



MTConnect® Standard

Part 5.0 – Interface Interaction Model

Version 2.5.0

Prepared for: MTConnect Institute

Prepared from: MTConnectSysMLModel.xml

Prepared on: January 5, 2026

MTConnect Specification and Materials

The Association for Manufacturing Technology (AMT) owns the copyright in this MTConnect Specification or Material. AMT grants to you a non-exclusive, non-transferable, revocable, non-sublicensable, fully-paid-up copyright license to reproduce, copy and redistribute this MTConnect Specification or Material, provided that you may only copy or redistribute the MTConnect Specification or Material in the form in which you received it, without modifications, and with all copyright notices and other notices and disclaimers contained in the MTConnect Specification or Material.

If you intend to adopt or implement an MTConnect Specification or Material in a product, whether hardware, software or firmware, which complies with an MTConnect Specification, you shall agree to the MTConnect Specification Implementer License Agreement (“Implementer License”) or to the MTConnect Intellectual Property Policy and Agreement (“IP Policy”). The Implementer License and IP Policy each sets forth the license terms and other terms of use for MTConnect Implementers to adopt or implement the MTConnect Specifications, including certain license rights covering necessary patent claims for that purpose. These materials can be found at www.MTConnect.org, or by contacting <mailto:info@MTConnect.org>.

MTConnect Institute and AMT have no responsibility to identify patents, patent claims or patent applications which may relate to or be required to implement a Specification, or to determine the legal validity or scope of any such patent claims brought to their attention. Each MTConnect Implementer is responsible for securing its own licenses or rights to any patent or other intellectual property rights that may be necessary for such use, and neither AMT nor MTConnect Institute have any obligation to secure any such rights.

This Material and all MTConnect Specifications and Materials are provided “as is” and MTConnect Institute and AMT, and each of their respective members, officers, affiliates, sponsors and agents, make no representation or warranty of any kind relating to these materials or to any implementation of the MTConnect Specifications or Materials in any product, including, without limitation, any expressed or implied warranty of noninfringement, merchantability, or fitness for particular purpose, or of the accuracy, reliability, or completeness of information contained herein. In no event shall MTConnect Institute or AMT be liable to any user or implementer of MTConnect Specifications or Materials for the cost of procuring substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, indirect, special or punitive damages or other direct damages, whether under contract, tort, warranty or otherwise, arising in any way out of access, use or inability to use the MTConnect Specification or other MTConnect Materials, whether or not they had advance notice of the possibility of such damage.

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	xlinkhref	56
6.1.11	x509	56
6.1.12	int32	57
6.1.13	int64	57
6.1.14	version	57
6.1.15	uint32	57
6.1.16	uint64	57
6.1.17	binary	57
6.1.18	double	57
6.1.19	Array	58
6.1.20	<>hasFormatSpecificRepresentation>>float3d	58
6.1.21	UUID	58
6.1.22	METER	58
6.2	Stereotypes	58
6.2.1	organizer	58
6.2.2	deprecated	58
6.2.3	extensible	58
6.2.4	informative	59
6.2.5	normative	59
6.2.6	observes	59

6.2.7	satisfiedBy	60
6.2.8	hasFormatSpecificRepresentation	60
6.2.9	valueType	60
6.2.10	isArray	60
6.2.11	MTConnectRequirementSpecification	60
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	59

List of Tables

Table 1: Commonly Observed DataItem Types for Interface	34
----------------------------------------------------------------	----

1 Purpose of This Document

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

5 This data model is based on an *interaction model* that defines the exchange of information
6 between pieces of equipment and is organized in the MTConnect Standard by `Interfaces`.
7

8 `Interfaces` is modeled as an extension to the *Device Information Model* and *Observation
9 Information Model*. `Interfaces` leverages similar rules and terminology as those
10 used to describe a component in the *Device Information Model*. `Interfaces` also uses
11 similar methods for reporting data to those used in the *MTConnectStreams Response Doc-
12 ument*.

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

16 Note: See *MTConnect Standard: Part 2.0 - Device Information Model* and
17 *MTConnect Standard: Part 3.0 - Observation Information Model* of the MT-
18 Connect Standard for information on how `Interfaces` is structured in the
19 *response documents* which are returned from an *agent* in response to a *probe
request*, *sample request*, or *current request*.
20

21 2 Terminology and Conventions

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

24 2.1 MTConnect References

25 [MTConnect Part 1.0] *MTConnect Standard Part 1.0 - Fundamentals*. Version 2.0.
26 [MTConnect Part 2.0] *MTConnect Standard: Part 2.0 - Device Information Model*. Ver-
27 sion 2.0.
28 [MTConnect Part 3.0] *MTConnect Standard: Part 3.0 - Observation Information Model*.
29 Version 2.0.
30 [MTConnect Part 5.0] *MTConnect Standard: Part 5.0 - Interface Interaction Model*. Ver-
31 sion 2.0.

32

33 3 Interface Interaction Model

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

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

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

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

- 54 • An *agent* which provides:
 - 55 • The data required to implement the *interaction model*.
 - 56 • Any other data from a piece of equipment needed to implement the *interface* – op-
57 erating states of the equipment, position information, execution modes, process in-
58 formation, etc.
 - 59 • A client software application that enables the piece of equipment to acquire and
60 interpret information from another piece of equipment.

61 3.1 Interfaces Architecture

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

64 piece of equipment. The *interaction model* used to implement *interfaces* is based on a
 65 *publish and subscribe* type of communications as described in *MTConnect Standard Part*
 66 *1.0 - Fundamentals* and utilizes a *request* and *response* information exchange mechanism.
 67 For *interfaces*, pieces of equipment must perform both the publish (*agent*) and subscribe
 68 (client) functions.

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

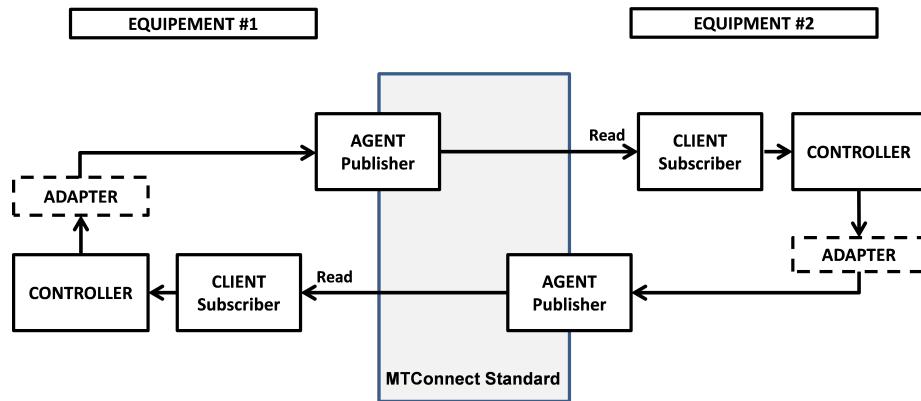


Figure 1: Data Flow Architecture for Interfaces

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

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

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

89 3.2 Request and Response Information Exchange

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

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

105 Note: The implementation details of how the *responder* piece of equipment
 106 reacts to the *request* and then completes the requested task are up to the im-
 107 plementer.

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

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

- 113 • The *request* transitions from READY to ACTIVE signaling that a service is needed.
- 114 • The *response* detects the transition of the *request*.
- 115 • The *response* transitions from READY to ACTIVE indicating that it is performing
 116 the action.
- 117 • Once the action has been performed, the *response* transitions to COMPLETE.
- 118 • The *request* detects the action is COMPLETE.
- 119 • The *request* transitions back to READY acknowledging that the service has been
 120 performed.

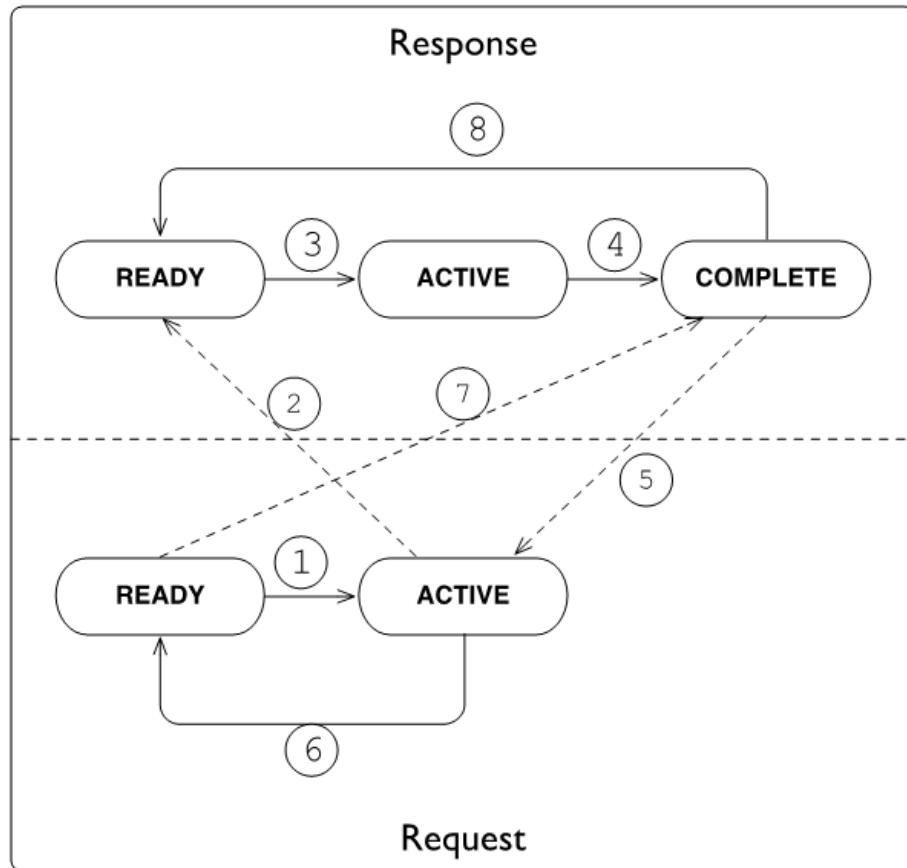


Figure 2: Request and Response Overview

121 • The *response* detects the *request* has returned to READY.

122 • In recognition of this acknowledgement, the *response* transitions back to READY.

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

125 3.3 Interface

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

127 3.3.1 Commonly Observed DataItem Types for Interface

128 *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

129 4 Interfaces for Device and Observation Information Mod- 130 els

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

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

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

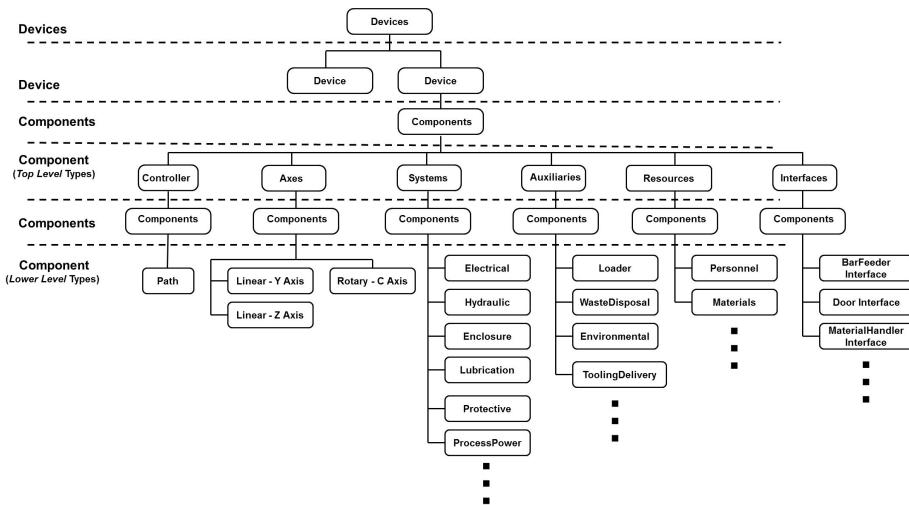


Figure 3: Interfaces in Entity Hierarchy

139 4.1 Interface Types

¹⁴⁰ The abstract `Interface` is realized by the following types listed in this section.

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

144 4.1.1 BarFeederInterface

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

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

150 4.1.2 ChuckInterface

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

153 The piece of equipment that is controlling the chuck **MUST** provide the data item `Chuck-`
154 `State` as part of the set of information provided.

155 4.1.3 DoorInterface

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

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

160 4.1.4 MaterialHandlerInterface

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

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

166 • Loading/unloading material or tooling

167 • Part inspection

168 • Testing

169 • Cleaning

170 A robot is a common example of a material handler.

171 4.2 Data for Interface

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

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

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

180 4.2.1 References for Interface

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

188 *References* *organizes* Reference elements.

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

191 *References* is an economical syntax for providing interface specific information with-
192 out directly duplicating the occurrence of the data. It provides a mechanism to include all
193 necessary information required for interaction and deterministic information flow between
194 pieces of equipment.

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

197 4.3 DataItem Types for Interface

198 Each `Interface` contains data items which are used to communicate information re-
 199 quired to execute the *interface*. When these data items are read by another piece of equip-
 200 ment, that piece of equipment can then determine the actions that it may take based upon
 201 that data.

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

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

208 4.3.1 Specific Data Items for the Interaction Model for Interface

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

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

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

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

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

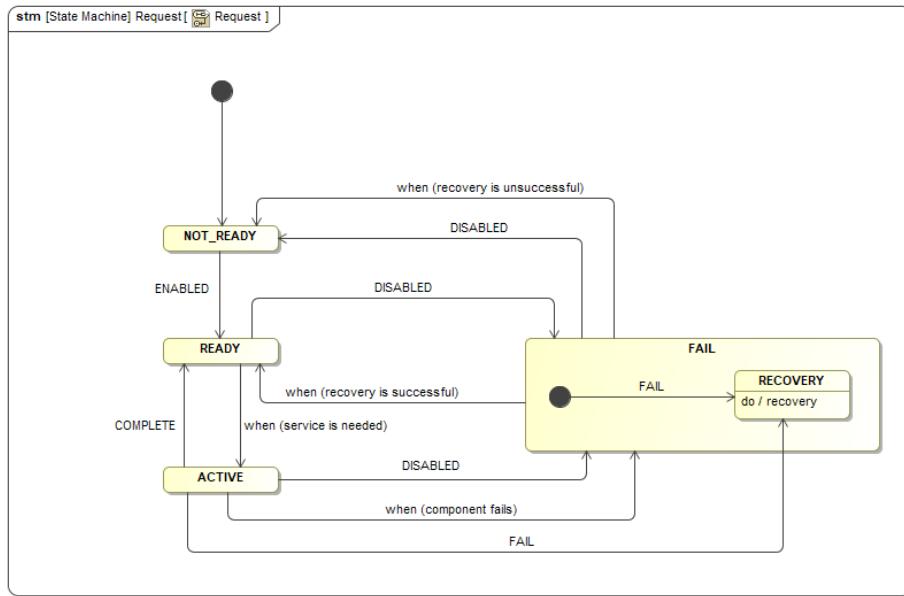


Figure 4: Request State Machine

232 4.3.2 CloseChuck

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

234 4.3.2.1 Subtypes of CloseChuck

235 • REQUEST

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

237 RequestStateEnum Enumeration:

238 – ACTIVE

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

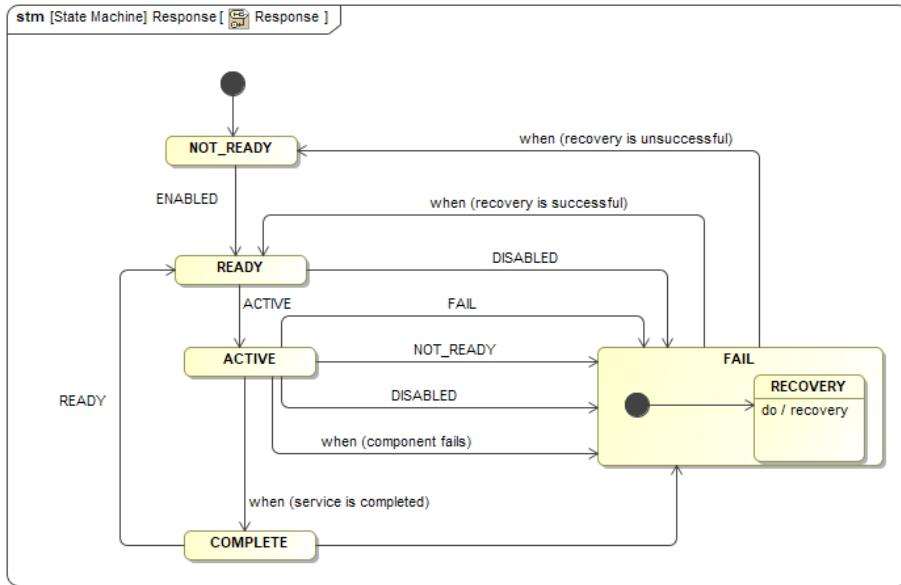


Figure 5: Response State Machine

241 - FAIL
 242 *requester* has detected a failure condition.

243 - NOT_READY
 244 *requester* is not ready to make a *request*.

245 - READY
 246 *requester* is prepared to make a *request*, but no *request* for service is required.

247 • RESPONSE
 248 operating state of the *response* to a *request* to close a chuck.

249 ResponseStateEnum Enumeration:

250 - ACTIVE
 251 *responder* has detected and accepted a *request* for a service and is in the process
 252 of performing the service, but the service has not yet been completed.

253 - COMPLETE
 254 *responder* has completed the actions required to perform the service.

255 - FAIL
 256 *responder* has detected a failure condition.

257 - NOT_READY
 258 *responder* is not ready to perform a service.

259 - READY
 260 *responder* is prepared to react to a *request*, but no *request* for service has been
 261 detected.

262 **4.3.3 CloseDoor**

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

264 **4.3.3.1 Subtypes of CloseDoor**

265 • REQUEST
 266 operating state of the *request* to close a door.
 267 The value of CloseDoor **MUST** be one of the RequestStateEnum enumera-
 268 tion.
 269 • RESPONSE
 270 operating state of the *response* to a *request* to close a door.
 271 The value of CloseDoor **MUST** be one of the ResponseStateEnum enumera-
 272 tion.

273 **4.3.4 InterfaceState**

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

276 InterfaceStateEnum Enumeration:

277 • DISABLED
 278 Interface is currently not operational.
 279 • ENABLED
 280 Interface is currently operational and performing as expected.

281 **4.3.5 MaterialChange**282 A subType **MUST** always be specified.283 **4.3.5.1 Subtypes of MaterialChange**

284 • REQUEST

285 operating state of the *request* to change the type of material or product being loaded
286 or fed to a piece of equipment.287 The value of MaterialChange **MUST** be one of the RequestStateEnum
288 enumeration.

289 • RESPONSE

290 operating state of the *response* to a *request* to change the type of material or product
291 being loaded or fed to a piece of equipment.292 The value of MaterialChange **MUST** be one of the ResponseStateEnum
293 enumeration.294 **4.3.6 MaterialFeed**295 A subType **MUST** always be specified.296 **4.3.6.1 Subtypes of MaterialFeed**

297 • REQUEST

298 operating state of the *request* to advance material or feed product to a piece of equip-
299 ment from a continuous or bulk source.300 The value of MaterialFeed **MUST** be one of the RequestStateEnum enum-
301 eration.

302 • RESPONSE

303 operating state of the *response* to a *request* to advance material or feed product to a
304 piece of equipment from a continuous or bulk source.305 The value of MaterialFeed **MUST** be one of the ResponseStateEnum enum-
306 eration.

307 **4.3.7 MaterialLoad**

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

309 **4.3.7.1 Subtypes of MaterialLoad**

310 • REQUEST

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

312 The value of MaterialLoad **MUST** be one of the RequestStateEnum enum-
313 eration.

314 • RESPONSE

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

316 The value of MaterialLoad **MUST** be one of the ResponseStateEnum enum-
317 eration.

318 **4.3.8 MaterialRetract**

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

320 **4.3.8.1 Subtypes of MaterialRetract**

321 • REQUEST

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

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

325 • RESPONSE

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

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

329 **4.3.9 MaterialUnload**

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

331 **4.3.9.1 Subtypes of MaterialUnload**

332 • REQUEST

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

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

336 • RESPONSE

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

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

340 **4.3.10 OpenChuck**

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

342 **4.3.10.1 Subtypes of OpenChuck**

343 • REQUEST

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

345 The value of OpenChuck **MUST** be one of the RequestStateEnum enumera-
346 tion.

347 • RESPONSE

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

349 The value of OpenChuck **MUST** be one of the ResponseStateEnum enumera-
350 tion.

351 **4.3.11 OpenDoor**

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

353 **4.3.11.1 Subtypes of OpenDoor**

354 • REQUEST

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

356 The value of OpenDoor **MUST** be one of the RequestStateEnum enumera-
357 tion.

358 • RESPONSE

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

360 The value of OpenDoor **MUST** be one of the ResponseStateEnum enumera-
361 tion.

362 **4.3.12 PartChange**

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

364 **4.3.12.1 Subtypes of PartChange**

365 • REQUEST

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

368 The value of PartChange **MUST** be one of the RequestStateEnum enumera-
369 tion.

370 • RESPONSE

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

373 The value of PartChange **MUST** be one of the ResponseStateEnum enumera-
374 tion.

375 5 Operation and Error Recovery

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

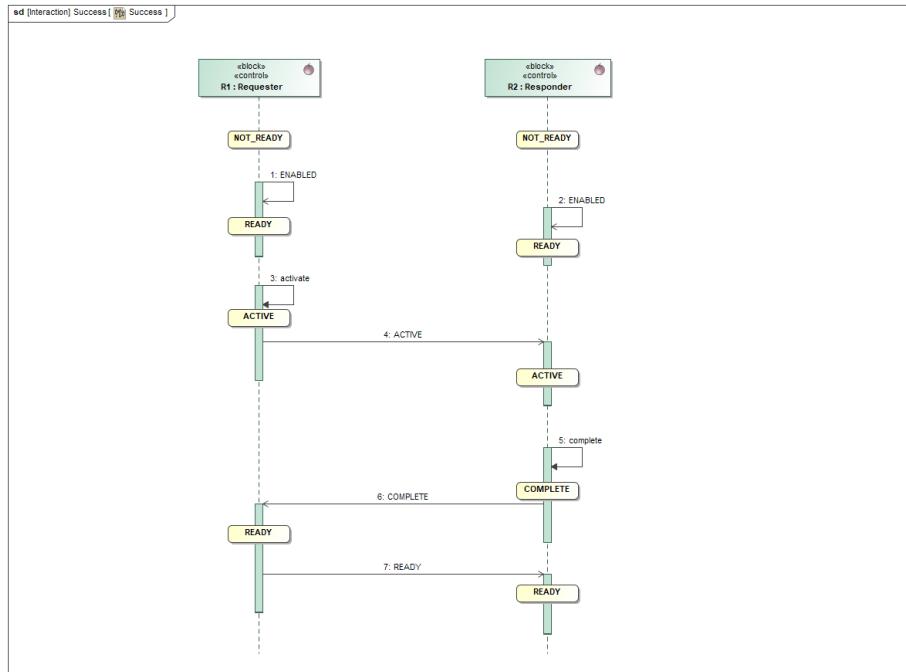


Figure 6: Success Scenario

380 5.1 Request and Response Failure Handling and Recovery

381 A significant feature of the *request and response interaction model* is the ability for ei-
 382 ther piece of equipment to detect a failure associated with either the *request* or *response*
 383 actions. When either a failure or unexpected action occurs, the *request* and the *response*
 384 portion of the *interaction model* can announce a FAIL state upon detecting a problem. The
 385 following are graphical models describing multiple scenarios where either the *requester* or
 386 or *responder* detects and reacts to a failure. In these examples, either the *requester* or
 387 or *responder* announces the detection of a failure by setting either the *request* or the *response*
 388 state to FAIL.

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

391 with the *interface* to their original state, and return to normal operation.
 392 The following sections are scenarios that describe how pieces of equipment may react to
 393 different types of failures and how they indicate when they are again ready to request a
 394 service or respond to a request for service after recovering from those failures:

395 5.1.1 Responder Fails Immediately

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

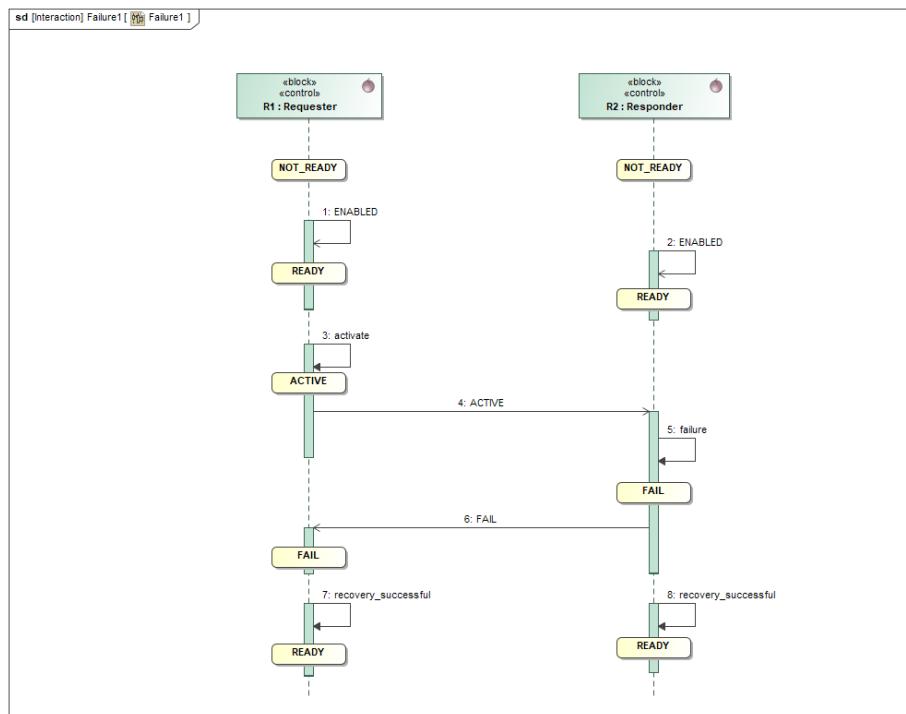


Figure 7: Responder Fails Immediately

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

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

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

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

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

413 5.1.2 Responder Fails While Providing a Service

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

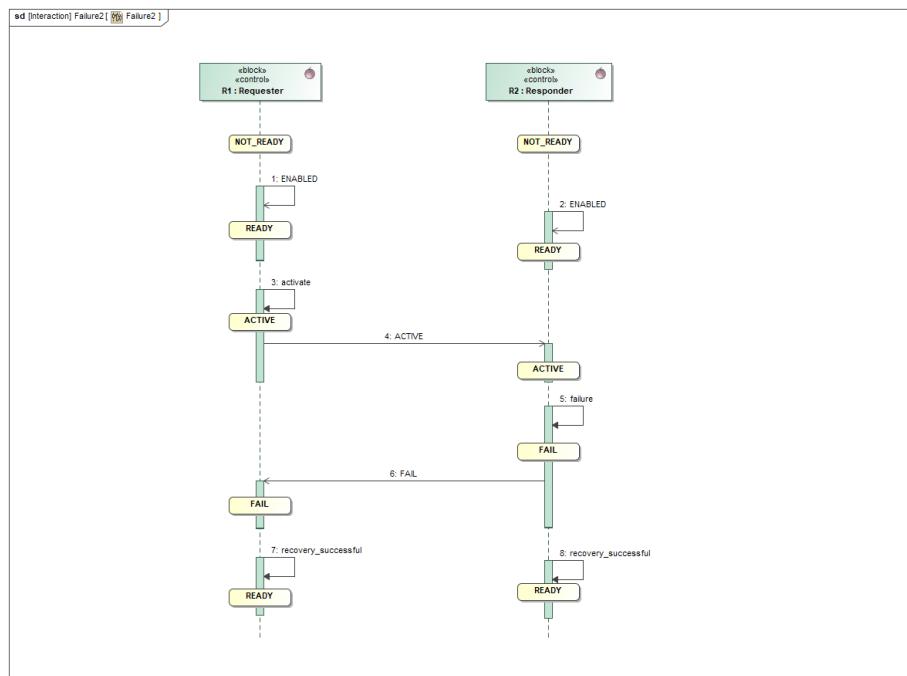


Figure 8: Responder Fails While Providing a Service

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

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

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

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

430 5.1.3 Requester Failure During a Service Request

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

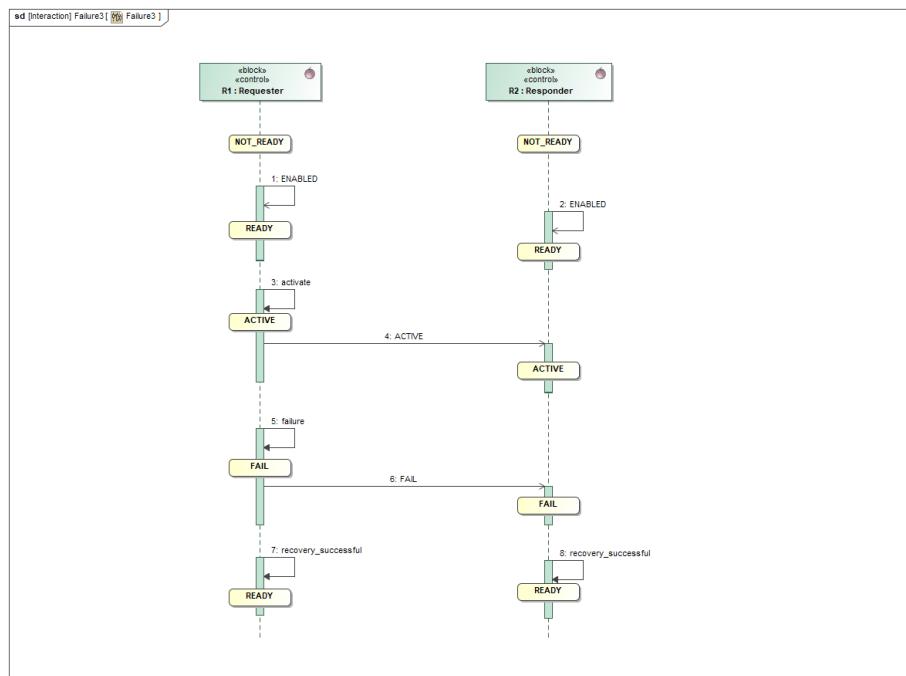


Figure 9: Requester Fails During a Service Request

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

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

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

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

445 5.1.4 Requester Changes to an Unexpected State While Responder is 446 Providing a Service

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

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

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

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

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

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

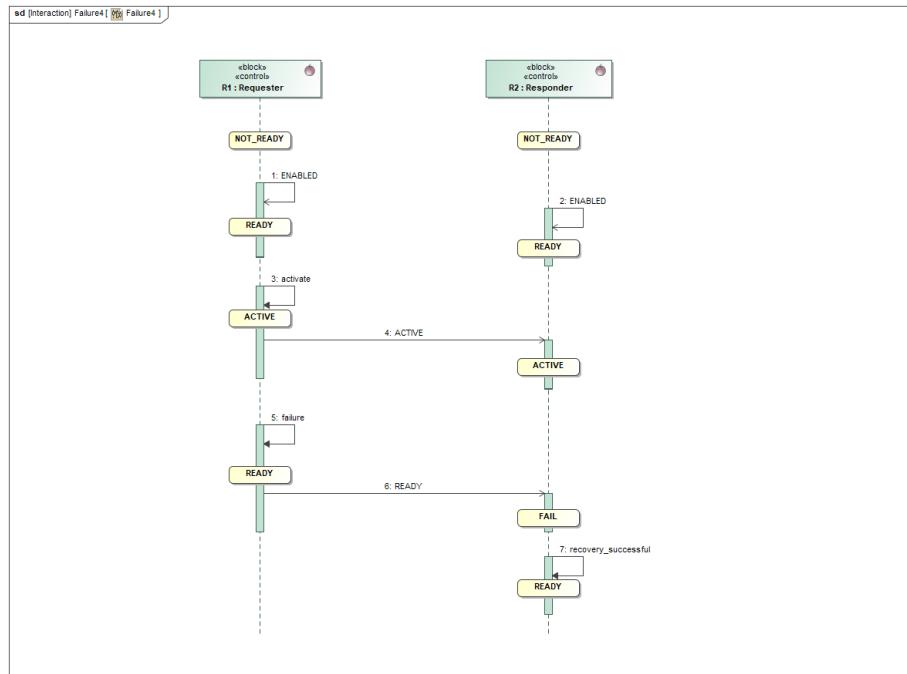


Figure 10: Requester Makes Unexpected State Change

467 **5.1.5 Responder Changes to an Unexpected State While Providing a**
 468 **Service**

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

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

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

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

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

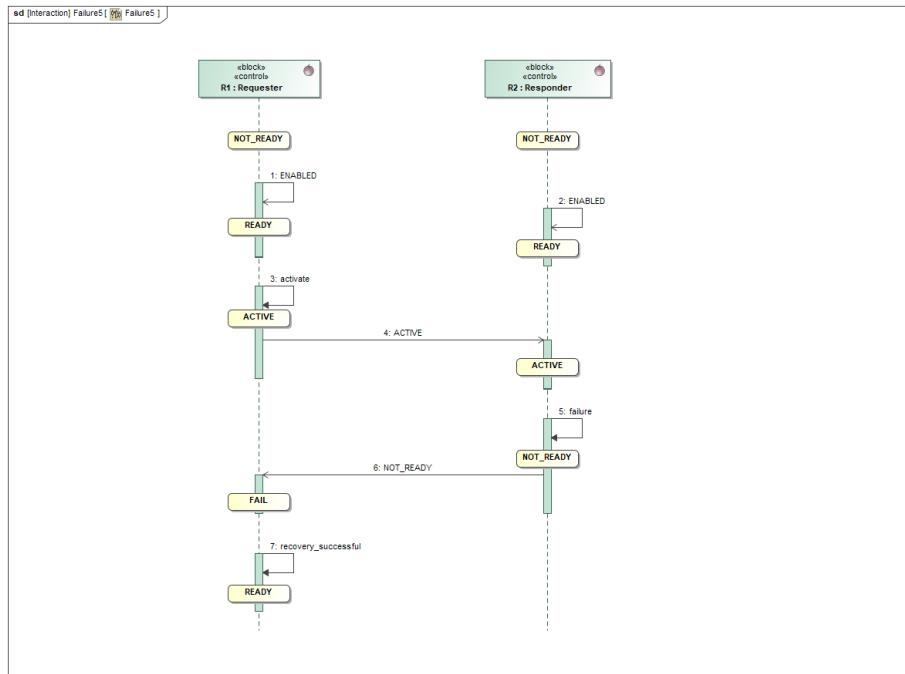


Figure 11: Responder Makes Unexpected State Change

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

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

489 5.1.6 Responder or Requester Become UNAVAILABLE or Experience a Loss of Communication

491 In this scenario, a failure occurs in the communications connection between the *responder*
 492 and *requester*. This failure may result from the `InterfaceState` from either piece of
 493 equipment returning a value of `UNAVAILABLE` or one of the pieces of equipment does
 494 not provide a heartbeat within the desired amount of time (See *MTConnect Standard Part*
 495 *1.0 - Fundamentals* for details on heartbeat).

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

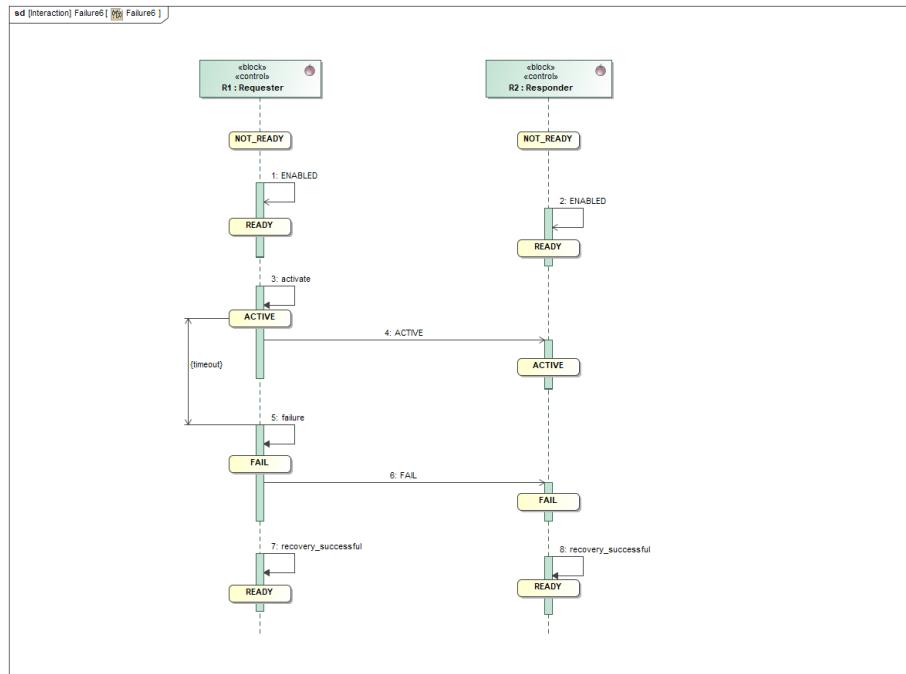


Figure 12: Requester - Responder Communication Failure 1

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

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

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

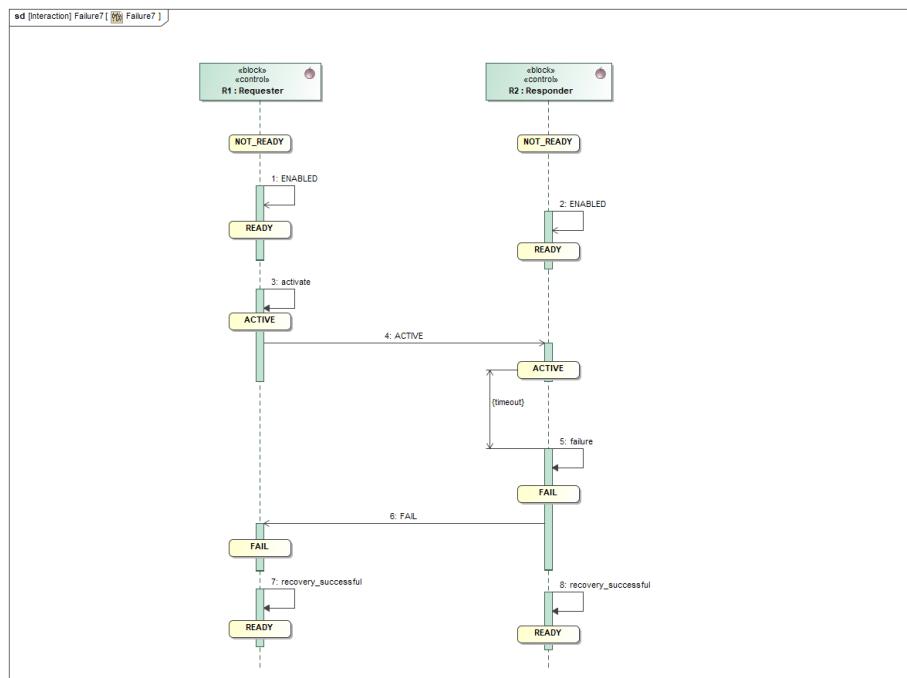


Figure 13: Requester - Responder Communication Failure 2

511 6 Profile

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

514 6.1 DataTypes

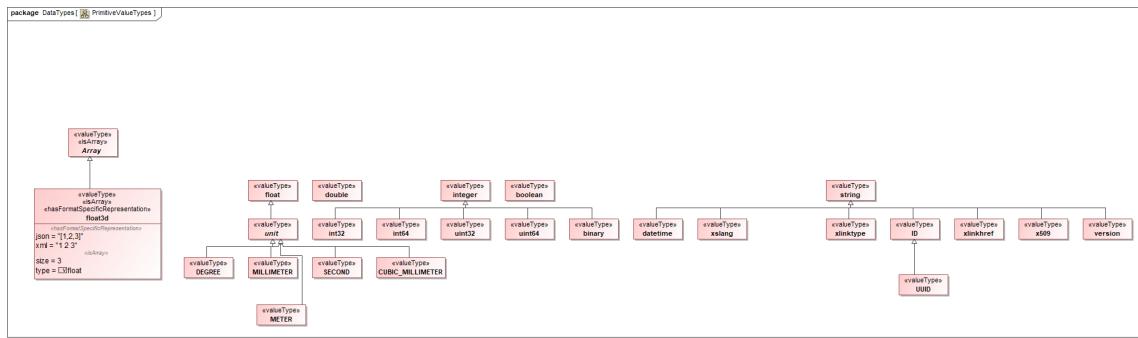


Figure 14: DataTypes

515 6.1.1 boolean

516 primitive type.

517 6.1.2 ID

518 string that represents an identifier (ID).

519 6.1.3 string

520 primitive type.

521 6.1.4 float

522 primitive type.

523 **6.1.5 `datetime`**

524 string that represents timestamp in ISO 8601 format.

525 **6.1.6 `integer`**

526 primitive type.

527 **6.1.7 `xlinktype`**

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

530 **6.1.8 `xslang`**

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

533 **6.1.9 `SECOND`**

534 float that represents time in seconds.

535 **6.1.10 `xlinkhref`**

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

538 **6.1.11 `x509`**

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

540 **6.1.12 int32**

541 32-bit integer.

542 **6.1.13 int64**

543 64-bit integer.

544 **6.1.14 version**

545 series of three numeric values, separated by a decimal point, representing a *major*, *minor*,
546 and *patch* number of the MTConnect Standard.

547 **6.1.15 uint32**

548 32-bit unsigned integer.

549 **6.1.16 uint64**

550 64-bit unsigned integer.

551 **6.1.17 binary**

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

553 **6.1.18 double**

554 primitive type.

555 **6.1.19 Array**

556 array.

557 **6.1.20 <>hasFormatSpecificRepresentation>>float3d**

558 array of size 3 and datatype float.

559 **6.1.21 UUID**

560 Universally Unique IDentifier. *Ref IETF:RFC-4122*

561 **6.1.22 METER**

562 float that represents measurement in meter.

563 **6.2 Stereotypes**

564 **6.2.1 organizer**

565 element that *organizes* other elements of a type.

566 **6.2.2 deprecated**

567 element that has been deprecated.

568 **6.2.3 extensible**

569 enumeration that can be extended.

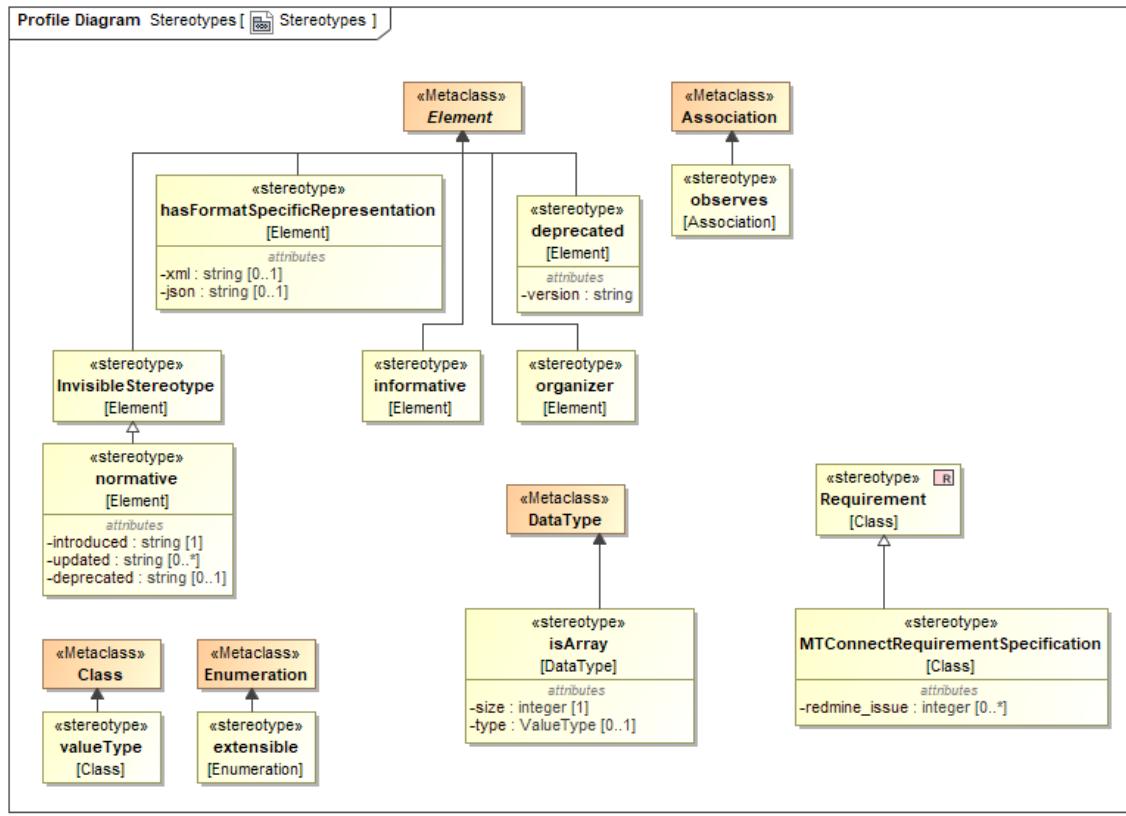


Figure 15: Stereotypes

570 6.2.4 informative

571 element that is descriptive and non-normative.

572 6.2.5 normative

573 element that has been added to the standard.

574 6.2.6 observes

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

576 **6.2.7 satisfiedBy**

577 **6.2.8 hasFormatSpecificRepresentation**

578 element that has format specific representation that might be different from the element's
579 SysML representation.

580 **6.2.9 valueType**

581 extends Class to be used as a SysML <<ValueType>>.

582 **6.2.10 isArray**

583 datatype that is an array.

584 **6.2.11 MTConnectRequirementSpecification**

585 MTConnect Requirement.

586 Appendices

587 A Bibliography

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

591 ISO TC 184/SC4/WG3 N1089. ISO/DIS 10303-238: Industrial automation systems and
592 integration Product data representation and exchange Part 238: Application Protocols: Ap-
593 plication interpreted model for computerized numerical controllers. Geneva, Switzerland,
594 2004.

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

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

601 International Organization for Standardization. ISO 6983/1 – Numerical Control of ma-
602 chines – Program format and definition of address words – Part 1: Data format for posi-
603 tioning, line and contouring control systems. Geneva, Switzerland, 1982.

604 Electronic Industries Association. ANSI/EIA-494-B-1992, 32 Bit Binary CL (BCL) and
605 7 Bit ASCII CL (ACL) Exchange Input Format for Numerically Controlled Machines.
606 Washington, D.C. 1992.

607 National Aerospace Standard. Uniform Cutting Tests - NAS Series: Metal Cutting Equip-
608 ment Specifications. Washington, D.C. 1969.

609 International Organization for Standardization. ISO 10303-11: 1994, Industrial automa-
610 tion systems and integration Product data representation and exchange Part 11: Descrip-
611 tion methods: The EXPRESS language reference manual. Geneva, Switzerland, 1994.

612 International Organization for Standardization. ISO 10303-21: 1996, Industrial automa-
613 tion systems and integration – Product data representation and exchange – Part 21: Imple-
614 mentation methods: Clear text encoding of the exchange structure. Geneva, Switzerland,
615 1996.

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

617 New York, 1984.

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

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

623 ASME/ANSI B5.54: Methods for Performance Evaluation of Computer Numerically Con-
624 trolled Machining Centers. 2005.

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

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

632 IEEE STD 1451.4-1994, Standard for a Smart Transducer Interface for Sensors and Ac-
633 tuators – Mixed-Mode Communication Protocols and Transducer Electronic Data Sheet
634 (TEDS) Formats, IEEE Instrumentation and Measurement Society, TC-9, The Institute of
635 Electrical and Electronics Engineers, Inc., New York, N.Y. 10016, SH95225, December
636 15, 2004.