MTConnect® Standard

Part 5.0 – Interface Interaction Model

Version 2.2.0

Prepared for: MTConnect Institute
Prepared from: `MTConnectSysMLModel.xml`
Prepared on: August 8, 2023

August 8, 2023

MTConnect Specification and Materials

The normative XMI is located at the following URL: `MTConnectSysMLModel.xml`

Table of Contents

1 Purpose of This Document **2**

2 Terminology and Conventions **3**
2.1 General Terms . . . 3
2.2 Information Model Terms . . . 10
2.3 Protocol Terms . . . 11
2.4 HTTP Terms . . . 13
2.5 XML Terms . . . 15
2.6 MTConnect Terms . . . 16
2.7 Acronyms . . . 17
2.8 MTConnect References . . . 29

3 Interface Interaction Model **30**
3.1 Interfaces Architecture . . . 30
3.2 Request and Response Information Exchange . . . 32
3.3 Interface . . . 33
3.3.1 Commonly Observed DataItem Types for Interface . . . 34

4 Interfaces for Device and Observation Information Models **35**
4.1 Interface Types . . . 35
4.1.1 BarFeederInterface . . . 36
4.1.2 ChuckInterface . . . 36
4.1.3 DoorInterface . . . 36
4.1.4 MaterialHandlerInterface . . . 36
4.2 Data for Interface . . . 37
4.2.1 References for Interface . . . 37
4.3 DataItem Types for Interface . . . 38
4.3.1 Specific Data Items for the Interaction Model for Interface . . . 38
4.3.2 CloseChuck . . . 39
4.3.3 CloseDoor . . . 41
4.3.4 InterfaceState . . . 41
4.3.5 MaterialChange . . . 42
4.3.6 MaterialFeed . . . 42
4.3.7 MaterialLoad . . . 43
4.3.8 MaterialRetract . . . 43
4.3.9 MaterialUnload . . . 43
4.3.10 OpenChuck . . . 44
4.3.11 OpenDoor . . . 44
4.3.12 PartChange . . . 45

5 Operation and Error Recovery **46**

5.1 Request and Response Failure Handling and Recovery 46
5.1.1 Responder Fails Immediately 47
5.1.2 Responder Fails While Providing a Service 48
5.1.3 Requester Failure During a Service Request 49
5.1.4 Requester Changes to an Unexpected State While Responder is Providing a Service 50
5.1.5 Responder Changes to an Unexpected State While Providing a Service 51
5.1.6 Responder or Requester Become UNAVAILABLE or Experience a Loss of Communication 52

6 Profile **55**
6.1 DataTypes 55
6.1.1 boolean 55
6.1.2 ID 55
6.1.3 string 55
6.1.4 float 55
6.1.5 datetime 56
6.1.6 integer 56
6.1.7 xlinktype 56
6.1.8 xslang 56
6.1.9 SECOND 56
6.1.10 IDREF 56
6.1.11 xlinkhref 56
6.1.12 x509 57
6.1.13 int32 57
6.1.14 int64 57
6.1.15 version 57
6.1.16 uint32 57
6.1.17 uint64 57
6.1.18 binary 57
6.1.19 double 58
6.2 Stereotypes 58
6.2.1 organizer 58
6.2.2 deprecated 58
6.2.3 extensible 58
6.2.4 informative 58
6.2.5 valueType 58
6.2.6 normative 58
6.2.7 observes 59

Appendices **61**

A Bibliography 61

Table of Figures

Figure 1: Data Flow Architecture for Interfaces 31
Figure 2: Request and Response Overview 33
Figure 3: Interfaces in Entity Hierarchy 35
Figure 4: Request State Machine 39
Figure 5: Response State Machine 40
Figure 6: Success Scenario 46
Figure 7: Responder Fails Immediately 47
Figure 8: Responder Fails While Providing a Service 48
Figure 9: Requester Fails During a Service Request 49
Figure 10:Requester Makes Unexpected State Change 51
Figure 11:Responder Makes Unexpected State Change 52
Figure 12:Requester - Responder Communication Failure 1 53
Figure 13:Requester - Responder Communication Failure 2 54
Figure 14:DataTypes 55
Figure 15:Stereotypes 60

List of Tables

Table 1: Commonly Observed DataItem Types for Interface 34

1 Purpose of This Document

This document, *MTConnect Standard: Part 5.0 - Interface Interaction Model* of the MTConnect Standard, defines a structured data model used to organize information required to coordinate inter-operations between pieces of equipment.

This data model is based on an *interaction model* that defines the exchange of information between pieces of equipment and is organized in the MTConnect Standard by `Interfaces`.

`Interfaces` is modeled as an extension to the *Device Information Model* and *Observation Information Model*. `Interfaces` leverages similar rules and terminology as those used to describe a component in the *Device Information Model*. `Interfaces` also uses similar methods for reporting data to those used in the *MTConnectStreams Response Document*.

As defined in *MTConnect Standard: Part 2.0 - Device Information Model*, `Interfaces` *organizes* the `Interface` types (see Figure 3). Each individual `Interface` contains data associated with the corresponding *interface*.

> Note: See *MTConnect Standard: Part 2.0 - Device Information Model* and *MTConnect Standard: Part 3.0 - Observation Information Model* of the MTConnect Standard for information on how `Interfaces` is structured in the *response documents* which are returned from an *agent* in response to a *probe request*, *sample request*, or *current request*.

2 Terminology and Conventions

Refer to *MTConnect Standard Part 1.0 - Fundamentals* for a dictionary of terms, reserved language, and document conventions used in the MTConnect Standard.

2.1 General Terms

adapter

optional piece of hardware or software that transforms information provided by a piece of equipment into a form that can be received by an *agent*.

agent

software that collects data published from one or more piece(s) of equipment, organizes that data in a structured manner, and responds to requests for data from client software systems by providing a structured response in the form of a *response document* that is constructed using the *semantic data model* of a Standard.

alarm limit

limit used to trigger warning or alarm indicators.

application

software or a program that is specific to the solution of an application problem. *Ref ISO/IEC 20944-1:2013*

archetype

archetype provides the requirements, constraints, and common properties for a type of *Asset*.

asset buffer

buffer for *Assets*.

attachment

connection by which one thing is associated with another.

buffer

section of an *agent* that provides storage for information published from pieces of equipment.

cartesian coordinate system

3D orthogonal coordinate system [(]ISO/IEC 19794-5:2011en).

characteristic

control placed on an element of a *feature* such as its size, location, or form, which may be a specification limit, a nominal with tolerance, or some other numerical or non-numerical control. *Ref QIF 3.0 3.4.29. Ref AS9102-B*.

client

application that sends *request* for information to an *agent*.

Note: Examples include software applications or a function that implements the *request* portion of an *interface interaction model*.

combined standard uncertainty

standard uncertainty of the result of a measurement when that result is obtained from the values of a number of other quantities, equal to the positive square root of a sum of terms, the terms being the variances or covariances of these other quantities weighted according to how the measurement result varies with changes in these quantities. *Ref JCGM 100:2008 2.3.4*

controlled vocabulary

restricted set of values that may be published for an observation.

data dictionary

listing of standardized terms and definitions used in *MTConnect Information Model*.

data model

organizes elements of data and standardizes how they relate to one another and to the properties of real-world entities.

data set

key-value pairs where each entry is uniquely identified by the *key*.

data source

piece of equipment that can produce data that is published to an *agent*.

deprecated

indication that specific content in an *MTConnect Document* is currently usable but is regarded as being obsolete or superseded.

deprecation warning

indication that specific content in an *MTConnect Document* may be changed to *deprecated* in a future release of the standard.

document

piece of written, printed, or electronic matter that provides information or evidence that serves as an official record.

electric current

rate of flow of electric charge.

element

constituent part or a basic unit of identifiable and definable data.

extensible

ability for an implementer to extend *MTConnect Information Model* by adding content not currently addressed in the MTConnect Standard.

feature

topological entity(ies) or design requirements related to a geometric model. *Ref QIF 3.0-3.4.59*

force

push or pull on a mass which results in an acceleration.

heartbeat

function that indicates to a *client* that the communications connection to an *agent* is still viable during times when there is no new data available to report often referred to as a "keep alive" message.

higher level

nested element that is above a lower level element.

implementation

specific instantiation of the MTConnect Standard.

information model

rules, relationships, and terminology that are used to define how information is structured.

instance

describes a set of *streaming data* in an *agent*. Each time an *agent* is restarted with an empty *buffer*, data placed in the *buffer* represents a new *instance* of the *agent*.

interaction model

model that defines how information is exchanged across an *interface* to enable interactions between independent systems.

interface

means by which communication is achieved between independent systems.

key

unique identifier in a *key-value pair* association.

key-value pair

association between an identifier referred to as the *key* and a value which taken together create a *key-value pair*.

location

place or named space associated with an object or that can be occupied by an object.

lower camel case

first word is lowercase and the remaining words are capitalized and all spaces between words are removed.

lower level

nested element that is below a higher level element.

lower limit

lower conformance boundary for a variable.

lower warning

lower boundary indicating increased concern and supervision may be required.

major

identifier representing a consistent set of functionalities defined by the MTConnect Standard.

maximum

numeric upper constraint.

message

communication in writing, in speech, or by signals.

metadata

data that provides information about other data.

minimum

numeric lower constraint.

minor

identifier representing a specific set of functionalities defined by the MTConnect Standard.

nominal

ideal or desired value for a variable.

organize

act of containing and owning one or more elements.

organizer

entity that *organizes* one or more elements.

parameter

variable that must be given a value during the execution of a program or a communications command.

part

discrete item that has both defined and measurable physical characteristics including mass, material, and features, and is created by applying one or more manufacturing process steps to a workpiece

pascal case

first letter of each word is capitalized and the remaining letters are in lowercase. All space is removed between letters

persistence

method for retaining or restoring information.

position

location that is represented by a point in space relative to a reference.

probe

instrument commonly used for measuring the physical geometrical characteristics of an object.

profile

extends a reference metamodel (such as Unified Modeling Language (UML)) by allowing to adapt or customize the metamodel with constructs that are specific to a particular domain, platform, or a software development method.

requester

entity that initiates a *request* for information in a communications exchange.

reset

act of reverting back the accumulated value or statistic to their initial value.

> Note: An *Observation* with a *data set* representation removes all *key-value pairs*, setting the *data set* to an empty set.

responder

entity that responds to a *request* for information in a communications exchange.

response document

electronic *document* published by an *MTConnect Agent* in response to a *probe request*, *current request*, *sample request* or *asset request*.

revision

supplemental identifier representing only organizational or editorial changes to a *minor* version document with no changes in the functionality described in that document.

schema

definition of the structure, rules, and vocabularies used to define the information published in an electronic document.

semantic data model

methodology for defining the structure and meaning for data in a specific logical way that can be interpreted by a software system.

sensing element

mechanism that provides a signal or measured value.

sequence number

primary key identifier used to manage and locate a specific piece of *streaming data* in an *agent*.

specification limit

limit defining a range of values designating acceptable performance for a variable.

spindle

mechanism that provides rotational capabilities to a piece of equipment.

Note: Typically used for either work holding, materials or cutting tools.

standard

document established by consensus that provides rules, guidelines, or characteristics for activities or their results.. *Ref ISO/IEC Guide 2:2004*

standard uncertainty

uncertainty of the result of a measurement expressed as a standard deviation. *Ref JCGM 100:2008 2.3.1*

stereotype

defines how an existing UML metaclass may be extended as part of a *profile*.

subtype

secondary or subordinate type of categorization or classification of information.

table

two dimensional set of values given by a set of *key-value pairs table entries*.

table cell

subdivision of a *table entry* representing a singular value.

table entry

subdivision of a *table* containing a set of *key-value pairs* representing *table cells*.

top level

element that represents the most significant physical or logical functions of a piece of equipment.

type

classification or categorization of information.

uncertainty

uncertainty (of measurement) parameter, associated with the result of a measurement, that characterizes the dispersion of the values that could reasonably be attributed to the measurand. *Ref JCGM 100:2008 2.2.3*

Note: Use of the term uncertainty refers to uncertainty of measurement.

upper limit

upper conformance boundary for a variable.

upper warning

upper boundary indicating increased concern and supervision may be required.

version

unique identifier of the administered item. *Ref ISO/IEC 11179-:2015*

2.2 Information Model Terms

Asset Information Model

information model that provides semantic models for *Assets*.

Device Information Model

information model that describes the physical and logical configuration for a piece of equipment and the data that may be reported by that equipment.

Error Information Model

information model that describes the *response document* returned by an *agent* when it encounters an error while interpreting a *request* for information from a *client* or when an *agent* experiences an error while publishing the *response* to a *request* for information.

MTConnect Information Model

information model that defines the semantics of the MTConnect Standard.

Observation Information Model

information model that describes the *streaming data* reported by a piece of equipment.

2.3 Protocol Terms

asset request

HTTP Request to the *agent* regarding *Assets*.

current request

request to an *agent* to produce an *MTConnectStreams Response Document* containing the *Observation Information Model* for a snapshot of the latest observations at the moment of the *request* or at a given *sequence number*.

data streaming

method for an *agent* to provide a continuous stream of information in response to a single *request* from a *client*.

MTConnect Request

request for information issued from a *client* to an *MTConnect Agent*.

MTConnect Response Document

response document published by an *MTConnect Agent*.

MTConnectAssets Response Document

response document published by an *MTConnect Agent* in response to an *asset request*.

MTConnectDevices Response Document

response document published by an *MTConnect Agent* in response to a *probe request*.

MTConnectErrors Response Document

response document published by an *MTConnect Agent* whenever it encounters an error while interpreting an *MTConnect Request*.

MTConnectStreams Response Document

response document published by an *MTConnect Agent* in response to a *current request* or a *sample request*.

probe request

request to an *agent* to produce an *MTConnectDevices Response Document* containing the *Device Information Model*.

protocol

set of rules that allow two or more entities to transmit information from one to the other.

publish

sending of messages in a *publish and subscribe* pattern.

publish and subscribe

asynchronous communication method in which messages are exchanged between applications without knowing the identity of the sender or recipient.

> Note: In the MTConnect Standard, a communications messaging pattern that may be used to publish *streaming data* from an *agent*.

request

communications method where a *client* transmits a message to an *agent*. That message instructs the *agent* to respond with specific information.

request and response

communications pattern that supports the transfer of information between an *agent* and a *client*.

response

response *interface* which responds to a *request*.

sample request

request to an *agent* to produce an *MTConnectStreams Response Document* containing the *Observation Information Model* for a set of timestamped observations made by *Components*.

streaming data

observations published by a piece of equipment defined by the equipment metadata.

subscribe

receiving messages in a *publish and subscribe* pattern.

transport protocol

set of capabilities that provide the rules and procedures used to transport information between an *agent* and a client software application through a physical connection.

2.4 HTTP Terms

HTTP Body

data bytes transmitted in an HTTP transaction message immediately following the headers. *Ref IETF:RFC-2616*

HTTP Error Message

response provided by an *agent* indicating that an *HTTP Request* is incorrectly formatted or identifies that the requested data is not available from the *agent*. *Ref IETF:RFC-2616*

HTTP Header

header of either an *HTTP Request* from a *client* or an *HTTP Response* from an *agent*. *Ref IETF:RFC-2616*

HTTP Header Field

components of the header section of request and response messages in an HTTP transaction. *Ref IETF:RFC-2616*

HTTP Message

consist of requests from client to server and responses from server to client. *Ref IETF:RFC-2616*

> Note: In MTConnect Standard, it describes the information that is exchanged between an *agent* and a *client*.

HTTP Messaging

interface for information exchange functionality. *Ref IETF:RFC-2616*

HTTP Method

portion of a command in an *HTTP Request* that indicates the desired action to be performed on the identified resource; often referred to as verbs. *Ref IETF:RFC-2616*

HTTP Query

portion of a request for information that more precisely defines the specific information to be published in response to the request. *Ref IETF:RFC-2616*

HTTP Request

request message from a client to a server includes, within the first line of that message, the method to be applied to the resource, the identifier of the resource, and the protocol version in use. *Ref IETF:RFC-2616*

Note: In MTConnect Standard, a request issued by a *client* to an *agent* requesting information defined in the *HTTP Request Line*.

HTTP Request Line

begins with a method token, followed by the Request-URI and the protocol version, and ending with CRLF. A CRLF is allowed in the definition of TEXT only as part of a header field continuation. *Ref IETF:RFC-2616*

Note: the first line of an *HTTP Request* describing a specific *response document* to be published by an *agent*.

HTTP Request Method

indicates the method to be performed on the resource identified by the Request-URI. *Ref IETF:RFC-2616*

HTTP Request URI

Uniform Resource Identifier that identifies the resource upon which to apply the request. *Ref IETF:RFC-2616*

HTTP Response

after receiving and interpreting a request message, a server responds with an HTTP response message. *Ref IETF:RFC-2616*

Note: In MTConnect Standard, the information published from an *agent* in reply to an *HTTP Request*.

HTTP Server

server that accepts *HTTP Request* from *client* and publishes *HTTP Response* as a reply to those *HTTP Request*. *Ref IETF:RFC-2616*

HTTP Status Code

3-digit integer result code of the attempt to understand and satisfy the request. *Ref IETF:RFC-2616*

HTTP Version

version of the HTTP protocol. *Ref IETF:RFC-2616*

2.5 XML Terms

abstract element

element that defines a set of common characteristics that are shared by a group of elements. An abstract entity cannot appear in a document. In a specific implementation, an abstract entity is replaced by a derived element that is itself not an abstract entity. The characteristics for the derived element are inherited from the abstract entity.

attribute

additional information or property for an *element*.

child element

element of a data modeling structure that illustrates the relationship between itself and the higher-level *parent element* within which it is contained.

document body

portion of the content of an *MTConnect Response Document* that is defined by the relative *MTConnect Information Model*. The *document body* contains the *structural elements* and *Observations* or *DataItems* reported in a *response document*.

document header

portion of the content of an *MTConnect Response Document* that provides information from an *agent* defining version information, storage capacity, protocol, and other information associated with the management of the data stored in or retrieved from the *agent*.

element name

descriptive identifier contained in both the `start-tag` and `end-tag` of an XML element that provides the name of the element.

namespace

organizes information into logical groups.

parent element

element of a data modeling structure that illustrates the relationship between itself and the lower-level *child element*.

root element

first *structural element* provided in a *response document* encoded using XML.

structural element

element that organizes information that represents the physical and logical parts and sub-parts of a piece of equipment.

XML Document

structured text file encoded using Extensible Markup Language (XML).

XML Schema

schema defining a specific document encoded in XML.

2.6 MTConnect Terms

Asset

asset that is used by the manufacturing process to perform tasks.

Note 1 to entry: An *Asset* relies upon an *Device* to provide observations and information about itself and the *Device* revises the information to reflect changes to the *Asset* during their interaction. Examples of *Assets* are cutting tools, Part Information, Manufacturing Processes, Fixtures, and Files.

Note 2 to entry: A singular `assetId`,`Asset` uniquely identifies an *Asset* throughout its lifecycle and is used to track and relate the *Asset* to other *Devices* and entities.

Note 3 to entry: *Assets* are temporally associated with a device and can be removed from the device without damage or alteration to its primary functions.

Component

engineered system part of a *Device* composed of zero or more *Components*

Composition

Component belonging to a *Component* and not composed of any *Components*.

Configuration

configuration for a *Component*

DataItem

observable observed by a *Component* that may make *Observations*

Device

Component not belonging to any *Component* that may have assets

MTConnect Agent

agent for the *MTConnect Information Model*.

MTConnect Document

document that represents a Part(s) of the MTConnect Standard.

MTConnect Event

observation of either a state or discrete value of the *Component*.

MTConnect Interface

interaction model for interoperability between pieces of equipment.

Observation

observation that provides telemetry data for a *DataItem*.

2.7 Acronyms

2D

two-dimensional

3D

three-dimensional

AI

artificial intelligence

ALM

application lifecycle management

AMT

The Association for Manufacturing Technology

ANSI

American National Standards Institute

AP

Application Protocol

API

application programming interface

ASME

American Society of Mechanical Engineers

ASTM

American Society for Testing and Materials

AWS

American Welding Society

BDD

block definition diagram

BOM

bill of materials

BST

Board on Standardization and Testing

C&R

cause and remedy

CA

certificate authority

CAD

computer-aided design

CAE

computer-aided engineering

CAI

computer-aided inspection

CAM

computer-aided manufacturing

CAx

computer-aided technologies

CDATA

Character Data

CFD

computational fluid dynamics

CM

configuration management

CMS

coordinate-measurement system

CNC

Computer Numerical Controller

CNRI

Corporation for National Research Initiatives

CPM

Core Product Model

CPM2

Revised Core Product Model

CPSC

Consumer Product Safety Commission

cUAV

configurable unmanned aerial vehicle

DARPA

Defense Advanced Research Projects Agency

DER

designated-engineering representative

DFM

design for manufacturing

DLA

Defense Logistics Agency

DMC

digital manufacturing certificate

DMSC

Dimensional Metrology Standards Consortium

DNS

Domain Name System

DoD

U.S. Department of Defense

DOI

Distributed Object Identifier

DRM

digital rights management

ECR

engineering change request

ERP

enterprise resource planning

FAA

Federal Aviation Administration

FAIR

first article inspection reporting

FDA

Food and Drug Administration

FEA

finite-element analysis

GD&T

geometric dimensions and tolerances

GID

global identifier

HMI

Human Machine Interface

HTML

Hypertext Markup Language

HTTP

Hypertext Transfer Protocol

HTTPS

Hypertext Transfer Protocol over Secure Sockets Layer

I/O

in-out

ID

identifier

IEEE

Institute of Electrical and Electronics Engineers

IIoT

industrial internet of things

INCOSE

International Council on Systems Engineering

IP

intellectual property

ISO

International Standards Organization

ISS

International Space Station

ISV

Independent Software Vendor

IT

information technology

ITU-T

Telecommunication Standardization Sector of the International Telecommunication Union

JSON

JavaScript Object Notation

JT

Jupiter Tesselation

LHS

Lifecycle Handler System

LIFT

Lifecycle Information Framework and Technology

LOI

Lifecycle Object Identifier

MAC

media access control

MADE

Manufacturing Automation and Design Engineering

MBD

model-based definition

MBE

Model-Based Enterprise

MBI

model-based inspection

MBM

model-based manufacturing

MBSD

model-based standards development

MBSE

model-based systems engineering

MEDALS

Military Engineering Data Asset Locator System

MES

manufacturing execution system

MOI

manufacturing object identifier

MOM

Message Orienged Middleware

MQTT

Message Queuing Telemetry Transport

MTC

Manufacturing Technology Centre

NASA

National Aeronautics and Space Administration

NC

numerical control

NIST

National Institute of Standards and Technology

NMTOKEN

Name Token

NNMI

National Network of Manufacturing Innovation

NSF

National Science Foundation

620 ***NTSC***

621 National Transportation Safety Board

622 ***OASIS***

623 Organization for the Advancement of Structured Information Standards

624 ***ODI***

625 Open Data Institute

626 ***OEM***

627 original equipment manufacturer

628 ***OOI***

629 Ocean Observatories Initiative

630 ***OPC***

631 OLE for Process Control

632 ***OSLC***

633 Open Services for Lifecycle Collaboration

634 ***OSTP***

635 Office of Science and Technology Policy

636 ***OT***

637 operational technology

638 ***OWL***

639 Ontology Web Language

640 ***PDF***

641 Portable Document Format

642 ***PDM***

643 product-data management

644 ***PDQ***

645 product-data quality

646 ***PHM***

647 prognosis and health monitoring

PI

principal investigator

PLC

Programmable Logic Controller

PLCS

Product Life Cycle Support

PLM

product lifecycle management

PLOT

product lifecycle of trust

PMI

product and manufacturing information

PMS

Production Management System

PRC

Product Representation Compact

PSI

Physical Science Informatics

PTAB

Primary Trustworthy Digital Repository Authorization Body Ltd.

QIF

Quality Information Framework

QMS

quality management system

QName

Qualified Name

RDF

Resource Description Framework

REST

Representational State Transfer

RII

receiving and incoming inspection

S/MIME

Secure/Multipurpose Internet Mail Extensions

SaaS

software-as-a-service

SAML

Security Assertion Markup Language

SC

Standards Committee

SCADA

Supervisory Control And Data Acquisition

SDO

Standards Development Organization

SFTP

Secure File Transfer Protocol

SKOS

Simple Knowledge Organization System

SLH

system lifecycle handler

SLR

systematic literature review

SME

small-to-medium enterprise

SMOPAC

Smart Manufacturing Operations Planning and Control

SMS Test Bed

Smart Manufacturing Systems Test Bed

SOA

service-oriented architecture

SPMM

semantic-based product metamodel

SSL

Secure Sockets Layer

STEP

Standard for the Exchange of Product Model Data

STEP AP242

Standard for the Exchange of Product Model Data Application Protocol 242

STL

Stereolithography

SysML

Systems Modeling Language

TCP/IP

Transmission Control Protocol/Internet Protocol

TDP

technical data package

TLS

Transport Layer Security

TSM

Total System Model

UA

Unified Architecture

UAL

Unified Architecture Language

UML
: Unified Modeling Language

URI
: Uniform Resource Identifier

URL
: Uniform Resource Locator

URN
: Uniform Resource Name

UTC
: Coordinated Universal Time

UUID
: Universally Unique Identifier

V&V
: verification and validation

W3C
: World Wide Web Consortium

WSN
: Wirth Syntax Notation

WWW
: World Wide Web

X.509-PKI
: Public Key Infrastructure

X.509-PMI
: Privilege Management Infrastructure

XML
: Extensible Markup Language

XPath
: XML Path Language

XSD
: XML Schema Definitions

2.8 MTConnect References

[MTConnect Part 1.0] *MTConnect Standard Part 1.0 - Fundamentals*. Version 2.0.

[MTConnect Part 2.0] *MTConnect Standard: Part 2.0 - Device Information Model*. Version 2.0.

[MTConnect Part 3.0] *MTConnect Standard: Part 3.0 - Observation Information Model*. Version 2.0.

[MTConnect Part 5.0] *MTConnect Standard: Part 5.0 - Interface Interaction Model*. Version 2.0.

3 Interface Interaction Model

In many manufacturing processes, multiple pieces of equipment must work together to perform a task. The traditional method for coordinating the activities between individual pieces of equipment is to connect them using a series of wires to communicate equipment states and demands for action. These interactions use simple binary ON/OFF signals to accomplished their intention.

In the MTConnect Standard, *interfaces* provides a means to replace this traditional method for interconnecting pieces of equipment with a structured *interaction model* that provides a rich set of information used to coordinate the actions between pieces of equipment. Implementers may utilize the information provided by this data model to (1) realize the interaction between pieces of equipment and (2) to extend the functionality of the equipment to improve the overall performance of the manufacturing process.

The *interaction model* used to implement *interfaces* provides a lightweight and efficient protocol, simplifies failure recovery scenarios, and defines a structure for implementing a Plug-And-Play relationship between pieces of equipment. By standardizing the information exchange using this higher-level semantic information model, an implementer may more readily replace a piece of equipment in a manufacturing system with any other piece of equipment capable of providing similar *interaction model* functions.

Two primary functions are required to implement the *interaction model* for an *interfaces* and manage the flow of information between pieces of equipment. Each piece of equipment needs to have the following:

- An *agent* which provides:
- The data required to implement the *interaction model*.
- Any other data from a piece of equipment needed to implement the *interface* – operating states of the equipment, position information, execution modes, process information, etc.
- A client software application that enables the piece of equipment to acquire and interpret information from another piece of equipment.

3.1 Interfaces Architecture

MTConnect Standard is based on a communications method that provides no direct way for one piece of equipment to change the state of or cause an action to occur in another

piece of equipment. The *interaction model* used to implement *interfaces* is based on a *publish and subscribe* type of communications as described in *MTConnect Standard Part 1.0 - Fundamentals* and utilizes a *request* and *response* information exchange mechanism. For *interfaces*, pieces of equipment must perform both the publish (*agent*) and subscribe (client) functions.

Note: The current definition of *interfaces* addresses the interaction between two pieces of equipment. Future releases of the MTConnect Standard may address the interaction between multiple (more than two) pieces of equipment.

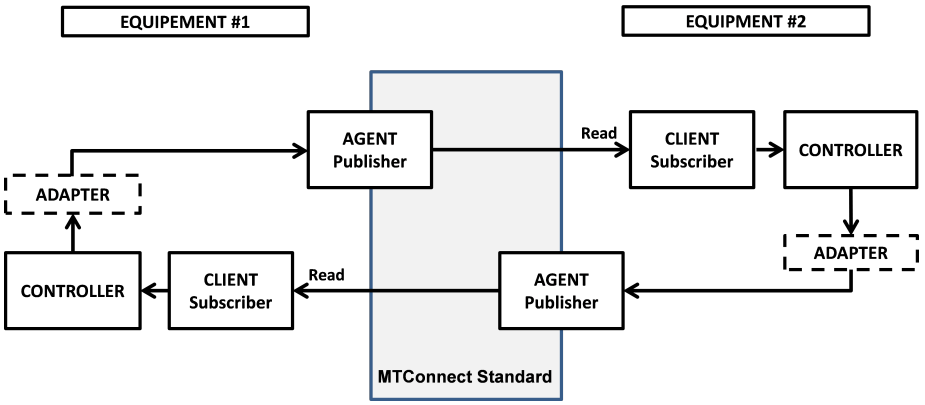

Figure 1: Data Flow Architecture for Interfaces

Note: The data flow architecture illustrated in Figure 1 was historically referred to in the MTConnect Standard as a read-read concept.

In the implementation of the *interaction model* for *interfaces*, two pieces of equipment can exchange information in the following manner. One piece of equipment indicates a *request* for service by publishing a type of *request* using a data item provided through an *agent* as defined in *Section 4.3 - DataItem Types for Interface*. The client associated with the second piece of equipment, which is subscribing to data from the first machine, detects and interprets that *request*. If the second machine chooses to take any action to fulfill this *request*, it can indicate its acceptance by publishing a *response* using a data item provided through its *agent*. The client on the first piece of equipment continues to monitor information from the second piece of equipment until it detects an indication that the *response* to the *request* has been completed or has failed.

An example of this type of interaction between pieces of equipment can be represented by a machine tool that wants the material to be loaded by a robot. In this example, the machine tool is the *requester*, and the robot is the *responder*. On the other hand, if the robot wants the machine tool to open a door, the robot becomes the *requester* and the machine tool the *responder*.

3.2 Request and Response Information Exchange

The DataItem elements defined by the *interaction model* each have a REQUEST and RESPONSE subtype. These subtypes identify if the data item represents a *request* or a *response*. Using these data items, a piece of equipment changes the state of its *request* or *response* to indicate information that can be read by the other piece of equipment. To aid in understanding how the *interaction model* functions, one can view this *interaction model* as a simple state machine.

The interaction between two pieces of equipment can be described as follows. When the *requester* wants an activity to be performed, it transitions its *request* state from a READY state to an ACTIVE state. In turn, when the client on the *responder* reads this information and interprets the *request*, the *responder* announces that it is performing the requested task by changing its response state to ACTIVE. When the action is finished, the *responder* changes its response state to COMPLETE. This pattern of *request* and *response* provides the basis for the coordination of actions between pieces of equipment. These actions are implemented using EVENT category data items. (See *Section 4.3 - DataItem Types for Interface* for details on the Event type data items defined for *interfaces*.)

> Note: The implementation details of how the *responder* piece of equipment reacts to the *request* and then completes the requested task are up to the implementer.

The initial condition of both the *request* and *response* states on both pieces of equipment is READY. The dotted lines indicate the on-going communications that occur to monitor the progress of the interactions between the pieces of equipment.

The interaction between the pieces of equipment as illustrated in Figure 2 progresses through the sequence listed below.

- The *request* transitions from READY to ACTIVE signaling that a service is needed.
- The *response* detects the transition of the *request*.
- The *response* transitions from READY to ACTIVE indicating that it is performing the action.
- Once the action has been performed, the *response* transitions to COMPLETE.
- The *request* detects the action is COMPLETE.
- The *request* transitions back to READY acknowledging that the service has been performed.

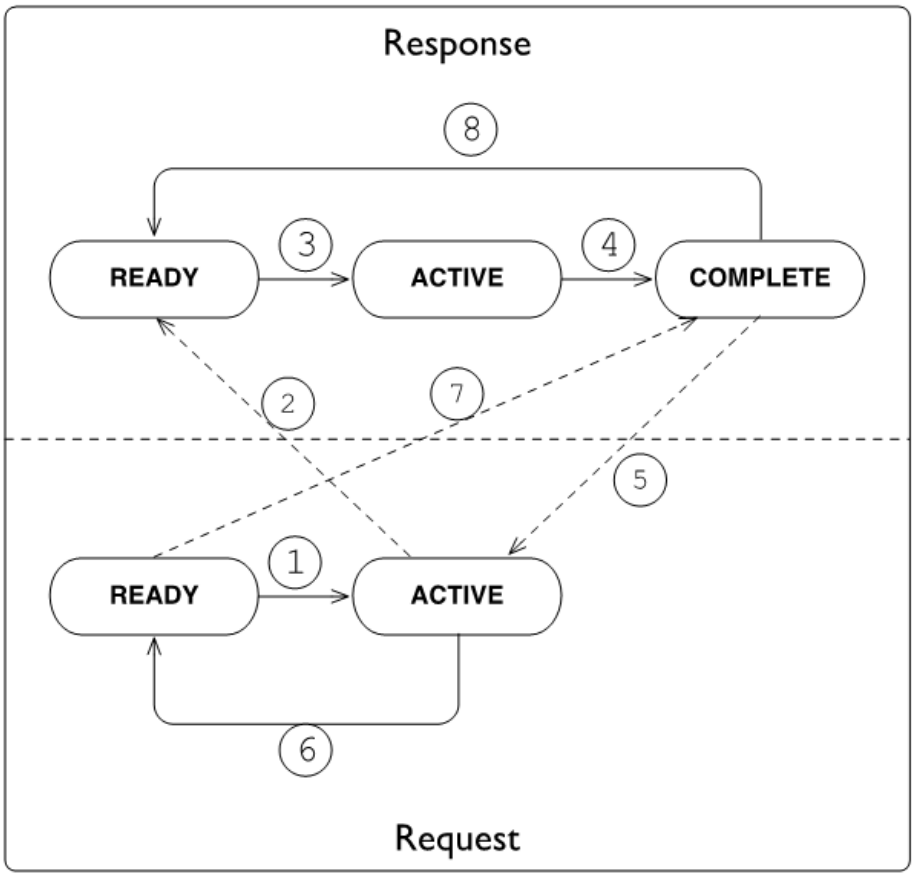

Figure 2: Request and Response Overview

- The *response* detects the *request* has returned to `READY`.
- In recognition of this acknowledgement, the *response* transitions back to `READY`.

After the final action has been completed, both pieces of equipment are back in the `READY` state indicating that they are able to perform another action.

3.3 Interface

abstract `Component` that coordinates actions and activities between pieces of equipment.

865 3.3.1 Commonly Observed DataItem Types for Interface

866 *Table 1* lists the Commonly Observed DataItem Types for `Interface`.

Commonly Observed DataItem Types	Multiplicity
`InterfaceState`	1

Table 1: Commonly Observed DataItem Types for Interface

4 Interfaces for Device and Observation Information Models

The *interaction model* for implementing *interfaces* is defined in the MTConnect Standard as an extension to the *Device Information Model* and *Observation Information Model*.

A piece of equipment **MAY** support multiple different *interfaces*. Each piece of equipment supporting *interfaces* **MUST** model the information associated with each *interface* as an `Interface` component. `Interface` is an abstract `Component` and is realized by `Interface` component types.

The Figure 3 illustrates where an `Interface` is modeled in the *Device Information Model* for a piece of equipment.

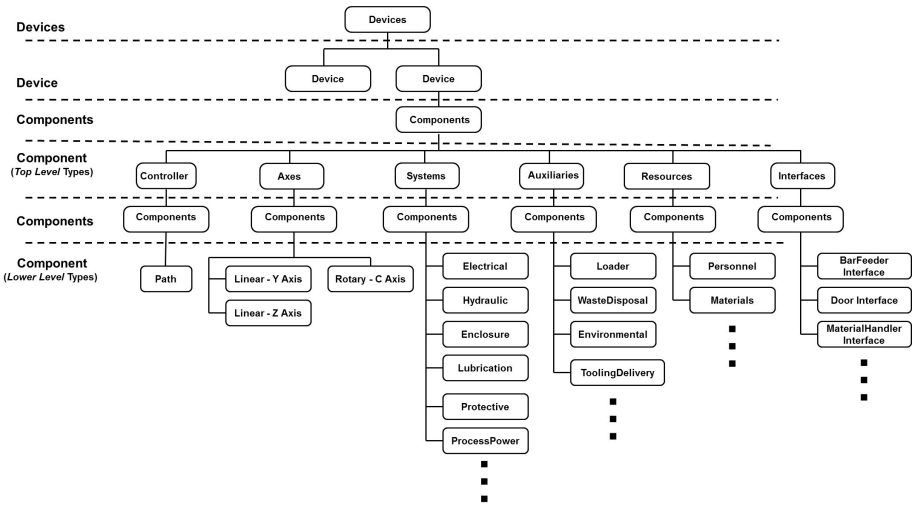

Figure 3: Interfaces in Entity Hierarchy

4.1 Interface Types

The abstract `Interface` is realized by the following types listed in this section.

In order to implement the *interaction model* for *interfaces*, each piece of equipment associated with an *interface* **MUST** provide the corresponding `Interface` type. A piece of equipment **MAY** support any number of unique *interfaces*.

4.1.1 BarFeederInterface

`Interface` that coordinates the operations between a bar feeder and another piece of equipment.

Bar feeder is a piece of equipment that pushes bar stock (i.e., long pieces of material of various shapes) into an associated piece of equipment – most typically a lathe or turning center.

4.1.2 ChuckInterface

`Interface` that coordinates the operations between two pieces of equipment, one of which controls the operation of a chuck.

The piece of equipment that is controlling the chuck **MUST** provide the data item `ChuckState` as part of the set of information provided.

4.1.3 DoorInterface

`Interface` that coordinates the operations between two pieces of equipment, one of which controls the operation of a door.

The piece of equipment that is controlling the door **MUST** provide data item `DoorState` as part of the set of information provided.

4.1.4 MaterialHandlerInterface

`Interface` that coordinates the operations between a piece of equipment and another associated piece of equipment used to automatically handle various types of materials or services associated with the original piece of equipment.

A material handler is a piece of equipment capable of providing any one, or more, of a variety of support services for another piece of equipment or a process like:

- Loading/unloading material or tooling
- Part inspection

- Testing
- Cleaning

A robot is a common example of a material handler.

4.2 Data for Interface

Each *interface* **MUST** provide the data associated with the specific *interface* to implement the *interaction model* and any additional data that may be needed by another piece of equipment to understand the operating states and conditions of the first piece of equipment as it applies to the *interface*.

Details on data items specific to the *interaction model* for each type of *interface* are provided in *Section 4.3 - DataItem Types for Interface*.

An implementer may choose any other data available from a piece of equipment to describe the operating states and other information needed to support an *interface*.

4.2.1 References for Interface

Some of the data items needed to support a specific *interface* may already be defined elsewhere in the *MTConnectDevices Response Document* for a piece of equipment. However, the implementer may not be able to directly associate this data with the *interface* since the MTConnect Standard does not permit multiple occurrences of a piece of data to be configured in an *MTConnectDevices Response Document*. `References` provides a mechanism for associating information defined elsewhere in the *information model* for a piece of equipment with a specific *interface*.

`References` *organizes* `Reference` elements.

`Reference` is a pointer to information that is associated with another entity defined elsewhere for a piece of equipment.

`References` is an economical syntax for providing interface specific information without directly duplicating the occurrence of the data. It provides a mechanism to include all necessary information required for interaction and deterministic information flow between pieces of equipment.

For more information on the References model, see *MTConnect Standard: Part 2.0 - Device Information Model*.

4.3 DataItem Types for Interface

Each Interface contains data items which are used to communicate information required to execute the *interface*. When these data items are read by another piece of equipment, that piece of equipment can then determine the actions that it may take based upon that data.

InterfaceState is a data item specifically defined for *interfaces*. It defines the operational state of the *interface*. This is an indicator identifying whether the *interface* is functioning or not. See *Section 4.3.4 - InterfaceState* for complete semantic details.

Some data items **MAY** be directly associated with the Interface element and others will be organized by a References element. It is up to an implementer to determine which additional data items are required for a particular *interface*.

4.3.1 Specific Data Items for the Interaction Model for Interface

A special set of data items have been defined to be used in conjunction with Interface. They provide information from a piece of equipment to *request* a service to be performed by another associated piece of equipment; and for the associated piece of equipment to indicate its progress in performing its *response* to the *request* for service. .

Many of the data items describing the services associated with an *interface* are paired to describe two distinct actions – one to *request* an action to be performed and a second to reverse the action or to return to an original state. For example, a DoorInterface will have two actions OpenDoor and CloseDoor. An example of an implementation of this would be a robot that indicates to a machine that it would like to have a door opened so that the robot could extract a part from the machine and then asks the machine to close that door once the part has been removed.

When these data items are used to describe a service associated with an *interface*, they **MUST** have one of the following two subType elements: REQUEST or RESPONSE. These **MUST** be specified to define whether the piece of equipment is functioning as the *requester* or *responder* for the service to be performed. The *requester* **MUST** specify the REQUEST subType for the data item and the *responder* **MUST** specify a corresponding RESPONSE subType for the data item to enable the coordination between the two pieces of equipment.

These data items and their associated `subType` provide the basic structure for implementing the *interaction model* for an *interface* and are defined in the following sections.

Figure 4 and Figure 5 show possible state transitions for a *request* and *response* respectively. The state machine diagrams provide the permissible values of the observations for the `DataItem` types listed in this section.

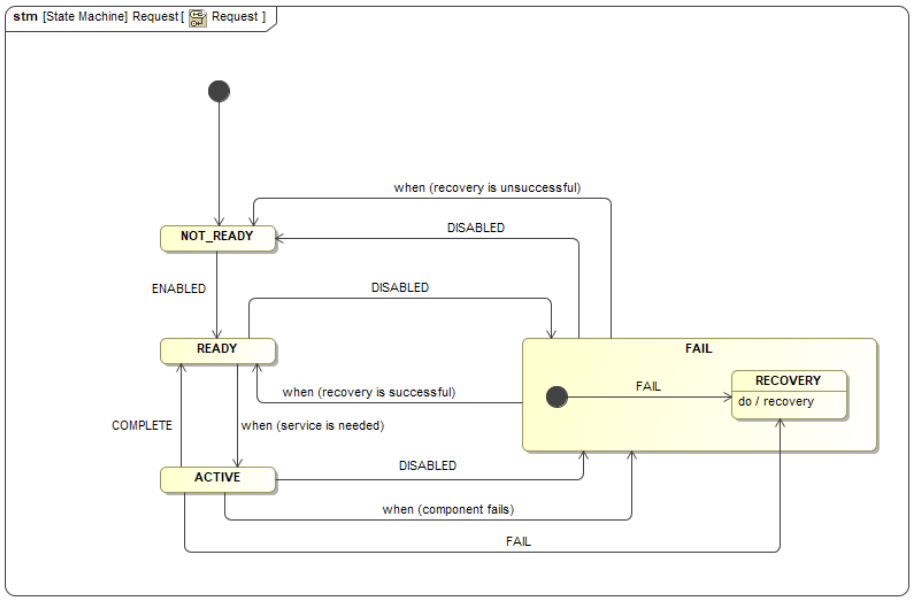

Figure 4: Request State Machine

4.3.2 CloseChuck

A `subType` **MUST** always be specified.

4.3.2.1 Subtypes of CloseChuck

- `REQUEST`

  operating state of the *request* to close a chuck.

  `RequestStateEnum` Enumeration:

  - `ACTIVE`

    requester has initiated a *request* for a service and the service has not yet been completed by the *responder*.

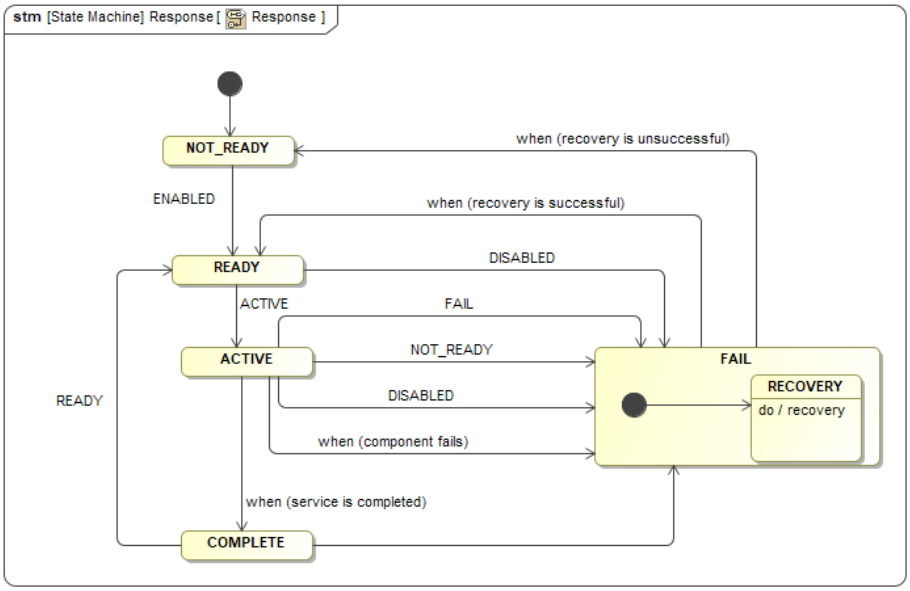

Figure 5: Response State Machine

- FAIL
  requester has detected a failure condition.
- NOT_READY
  requester is not ready to make a *request*.
- READY
  requester is prepared to make a *request*, but no *request* for service is required.

- RESPONSE

  operating state of the *response* to a *request* to close a chuck.

  ResponseStateEnum Enumeration:

  - ACTIVE
    responder has detected and accepted a *request* for a service and is in the process of performing the service, but the service has not yet been completed.
  - COMPLETE
    responder has completed the actions required to perform the service.
  - FAIL
    responder has detected a failure condition.
  - NOT_READY
    responder is not ready to perform a service.

- READY

  responder is prepared to react to a *request*, but no *request* for service has been detected.

4.3.3 CloseDoor

A subType **MUST** always be specified.

4.3.3.1 Subtypes of CloseDoor

- REQUEST

  operating state of the *request* to close a door.

  The value of CloseDoor **MUST** be one of the RequestStateEnum enumeration.

- RESPONSE

  operating state of the *response* to a *request* to close a door.

  The value of CloseDoor **MUST** be one of the ResponseStateEnum enumeration.

4.3.4 InterfaceState

When the InterfaceState is DISABLED, the state of all data items that are specific for the *interaction model* associated with that Interface **MUST** be set to NOT_READY.

InterfaceStateEnum Enumeration:

- DISABLED

  Interface is currently not operational.

- ENABLED

  Interface is currently operational and performing as expected.

4.3.5 MaterialChange

A `subType` **MUST** always be specified.

4.3.5.1 Subtypes of MaterialChange

- `REQUEST`

  operating state of the *request* to change the type of material or product being loaded or fed to a piece of equipment.

  The value of `MaterialChange` **MUST** be one of the `RequestStateEnum` enumeration.

- `RESPONSE`

  operating state of the *response* to a *request* to change the type of material or product being loaded or fed to a piece of equipment.

  The value of `MaterialChange` **MUST** be one of the `ResponseStateEnum` enumeration.

4.3.6 MaterialFeed

A `subType` **MUST** always be specified.

4.3.6.1 Subtypes of MaterialFeed

- `REQUEST`

  operating state of the *request* to advance material or feed product to a piece of equipment from a continuous or bulk source.

  The value of `MaterialFeed` **MUST** be one of the `RequestStateEnum` enumeration.

- `RESPONSE`

  operating state of the *response* to a *request* to advance material or feed product to a piece of equipment from a continuous or bulk source.

  The value of `MaterialFeed` **MUST** be one of the `ResponseStateEnum` enumeration.

4.3.7 MaterialLoad

A `subType` **MUST** always be specified.

4.3.7.1 Subtypes of MaterialLoad

- `REQUEST`

  operating state of the *request* to load a piece of material or product.

  The value of `MaterialLoad` **MUST** be one of the `RequestStateEnum` enumeration.

- `RESPONSE`

  operating state of the *response* to a *request* to load a piece of material or product.

  The value of `MaterialLoad` **MUST** be one of the `ResponseStateEnum` enumeration.

4.3.8 MaterialRetract

A `subType` **MUST** always be specified.

4.3.8.1 Subtypes of MaterialRetract

- `REQUEST`

  operating state of the *request* to remove or retract material or product.

  The value of `MaterialRetract` **MUST** be one of the `RequestStateEnum` enumeration.

- `RESPONSE`

  operating state of the *response* to a *request* to remove or retract material or product.

  The value of `MaterialRetract` **MUST** be one of the `ResponseStateEnum` enumeration.

4.3.9 MaterialUnload

A `subType` **MUST** always be specified.

4.3.9.1 Subtypes of MaterialUnload

- REQUEST

  operating state of the *request* to unload a piece of material or product.

  The value of MaterialUnload **MUST** be one of the RequestStateEnum enumeration.

- RESPONSE

  operating state of the *response* to a *request* to unload a piece of material or product.

  The value of MaterialUnload **MUST** be one of the ResponseStateEnum enumeration.

4.3.10 OpenChuck

A subType **MUST** always be specified.

4.3.10.1 Subtypes of OpenChuck

- REQUEST

  operating state of the *request* to open a chuck.

  The value of OpenChuck **MUST** be one of the RequestStateEnum enumeration.

- RESPONSE

  operating state of the *response* to a *request* to open a chuck.

  The value of OpenChuck **MUST** be one of the ResponseStateEnum enumeration.

4.3.11 OpenDoor

A subType **MUST** always be specified.

4.3.11.1 Subtypes of OpenDoor

- `REQUEST`

  operating state of the *request* to open a door.

  The value of `OpenDoor` **MUST** be one of the `RequestStateEnum` enumeration.

- `RESPONSE`

  operating state of the *response* to a *request* to open a door.

  The value of `OpenDoor` **MUST** be one of the `ResponseStateEnum` enumeration.

4.3.12 PartChange

A `subType` **MUST** always be specified.

4.3.12.1 Subtypes of PartChange

- `REQUEST`

  operating state of the *request* to change the part or product associated with a piece of equipment to a different part or product.

  The value of `PartChange` **MUST** be one of the `RequestStateEnum` enumeration.

- `RESPONSE`

  operating state of the *response* to a *request* to change the part or product associated with a piece of equipment to a different part or product.

  The value of `PartChange` **MUST** be one of the `ResponseStateEnum` enumeration.

1113 5 Operation and Error Recovery

1114 The *request and response* state model implemented for *interfaces* may also be represented
1115 by a graphical model. The scenario in Figure 6 demonstrates the state transitions that occur
1116 during a successful *request* for service and the resulting *response* to fulfill that service
1117 *request*.

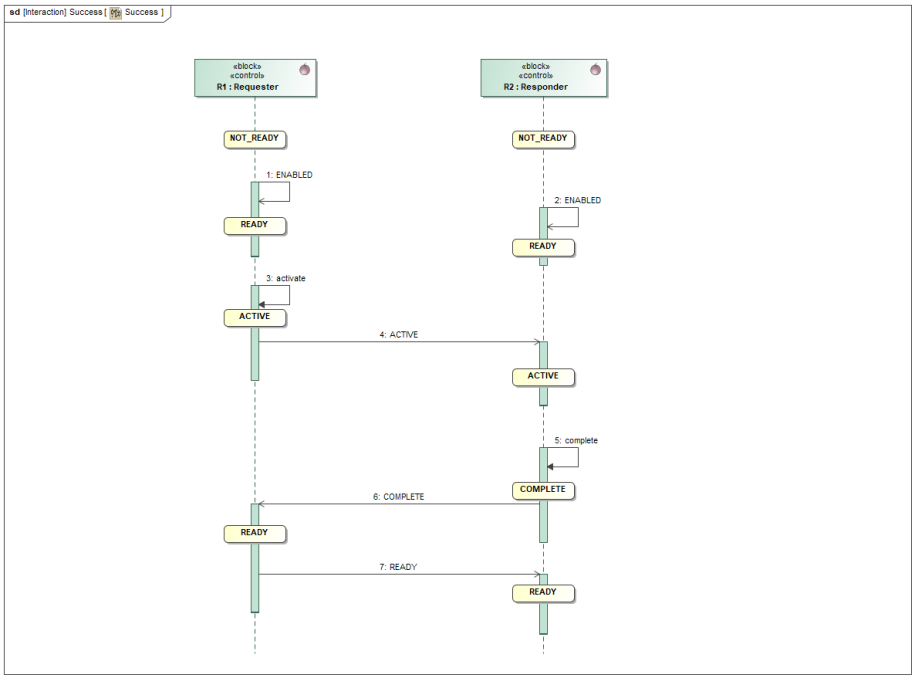

Figure 6: Success Scenario

1118 5.1 Request and Response Failure Handling and Recovery

1119 A significant feature of the *request and response interaction model* is the ability for ei-
1120 ther piece of equipment to detect a failure associated with either the *request* or *response*
1121 actions. When either a failure or unexpected action occurs, the *request* and the *response*
1122 portion of the *interaction model* can announce a `FAIL` state upon detecting a problem. The
1123 following are graphical models describing multiple scenarios where either the *requester*
1124 or *responder* detects and reacts to a failure. In these examples, either the *requester* or
1125 *responder* announces the detection of a failure by setting either the *request* or the *response*
1126 state to `FAIL`.

1127 Once a failure is detected, the *interaction model* provides information from each piece of
1128 equipment as they attempt to recover from a failure, reset all of their functions associated

with the *interface* to their original state, and return to normal operation.

The following sections are scenarios that describe how pieces of equipment may react to different types of failures and how they indicate when they are again ready to request a service or respond to a request for service after recovering from those failures:

5.1.1 Responder Fails Immediately

In this scenario, a failure is detected by the *responder* immediately after a *request* for service has been initiated by the *requester*.

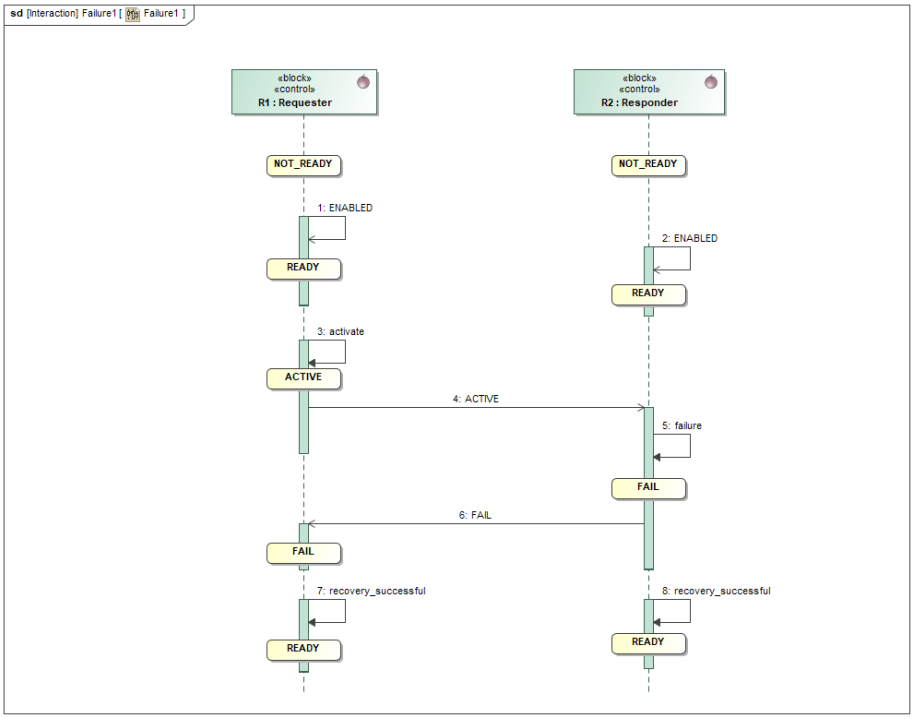

Figure 7: Responder Fails Immediately

In this case, the *request* transitions to `ACTIVE` and the *responder* immediately detects a failure before it can transition the *response* state to `ACTIVE`. When this occurs, the *responder* transitions the *response* state to `FAIL`.

After detecting that the *responder* has transitioned its state to `FAIL`, the *requester* **MUST** change its state to `FAIL`.

The *requester*, as part of clearing a failure, resets any partial actions that were initiated and attempts to return to a condition where it is again ready to request a service. If the recovery

is successful, the *requester* changes its state from FAIL to READY. If for some reason the *requester* cannot return to a condition where it is again ready to request a service, it transitions its state from FAIL to NOT_READY.

The *responder*, as part of clearing a failure, resets any partial actions that were initiated and attempts to return to a condition where it is again ready to perform a service. If the recovery is successful, the *responder* changes its *response* state from FAIL to READY. If for some reason the *responder* is not again prepared to perform a service, it transitions its state from FAIL to NOT_READY.

5.1.2 Responder Fails While Providing a Service

This is the most common failure scenario. In this case, the *responder* will begin the actions required to provide a service. During these actions, the *responder* detects a failure and transitions its *response* state to FAIL.

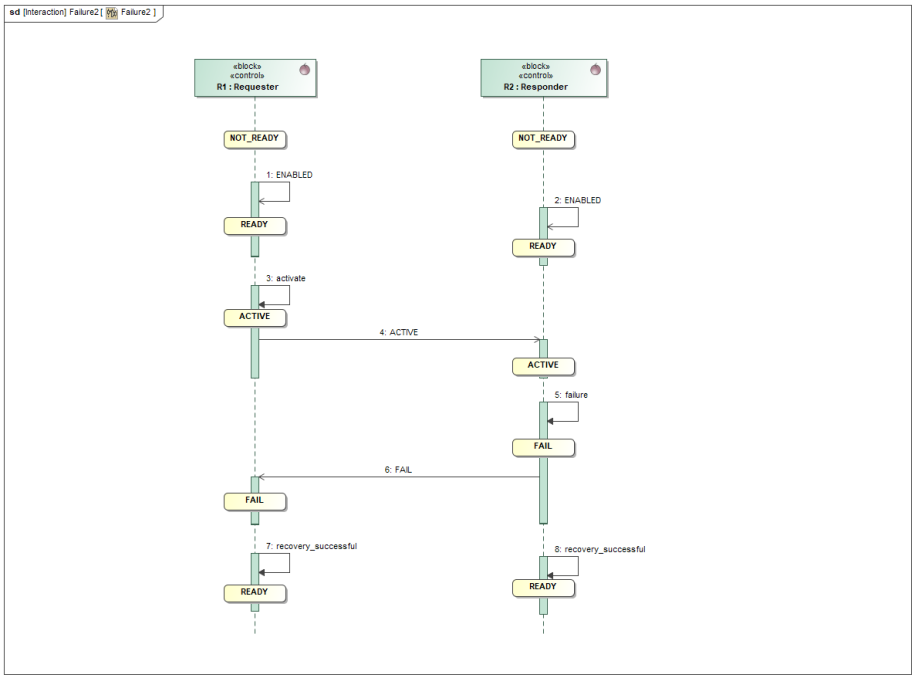

Figure 8: Responder Fails While Providing a Service

When a *requester* detects a failure of a *responder*, it transitions it state from ACTIVE to FAIL.

The *requester* resets any partial actions that were initiated and attempts to return to a condition where it is again ready to request a service. If the recovery is successful, the

requester changes its state from FAIL to READY if the failure has been cleared and it is again prepared to request another service. If for some reason the *requester* cannot return to a condition where it is again ready to request a service, it transitions its state from FAIL to NOT_READY.

The *responder*, as part of clearing a failure, resets any partial actions that were initiated and attempts to return to a condition where it is again ready to perform a service. If the recovery is successful, the *responder* changes its *response* state from FAIL to READY if it is again prepared to perform a service. If for some reason the *responder* is not again prepared to perform a service, it transitions its state from FAIL to NOT_READY.

5.1.3 Requester Failure During a Service Request

In this scenario, the *responder* will begin the actions required to provide a service. During these actions, the *requester* detects a failure and transitions its *request* state to FAIL.

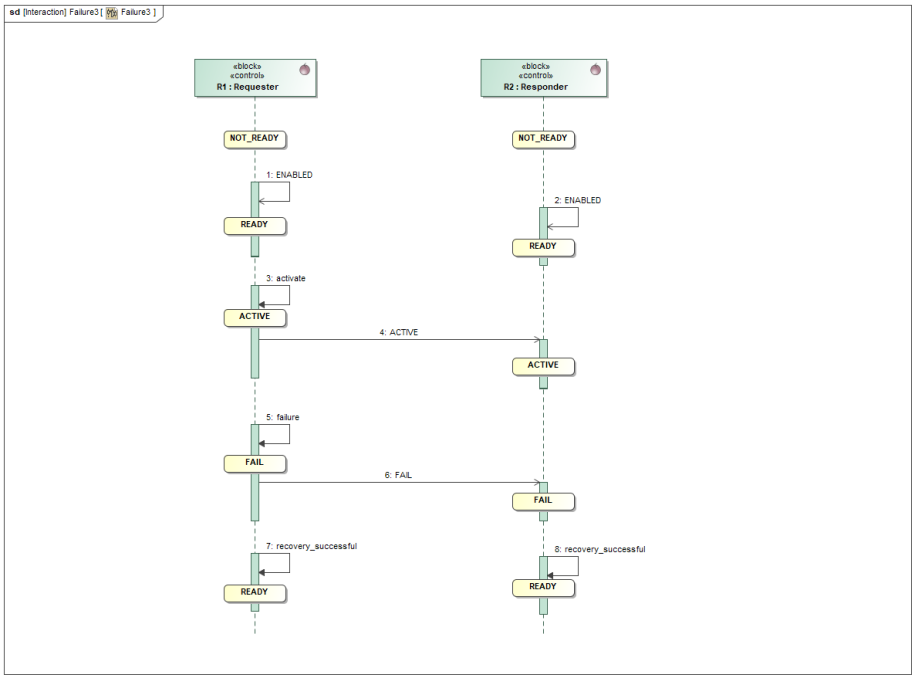

Figure 9: Requester Fails During a Service Request

When the *responder* detects that the *requester* has transitioned its *request* state to FAIL, the *responder* also transitions its *response* state to FAIL.

The *requester*, as part of clearing a failure, resets any partial actions that were initiated and attempts to return to a condition where it is again ready to request a service. If the recovery

is successful, the *requester* changes its state from FAIL to READY. If for some reason the *requester* cannot return to a condition where it is again ready to request a service, it transitions its state from FAIL to NOT_READY.

The *responder*, as part of clearing a failure, resets any partial actions that were initiated and attempts to return to a condition where it is again ready to perform a service. If the recovery is successful, the *responder* changes its *response* state from FAIL to READY. If for some reason the *responder* is not again prepared to perform a service, it transitions its state from FAIL to NOT_READY.

5.1.4 Requester Changes to an Unexpected State While Responder is Providing a Service

In some cases, a *requester* may transition to an unexpected state after it has initiated a *request* for service.

As demonstrated in Figure 10, the *requester* has initiated a *request* for service and its *request* state has been changed to ACTIVE. The *responder* begins the actions required to provide the service. During these actions, the *requester* transitions its *request* state back to READY before the *responder* can complete its actions. This **SHOULD** be regarded as a failure of the *requester*.

In this case, the *responder* reacts to this change of state of the *requester* in the same way as though the *requester* had transitioned its *request* state to FAIL (i.e., the same as in Scenario 3 above).

At this point, the *responder* then transitions its *response* state to FAIL.

The *responder* resets any partial actions that were initiated and attempts to return to its original condition where it is again ready to perform a service. If the recovery is successful, the *responder* changes its *response* state from FAIL to READY. If for some reason the *responder* is not again prepared to perform a service, it transitions its state from FAIL to NOT_READY.

> Note: The same scenario exists if the *requester* transitions its *request* state to NOT_READY. However, in this case, the *requester* then transitions its *request* state to READY after it resets all of its functions back to a condition where it is again prepared to make a *request* for service.

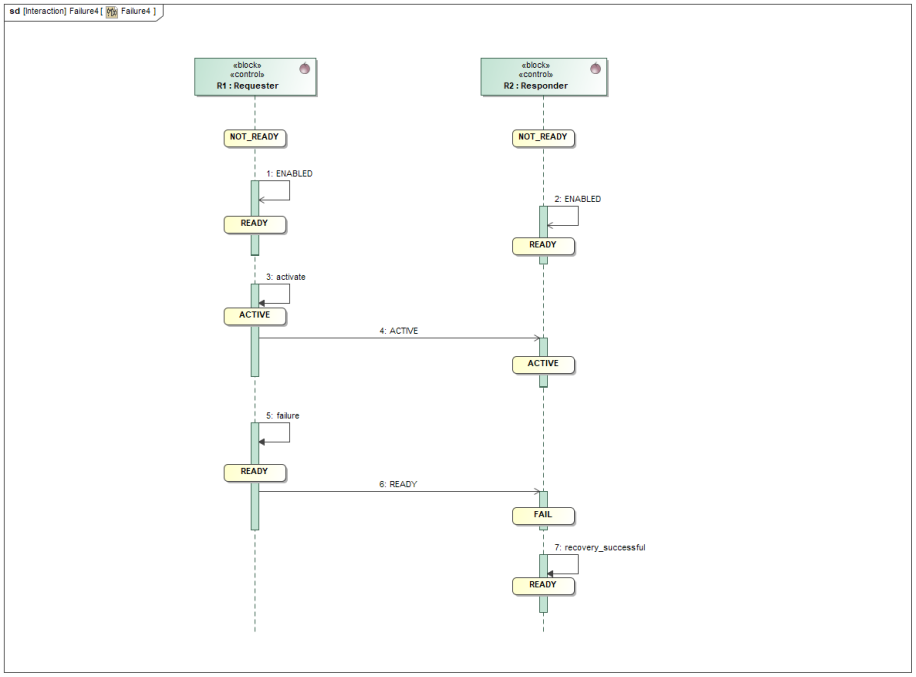

Figure 10: Requester Makes Unexpected State Change

5.1.5 Responder Changes to an Unexpected State While Providing a Service

Similar to Scenario 5, a *responder* may transition to an unexpected state while providing a service.

As demonstrated in Figure 11, the *responder* is performing the actions to provide a service and the *response* state is ACTIVE. During these actions, the *responder* transitions its state to NOT_READY before completing its actions. This should be regarded as a failure of the *responder*.

Upon detecting an unexpected state change of the *responder*, the *requester* transitions its state to FAIL.

The *requester* resets any partial actions that were initiated and attempts to return to a condition where it is again ready to request a service. If the recovery is successful, the *requester* changes its state from FAIL to READY. If for some reason the *requester* cannot return to a condition where it is again ready to request a service, it transitions its state from FAIL to NOT_READY.

Since the *responder* has failed to an invalid state, the condition of the *responder* is un-

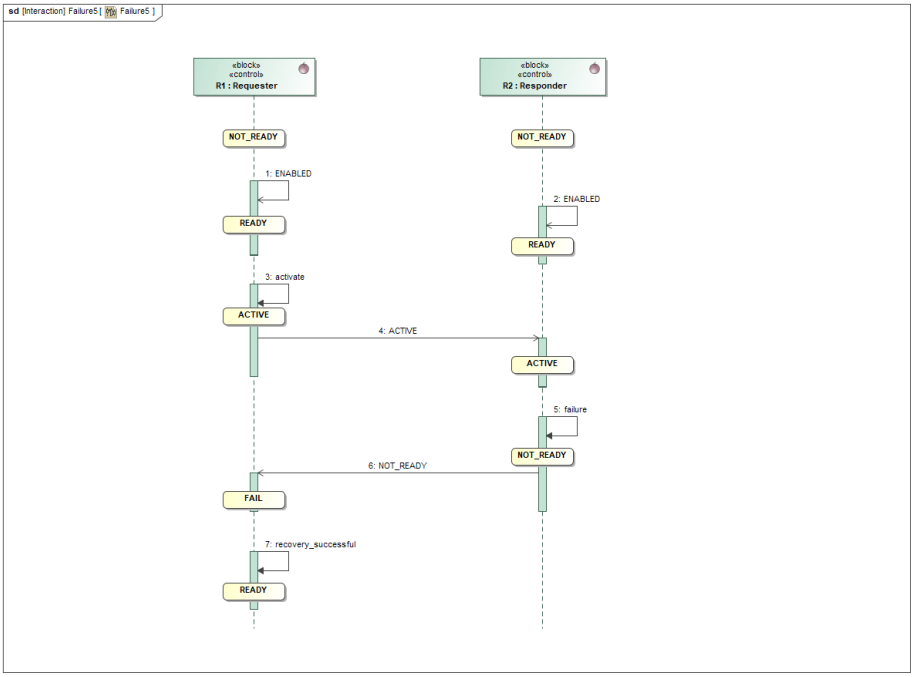

Figure 11: Responder Makes Unexpected State Change

known. Where possible, the *responder* should try to reset to an initial state.

The *responder*, as part of clearing the cause for the change to the unexpected state, should attempt to reset any partial actions that were initiated and then return to a condition where it is again ready to perform a service. If the recovery is successful, the *responder* changes its *response* state from the unexpected state to `READY`. If for some reason the *responder* is not again prepared to perform a service, it maintains its state as `NOT_READY`.

5.1.6 Responder or Requester Become UNAVAILABLE or Experience a Loss of Communication

In this scenario, a failure occurs in the communications connection between the *responder* and *requester*. This failure may result from the `InterfaceState` from either piece of equipment returning a value of `UNAVAILABLE` or one of the pieces of equipment does not provide a heartbeat within the desired amount of time (See *MTConnect Standard Part 1.0 - Fundamentals* for details on heartbeat).

When one of these situations occurs, each piece of equipment assumes that there has been a failure of the other piece of equipment.

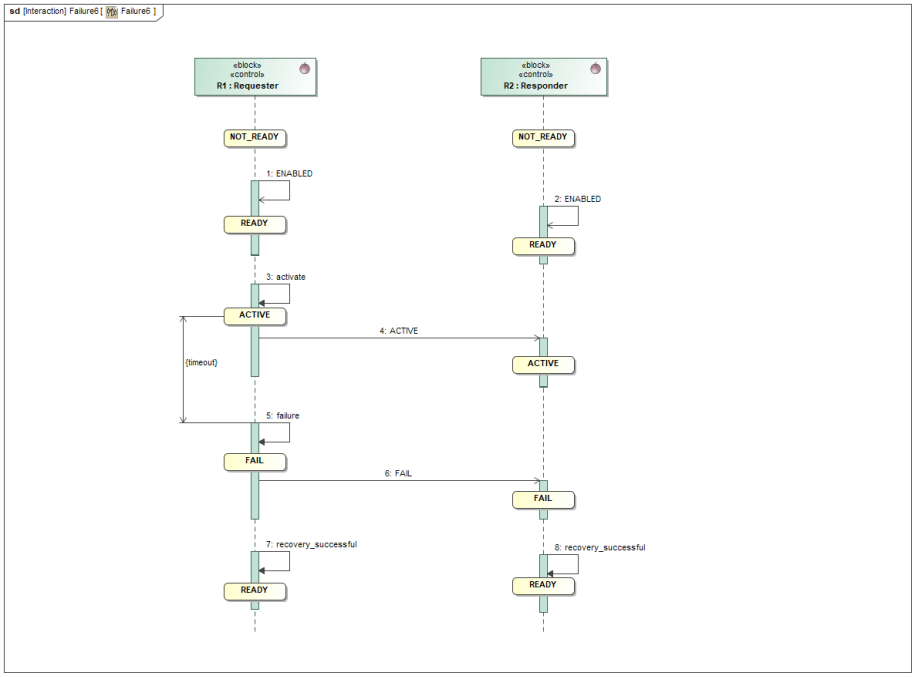

Figure 12: Requester - Responder Communication Failure 1

When normal communications are re-established, neither piece of equipment should assume that the *request and response* state of the other piece of equipment remains valid. Both pieces of equipment should set their state to FAIL.

The *requester*, as part of clearing its FAIL state, resets any partial actions that were initiated and attempts to return to a condition where it is again ready to request a service. If the recovery is successful, the *requester* changes its state from FAIL to READY. If for some reason the *requester* cannot return to a condition where it is again ready to request a service, it transitions its state from FAIL to NOT_READY.

The *responder*, as part of clearing its FAIL state, resets any partial actions that were initiated and attempts to return to a condition where it is again ready to perform a service. If the recovery is successful, the *responder* changes its *response* state from FAIL to READY. If for some reason the *responder* is not again prepared to perform a service, it transitions its state from FAIL to NOT_READY.

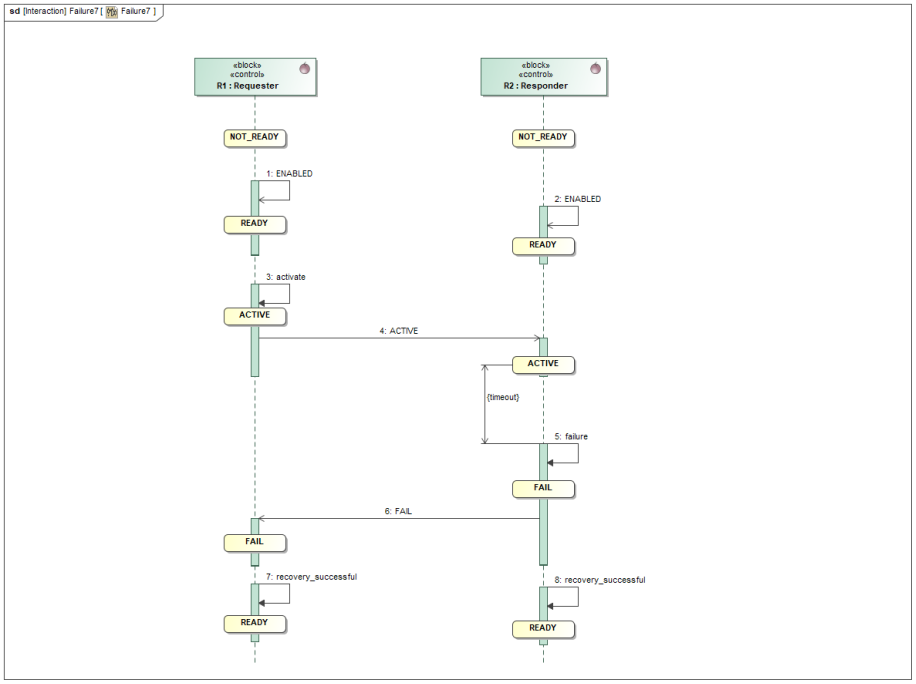

Figure 13: Requester - Responder Communication Failure 2

6 Profile

MTConnect Profile is a *profile* that extends the Systems Modeling Language (SysML) metamodel for the MTConnect domain using additional data types and *stereotypes*.

6.1 DataTypes

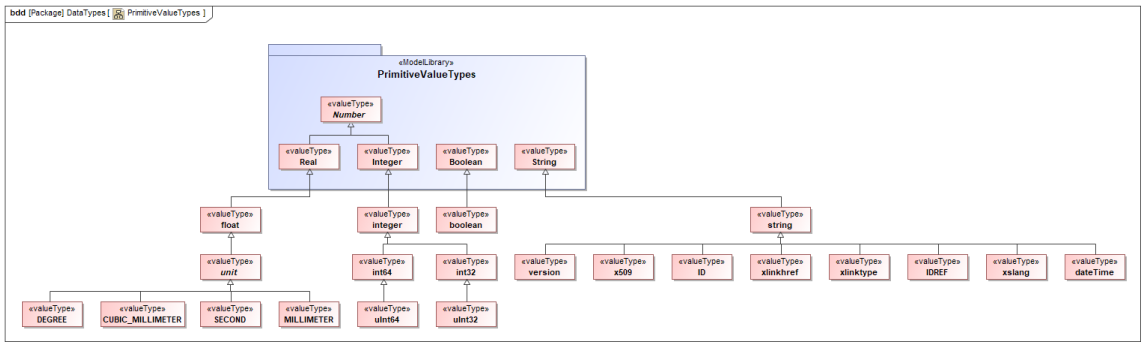

Figure 14: DataTypes

6.1.1 boolean

primitive type.

6.1.2 ID

string that represents an identifier (ID).

6.1.3 string

primitive type.

6.1.4 float

primitive type.

6.1.5 datetime

string that represents timestamp in ISO 8601 format.

6.1.6 integer

primitive type.

6.1.7 xlinktype

string that represents the type of an XLink element. See `https://www.w3.org/TR/xlink11/`.

6.1.8 xslang

string that represents a language tag. See `http://www.ietf.org/rfc/rfc4646.txt`.

6.1.9 SECOND

float that represents time in seconds.

6.1.10 IDREF

string that represents a reference to an `ID`.

6.1.11 xlinkhref

string that represents the locator attribute of an XLink element. See `https://www.w3.org/TR/xlink11/`.

6.1.12 x509

string that represents an `x509` data block. *Ref ISO/IEC 9594-8:2020*.

6.1.13 int32

32-bit integer.

6.1.14 int64

64-bit integer.

6.1.15 version

series of four numeric values, separated by a decimal point, representing a *major*, *minor*, and *revision* number of the MTConnect Standard and the revision number of a specific *schema*.

6.1.16 uint32

32-bit unsigned integer.

6.1.17 uint64

64-bit unsigned integer.

6.1.18 binary

base-2 numeral system or binary numeral system represented by two digits: “0” and “1”.

6.1.19 double

primitive type.

6.2 Stereotypes

6.2.1 organizer

element that *organizes* other elements of a type.

6.2.2 deprecated

element that has been deprecated.

6.2.3 extensible

enumeration that can be extended.

6.2.4 informative

element that is descriptive and non-normative.

6.2.5 valueType

extends SysML `<<ValueType>>` to include `Class` as a value type.

6.2.6 normative

element that has been added to the standard.

6.2.7 observes

association in which a *Component* makes *Observations* about an observable *DataItem*.

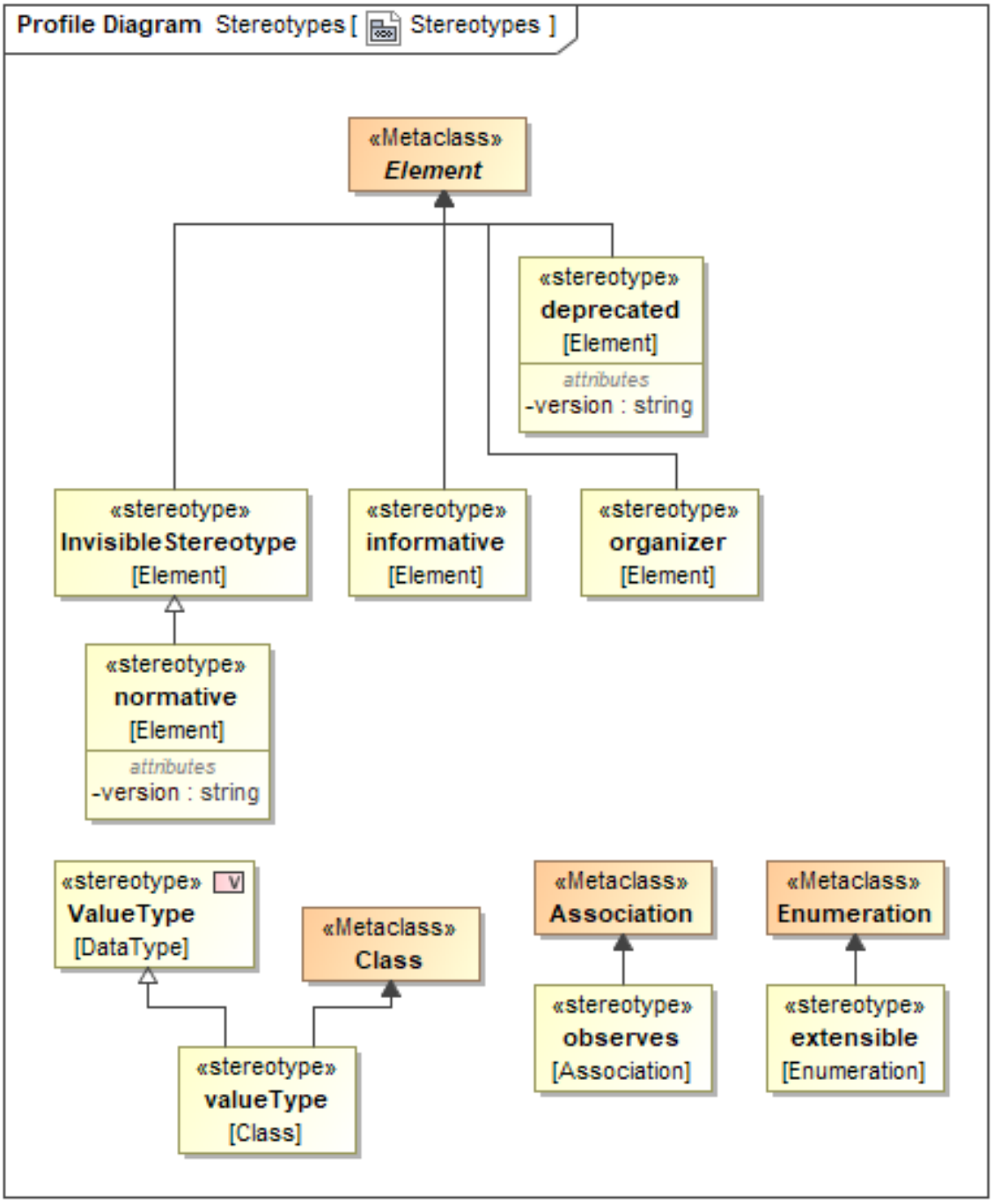

Figure 15: Stereotypes

Appendices

A Bibliography

Engineering Industries Association. EIA Standard - EIA-274-D, Interchangeable Variable, Block Data Format for Positioning, Contouring, and Contouring/Positioning Numerically Controlled Machines. Washington, D.C. 1979.

ISO TC 184/SC4/WG3 N1089. ISO/DIS 10303-238: Industrial automation systems and integration Product data representation and exchange Part 238: Application Protocols: Application interpreted model for computerized numerical controllers. Geneva, Switzerland, 2004.

International Organization for Standardization. ISO 14649: Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 10: General process data. Geneva, Switzerland, 2004.

International Organization for Standardization. ISO 14649: Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers – Part 11: Process data for milling. Geneva, Switzerland, 2000.

International Organization for Standardization. ISO 6983/1 – Numerical Control of machines – Program format and definition of address words – Part 1: Data format for positioning, line and contouring control systems. Geneva, Switzerland, 1982.

Electronic Industries Association. ANSI/EIA-494-B-1992, 32 Bit Binary CL (BCL) and 7 Bit ASCII CL (ACL) Exchange Input Format for Numerically Controlled Machines. Washington, D.C. 1992.

National Aerospace Standard. Uniform Cutting Tests - NAS Series: Metal Cutting Equipment Specifications. Washington, D.C. 1969.

International Organization for Standardization. ISO 10303-11: 1994, Industrial automation systems and integration Product data representation and exchange Part 11: Description methods: The EXPRESS language reference manual. Geneva, Switzerland, 1994.

International Organization for Standardization. ISO 10303-21: 1996, Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure. Geneva, Switzerland, 1996.

H.L. Horton, F.D. Jones, and E. Oberg. Machinery's Handbook. Industrial Press, Inc.

New York, 1984.

International Organization for Standardization. ISO 841-2001: Industrial automation systems and integration - Numerical control of machines - Coordinate systems and motion nomenclature. Geneva, Switzerland, 2001.

ASME B5.57: Methods for Performance Evaluation of Computer Numerically Controlled Lathes and Turning Centers, 1998.

ASME/ANSI B5.54: Methods for Performance Evaluation of Computer Numerically Controlled Machining Centers. 2005.

OPC Foundation. OPC Unified Architecture Specification, Part 1: Concepts Version 1.00. July 28, 2006.

IEEE STD 1451.0-2007, Standard for a Smart Transducer Interface for Sensors and Actuators – Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats, IEEE Instrumentation and Measurement Society, TC-9, The Institute of Electrical and Electronics Engineers, Inc., New York, N.Y. 10016, SH99684, October 5, 2007.

IEEE STD 1451.4-1994, Standard for a Smart Transducer Interface for Sensors and Actuators – Mixed-Mode Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats, IEEE Instrumentation and Measurement Society, TC-9, The Institute of Electrical and Electronics Engineers, Inc., New York, N.Y. 10016, SH95225, December 15, 2004.