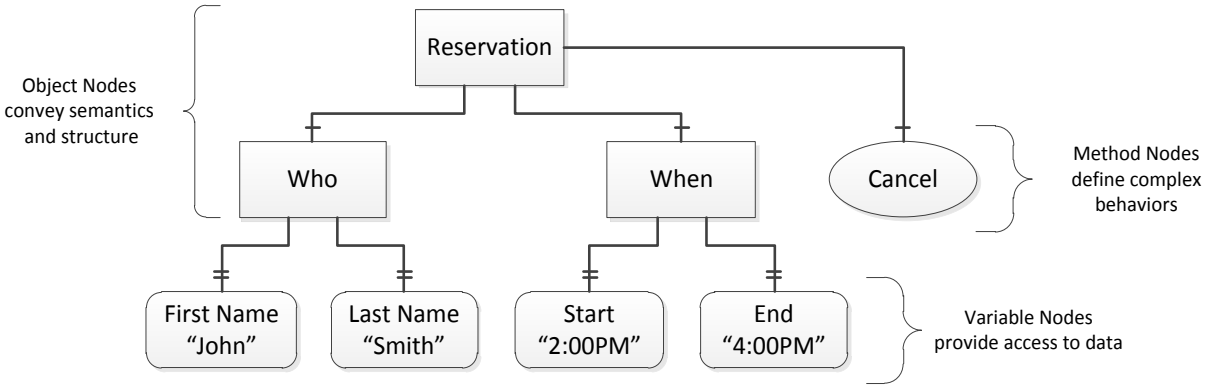

Figure 6 – A Basic Object in an OPC UA Address Space

Object and Variable Nodes are called Instance Nodes always reference a Type Definition (ObjectType or VariableType) Node which describes their semantics and structure. Figure 7 illustrates the relationship between an Instance and its Type Definition.

The Type Nodes are templates that define all of the children that can be present in an Instance of the Type. In the example in Figure 7 the PersonType ObjectType defines two children: First Name and Last Name. All instances of PersonType are expected to have the same children with the same BrowseNames. Within a Type the BrowseNames uniquely identify the child. This means Client applications can be designed to search for children based on the BrowseNames from the Type instead of NodeIds. This eliminates the need for manual reconfiguration of systems if a Client uses Types that multiple devices implement.

OPC UA also supports the concept of sub typing. This allows a modeler to take an existing type and extend it. There are rule regarding sub typing defined in [UA Part 3], but in general they allow the extension of a given type or the restriction of a DataType. For example the modeler may decide that the existing ObjectType in some cases needs an additional variable. The modeler can create a subtype the object and add the variable. A client that is expecting the parent type can treat the new type as if it was of the parent type. With regard to DataTypes, if a variable is defined to have a numeric value, a sub type could restrict the Value to a float.

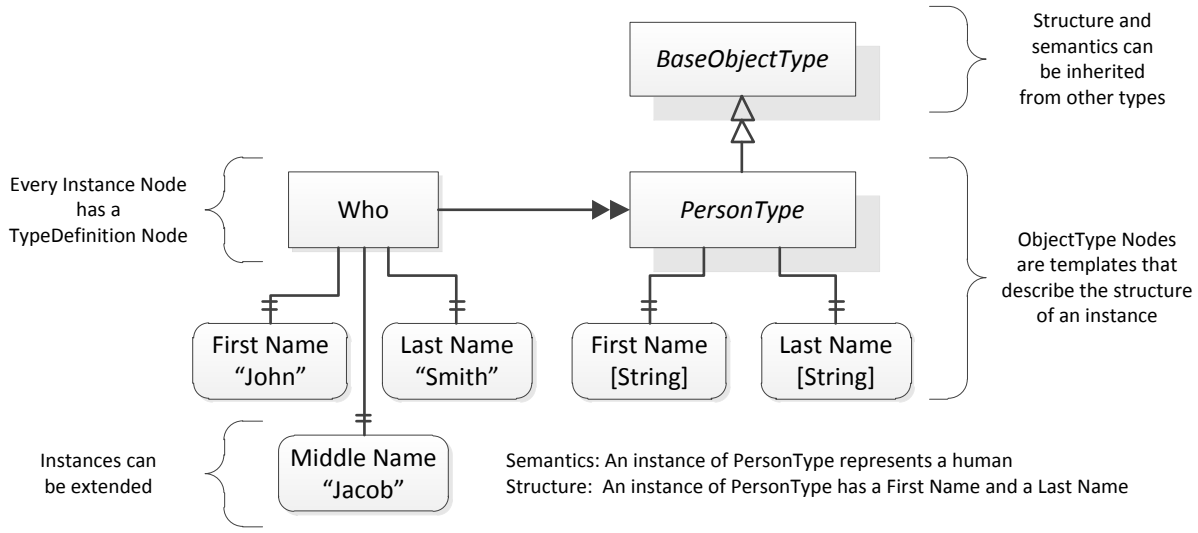

Figure 7 – The Relationship between Type Definitions and Instances

References allow Nodes to be connected together in ways that describe their relationships. All References have a ReferenceType that specifies the semantics of the relationship. References can be hierarchical or non-hierarchical. Hierarchical references are used to create the structure of Objects. Non-hierarchical are used to create arbitrary associations. Applications can define their own ReferenceTypes by creating subtypes of the existing ReferenceType. Subtypes inherit the semantics of the parent but may add additional restrictions. Figure 8 depicts several references connecting different Objects.

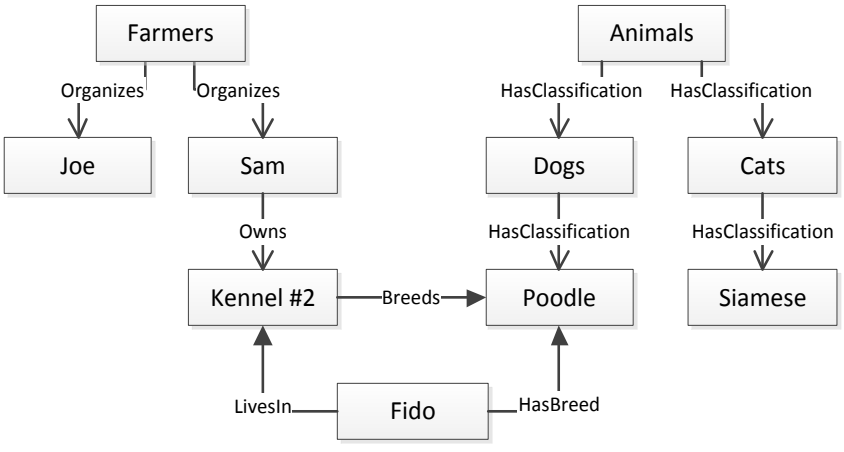

Figure 8 – Examples of References between Objects

The figures above use a notation that was developed for the OPC UA specification. The notation is summarized in Figure 9. UML representations can also be used; however, the OPC UA

notation is less ambiguous because there is a direct mapping from the elements in the figures to Nodes in the address space of an OPC UA server.

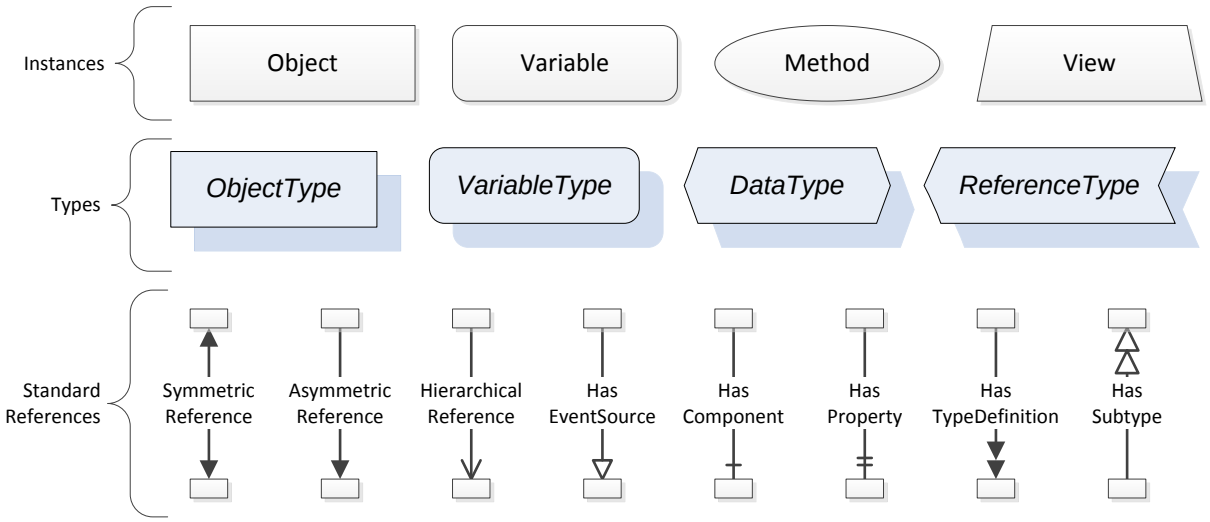

Figure 9 – The OPC UA Information Model Notation

A complete description of the different types of Nodes and References can be found in [UA Part 3] and the base OPC UA Address space is described in [UA Part 5]..

OPC UA specification defines a very wide range of functionality in its basic information model. It is not expect that all clients or servers support all functionality in the OPC UA specifications. OPC UA includes the concept of profiles, which segment the functionality into testable certifiable units. This allows the development of companion specification (such as MTConnet-OPC UA) that can describe the subset of functionality that is expected to be implemented. The profiles do not restrict functionality, but generate requirements for a minimum set of functionality (see [UA Part 7])

4.3.2 Namespaces

OPC UA allows information from many different sources to be combined into a single coherent address space. Namespaces are used to make this possible by eliminating naming and id conflicts between information from different sources. Namespaces in OPC UA have a globally unique string called a NamespaceURI and a locally unique integer called a NamespaceIndex. The NamespaceIndex is only unique within the context of a Session between an OPC UA Client and an OPC UA Server. All of the web services defined for OPC UA use the NamespaceIndex to specify the Namespace for qualified values.

There are two types of values in OPC UA that are qualified with Namespaces: NodeIds and QualifiedNames. NodeIds are globally unique identifiers for Nodes. This means the same Node with the same NodeId can appear in many Servers. This, in turn, means Clients can have built in

knowledge of some Nodes. OPC UA Information Models generally define globally unique NodeIds for the TypeDefinitions defined by the Information Model.

QualifiedNames are non-localized names qualified with a Namespace. They are used for the BrowseNames of Nodes and allow the same Names to be used by different information models without conflict. The BrowseName is used to identify the children within a TypeDefinitions. Instances of a TypeDefinition are expected to have children with the same BrowseNames. TypeDefinitions are not allowed to have children with duplicate BrowseNames; however, Instances do not have that restriction.

All TypeDefinitions and BrowseNames defined by this specification are qualified by the MTConnect Namespace (“urn:mtconnect.com:MTConnectDevices:1.1”) unless stated otherwise.

4.3.3 Companion Specifications

An OPC UA companion specification for an industry specific vertical market describes an information model by defining ObjectTypes, VariableTypes, DataTypes and ReferenceTypes that represent the concepts used in the vertical market. Table 1 contains an example of an ObjectType definition.

Table 1 – Example *ObjectType* Definition

Attribute	**Value**				
BrowseName	WidgetType				
IsAbstract	True				
Reference	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Inherit the *Properties* of the *TopologyElementType* from [UA DI].					
HasProperty	Variable	Color	String	PropertyType	Mandatory
HasProperty	Variable	Flavor	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	Rank	Int32	PropertyType	Mandatory

The BrowseName is a non-localized name for an ObjectType.

IsAbstract is a flag indicating whether instances of the ObjectType can be created.

The bottom of the table lists the child nodes for the type. The Reference is the type of reference between the Object instance and the child Node. The NodeClass is the class of Node. The BrowseName is the non-localized name for the child. The DataType is the structure of the Value accessible via the Node (only used for Variable NodeClass Nodes) and the TypeDefinition is the ObjectType or VariableType for the child.

The ModellingRule indicates whether a child is Mandatory or Optional. It can also indicate cardinality. Note that the BrowseName is not defined if the cardinality is greater than 1:1.

Figure 10 visually depicts the ObjectType defined in Table 1 along with an instance of the ObjectType.

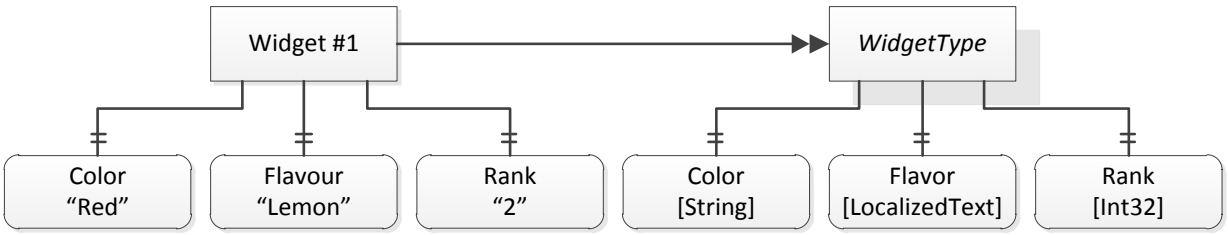

Figure 10 – A Visual Representation of the Sample ObjectType

4.3.4 OPC UA for Devices (DI)

OPC UA for Devices (DI) (see [UA DI]) is an information model for a generic device containing multiple components called ‘Functional Groups’. Figure 11 illustrates the structure of a *TopologyElementType* which is the base type used to construct devices and their groups. An MTConnect Device is a subtype of the DI *DeviceType*.

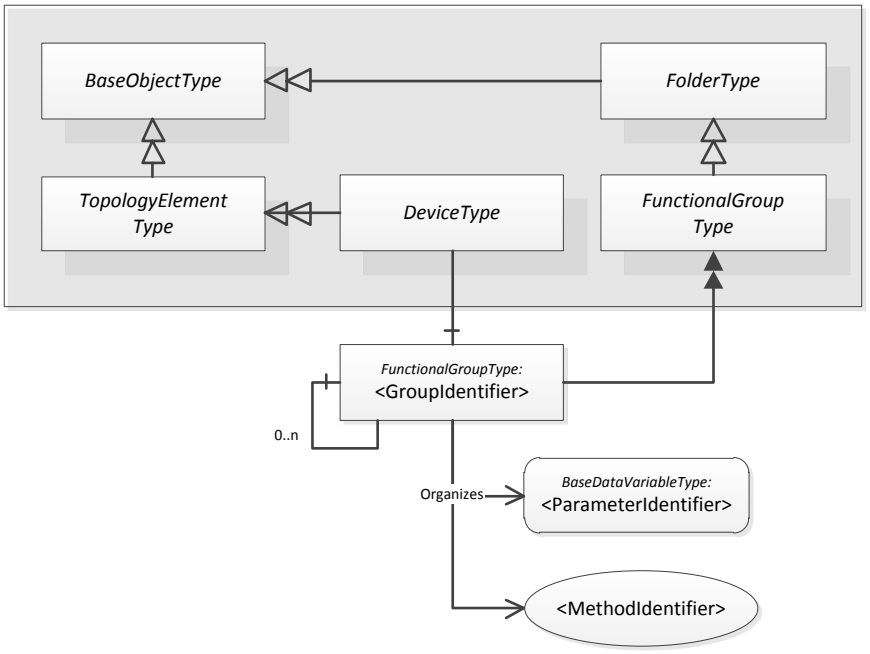

Figure 11 – The OPC UA for Devices (DI) Model

5 MTConnect in OPC UA

5.1 Devices

The top level container for an MTConnect Agent is the Devices element. This is mapped to the DI DeviceSet object which appears at the root of the Objects folder.

An OPC DI DeviceSet can contain devices that conform to other OPC UA Device Profiles such as PLCOpen. The structure of an MTConnect Device is described below. Generic OPC UA Clients should be able to treat all devices in the same way. MTConnect aware clients may ignore devices that do not comply with the MTConnect OPC UA profile. Figure 12 illustrates the top of the address space for an MTConnect enabled OPC UA Server.

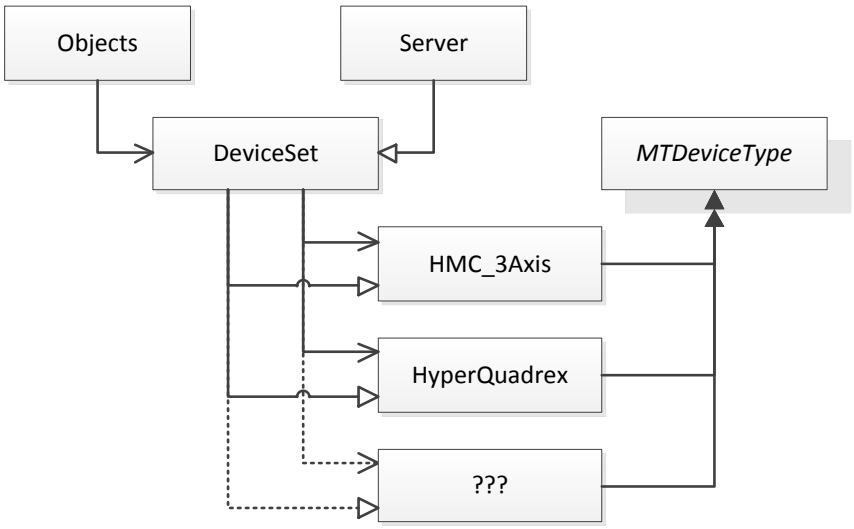

Figure 12 – MTConnect Devices in a UA Address Space

Figure 12 depicts two separate hierarchies. The Organizes hierarchy from Objects Node is used to discover the structure of the devices. The HasNotifier hierarchy from the Server Node is used when subscribing to events (see discussion later in document).

5.2 Component

An MTConnect Component element is the container for Machine Tool information that can be accessed via MTConnect. Each Component is mapped to an instance of an Object Node with a TypeDefinition that is *MTComponentType* or one of its subtypes.

The *MTComponentType* ObjectType is defined in Table 2.

Table 2 – *MTComponentType* Definition

Attribute	**Value**				
BrowseName	MTComponentType				
IsAbstract	False				
Reference	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Inherit the children of the *TopologyElementType* which is defined in [UA DI] .					
HasProperty	Variable	Uuid	String	PropertyType	Mandatory
HasProperty	Variable	NativeName	String	PropertyType	Optional
HasProperty	Variable	Station	String	PropertyType	Optional
HasComponent	Object	DataItems		FunctionalGroupType	Mandatory
HasComponent	Object	Conditions		FunctionalGroupType	Optional
HasComponent HasNotifier	Object	<server defined>		MTComponentType	Optional (0..N)

The Uuid, NativeName and Station properties correspond to their definitions in [MT Part 2].

The BrowseName of an *MTComponentType* instance is always the value of the 'name' attribute.

The DataItems Object is a container for the MTConnect Sample and Event DataItems supported by the Component.

The Conditions Object is a container for the MTConnect Condition DataItems supported by the Component.

The MTConnect specification defines many standard Components. A complete set of TypeDefinitions for the standard components is not provided in this specification in order to avoid transcription errors. Instead the following design rules should be followed when creating an Object Node from an MTConnect Component element:

1) The BrowseName is the 'name' attribute;
2) The Description is the CDATA from the 'Description' element;
3) The TypeDefinitionId is the NodeId of the *MTComponentType* identified by the QName;
4) Any DataItems are added as children of the DataItems or Conditions folders;
5) Any sub-Components are added as targets of HasComponent and HasNotifier references;

The QName of a Component element identifies the subtype of *MTComponentType.* If this QName is not already recognized by the UA Server it can create a new ObjectType Node by following these design rules:

1) The BrowseName is the QName with 'ComponentType' appended;
2) Add a HasSubtype reference from *MTComponentType;*

The NodeId of the dynamically created ObjectType is server dependent. This specification will define NodeIds for the Component types known at the time this document was written.

5.3 Device

An MTConnect Device is the container for all of the components for a piece of equipment. Each Device is mapped to an instance of an Object Node with a TypeDefinition that is *MTDeviceType* or one of its subtypes. The *MTDeviceType* ObjectType is defined in Table 3.

Table 3 – *MTDeviceType* Definition

Attribute	**Value**				
BrowseName	MTDeviceType				
IsAbstract	False				
Reference	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Inherit the *Properties* of the *DeviceType* which is defined in [UA DI]					
HasProperty	Variable	Uuid	String	PropertyType	Mandatory
HasProperty	Variable	NativeName	String	PropertyType	Optional
HasProperty	Variable	Station	String	PropertyType	Optional
HasComponent	Object	DataItems		FunctionalGroupType	Mandatory
HasComponent	Object	Conditions		FunctionalGroupType	Optional
HasComponent	Variable	Availability	String	AvailabilityItemType	Mandatory
HasComponent HasNotifier	Object	<server defined>		MTComponentType	Optional (0..N)

The Uuid, NativeName and Station properties correspond to their definitions in [MT Part 2].

The DataItems Functional Group is a container for the MTConnect Sample and Event DataItems supported by the Device.

The Conditions Functional Group is a container for the MTConnect Condition DataItems supported by the Device.

The Availability Node is the target of a HasComponent reference from the Device and the DataItems folder. This allows clients to treat it as any other DataItem.

The DI DeviceType defines several properties for device metadata. The Description element may contain attributes which can be used to set these properties. Specifically:

1) The value of the Manufacturer property is the ‘manufacturer’ attribute;
2) The value of the SerialNumber property is the ‘serialNumber’ attribute;

Any Components of the Device are added as targets of HasComponent and HasNotifier references.

5.4 DataItems

5.4.1 General

An MTConnect DataItem describes a piece of information that can be collected from a component. Each DataItem is mapped to an instance of a Variable Node with a TypeDefinition that is a subtype of *MTDataItemType*. The *MTDataItemType* VariableType is defined in Table 4.

Table 4 – *MTDataItemType* Definition

Attribute	**Value**				
BrowseName	MTDataItemType				
IsAbstract	True				
DataType	BaseDataType				
ValueRank	Scalar				
Reference	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Inherit the *Properties* of the *DataItemType* which is defined in [UA Part 8].					
HasProperty	Variable	CoordinateSystem	String	PropertyType	Optional
HasProperty	Variable	Source	String	PropertyType	Optional

The CoordinateSystem property corresponds to the attribute defined in [MT Part 2].

The Source property is the contents of the Source element.

Each MTConnect DataItem defines a category, type and subtype attributes. These three values uniquely identify a subtype of *MTDataItemType*. The mappings for SAMPLE and EVENT categories are:

- SAMPLE: *MTSampleDataItemType*
- EVENT: *MTEventDataItemType*

The mappings for CONDITION category is discussed in the section on Conditions below.

The type is mapped to a subtype of the category. The following design rules are followed when creating the VariableType Node from type attribute:

1) The BrowseName is camel case form of type with ‘ItemType’ appended;
2) Add a HasSubtype reference from VariableType for the category;

‘Camel Case’ refers to the convention where multiple words are combined to make a symbol by removing spaces and capitalizing the first letter of each word (e.g. ‘PathFeedrate’).

The subtype is mapped to a subtype of the VariableType for the type. The following design rules are followed when creating the VariableType Node from subtype attribute:

1) The BrowseName is camel case form of subtype with the BrowseName of the parent type appended (e.g. ActualRotaryVelocityType);
2) Add a HasSubtype reference from VariableType for the type;

The NodeId of the dynamically created VariableType are server dependent. This specification will define NodeIds for the DataItem types known at the time this document was written.

Figure 13 depicts an MTConnect device with some Components, DataItems and their TypeDefinitions.

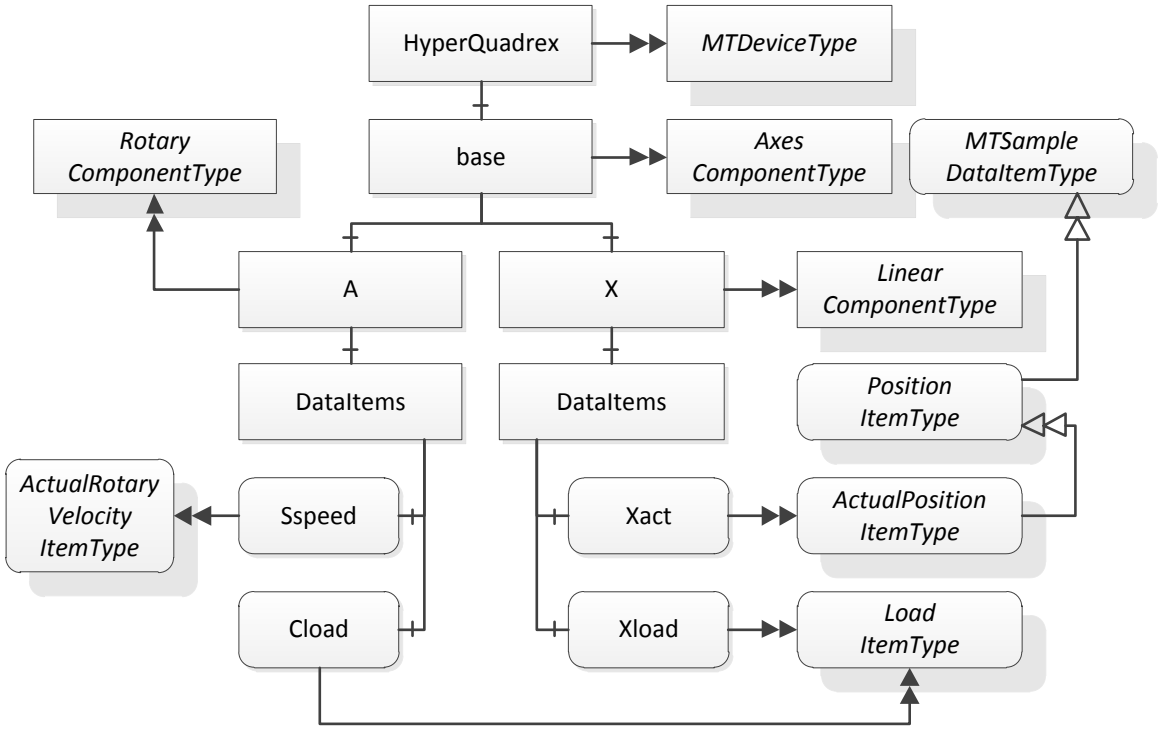

Figure 13 – MTConnect DataItems in the UA Address Space

5.4.2 Sample DataItems

The *MTSampleDataItemType* VariableType is defined in Table 5.

Table 5 – *MTSampleDataItemType* Definition

Attribute	**Value**				
BrowseName	MTSampleDataItemType				
IsAbstract	False				
DataType	Number				
ValueRank	Scalar				
Reference	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Inherit the *Properties* of the *MTDataItemType* which is defined in Table 4.					
HasProperty	Variable	NativeUnits	String	PropertyType	Optional
HasProperty	Variable	NativeScale	Float	PropertyType	Optional
HasProperty	Variable	EURange	Range	PropertyType	Optional
HasProperty	Variable	EngineeringUnits	EUInformation	PropertyType	Optional
HasProperty	Variable	ValuePrecision	Double	PropertyType	Optional

The Units, NativeUnits, NativeScale and CoordinateSystem properties correspond to the attribute definitions in [MT Part 2].

The elements of the EnumStrings array are the contents of the Constraints/Value element.

The EURange property comes from the Constraints/Minimum and Constraints/Maximum elements.

The EngineeringUnits property is the same as the Units property except it qualifies the unit names with the MTConnect NamespaceURI.

The ValuePrecision comes from the significantDigits attribute.

5.4.3 Event DataItems

The *MTEventDataItemType* VariableType is defined in Table 6.

Table 6 – *MTEventDataItemType* Definition

Attribute	**Value**				
BrowseName	MTEventDataItemType				
IsAbstract	False				
DataType	String				
ValueRank	Scalar				
Reference	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Inherit the *Properties* of the *MTDataItemType* which is defined in Table 4.					
HasProperty	Variable	EnumStrings	LocalizedText[]	PropertyType	Optional

The elements of the EnumStrings array are the contents of the Constraints/Value element.

An example the mapping from MTConnect DataItems to OPC UA Nodes is shown in Figure 13.

5.4.4 Conditions

MTConnect Conditions are DataItems which report the state of health or functionality for a component. Semantically they are equivalent to Conditions in OPC UA which are Object Nodes

that produce Events. OPC UA Conditions also appear in the address space and can be accessed like any other Variable.

Each MTConnect Condition is represented by an Object Node with a TypeDefinition *MTConditionType.* The *MTConditionType* ObjectType is defined in Table 7.

Table 7 – *MTConditionType* Definition

Attribute	**Value**				
BrowseName	MTConditionType				
IsAbstract	False				
Reference	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Inherit the children of the *ConditionType* which is defined in [UA Part 9].					
HasComponent	Variable	CurrentState	String	BaseDataVariableType	Mandatory
HasComponent	Variable	ActiveState	LocalizedText	TwoStateVariableType	Mandatory
HasComponent	Object	LimitState		ExclusiveLimit StateMachineType	Optional
HasProperty	Variable	NativeCode	String	PropertyType	Optional
HasProperty	Variable	NativeSeverity	String	PropertyType	Optional

The NativeSeverity and NativeCode properties correspond to the attribute definitions in [MT Part 3].

The CurrentState is the MTConnect Condition state: Unavailable, Normal, Warning or Fault. The Severity property (inherited from ConditionType) should set based on the value of this Variable according to these rules:

- Unavailable: 1
- Normal: <= 100
- Warning: > 100 && <=500
- Fault: > 500

The ActiveState Variable indicates whether the condition is in a state that requires attention. This Variable is ‘Active’ if the MTConnect Condition state is Warning or Fault. If the MTConnect Condition state is Normal or Unavailable this Variable is ‘Inactive’.

The LimitState is used to specify the MTConnect Condition qualifier attribute. If the qualifier is ‘HIGH’ the LimitState/CurrentState Variable is ‘High’. If the qualifier is ‘LOW’ then the LimitState/CurrentState Variable is ‘Low’.

The Message property (inherited from BaseEventType) contains contents of the MTConnect Condition element. If no contents were provided the Server shall choose a suitable default message.

The BrowseName of the Condition is value of the name attribute for the DataItem. If the name attribute is missing the camel case version of the type attribute is used instead.

Many MTConnect Conditions may be related to an instance of *MTSampleDataItemType* which provides access to the value that can trigger the Condition. If these relationships are known the Server should provide a HasCondition reference from the Variable for the DataItem to the corresponding Condition Objects. Figure 14 depicts an example of a component with a Position DataItem with an associated Condition.

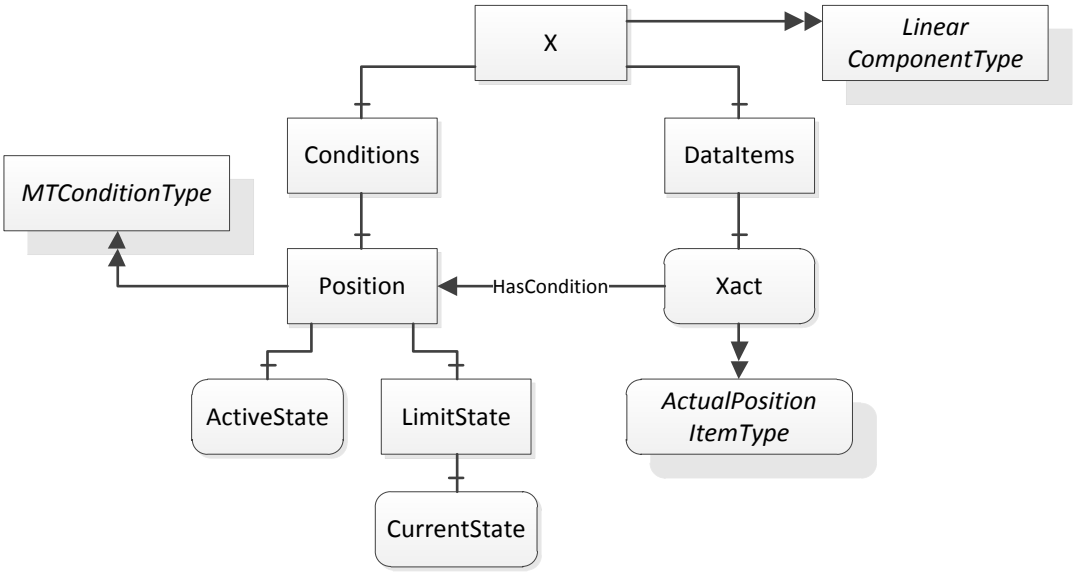

Figure 14 – MTConnect Conditions in the UA Address Space

The ConditionClassId property (inherited from ConditionType) is used to represent the type attribute from the DataItem. Each possible type is used to create a subtype of *MTConditionClassType* with the BrowseName equal to camel case form of the type. Table 8 defines the *MTConditionClassType* ObjectType.

Table 8 – *MTConditionClassType* Definition

Attribute	**Value**				
BrowseName	MTConditionClassType				
IsAbstract	True				
Reference	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Inherit the children of the *BaseConditionClassType* which is defined in [UA Part 9].					

5.5 Streams

MTConnect streams are the mechanism used to receive updates to DataItems. In OPC UA Subscriptions are designed to provide the same functionality.

There are two types of subscriptions in OPC UA: DataChange and Event.

A DataChange Subscription can be created with the following steps:

- Browse the Address Space and read the NodeIds for the DataItems of interest;
- Create a Subscription;
- Create a MonitoredItem for each DataItem of interest;
- Tell the Server to send Notifications when one or more of the DataItems change;

Each Notification has the current value for a DataItem, the Timestamp and StatusCode.

The MonitoredItem allows the Client to specify a different SamplingInterval and QueueSize for each DataItem. The MinimumSamplingInterval attribute for each DataItem specifies the fastest supported SamplingInterval. If a UA Server is a front end for a MT Agent the MinimumSamplingInterval is the value of the samplingRate attribute which is on the Device or Component elements.

The PublishingInterval for a Subscription controls how frequently Notifications are returned to the Client. If this value is longer than SamplingInterval then the Server will buffer changes and return multiple Notifications in one message.

An Event Subscription can be created with the following steps:

- Browse the Address Space and read the NodeIds for the Devices or Components of interest;
- Create a Subscription;
- Create a MonitoredItem for each Device or Component of interest;
- Select the fields to return for each event;
- Specify additional filter criteria for the events to return;
- Tell the Server to send Notifications when the state of one or more events occur;

Events in OPC UA propagate up the HasNotifier hierarchy. This means subscribing to a parent Node in a hierarchy (i.e. a Device) will request events for all Components under that Node. If a Client wants to receive all events for all devices it can subscribe to the Server node.

Each Condition Event is a snapshot of the *MTConditionType* Object. The Client must select the fields from the Condition that it wishes to receive in the update by specifying the BrowsePath (a sequence of BrowseNames). The values of the variables in the address space contain the values from the last event sent. These values can be subscribed to like any other DataType but there is no guarantee that the Client will receive a synchronized snapshot of the Condition.

Table 9 provides an example of an Event produced from the following Condition element:

<Fault type="MOTION_PROGRAM" dataitemid="cc2" sequence="25" qualifier=”HIGH” nativeCode="PR1123" timestamp="...">Syntax error on line 107</Fault>

Table 9 – An Example Condition Event

BrowsePath	Value	Notes
<blank>	<server defined>	Identifies the Condition Node in the Address space.
EventId	<server defined>	Equivalent to the sequence number but an opaque value.
ConditionName	MotionProgram	From the name attribute,
ConditionClassName	MotionProgram	From the type attribute.
Message	Syntax error on line 107	The contents of the Condition element.
Time	…	The timestamp attribute.
Severity	700	Inferred from the current state.
CurrentState	Fault	From the QName.
ActiveState/Id	True	Inferred from the QName.
LimitState/CurrentState	High	From the qualifier attribute.
NativeCode	PR1123	From the nativeCode attribute.

A Client can request any or all of the fields defined by the children of the *MTConditionType ObjectType.*

OPC UA Event Filters allow for SQL like filtering on events. For example, the following filter would select only those events for MOTION_PROGRAMS in the Fault state:

(ConditionClassName = ‘MotionProgram’) AND (CurrentState = ‘Fault’)

OPC UA does not define syntax for filters. Instead it provides a generic structure that stores a pre-parsed expression tree. Client applications are free to use whatever syntax they feel is appropriate.