Project Cerberus PLDM Firmware Update 1.0

Contents

1 Introduction

This document details the execution flow of a Platform Level Data Model firmware update within the Project Cerberus framework. It outlines the procedural steps, modules, and interactions between various components involved in the firmware update process. By leveraging PLDM, Project Cerberus ensures a seamless, secure, and reliable firmware update process, reinforcing the overall security posture of the hardware platform.

Understanding the execution flow of a PLDM firmware update within Project Cerberus is essential for developers, system administrators, and security professionals who aim to implement or maintain secure firmware management practices. This introduction sets the stage for a comprehensive exploration of how Project Cerberus employs PLDM to achieve its security objectives in firmware management.

1.1 Additional Documents

The following documents provide additional details and clarification left out of this document for the sake of brevity. These are crucial to understand much of the terminology and processes explained in this document.

DMTF DSP0236, MCTP Base Specification 1.2.0

http://dmtf.org/sites/default/files/standards/documents/DSP0236_1.2.0.pdf

DMTF DSP0240, Platform Level Data Model (PLDM) Base Specification 1.0

http://dmtf.org/sites/default/files/standards/documents/DSP0240_1.0.0.pdf

DMTF DSP0241, Platform Level Data Model (PLDM) Over MCTP Binding Specification 1.0

http://dmtf.org/sites/default/files/standards/documents/DSP0241_1.0.0.pdf

DMTF DSP0267, Platform Level Data Model (PLDM) for Firmware Update Specification 1.0.1

https://dmtf.org/sites/default/files/standards/documents/DSP0267_1.0.1.pdf

Microsoft Azure Project Cerberus Goals and Design https://github.com/opencomputeproject/Project_Olympus/tree/master/Project_Cerberus

2 Project Cerberus PLDM Firmware Update

The core functionality for performing a PLDM firmware update is contained within the core/pldm folder in Cerberus. The source code layout is as follows:

Source	Description
cmd_interface_pldm	Derived type to handle the processing of PLDM type commands from the MCTP interface.
pldm_fwup_handler	A handler for performing the full firmware update.
pldm_fwup_manager	Module for managing context and state during a firmware update.
pldm_fwup_protocol_commands	Functions to process and generate PLDM firmware update request and response type messages.
pldm_fwup_protocol	Header containing various structures, macros, and enumerations used by other parts of PLDM.

2.1 PLDM Command Interface

Project Cerberus defines a generic command interface called `cmd_interface` for processing requests and responses in a command protocol. `cmd_interface_pldm` extends `cmd_interface` to handle PLDM specific commands. It inherits the properties and function pointers from `cmd_interface` which are then defined during its initialization. Currently `cmd_interface_pldm` only processes PLDM firmware update command types, but can be further extended to process others such as PLDM for FRU commands.

2.1.1 Command Interfaces and MCTP

Cerberus uses MCTP as the protocol for which messages are exchanged throughout a Cerberus managed subsystem. MCTP is a flexible standard that can encapsulate other protocols such as Cerberus's own command protocol, SPDM, or in this case PLDM. During the processing of MCTP packets Cerberus will interpret the MCTP header and extract the message type field which descries the type of payload that packet is carrying. The payload is then passed along to its respective command interface for further processing.

2.2 PLDM FWUP Manager

The `pldm_fwup_manager` is a system for managing the state of a PLDM-based firmware update and allowing other parts of the Cerberus system to modify or view the information present in the firmware update commands. An instance of it is passed along to the 'cmd_interface_pldm' so that during the processing of firmware update commands the information needed to populate or save the fields of the commands can be accomplished.

2.3 PLDM FWUP Protocol Commands

The `pldm_fwup_protocol_commands' contain functions which perform the actual decoding and encoding of PLDM commands saving information to or populating message fields with information from the PLDM FWUP manager. For example,  `pldm_fwup_process_request_update_request()`

function is used to process incoming RequestUpdate PLDM commands saving information in the request to the manager and extracting the manager's context to generate a RequestUpdate response.

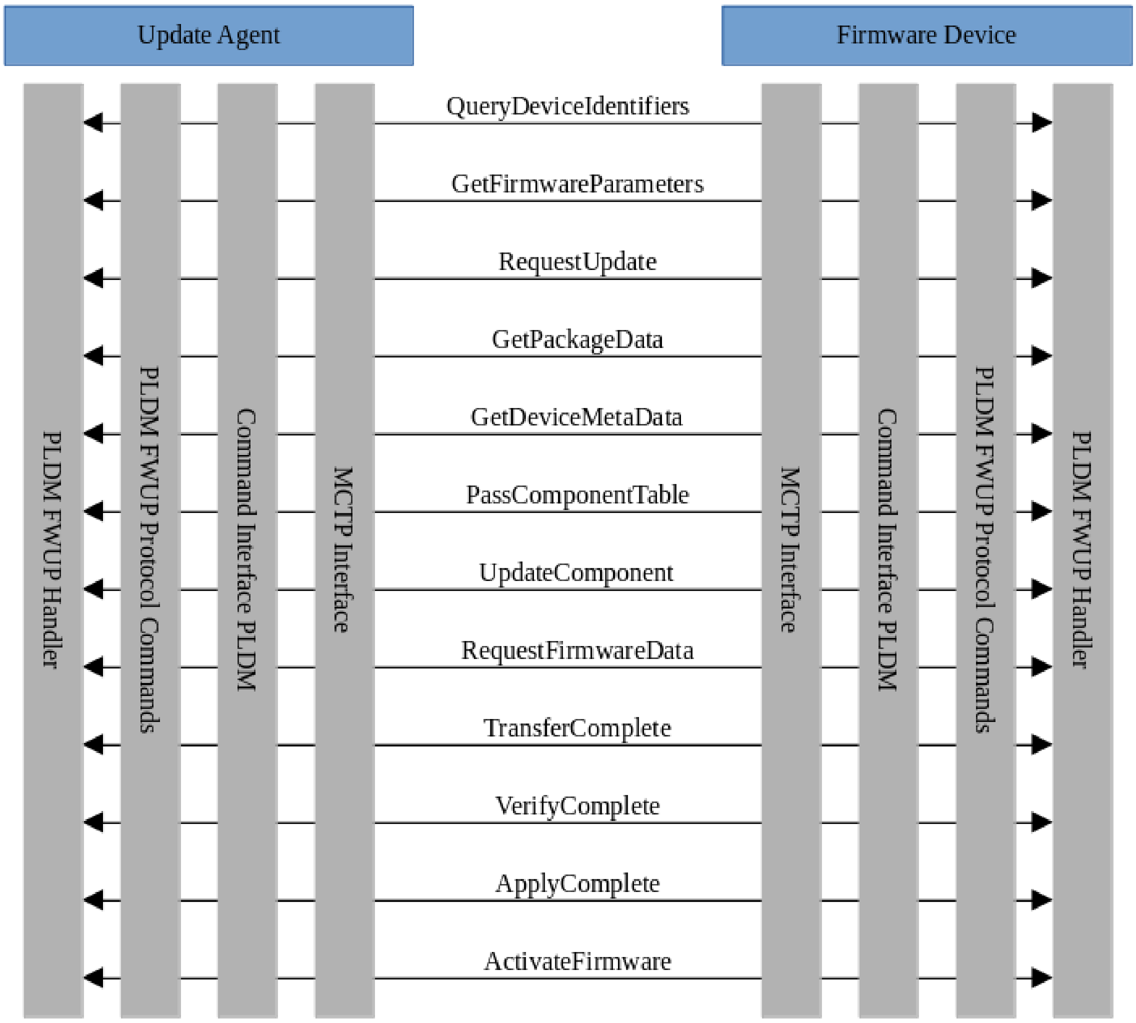
2.4 PLDM FWUP Handler

The `pldm_fwup_handler` is the main driver code of a firmware update. The handler mainly calls the generate request API of the PLDM FWUP protocol commands and the MCTP interface API to generate, send, receive, process, and respond to PLDM firmware update commands.

Important is the fact that at any time Cerberus can be operating as either Update Agent (UA), performing firmware update on another device it manages or as the actual Firmware Device (FD) being updated. As such the `pldm_fwup_handler` interface contains two function pointers: `run_update_ua` for updating another device in the subsystem and 'start_update_fd` for updating Cerberus's own firmware as directed by another UA.

3 Cerberus PLDM Firmware Update Flow

The PLDM firmware update is performed in a sequential manner. Implementation for parallel operation and message exchange is left up to the user of this document. Additionally, an interface for handling the Firmware Update Package has not been implemented and is also left up to the user of this document to configure. Additionally, the PLDM FWUP manager must be initialized and the PLDM commander interface must obtain a reference to the manager prior to the start of the firmware update process.



3.1 Cerberus Operating as UA Flow

1. First Cerberus will check if it needs to send the two inventory commands QueryDeviceIdentifiers and GetFirmwareParameters based on a control boolean passed to `run_update_ua`. Cerberus often sends these commands if it does not have any of the device descriptors such as the PCI device ID or the PCI vendor ID or if it does not have any information about the current firmware residing on the device.

   @Note: In the figures Company represent Server/Update Agent and PRODUCT represents Clients/Firmware Device.

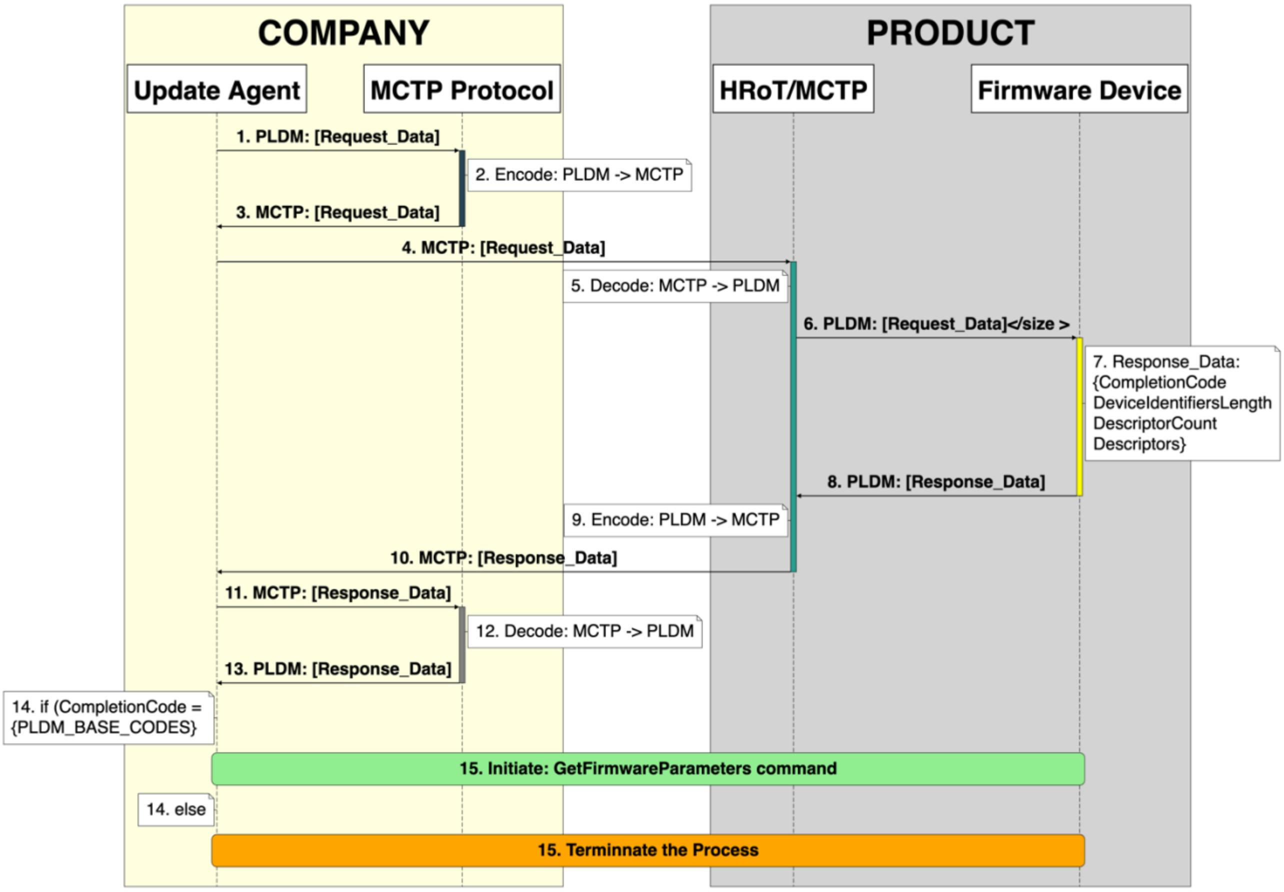

COMPANY
PRODUCT
Update Agent
MCTP Protocol
HRoT/MCTP
Firmware Device
1. PLDM: [Request_Data]
2. Encode: PLDM -> MCTP
3. MCTP: [Request_Data]
4. MCTP: [Request_Data]
5. Decode: MCTP -> PLDM
6. PLDM: [Request_Data]</size >
7. Response_Data: {CompletionCode DeviceIdentifiersLength DescriptorCount Descriptors}
8. PLDM: [Response_Data]
9. Encode: PLDM -> MCTP
10. MCTP: [Response_Data]
11. MCTP: [Response_Data]
12. Decode: MCTP -> PLDM
13. PLDM: [Response_Data]
14. if (CompletionCode = {PLDM_BASE_CODES}
15. Initiate: GetFirmwareParameters command
14. else
15. Terminnate the Process

Figure 1: QueryDeviceIdentifiers Sequence Diagram

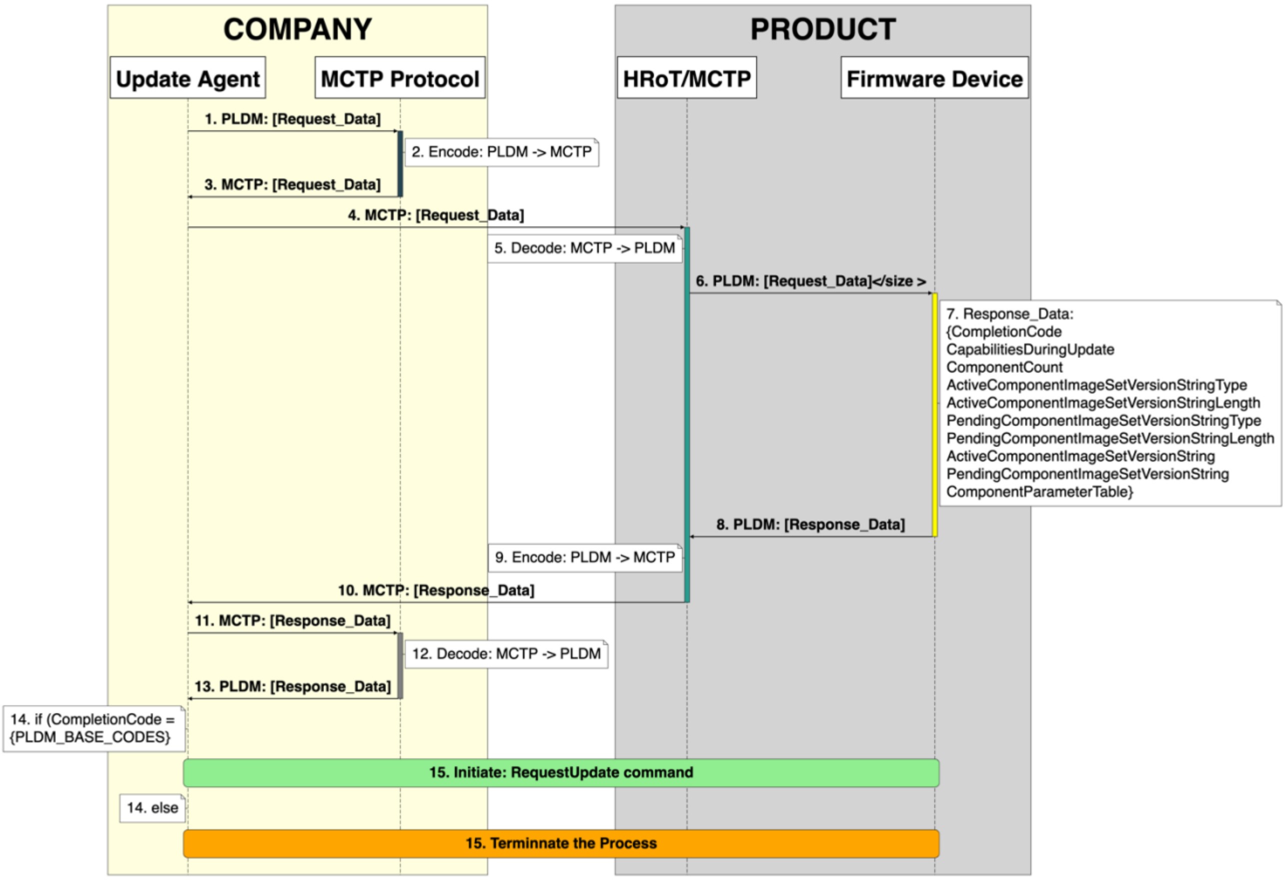

COMPANY
PRODUCT
Update Agent
MCTP Protocol
HRoT/MCTP
Firmware Device
1. PLDM: [Request_Data]
2. Encode: PLDM -> MCTP
3. MCTP: [Request_Data]
4. MCTP: [Request_Data]
5. Decode: MCTP -> PLDM
6. PLDM: [Request_Data]</size >
7. Response_Data: {CompletionCode CapabilitiesDuringUpdate ComponentCount ActiveComponentImageSetVersionStringType ActiveComponentImageSetVersionStringLength PendingComponentImageSetVersionStringType PendingComponentImageSetVersionStringLength ActiveComponentImageSetVersionString PendingComponentImageSetVersionString ComponentParameterTable}
8. PLDM: [Response_Data]
9. Encode: PLDM -> MCTP
10. MCTP: [Response_Data]
11. MCTP: [Response_Data]
12. Decode: MCTP -> PLDM
13. PLDM: [Response_Data]
14. if (CompletionCode = {PLDM_BASE_CODES}
15. Initiate: RequestUpdate command
14. else
15. Terminnate the Process

Figure 2: GetFirmwareParameters Sequence Diagram

2. After sending the two inventory commands Cerberus will then issue the RequestUpdate command signaling to the device that a firmware update is eminent. The RequestUpdate command passes some necessary information such as the maximum transfer size and number of outstanding requests the device is allowed to issue to Cerberus. Both of these parameters can be configured in the `platform_config` of the projects folder. The length of the package data, the version of the component image set, and the number of components Cerberus will update is also passed to the device.

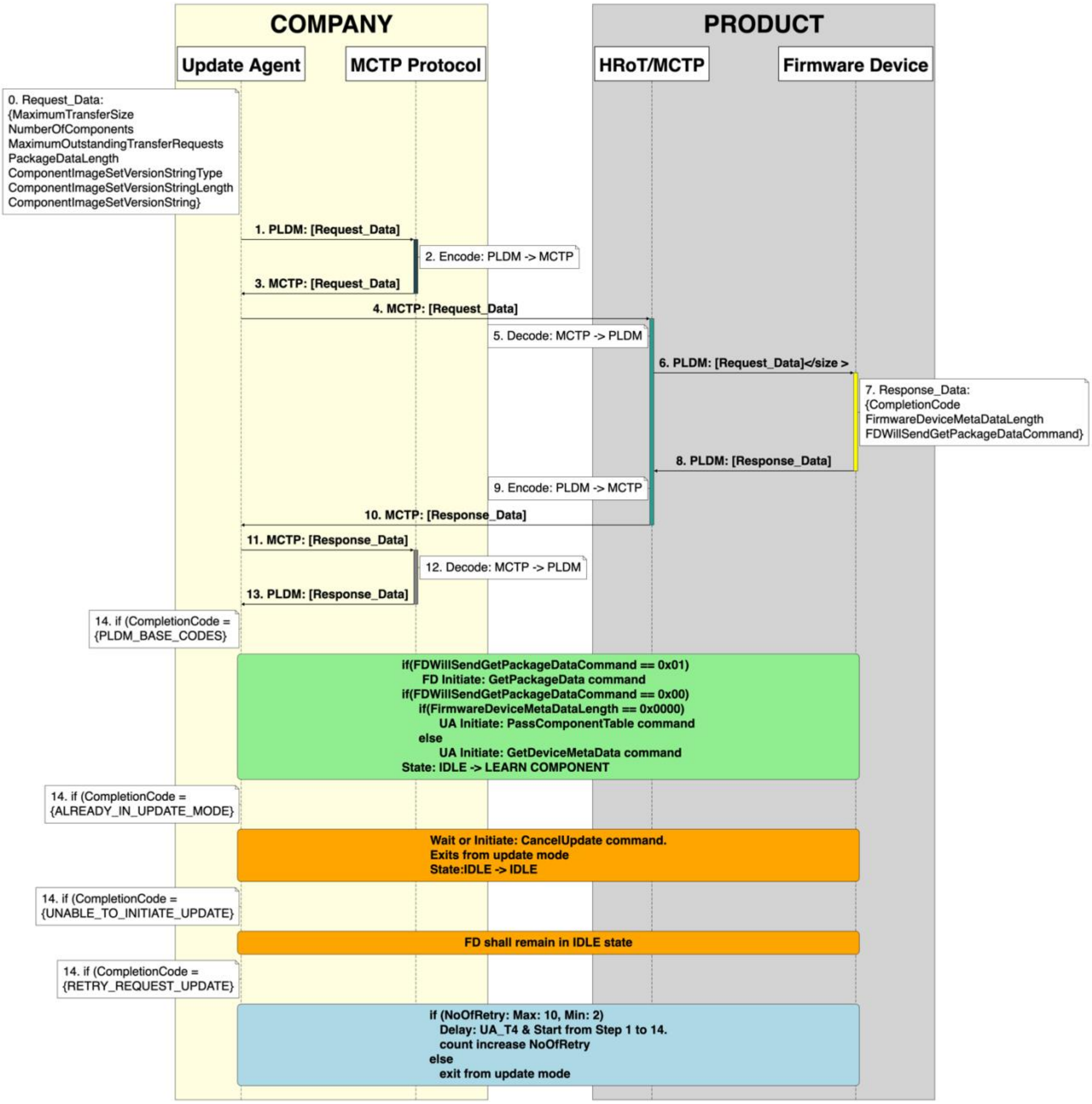
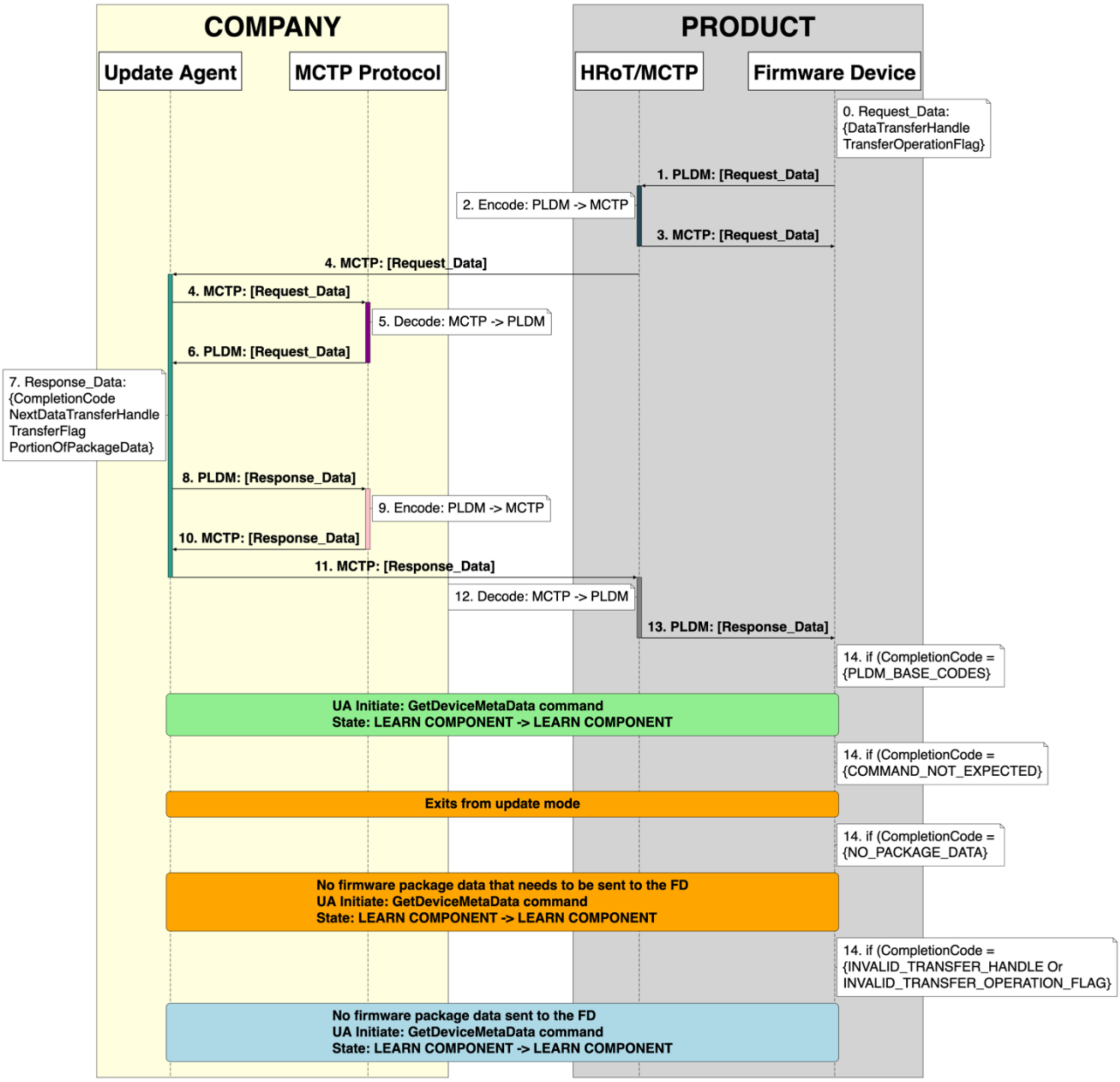

COMPANY
PRODUCT
Update Agent
MCTP Protocol
HRoT/MCTP
Firmware Device
0. Request_Data:
{MaximumTransferSize
NumberOfComponents
MaximumOutstandingTransferRequests
PackageDataLength
ComponentImageSetVersionStringType
ComponentImageSetVersionStringLength
ComponentImageSetVersionString}
1. PLDM: [Request_Data]
2. Encode: PLDM -> MCTP
3. MCTP: [Request_Data]
4. MCTP: [Request_Data]
5. Decode: MCTP -> PLDM
6. PLDM: [Request_Data]</size >
7. Response_Data:
{CompletionCode
FirmwareDeviceMetaDataLength
FDWillSendGetPackageDataCommand}
8. PLDM: [Response_Data]
9. Encode: PLDM -> MCTP
10. MCTP: [Response_Data]
11. MCTP: [Response_Data]
12. Decode: MCTP -> PLDM
13. PLDM: [Response_Data]
14. if (CompletionCode =
{PLDM_BASE_CODES}
if(FDWillSendGetPackageDataCommand == 0x01)
FD Initiate: GetPackageData command
if(FDWillSendGetPackageDataCommand == 0x00)
if(FirmwareDeviceMetaDataLength == 0x0000)
UA Initiate: PassComponentTable command
else
UA Initiate: GetDeviceMetaData command
State: IDLE -> LEARN COMPONENT
14. if (CompletionCode =
{ALREADY_IN_UPDATE_MODE}
Wait or Initiate: CancelUpdate command.
Exits from update mode
State:IDLE -> IDLE
14. if (CompletionCode =
{UNABLE_TO_INITIATE_UPDATE}
FD shall remain in IDLE state
14. if (CompletionCode =
{RETRY_REQUEST_UPDATE}
if (NoOfRetry: Max: 10, Min: 2)
Delay: UA_T4 & Start from Step 1 to 14.
count increase NoOfRetry
else
exit from update mode

Figure 3: RequestUpdate Sequence Diagram

3. If package data was indicated then the device will communicate to Cerberus that it will send the GetPackageData command in the response to RequestUpdate along with the length of any metadata the device needs Cerberus to retain. The device will proceed to issue the GetPackageData command, if need be, with Cerberus responding with the package data read

from flash memory. Once all the package data is transferred Cerberus will then issue the GetDeviceMetaData command, again if need be. In the current implementation this metadata is written to a region in flash although depending on what the metadata is it could be written to a structure or any other volatile memory.



COMPANY
PRODUCT
Update Agent
MCTP Protocol
HRoT/MCTP
Firmware Device
0. Request_Data: {DataTransferHandle TransferOperationFlag}
1. PLDM: [Request_Data]
2. Encode: PLDM -> MCTP
3. MCTP: [Request_Data]
4. MCTP: [Request_Data]
4. MCTP: [Request_Data]
5. Decode: MCTP -> PLDM
6. PLDM: [Request_Data]
7. Response_Data: {CompletionCode NextDataTransferHandle TransferFlag PortionOfPackageData}
8. PLDM: [Response_Data]
9. Encode: PLDM -> MCTP
10. MCTP: [Response_Data]
11. MCTP: [Response_Data]
12. Decode: MCTP -> PLDM
13. PLDM: [Response_Data]
14. if (CompletionCode = {PLDM_BASE_CODES}
UA Initiate: GetDeviceMetaData command
State: LEARN COMPONENT -> LEARN COMPONENT
14. if (CompletionCode = {COMMAND_NOT_EXPECTED}
Exits from update mode
14. if (CompletionCode = {NO_PACKAGE_DATA}
No firmware package data that needs to be sent to the FD
UA Initiate: GetDeviceMetaData command
State: LEARN COMPONENT -> LEARN COMPONENT
14. if (CompletionCode = {INVALID_TRANSFER_HANDLE Or INVALID_TRANSFER_OPERATION_FLAG}
No firmware package data sent to the FD
UA Initiate: GetDeviceMetaData command
State: LEARN COMPONENT -> LEARN COMPONENT

Figure 4: GetPackageData Sequence Diagram

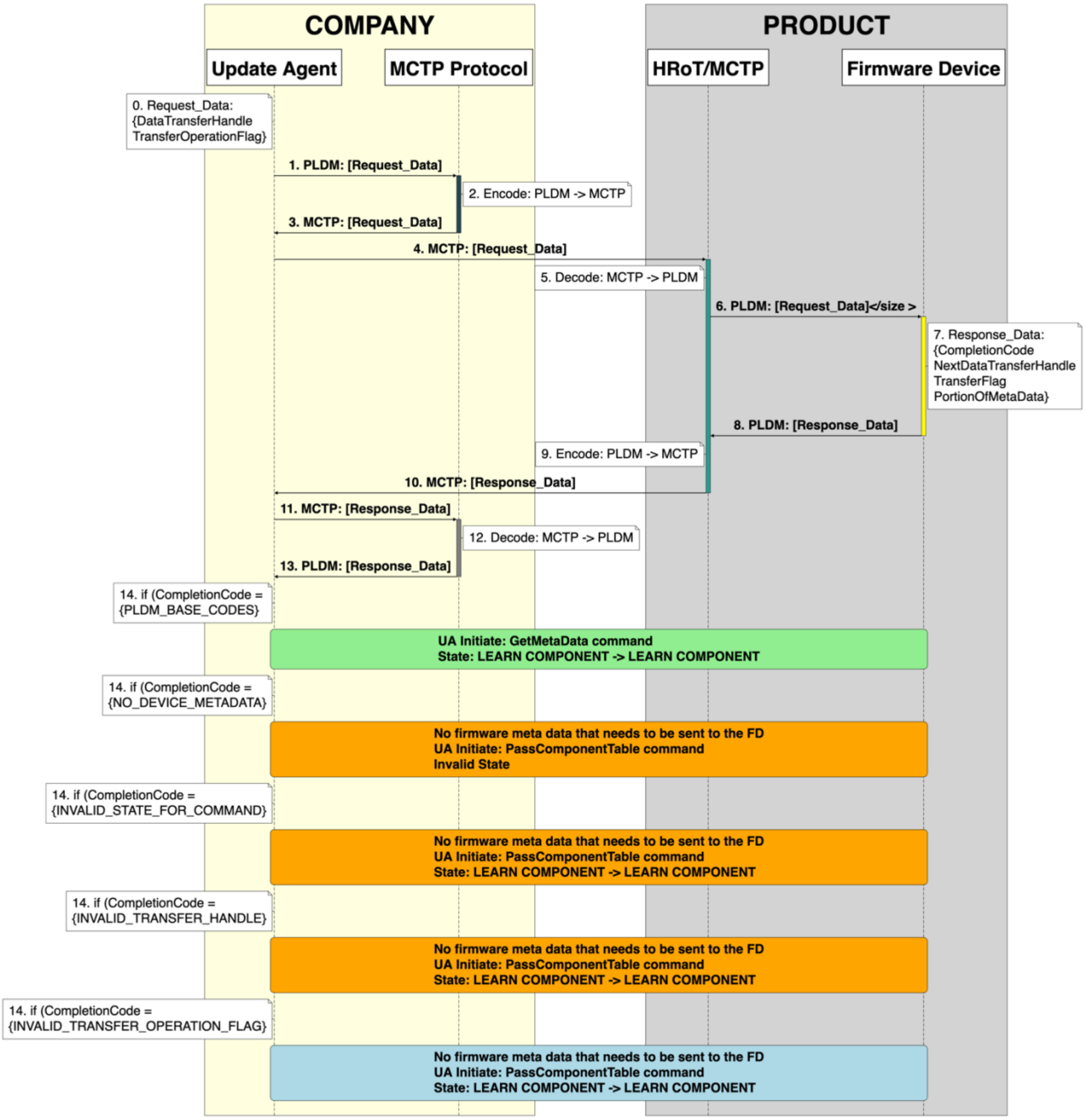
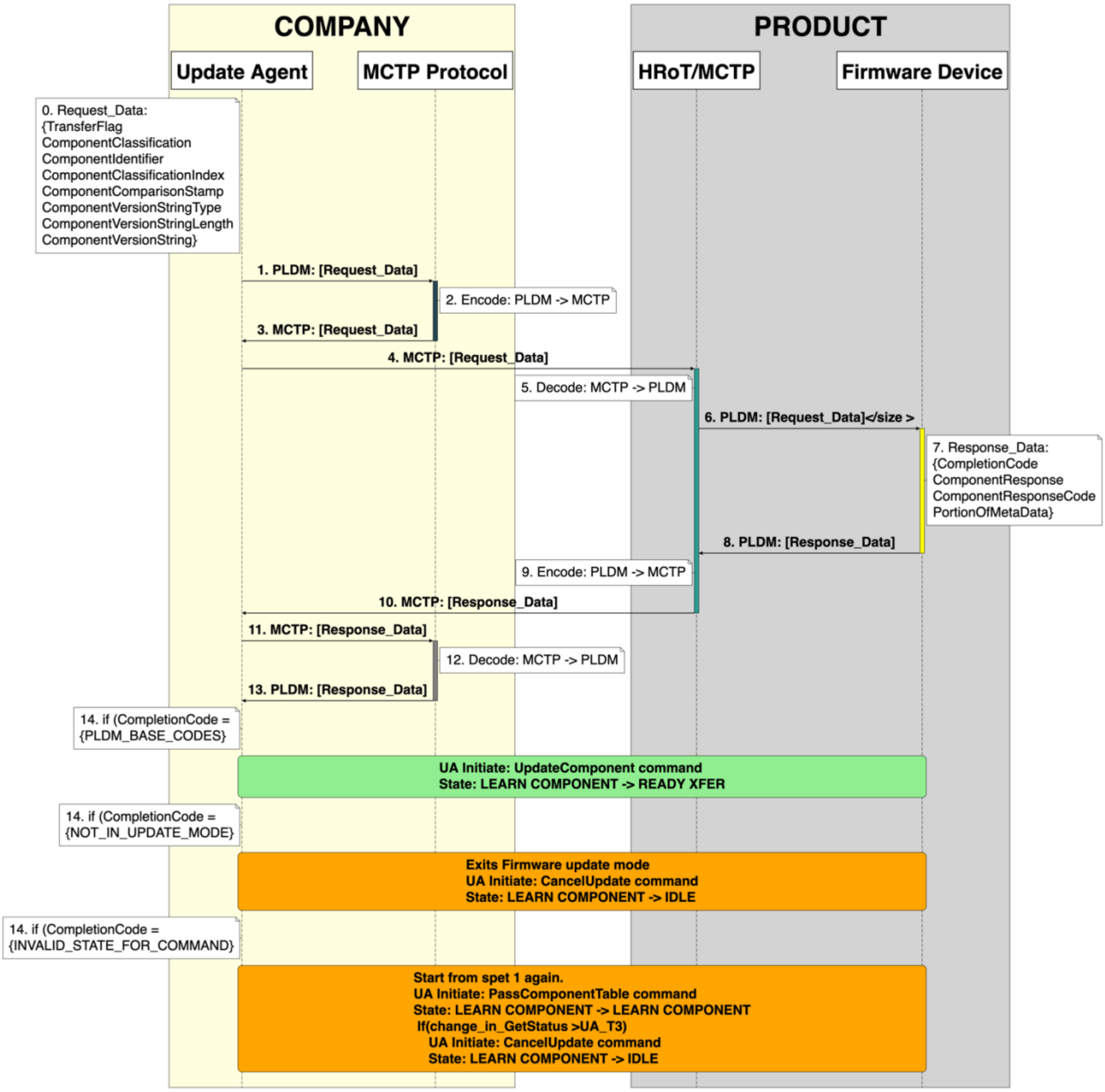

COMPANY
PRODUCT
Update Agent
MCTP Protocol
HRoT/MCTP
Firmware Device
0. Request_Data: {DataTransferHandle TransferOperationFlag}
1. PLDM: [Request_Data]
2. Encode: PLDM -> MCTP
3. MCTP: [Request_Data]
4. MCTP: [Request_Data]
5. Decode: MCTP -> PLDM
6. PLDM: [Request_Data]</size >
7. Response_Data: {CompletionCode NextDataTransferHandle TransferFlag PortionOfMetaData}
8. PLDM: [Response_Data]
9. Encode: PLDM -> MCTP
10. MCTP: [Response_Data]
11. MCTP: [Response_Data]
12. Decode: MCTP -> PLDM
13. PLDM: [Response_Data]
14. if (CompletionCode = {PLDM_BASE_CODES}
UA Initiate: GetMetaData command
State: LEARN COMPONENT -> LEARN COMPONENT
14. if (CompletionCode = {NO_DEVICE_METADATA}
No firmware meta data that needs to be sent to the FD
UA Initiate: PassComponentTable command
Invalid State
14. if (CompletionCode = {INVALID_STATE_FOR_COMMAND}
No firmware meta data that needs to be sent to the FD
UA Initiate: PassComponentTable command
State: LEARN COMPONENT -> LEARN COMPONENT
14. if (CompletionCode = {INVALID_TRANSFER_HANDLE}
No firmware meta data that needs to be sent to the FD
UA Initiate: PassComponentTable command
State: LEARN COMPONENT -> LEARN COMPONENT
14. if (CompletionCode = {INVALID_TRANSFER_OPERATION_FLAG}
No firmware meta data that needs to be sent to the FD
UA Initiate: PassComponentTable command
State: LEARN COMPONENT -> LEARN COMPONENT

Figure 5: GetDeviceMetaData Sequence Diagram

4. After the GetPackageData and GetDeviceMetaData commands, Cerberus will transfer the component table to the device via the PassComponentTable command. The component table contains information about each firmware component which the device uses to check against the firmware components transferred by Cerberus. During the response to each PassComponentTable command the device will indicate its comparability with the component. The implementation does not assume what to do if there is an error in compatibility. The handling of these compatibility codes are left up to the user.



COMPANY
PRODUCT
Update Agent
MCTP Protocol
HRoT/MCTP
Firmware Device
0. Request_Data: {TransferFlag ComponentClassification ComponentIdentifier ComponentClassificationIndex ComponentComparisonStamp ComponentVersionStringType ComponentVersionStringLength ComponentVersionString}
1. PLDM: [Request_Data]
2. Encode: PLDM -> MCTP
3. MCTP: [Request_Data]
4. MCTP: [Request_Data]
5. Decode: MCTP -> PLDM
6. PLDM: [Request_Data]</size >
7. Response_Data: {CompletionCode ComponentResponse ComponentResponseCode PortionOfMetaData}
8. PLDM: [Response_Data]
9. Encode: PLDM -> MCTP
10. MCTP: [Response_Data]
11. MCTP: [Response_Data]
12. Decode: MCTP -> PLDM
13. PLDM: [Response_Data]
14. if (CompletionCode = {PLDM_BASE_CODES}
UA Initiate: UpdateComponent command
State: LEARN COMPONENT -> READY XFER
14. if (CompletionCode = {NOT_IN_UPDATE_MODE}
Exits Firmware update mode
UA Initiate: CancelUpdate command
State: LEARN COMPONENT -> IDLE
14. if (CompletionCode = {INVALID_STATE_FOR_COMMAND}
Start from spet 1 again.
UA Initiate: PassComponentTable command
State: LEARN COMPONENT -> LEARN COMPONENT
If(change_in_GetStatus >UA_T3)
UA Initiate: CancelUpdate command
State: LEARN COMPONENT -> IDLE

Figure 6: PassComponentTable Sequence Diagram

5. After the component table has been passed to the device, Cerberus will issue the UpdateComponent command telling the firmware device which component will be updated next. Similar to PassComponentTable, the device will respond with compatibility codes.

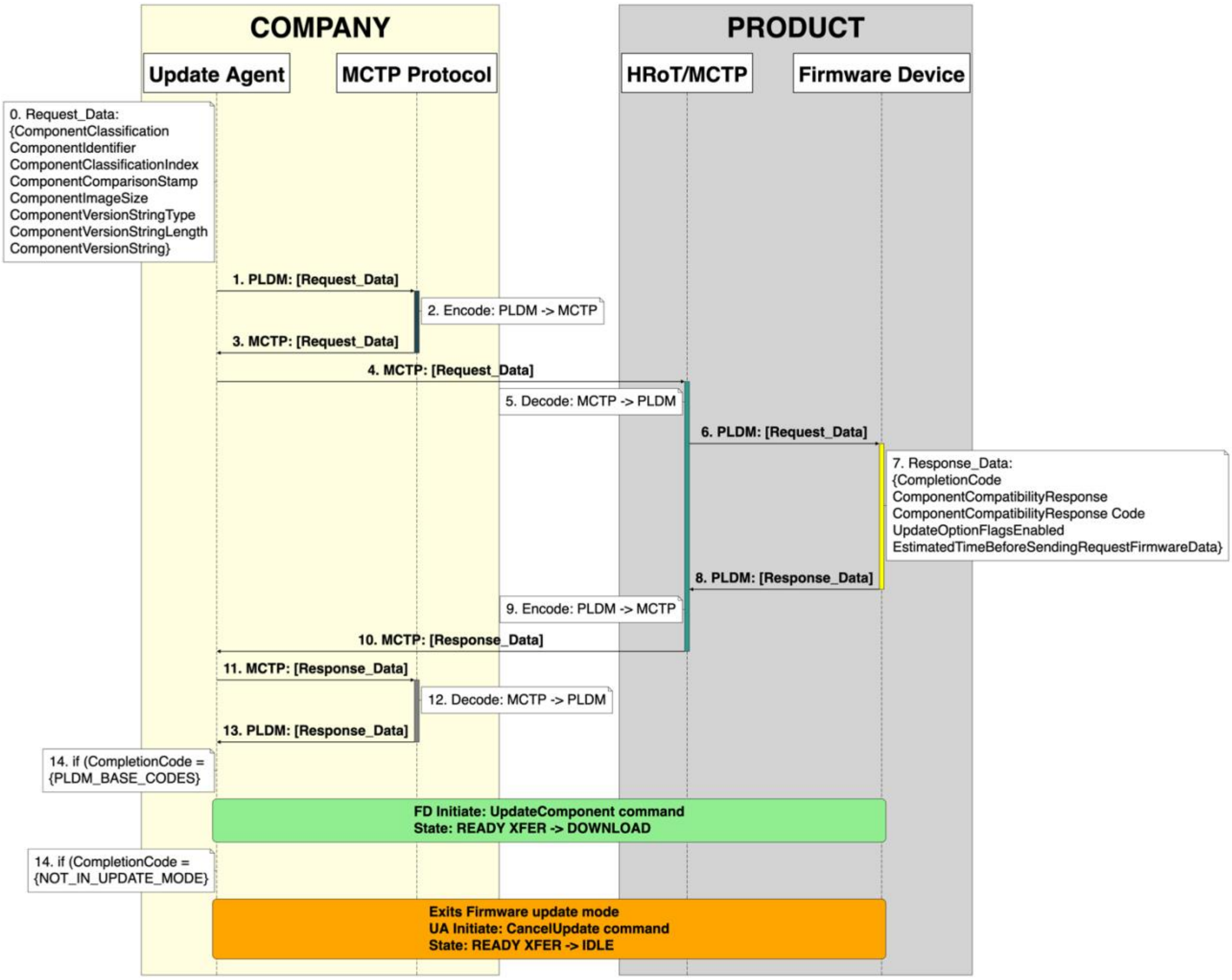

COMPANY
PRODUCT
Update Agent
MCTP Protocol
HRoT/MCTP
Firmware Device
0. Request_Data: {ComponentClassification ComponentIdentifier ComponentClassificationIndex ComponentComparisonStamp ComponentImageSize ComponentVersionStringType ComponentVersionStringLength ComponentVersionString}
1. PLDM: [Request_Data]
2. Encode: PLDM -> MCTP
3. MCTP: [Request_Data]
4. MCTP: [Request_Data]
5. Decode: MCTP -> PLDM
6. PLDM: [Request_Data]
7. Response_Data: {CompletionCode ComponentCompatibilityResponse ComponentCompatibilityResponse Code UpdateOptionFlagsEnabled EstimatedTimeBeforeSendingRequestFirmwareData}
8. PLDM: [Response_Data]
9. Encode: PLDM -> MCTP
10. MCTP: [Response_Data]
11. MCTP: [Response_Data]
12. Decode: MCTP -> PLDM
13. PLDM: [Response_Data]
14. if (CompletionCode = {PLDM_BASE_CODES}
FD Initiate: UpdateComponent command
State: READY XFER -> DOWNLOAD
14. if (CompletionCode = {NOT_IN_UPDATE_MODE}
Exits Firmware update mode
UA Initiate: CancelUpdate command
State: READY XFER -> IDLE

Figure 7: UpdateComponent Sequence Diagram

6. Cerberus will now wait to receive RequestFirmwareData commands from the device. Upon each command Cerberus will respond with the portion of the firmware component image specified by the device.

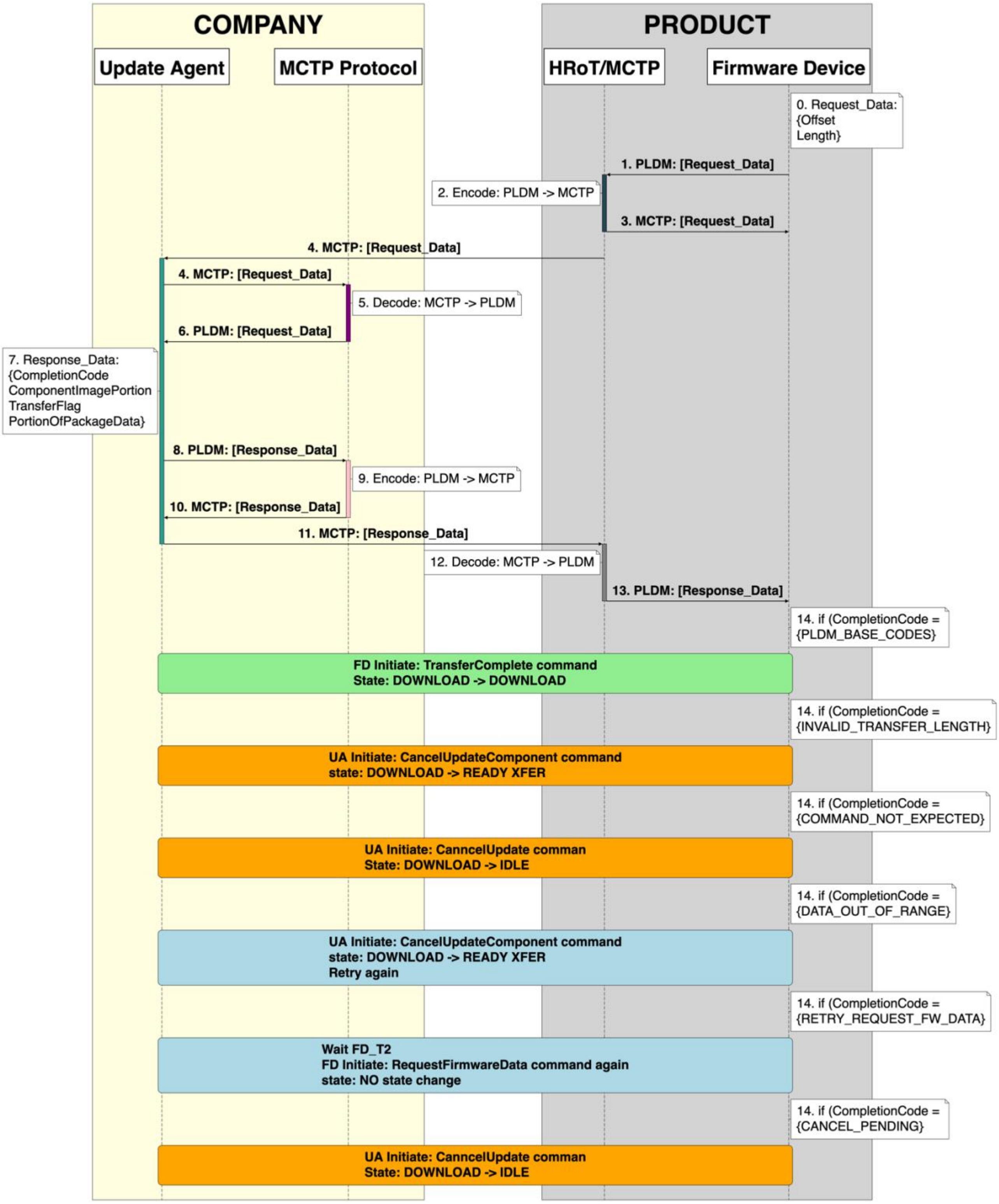

COMPANY
PRODUCT
Update Agent
MCTP Protocol
HRoT/MCTP
Firmware Device
0. Request_Data: {Offset Length}
1. PLDM: [Request_Data]
2. Encode: PLDM -> MCTP
3. MCTP: [Request_Data]
4. MCTP: [Request_Data]
4. MCTP: [Request_Data]
5. Decode: MCTP -> PLDM
6. PLDM: [Request_Data]
7. Response_Data: {CompletionCode ComponentImagePortion TransferFlag PortionOfPackageData}
8. PLDM: [Response_Data]
9. Encode: PLDM -> MCTP
10. MCTP: [Response_Data]
11. MCTP: [Response_Data]
12. Decode: MCTP -> PLDM
13. PLDM: [Response_Data]
14. if (CompletionCode = {PLDM_BASE_CODES}
FD Initiate: TransferComplete command
State: DOWNLOAD -> DOWNLOAD
14. if (CompletionCode = {INVALID_TRANSFER_LENGTH}
UA Initiate: CancelUpdateComponent command
state: DOWNLOAD -> READY XFER
14. if (CompletionCode = {COMMAND_NOT_EXPECTED}
UA Initiate: CanncelUpdate comman
State: DOWNLOAD -> IDLE
14. if (CompletionCode = {DATA_OUT_OF_RANGE}
UA Initiate: CancelUpdateComponent command
state: DOWNLOAD -> READY XFER
Retry again
14. if (CompletionCode = {RETRY_REQUEST_FW_DATA}
Wait FD_T2
FD Initiate: RequestFirmwareData command again
state: NO state change
14. if (CompletionCode = {CANCEL_PENDING}
UA Initiate: CanncelUpdate comman
State: DOWNLOAD -> IDLE

Figure 8: RequestFirmwareData Sequence Diagram

7. The device will issue the TransferComplete command once it obtains the entire component image or it there was an error during the transfer. How the transfer result is set is up to the user (it currently defaults to success). It could be that the function simply exits or that Cerberus will issue a CancelUpdateComponent command to the device.

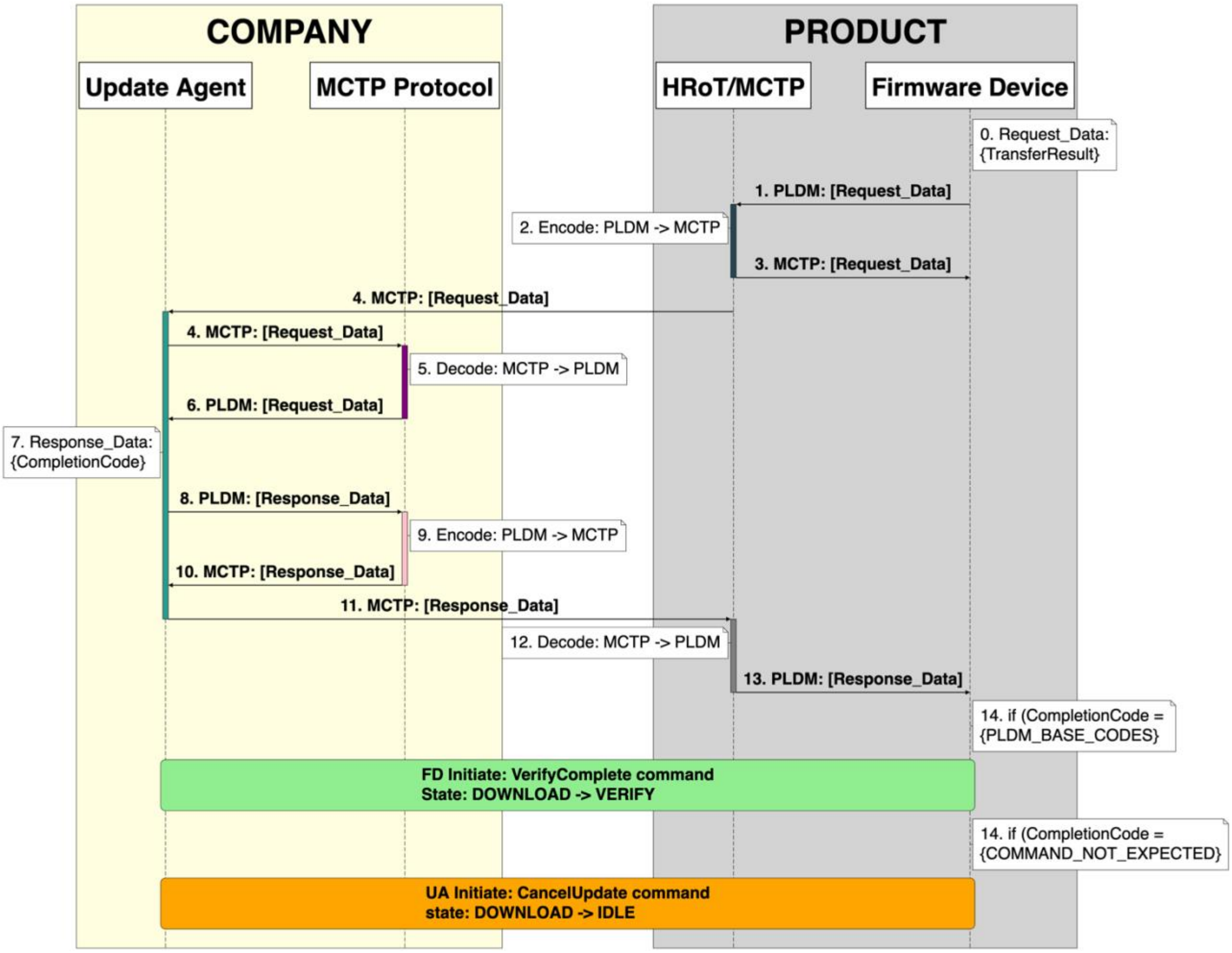

COMPANY
PRODUCT
Update Agent
MCTP Protocol
HRoT/MCTP
Firmware Device
0. Request_Data: {TransferResult}
1. PLDM: [Request_Data]
2. Encode: PLDM -> MCTP
3. MCTP: [Request_Data]
4. MCTP: [Request_Data]
4. MCTP: [Request_Data]
5. Decode: MCTP -> PLDM
6. PLDM: [Request_Data]
7. Response_Data: {CompletionCode}
8. PLDM: [Response_Data]
9. Encode: PLDM -> MCTP
10. MCTP: [Response_Data]
11. MCTP: [Response_Data]
12. Decode: MCTP -> PLDM
13. PLDM: [Response_Data]
14. if (CompletionCode = {PLDM_BASE_CODES}
FD Initiate: VerifyComplete command
State: DOWNLOAD -> VERIFY
14. if (CompletionCode = {COMMAND_NOT_EXPECTED}
UA Initiate: CancelUpdate command
state: DOWNLOAD -> IDLE

Figure 9: TransferComplete Sequence Diagram

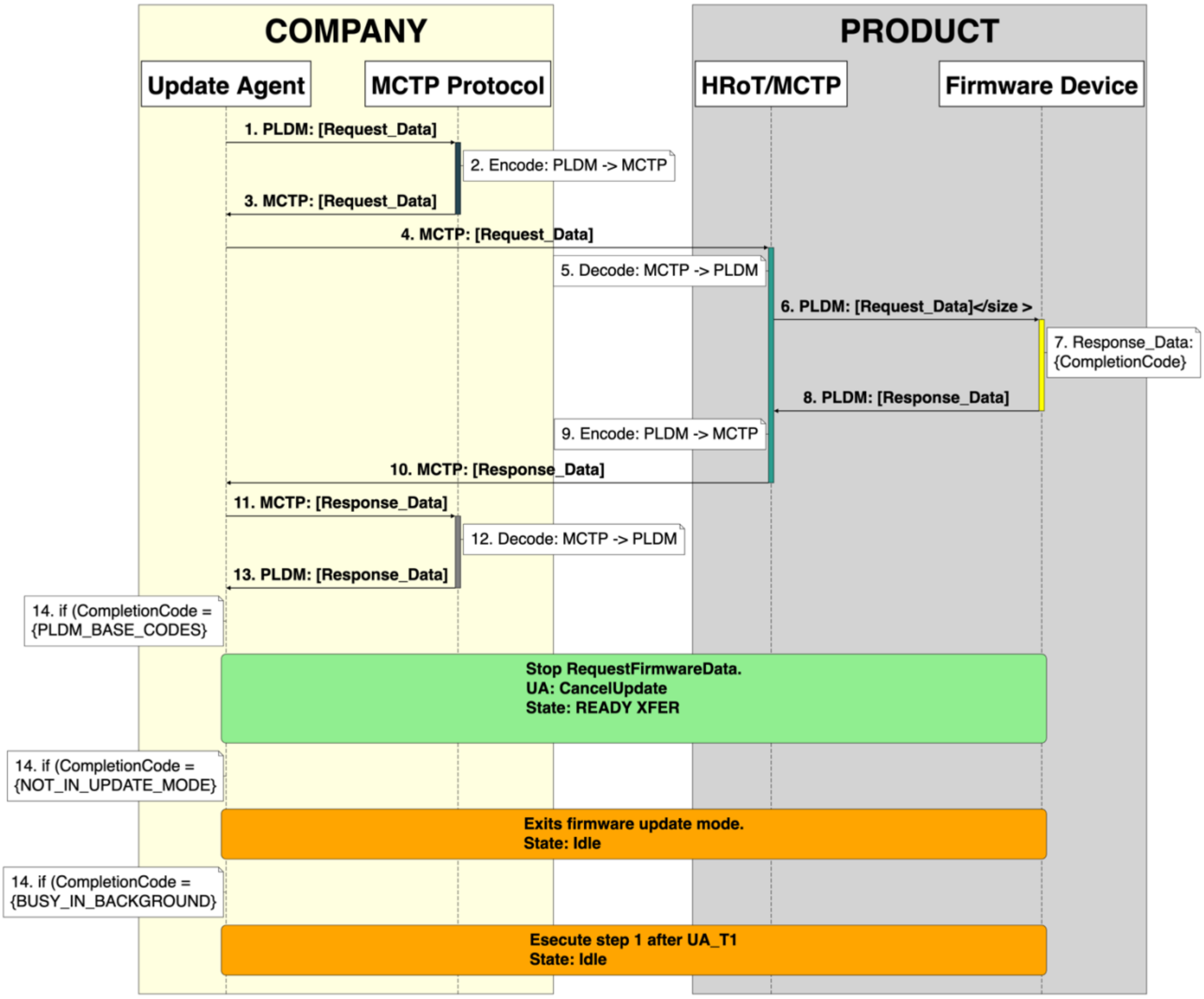

COMPANY
PRODUCT
Update Agent
MCTP Protocol
HRoT/MCTP
Firmware Device
1. PLDM: [Request_Data]
2. Encode: PLDM -> MCTP
3. MCTP: [Request_Data]
4. MCTP: [Request_Data]
5. Decode: MCTP -> PLDM
6. PLDM: [Request_Data]</size >
7. Response_Data: {CompletionCode}
8. PLDM: [Response_Data]
9. Encode: PLDM -> MCTP
10. MCTP: [Response_Data]
11. MCTP: [Response_Data]
12. Decode: MCTP -> PLDM
13. PLDM: [Response_Data]
14. if (CompletionCode = {PLDM_BASE_CODES}
Stop RequestFirmwareData.
UA: CancelUpdate
State: READY XFER
14. if (CompletionCode = {NOT_IN_UPDATE_MODE}
Exits firmware update mode.
State: Idle
14. if (CompletionCode = {BUSY_IN_BACKGROUND}
Esecute step 1 after UA_T1
State: Idle

Figure 10: CancelUpdateComponent Sequence Diagram

8. Cerberus will now wait for the device to verify the firmware image. The current handler will simply wait for a certain number of milliseconds for the VerifyComplete command and exit if a time out is reached. A more robust mechanism is left up to the reader and could include something like periodically sending the GetStatus command to the device to poll the status of the verification. Additionally, the device may respond with an error verify result which the user is left to correct for.

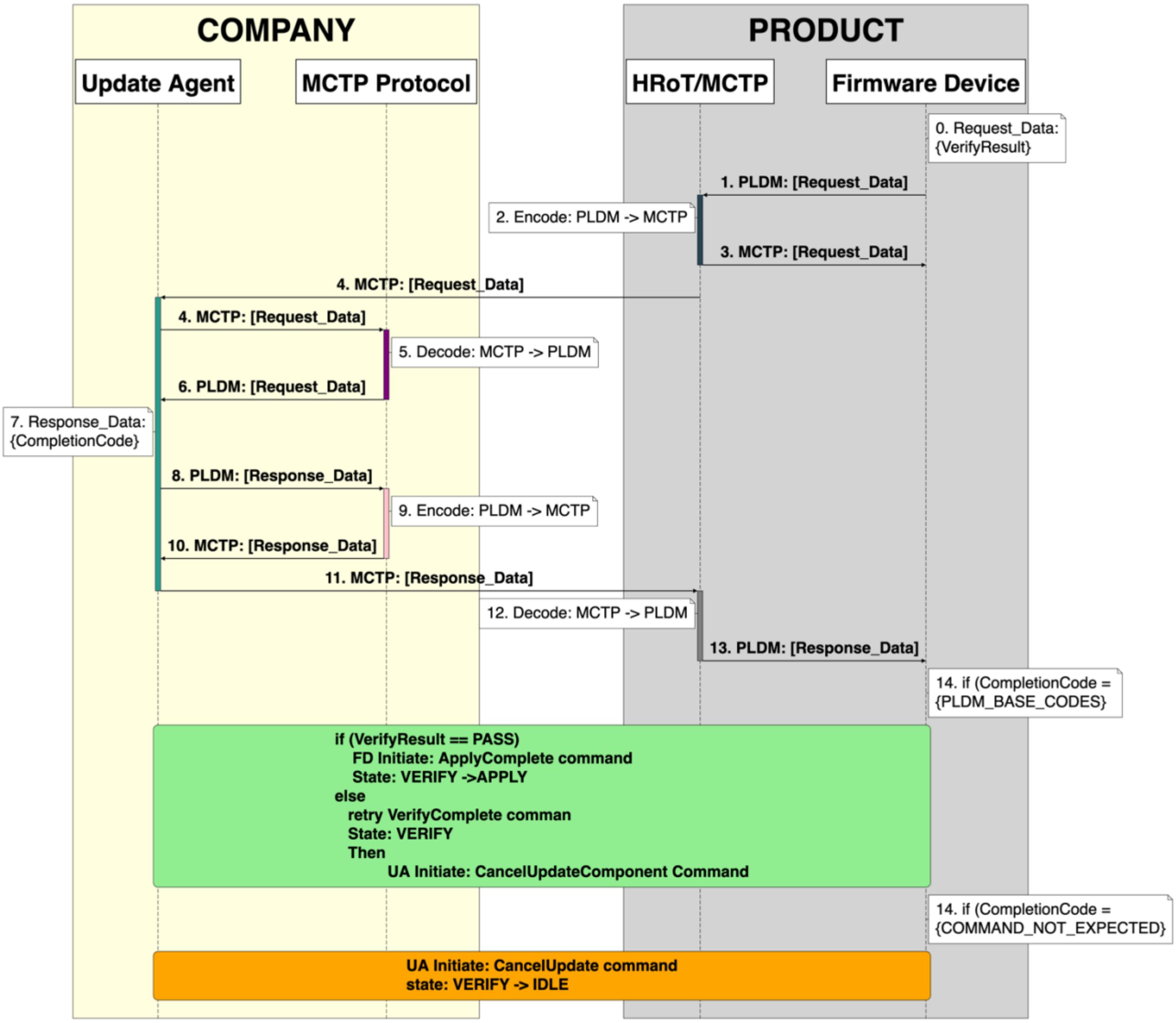

COMPANY
Update Agent
MCTP Protocol
PRODUCT
HRoT/MCTP
Firmware Device
0. Request_Data: {VerifyResult}
1. PLDM: [Request_Data]
2. Encode: PLDM -> MCTP
3. MCTP: [Request_Data]
4. MCTP: [Request_Data]
4. MCTP: [Request_Data]
5. Decode: MCTP -> PLDM
6. PLDM: [Request_Data]
7. Response_Data: {CompletionCode}
8. PLDM: [Response_Data]
9. Encode: PLDM -> MCTP
10. MCTP: [Response_Data]
11. MCTP: [Response_Data]
12. Decode: MCTP -> PLDM
13. PLDM: [Response_Data]
14. if (CompletionCode = {PLDM_BASE_CODES}
if (VerifyResult == PASS)
FD Initiate: ApplyComplete command
State: VERIFY ->APPLY
else
retry VerifyComplete comman
State: VERIFY
Then
UA Initiate: CancelUpdateComponent Command
14. if (CompletionCode = {COMMAND_NOT_EXPECTED}
UA Initiate: CancelUpdate command
state: VERIFY -> IDLE

Figure 11: VerifyComplete Sequence Diagram

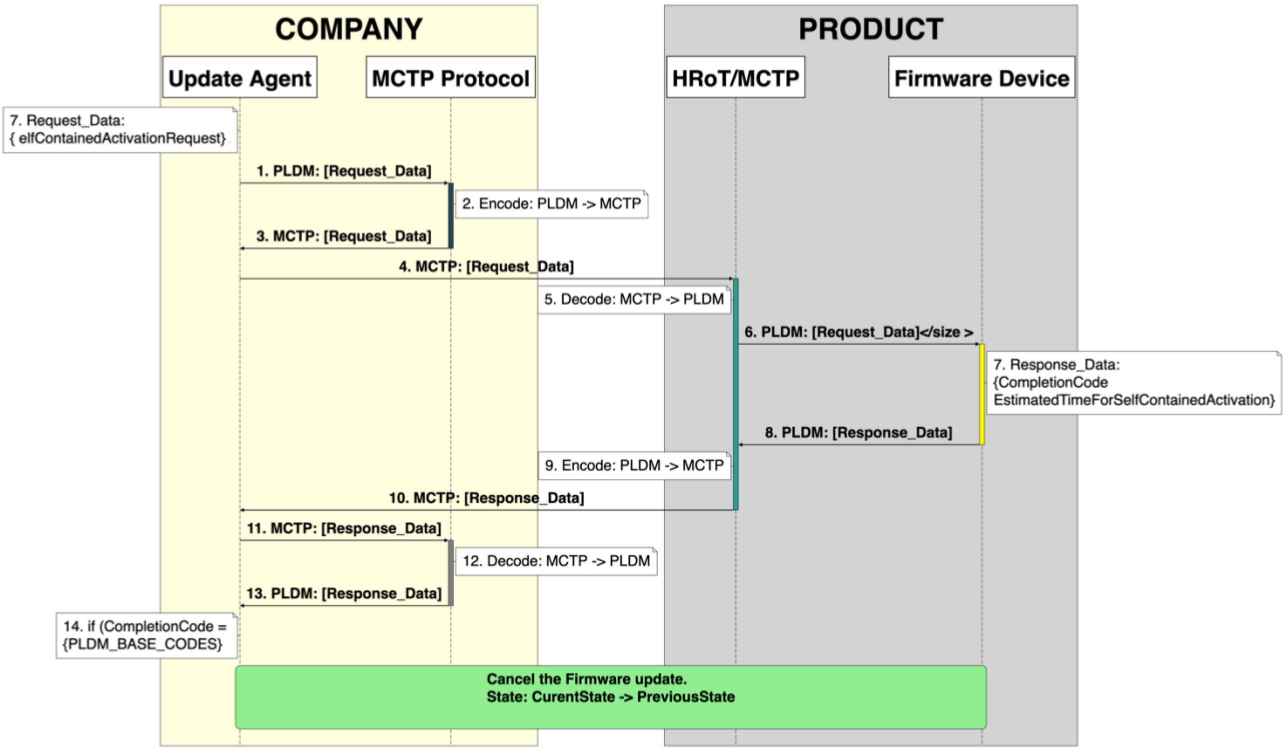


Figure 12: GetStatus Sequence Diagram

9. After verification is completed Cerberus will again wait for the device to apply the firmware image. Just like VerifyComplete a more robust mechanism of waiting for the ApplyComplete command is needed and the apply result that is apart of the command.

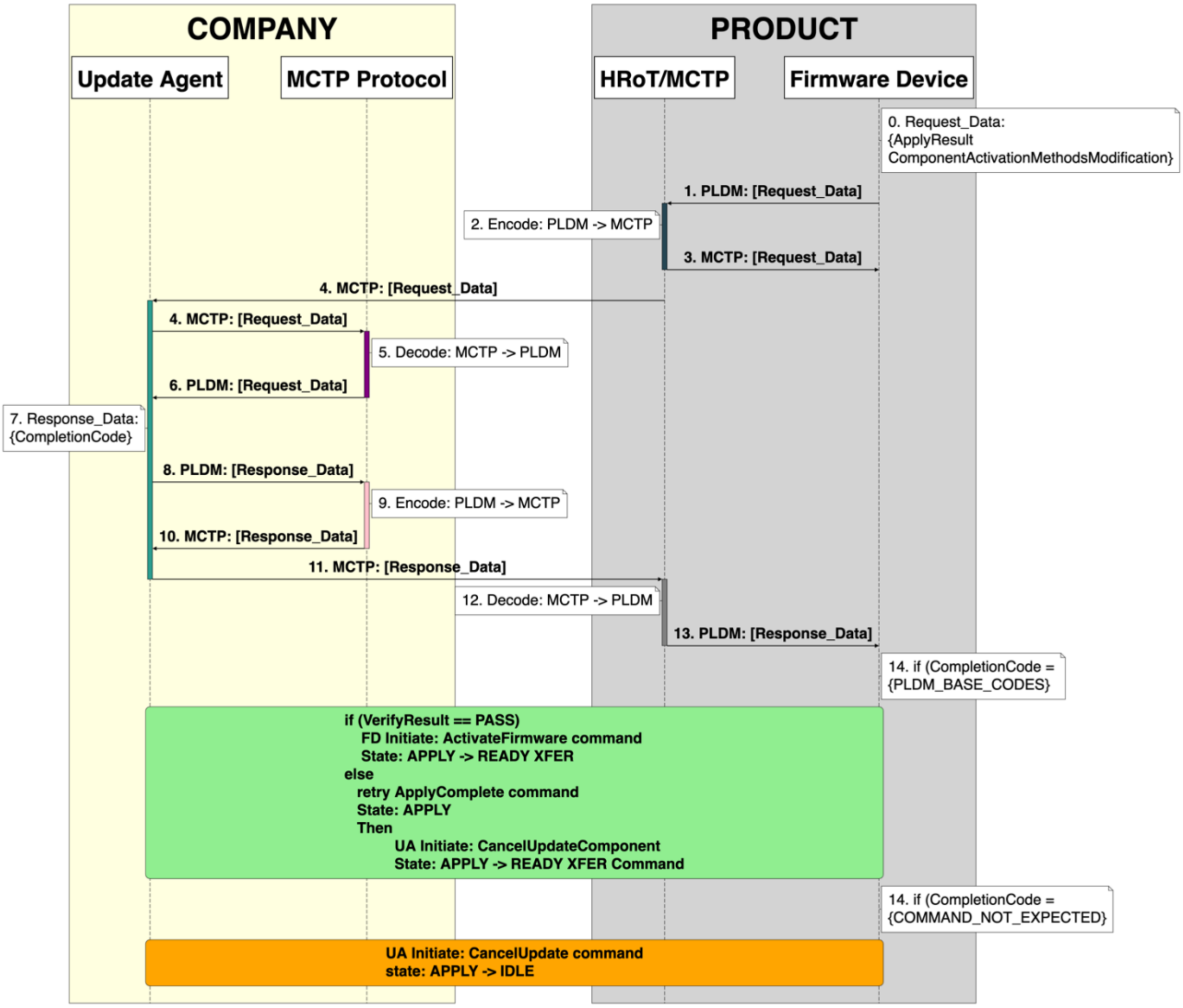

COMPANY
PRODUCT
Update Agent
MCTP Protocol
HRoT/MCTP
Firmware Device
0. Request_Data: {ApplyResult ComponentActivationMethodsModification}
1. PLDM: [Request_Data]
2. Encode: PLDM -> MCTP
3. MCTP: [Request_Data]
4. MCTP: [Request_Data]
4. MCTP: [Request_Data]
5. Decode: MCTP -> PLDM
6. PLDM: [Request_Data]
7. Response_Data: {CompletionCode}
8. PLDM: [Response_Data]
9. Encode: PLDM -> MCTP
10. MCTP: [Response_Data]
11. MCTP: [Response_Data]
12. Decode: MCTP -> PLDM
13. PLDM: [Response_Data]
14. if (CompletionCode = {PLDM_BASE_CODES}
if (VerifyResult == PASS)
FD Initiate: ActivateFirmware command
State: APPLY -> READY XFER
else
retry ApplyComplete command
State: APPLY
Then
UA Initiate: CancelUpdateComponent
State: APPLY -> READY XFER Command
14. if (CompletionCode = {COMMAND_NOT_EXPECTED}
UA Initiate: CancelUpdate command
state: APPLY -> IDLE

Figure 13: ApplyComplete Sequence Diagram

10. Steps 5 through 9 are repeated for every firmware component that Cerberus needs to update.

11. After all firmware images have been transferred, verified, and applied, Cerberus will issue the ActivateFirmware command. The command contains a boolean flag telling the device whether to activate any self-contained components specified during the UpdateComponent command. Setting this flag is left up to the user.

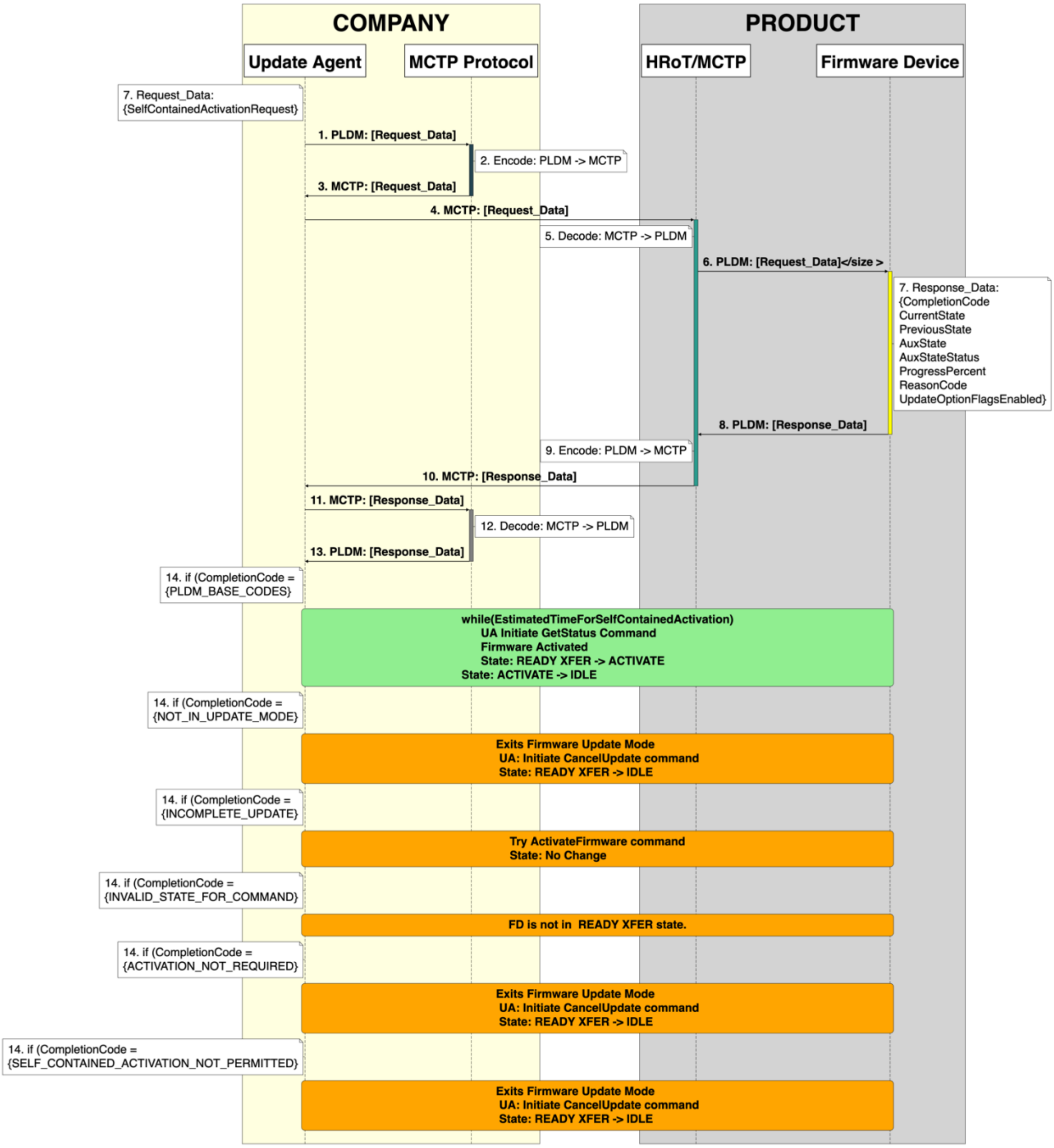

COMPANY
PRODUCT
Update Agent
MCTP Protocol
HRoT/MCTP
Firmware Device
7. Request_Data: {SelfContainedActivationRequest}
1. PLDM: [Request_Data]
2. Encode: PLDM -> MCTP
3. MCTP: [Request_Data]
4. MCTP: [Request_Data]
5. Decode: MCTP -> PLDM
6. PLDM: [Request_Data]</size >
7. Response_Data: {CompletionCode CurrentState PreviousState AuxState AuxStateStatus ProgressPercent ReasonCode UpdateOptionFlagsEnabled}
8. PLDM: [Response_Data]
9. Encode: PLDM -> MCTP
10. MCTP: [Response_Data]
11. MCTP: [Response_Data]
12. Decode: MCTP -> PLDM
13. PLDM: [Response_Data]
14. if (CompletionCode = {PLDM_BASE_CODES}
while(EstimatedTimeForSelfContainedActivation)
UA Initiate GetStatus Command
Firmware Activated
State: READY XFER -> ACTIVATE
State: ACTIVATE -> IDLE
14. if (CompletionCode = {NOT_IN_UPDATE_MODE}
Exits Firmware Update Mode
UA: Initiate CancelUpdate command
State: READY XFER -> IDLE
14. if (CompletionCode = {INCOMPLETE_UPDATE}
Try ActivateFirmware command
State: No Change
14. if (CompletionCode = {INVALID_STATE_FOR_COMMAND}
FD is not in READY XFER state.
14. if (CompletionCode = {ACTIVATION_NOT_REQUIRED}
Exits Firmware Update Mode
UA: Initiate CancelUpdate command
State: READY XFER -> IDLE
14. if (CompletionCode = {SELF_CONTAINED_ACTIVATION_NOT_PERMITTED}
Exits Firmware Update Mode
UA: Initiate CancelUpdate command
State: READY XFER -> IDLE

Figure 14: ActivateFirmware Sequence Diagram

3.2 Cerberus Operating as FD Flow

1. Cerberus will wait for the first command from the UA. If this command was one of the two inventory commands then Cerberus knows it needs to receive a second inventory command from the UA.
2. If the two inventory commands were issued or if none were, Cerberus will receive the RequestUpdate command storing its information in the manager. During the creation of the Request Update response, Cerberus will indicate if it will send the GetPackageData command based on if the UA specified there was package data present. Cerberus will also

respond with the length of the metadata it wishes the UA to retain. This metadata is left up to the user of this document if there choose to do so. Additionally, it is assumed that this metadata is stored on flash though this can also be freely changed to some structure or other volatile memory.

3. After the RequestUpdate command Cerberus will proceed to issue the GetPackageData command and respond to the GetDeviceMetaData commands if need be.
4. After receiving package data and transferring metadata, Cerberus will now wait to receive the component table from the UA via PassComponentTable commands. The component table is saved to the manager and contains information about each firmware component the UA wishes to update. Cerberus will respond with a compatibility code for each component. The implementation provides some checks for this compatibility code although further comparisons are needed by the user.
5. Cerberus now receives the UpdateComponent command indicating which component in the table should be updated next. Cerberus responds back with another compatibility code that requires additional checking from the user.
6. Cerberus will now issue the RequestFirmwareData command with the offset and length of the portion of the firmware image Cerberus is requesting. The offset and length is saved in the manager and upon receiving a response Cerberus will immediate write the portion of the image to flash memory.
7. Once component image has been transferred Cerberus will issue the TransferComplete command. The handler assumes a successful transfer with no additional checks being performed for the result field of TransferComplete, this is left up to the user.
8. Cerberus will now perform a verification of the received image. This verification mechanism is left up to the user to implement and subsequently the assignment of the result field in the VerifyComplete command (which is by default set to success). One example of verification is that Cerberus issues the GetMetaData command with the UA responding with hash codes or signatures of the firmware image. Cerberus would then take these and compare them to the hash codes or signatures it generated with the received component image.

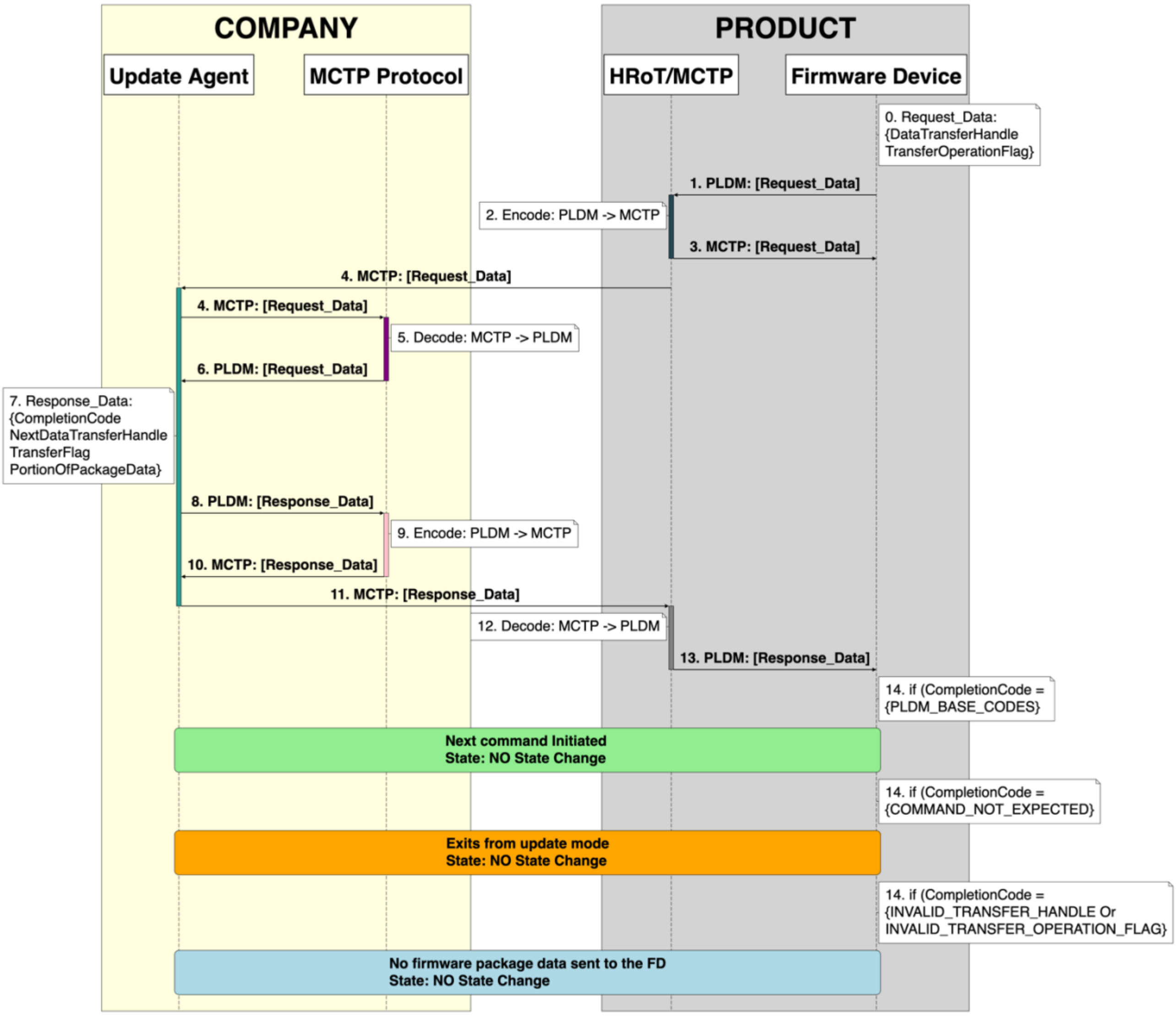

COMPANY
PRODUCT
Update Agent
MCTP Protocol
HRoT/MCTP
Firmware Device
0. Request_Data: {DataTransferHandle TransferOperationFlag}
1. PLDM: [Request_Data]
2. Encode: PLDM -> MCTP
3. MCTP: [Request_Data]
4. MCTP: [Request_Data]
4. MCTP: [Request_Data]
5. Decode: MCTP -> PLDM
6. PLDM: [Request_Data]
7. Response_Data: {CompletionCode NextDataTransferHandle TransferFlag PortionOfPackageData}
8. PLDM: [Response_Data]
9. Encode: PLDM -> MCTP
10. MCTP: [Response_Data]
11. MCTP: [Response_Data]
12. Decode: MCTP -> PLDM
13. PLDM: [Response_Data]
14. if (CompletionCode = {PLDM_BASE_CODES}
Next command Initiated
State: NO State Change
14. if (CompletionCode = {COMMAND_NOT_EXPECTED}
Exits from update mode
State: NO State Change
14. if (CompletionCode = {INVALID_TRANSFER_HANDLE Or INVALID_TRANSFER_OPERATION_FLAG}
No firmware package data sent to the FD
State: NO State Change

Figure 15: GetMetaData Sequence Diagram

9. After verification is successfully completed, Cerberus will now apply the firmware image. However, because in the current implementation the firmware image is immediately written to flash the apply stage become semi-obsolete. As such, Cerberus will simply send a successful ApplyComplete command to the UA. That said, the user could treat the flash region where the image is written to be temporary and that during the apply stage the image is transferred to another more permanent region of flash. Additionally, the assignment of the component update options modification field is left up to the user and the apply result field is default set to success.

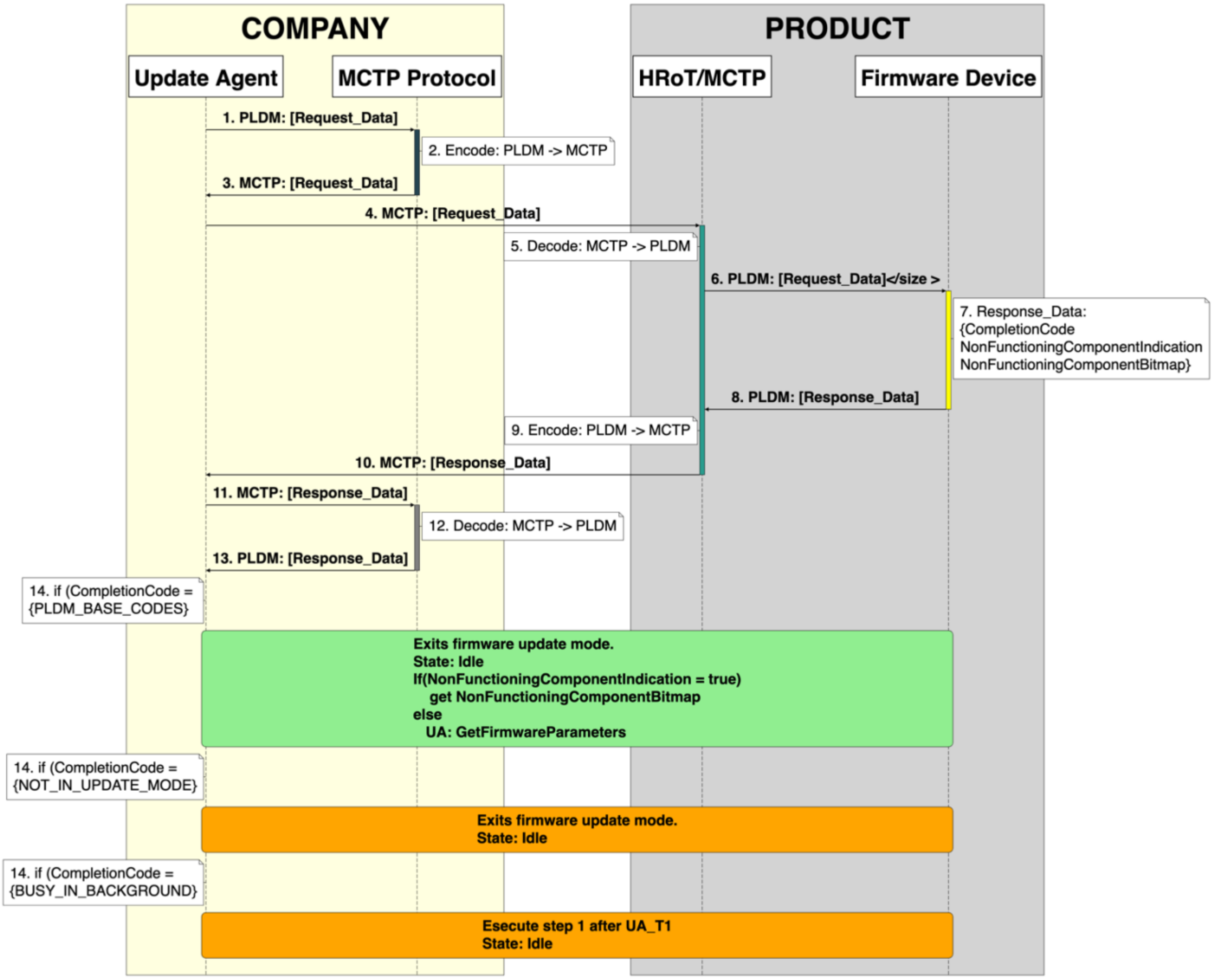


Figure 16: CancelUpdate Sequence Diagram

10. Steps 5 through 9 are repeated for every component that needs to be updated.

11. Finally, Cerberus expects to receive the ActivateFirmware command. How the firmware is activated is left up to the user of this document. The estimated time for activation field in Cerberus's response to ActivateFirmware can be configured in the `platform_config` on the projects folder.