MuleSoft®

API-led Integration Patterns

1 Summary

The purpose of this page is to describe the API-led Integration patterns that have been identified to ensure that best practices are followed in their actual implementation. Please note that these patterns are end-to-end orchestrations that may or may not combine both synchronous and asynchronous interactions and may or may not span application boundaries.

2 Assumptions

It is assumed that a CDM (Common Data Model) is the default message format used by the MuleSoft APIs during the description of these patterns.

3 Synchronous API

This is the simplest of all patterns where the entire pattern is fulfilled by a number of calls within the three MuleSoft layers (Experience, Process and System) and the pattern is completely stateless. The CDM message is sent through all the three layers in this case. The main tasks associated with this pattern are the following:

1. **Experience API:** acts as the entry point for the source system/API consumer and delegates the request to the Process API.
2. **Process API:** provides an abstraction layer only (in this case) by re-using the underlying System API to perform the business logic. This abstraction layer can be easily modified in the future if further orchestration of multiple System APIs is required (as described in section Synchronous API with Orchestration), without impacting the related Experience API.
3. **System API:** transforms the message to/from the target system format, returning a CDM message as a response.

This pattern is NOT recommended for longer-lived orchestrations. Longer-lived orchestrations are orchestrations where the invocation of multiple activities may require a time that is much larger than the time that the original consumer is willing to wait.

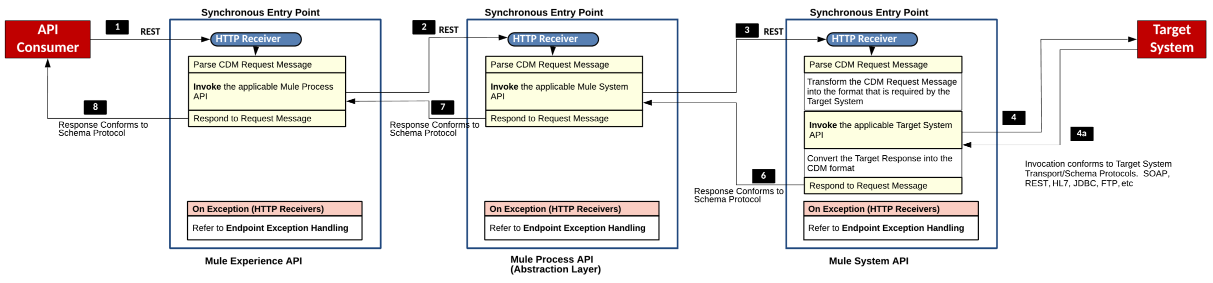

API Consumer
1
REST
Synchronous Entry Point
HTTP Receiver
Parse CDM Request Message
Invoke the applicable Mule Process API
Respond to Request Message
8
Response Conforms to Schema Protocol
On Exception (HTTP Receivers)
Refer to Endpoint Exception Handling
Mule Experience API
2
REST
Synchronous Entry Point
HTTP Receiver
Parse CDM Request Message
Invoke the applicable Mule System API
Respond to Request Message
7
Response Conforms to Schema Protocol
On Exception (HTTP Receivers)
Refer to Endpoint Exception Handling
Mule Process API (Abstraction Layer)
3
REST
Synchronous Entry Point
HTTP Receiver
Parse CDM Request Message
Transform the CDM Request Message into the format that is required by the Target System
Invoke the applicable Target System API
Convert the Target Response into the CDM format
Respond to Request Message
6
Response Conforms to Schema Protocol
On Exception (HTTP Receivers)
Refer to Endpoint Exception Handling
Mule System API
4
4a
Invocation conforms to Target System Transport/Schema Protocols. SOAP, REST, HL7, JDBC, FTP, etc
Target System

Figure 1 – Integration Pattern – Synchronous API

4 Synchronous API with Orchestration

This pattern is very similar to the pattern in the previous section Synchronous API with one main difference. The Mule Process API, in this case, is orchestrating the underlying business logic rather than providing a simple abstraction layer. The orchestration steps consist of calling first the System API 1, then the System API 2 and finally aggregating the API responses into a CDM message before returning it to the Experience API. It has to be noted that the System APIs, in this scenario, are invoked sequentially as the output from the System API 1 is required as part of the input for the System API 2. If sequencing is not required for the orchestration, then the System APIs can be invoked in parallel following the **Scatter-Gather** pattern.

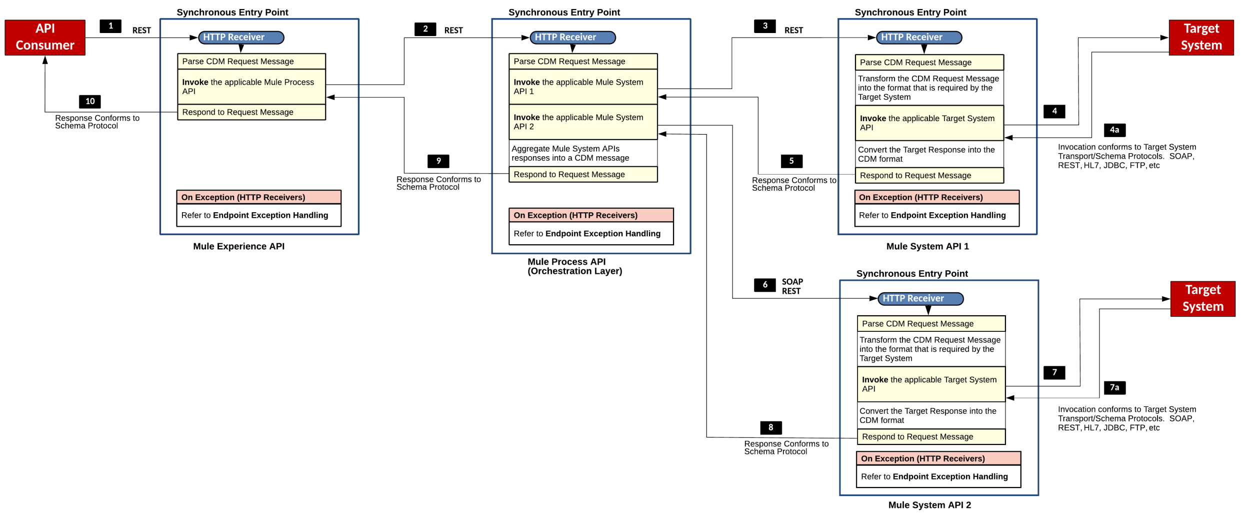

API Consumer
1
REST
10
Response Conforms to Schema Protocol
Synchronous Entry Point
HTTP Receiver
Parse CDM Request Message
Invoke the applicable Mule Process API
Respond to Request Message
On Exception (HTTP Receivers)
Refer to Endpoint Exception Handling
Mule Experience API
2
REST
9
Response Conforms to Schema Protocol
Synchronous Entry Point
HTTP Receiver
Parse CDM Request Message
Invoke the applicable Mule System API 1
Invoke the applicable Mule System API 2
Aggregate Mule System APIs responses into a CDM message
Respond to Request Message
On Exception (HTTP Receivers)
Refer to Endpoint Exception Handling
Mule Process API (Orchestration Layer)
3
REST
5
Response Conforms to Schema Protocol
Synchronous Entry Point
HTTP Receiver
Parse CDM Request Message
Transform the CDM Request Message into the format that is required by the Target System
Invoke the applicable Target System API
Convert the Target Response into the CDM format
Respond to Request Message
On Exception (HTTP Receivers)
Refer to Endpoint Exception Handling
Mule System API 1
4
4a
Invocation conforms to Target System Transport/Schema Protocols. SOAP, REST, HL7, JDBC, FTP, etc
Target System
6
SOAP
REST
8
Response Conforms to Schema Protocol
Synchronous Entry Point
HTTP Receiver
Parse CDM Request Message
Transform the CDM Request Message into the format that is required by the Target System
Invoke the applicable Target System API
Convert the Target Response into the CDM format
Respond to Request Message
On Exception (HTTP Receivers)
Refer to Endpoint Exception Handling
Mule System API 2
7
7a
Invocation conforms to Target System Transport/Schema Protocols. SOAP, REST, HL7, JDBC, FTP, etc
Target System

Figure 2 – Integration Pattern – Synchronous API with Orchestration

5 Synchronous API with Caching

This pattern is very similar to the pattern in section Synchronous API with one main difference. The Mule Process API, in this case, is caching the result of the call to the underlying System API for storing and re-using frequently called data. Using caching reduces the processing load on the Mule APIs/target systems and increases the speed of message processing. The caching mechanism is applied in the Process layer so that a fresh copy of the data can always be retrieved by the System layer. The cache refreshing can be either scheduled (i.e. a scheduler refreshes the cache every **x** seconds/minutes/hours) or be triggered when the TTL configured in the cache is expired. In MuleSoft, the Cache scope uses a Caching Strategy that stores data as **key-value pairs** in the Mule Object Store. The Object Store can be configured in its **in-memory** format, which means that a shutdown of the Mule application will result in a loss of the last cached entries, or in its **persistent** format, where the data is not lost and persisted up to 30 days. In general, the Cache scope compares the newly generated key to cached responses that it has previously processed and saves it in the Object Store:

1. If there is no cached response event (**cache miss**), the Cache scope processes the new message and produces a response. It also saves the resulting response in the Object Store.
2. If there is a cached response event (**cache hit**), the Caching Strategy generates a response that combines data from both the new request and the cached response

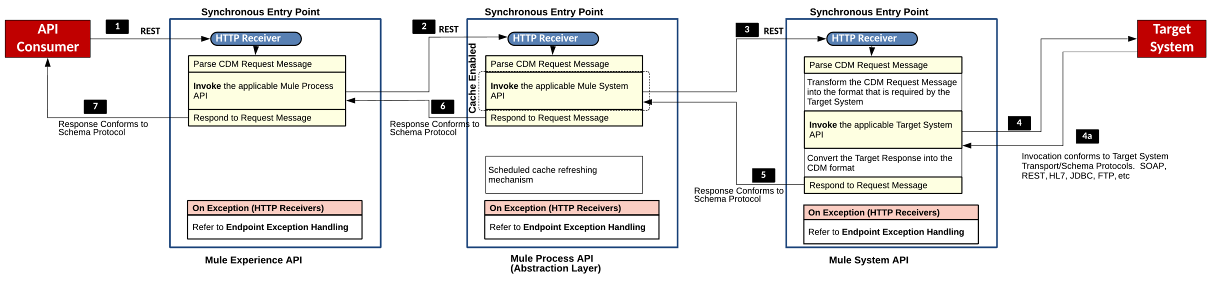

API Consumer
1
REST
Synchronous Entry Point
HTTP Receiver
Parse CDM Request Message
Invoke the applicable Mule Process API
Respond to Request Message
7
Response Conforms to Schema Protocol
On Exception (HTTP Receivers)
Refer to Endpoint Exception Handling
Mule Experience API
2
REST
Synchronous Entry Point
Cache Enabled
HTTP Receiver
Parse CDM Request Message
Invoke the applicable Mule System API
Respond to Request Message
6
Response Conforms to Schema Protocol
Scheduled cache refreshing mechanism
On Exception (HTTP Receivers)
Refer to Endpoint Exception Handling
Mule Process API (Abstraction Layer)
3
REST
Synchronous Entry Point
HTTP Receiver
Parse CDM Request Message
Transform the CDM Request Message into the format that is required by the Target System
Invoke the applicable Target System API
Convert the Target Response into the CDM format
Respond to Request Message
5
Response Conforms to Schema Protocol
On Exception (HTTP Receivers)
Refer to Endpoint Exception Handling
Mule System API
4
4a
Invocation conforms to Target System Transport/Schema Protocols. SOAP, REST, HL7, JDBC, FTP, etc
Target System

Figure 3 – Integration Pattern – Synchronous API with Caching

6 Synchronous API for non-CDM Consumers

This pattern is very similar to the pattern in section Synchronous API with one main difference. In this case, the request message sent from the Experience API's consumer is not in CDM format. Upon receiving the request, the Experience API transforms the message into the CDM format and invokes the applicable Mule Process API. The CDM response message received from the Process API is then converted back by the Experience API to the applicable message format expected by the consumer.

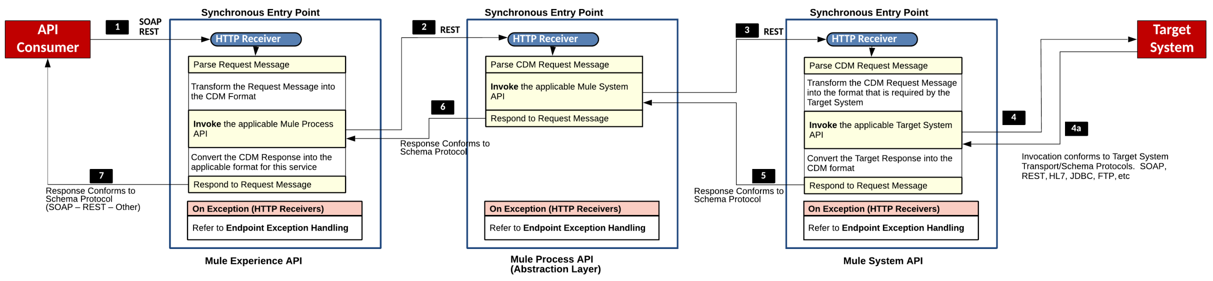

API Consumer
1
SOAP
REST
Synchronous Entry Point
HTTP Receiver
Parse Request Message
Transform the Request Message into the CDM Format
Invoke the applicable Mule Process API
Convert the CDM Response into the applicable format for this service
Respond to Request Message
7
Response Conforms to Schema Protocol (SOAP – REST – Other)
On Exception (HTTP Receivers)
Refer to **Endpoint Exception Handling**
Mule Experience API
2
REST
Synchronous Entry Point
HTTP Receiver
Parse CDM Request Message
Invoke the applicable Mule System API
Respond to Request Message
6
Response Conforms to Schema Protocol
On Exception (HTTP Receivers)
Refer to **Endpoint Exception Handling**
Mule Process API (Abstraction Layer)
3
REST
Synchronous Entry Point
HTTP Receiver
Parse CDM Request Message
Transform the CDM Request Message into the format that is required by the Target System
Invoke the applicable Target System API
Convert the Target Response into the CDM format
Respond to Request Message
5
Response Conforms to Schema Protocol
On Exception (HTTP Receivers)
Refer to **Endpoint Exception Handling**
Mule System API
4
4a
Target System
Invocation conforms to Target System Transport/Schema Protocols. SOAP, REST, HL7, JDBC, FTP, etc

Figure 4 – Integration Pattern – Synchronous API for non-CDM Consumers

7 Synchronous Fire and Forget (Async Push)

This pattern can be used when the CDM message that is originating from the source is intended for multiple target systems and source system does not need to wait for a response (e.g., information related to created, updated or deleted entities) from target systems. The figure below highlights the various runtime blocks that participate in the fulfilment of the overall pattern.

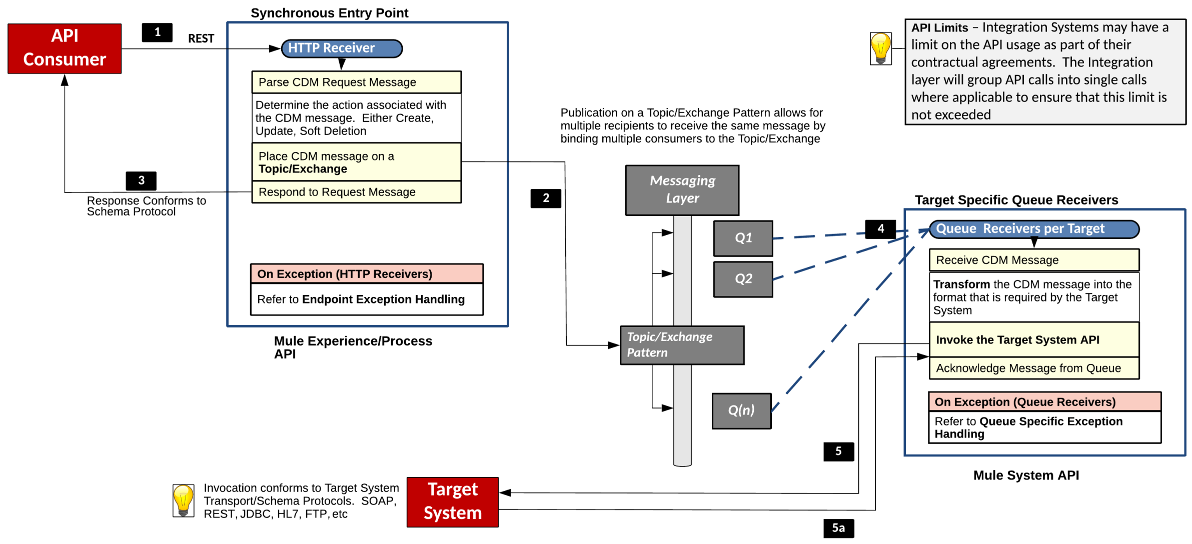

API Consumer
1
REST
Synchronous Entry Point
HTTP Receiver
Parse CDM Request Message
Determine the action associated with the CDM message. Either Create, Update, Soft Deletion
Place CDM message on a Topic/Exchange
Respond to Request Message
3
Response Conforms to Schema Protocol
On Exception (HTTP Receivers)
Refer to Endpoint Exception Handling
Mule Experience/Process API
2
Publication on a Topic/Exchange Pattern allows for multiple recipients to receive the same message by binding multiple consumers to the Topic/Exchange
Messaging Layer
Q1
Q2
Topic/Exchange Pattern
Q(n)
API Limits – Integration Systems may have a limit on the API usage as part of their contractual agreements. The Integration layer will group API calls into single calls where applicable to ensure that this limit is not exceeded
Target Specific Queue Receivers
4
Queue Receivers per Target
Receive CDM Message
Transform the CDM message into the format that is required by the Target System
Invoke the Target System API
Acknowledge Message from Queue
On Exception (Queue Receivers)
Refer to Queue Specific Exception Handling
Mule System API
5
5a
Invocation conforms to Target System Transport/Schema Protocols. SOAP, REST, JDBC, HL7, FTP, etc
Target System

Figure 5 – Integration Pattern – Synchronous Fire and Forget (Async Push)

The notes below provide additional detail for each of the runtime blocks in the figure above. Some key points in relation to each of the runtime blocks have been highlighted which must be adhered to as part of the implementation.

7.1 Synchronous Entry Point (Mule Experience/Process API)

The steps associated with this block are as below:

1. The integration layer exposes a synchronous Mule Experience/Process API that is responsible for the reception of the CDM messages from the source system. The Mule API will determine the action associated with the object (Create, Update, Soft Deletion).
2. It is the responsibility of the source system that the **sequencing** associated with the incoming CDM messages is honoured. It will be the responsibility of the integration layer to ensure that the sequence honouring as part of the asynchronous processing is handled.
3. The CDM message is then placed on a **Topic/Exchange pattern** within the Messaging Layer. The configuration of the topic/exchange pattern will allow the CDM message to be received by multiple consumers/queues subscribed/bound to the topic/exchange. Queue processing will ensure **Guaranteed Delivery**.
4. At this point in time, the responsibilities of this application boundary are finalised and the target queue receivers will then commence their processing.

5. The exception handling associated with this block is related to the HTTP exceptions and is further detailed in the section Endpoint Exception Handling.

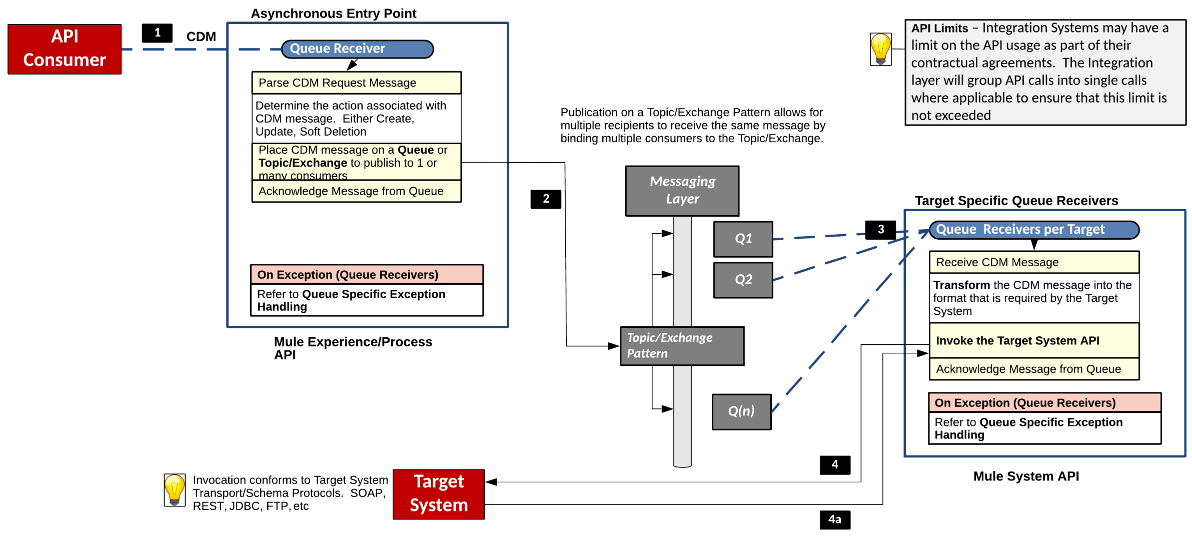
7.2 Target Specific Queue Receivers (Mule System API)

The steps associated with this block are as below:

1. There will be a queue receiver associated **for each target system**. It is likely that these will be separated into their **own application boundaries (Mule System API)**. On the reception on the queue of the CDM message, the CDM format will be transformed to the target format and the applicable target API (**over whatever supported protocol and security requirements**) will be invoked. There will be a single queue per target to handle all operations (create, update, etc.) to ensure that **sequencing** is honoured.
2. In the event of **creating invocations**, the API invocation will likely return the ID associated with the object that was created in the target system. If the source system requires the external IDs to be stored in the source system as well, then a correlation to the source system via the integration layer will be required. The correlation can occur by creating an **ID correlation CDM** message and place it on a queue that is intended for the correlation processing.
3. The exception handling associated with this block is related to Queue Receiver exceptions and it is further detailed in the section Queue Specific Exception Handling. Queue Specific Exception Handling will also cater for **Guaranteed Delivery**.

8 Asynchronous Publisher/Subscriber

This pattern is very similar to the pattern in the section Synchronous Fire and Forget (Async Push) with the main difference that the Mule Experience/Process API receives a CDM message on an Asynchronous Queue endpoint rather than HTTP endpoint.


API Consumer
1
CDM
Asynchronous Entry Point
Queue Receiver
Parse CDM Request Message
Determine the action associated with CDM message. Either Create, Update, Soft Deletion
Place CDM message on a Queue or Topic/Exchange to publish to 1 or many consumers
Acknowledge Message from Queue
On Exception (Queue Receivers)
Refer to Queue Specific Exception Handling
Mule Experience/Process API
2
Messaging Layer
Q1
Q2
Topic/Exchange Pattern
Q(n)
3
Target Specific Queue Receivers
Queue Receivers per Target
Receive CDM Message
Transform the CDM message into the format that is required by the Target System
Invoke the Target System API
Acknowledge Message from Queue
On Exception (Queue Receivers)
Refer to Queue Specific Exception Handling
Mule System API
4
4a
Target System

Publication on a Topic/Exchange Pattern allows for multiple recipients to receive the same message by binding multiple consumers to the Topic/Exchange.

API Limits – Integration Systems may have a limit on the API usage as part of their contractual agreements. The Integration layer will group API calls into single calls where applicable to ensure that this limit is not exceeded

Invocation conforms to Target System Transport/Schema Protocols. SOAP, REST, JDBC, FTP, etc

Figure 6 – Integration Pattern – Asynchronous Publisher/Subscriber

9 Asynchronous Poll

This pattern is used when the retrieved data that is originating from the source is intended for multiple target systems. It is a typical single source to multiple target systems synchronisation pattern. The figure below highlights the various runtime blocks that participate in the fulfilment of the overall pattern. It is important to note that the scheduled entry point must maintain the state of retrievals.

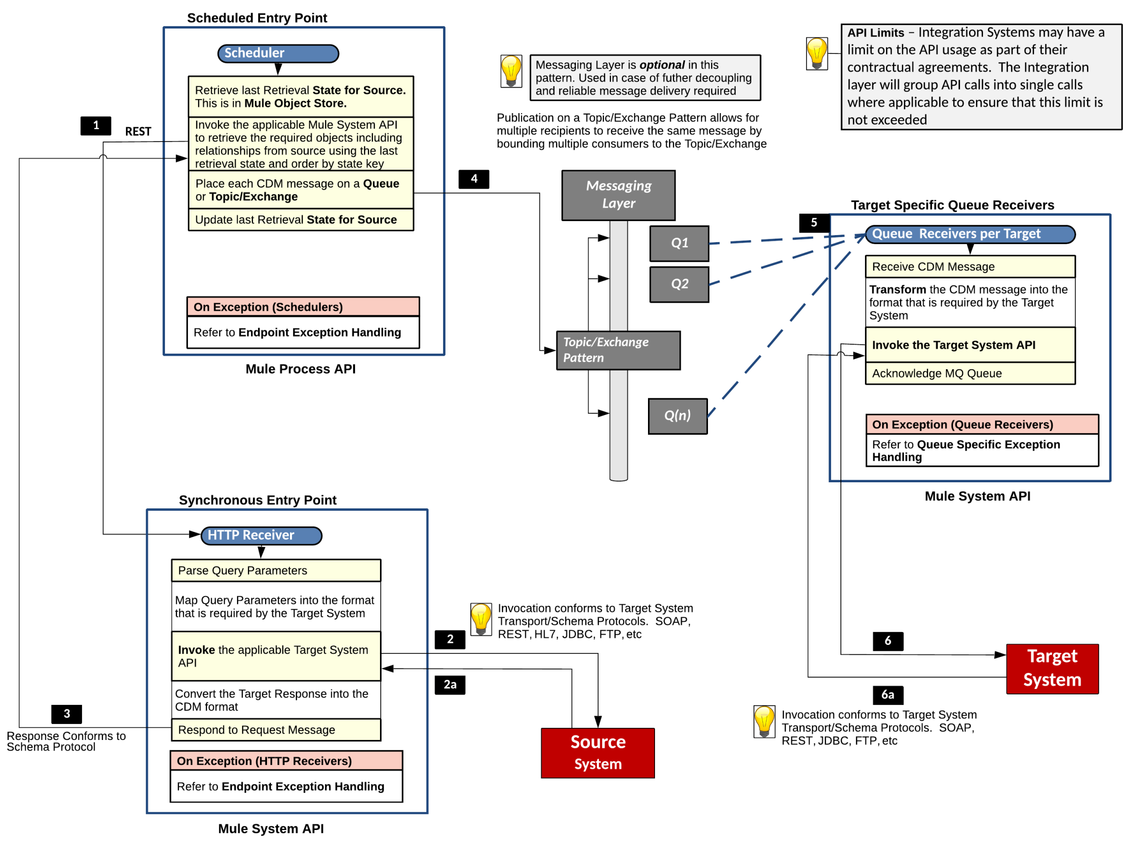

Scheduled Entry Point
Scheduler
Retrieve last Retrieval State for Source. This is in Mule Object Store.
Invoke the applicable Mule System API to retrieve the required objects including relationships from source using the last retrieval state and order by state key
Place each CDM message on a Queue or Topic/Exchange
Update last Retrieval State for Source
On Exception (Schedulers)
Refer to Endpoint Exception Handling
Mule Process API
1
REST
Messaging Layer is optional in this pattern. Used in case of futher decoupling and reliable message delivery required
Publication on a Topic/Exchange Pattern allows for multiple recipients to receive the same message by bounding multiple consumers to the Topic/Exchange
API Limits – Integration Systems may have a limit on the API usage as part of their contractual agreements. The Integration layer will group API calls into single calls where applicable to ensure that this limit is not exceeded
4
Messaging Layer
Q1
Q2
Topic/Exchange Pattern
Q(n)
Target Specific Queue Receivers
5
Queue Receivers per Target
Receive CDM Message
Transform the CDM message into the format that is required by the Target System
Invoke the Target System API
Acknowledge MQ Queue
On Exception (Queue Receivers)
Refer to Queue Specific Exception Handling
Mule System API
Synchronous Entry Point
HTTP Receiver
Parse Query Parameters
Map Query Parameters into the format that is required by the Target System
Invoke the applicable Target System API
Convert the Target Response into the CDM format
Respond to Request Message
On Exception (HTTP Receivers)
Refer to Endpoint Exception Handling
Mule System API
3
Response Conforms to Schema Protocol
2
2a
Invocation conforms to Target System Transport/Schema Protocols. SOAP, REST, HL7, JDBC, FTP, etc
Source System
6
6a
Target System
Invocation conforms to Target System Transport/Schema Protocols. SOAP, REST, JDBC, FTP, etc

Figure 7 – Integration Pattern – Asynchronous Poll

The notes below provide additional detail for each of the runtime blocks in the figure above. Some key points in relation to each of the runtime blocks have been highlighted which must be adhered to as part of the implementation.

9.1 Scheduled Entry Point (Mule Process API)

The steps associated with this block are as below:

1. The Mule Process API will periodically schedule **(single threaded)** the retrieval of all the changed objects on the source system side by invoking the applicable Mule System API. The retrieval process will honour the sequencing based on certain attributes such as the

last **updated timestamp** or last **ID** of the object of interest. The retrieval process will also maintain the last timestamp/ID associated with the newest record that has been retrieved as part of the retrieval process. The maintenance of the state of the retrieval will require a persistent layer. The Mule Object Store module will be leveraged to maintain this state.
2. The retrieved objects and their relationships along with their derived action (Create, Update, etc.) returned in CDM format are then placed to a **Topic/Exchange Pattern**. Queue processing ensures **Guaranteed Delivery.**
3. In the event that the change in an object is at a related record level, i.e. the attributes of the parent have not changed but because the retrieval is at the parent level, its attributes are retrieved, the CDM format must have the ability to highlight the action associated with the record. The delta identification will then allow the target queue receivers to ensure that the correct API is called at the target level.
4. The state of the retrieval will be updated to highlight the last retrieval record. This will ensure that subsequent retrievals do not retrieve duplicate records. At this point in time, the responsibilities of this application boundary are finalised and the target queue receivers will then commence their processing.
5. The exception handling associated with this block is related to schedule exception handling and is further detailed in the section Endpoint Exception Handling.

9.2 Target Specific Queue Receivers

As per the steps outlined in section Synchronous Fire and Forget (Async Push).

10 Asynchronous Batch Processing

This pattern is used when the retrieved data that is originating from the source needs to be processed by a number of ETL activities before being sent to the target system. This may involve a full data sync or be related to delta changes only. The figure below highlights the various runtime blocks that participate in the fulfilment of the overall pattern. It is important to note that, whilst Mule has ETL capabilities, it is not an ETL tool as complex joining, aggregations, pivoting, etc. are not supported nor provided like a traditional ETL platform.

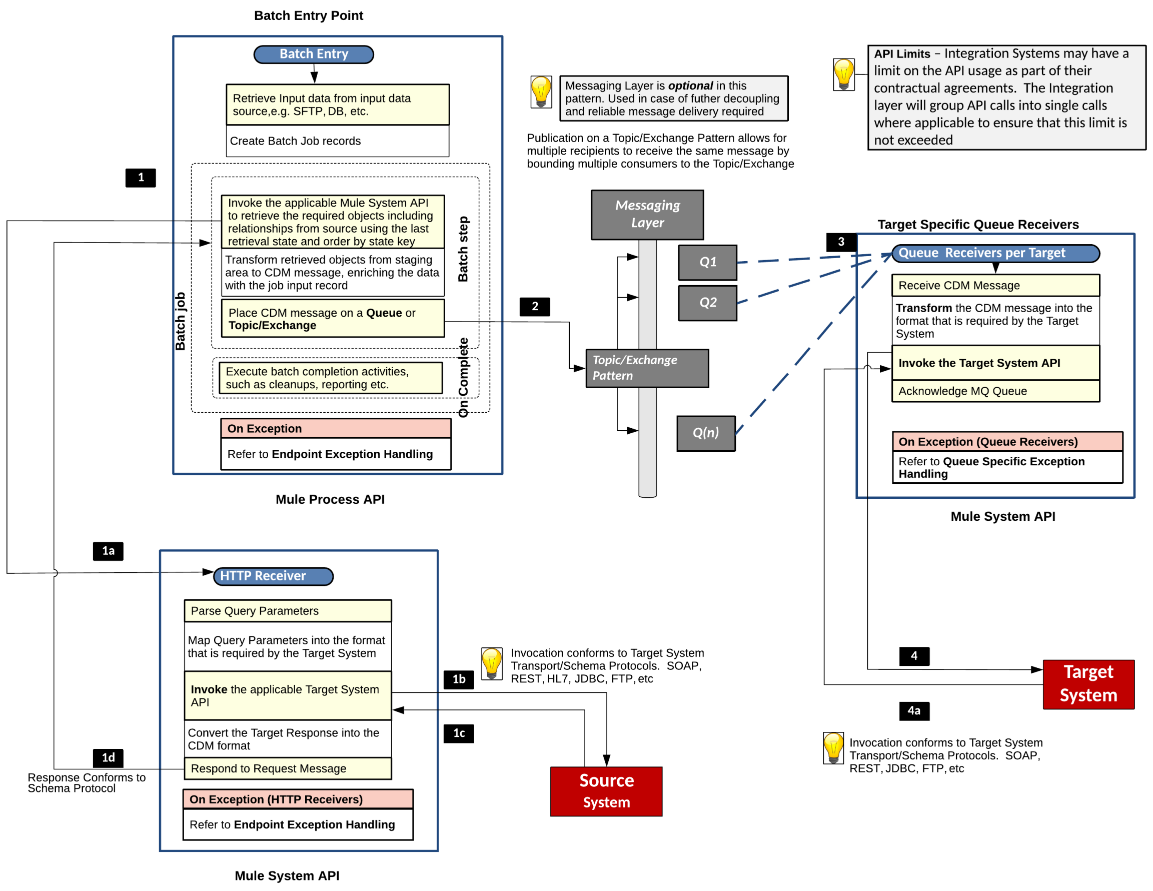

Batch Entry Point
Batch Entry
Retrieve Input data from input data source,e.g. SFTP, DB, etc.
Create Batch Job records
1
Invoke the applicable Mule System API to retrieve the required objects including relationships from source using the last retrieval state and order by state key
Transform retrieved objects from staging area to CDM message, enriching the data with the job input record
Place CDM message on a Queue or Topic/Exchange
Batch step
Batch job
Execute batch completion activities, such as cleanups, reporting etc.
On Complete
On Exception
Refer to Endpoint Exception Handling
Mule Process API
Messaging Layer is optional in this pattern. Used in case of futher decoupling and reliable message delivery required
Publication on a Topic/Exchange Pattern allows for multiple recipients to receive the same message by bounding multiple consumers to the Topic/Exchange
Messaging Layer
Q1
Q2
2
Topic/Exchange Pattern
Q(n)
API Limits – Integration Systems may have a limit on the API usage as part of their contractual agreements. The Integration layer will group API calls into single calls where applicable to ensure that this limit is not exceeded
Target Specific Queue Receivers
3
Queue Receivers per Target
Receive CDM Message
Transform the CDM message into the format that is required by the Target System
Invoke the Target System API
Acknowledge MQ Queue
On Exception (Queue Receivers)
Refer to Queue Specific Exception Handling
Mule System API
1a
HTTP Receiver
Parse Query Parameters
Map Query Parameters into the format that is required by the Target System
Invoke the applicable Target System API
Convert the Target Response into the CDM format
Respond to Request Message
On Exception (HTTP Receivers)
Refer to Endpoint Exception Handling
Mule System API
1d
Response Conforms to Schema Protocol
1b
1c
Invocation conforms to Target System Transport/Schema Protocols. SOAP, REST, HL7, JDBC, FTP, etc
Source System
4
4a
Target System
Invocation conforms to Target System Transport/Schema Protocols. SOAP, REST, HL7, JDBC, FTP, etc

Figure 8 – Integration Pattern – Asynchronous Batch Processing

The notes below provide additional detail for each of the runtime blocks in the figure above. Some key points in relation to each of the runtime blocks have been highlighted which must be adhered to as part of the implementation.

10.1 Batch Entry Point (Mule Process API)

The steps associated with this block are as below:

1. The Mule Process API will periodically run the batch either triggered by a **scheduler** with predefined interval or triggered by a **polling inbound component** after retrieving the data e.g. SFTP. In case of SFTP, the CSV file (or a file in a different format) from an SFTP server is retrieved and moved to a **temporary** working directory. The CSV records are parsed and

sorted by ascending order using the **ID** of the object of interest. The batch triggered by the scheduler retrieves the data from the input data source and creates the record set to be submitted to the batch job.

2. A **Batch Job** kicks in to process the retrieved records from the Staging Area. In Mule, a Batch Job consists of a number of **Batch Steps**, where each record is processed separately and moved across the steps in an asynchronous and paralleled fashion, and an **On Complete** phase which executes after all the Batch Steps are completed. In the Batch Step, the details are retrieved by invoking the System API, each retrieved record is **enriched** with the related input record, **transformed** to a CDM message and then placed to a **Topic/Exchange Pattern**. Queue processing ensures **Guaranteed Delivery.**
3. After all the records are processed by the Batch Step, the teardown/cleanup activities are performed e.g. in case of SFTP inbound, the CSV file is moved to a **processed** directory during the On Complete phase. At this point in time, the responsibilities of this application boundary are finalised and the target queue receivers will then commence their processing for the **transform** and **load** activities.
4. The exception handling associated with SFTP inbound is further detailed in the section Endpoint Exception Handling.

10.2 Target Specific Queue Receivers

As per the steps outlined in section Synchronous Fire and Forget (Async Push).

11 Endpoint Exception Handling

The endpoints that are used in the MuleSoft applications have in some cases a specific requirement for exception handling and these have been highlighted below (queue endpoints are specific and described in their own section following this one).

HTTP Endpoints

- Ensure the correct response that conforms to either SOAP, XML or JSON schema is returned.
- Ensure the correct HTTP status codes and reasons are returned in the response.

Scheduler/Poller Endpoints

- Ensure the **last retrieval** state is **not** updated in a state-full scheduled scenario.
- In the case of Poller with ObjectStore usage (e.g. Batch entry point), log the error.

SFTP Endpoints

- Ensure that the file is not deleted or moved to a ‘processed’ directory but rather to an ‘error’ directory.

12 Queue Specific Exception Handling

In dealing with queue patterns, one of the primary objectives is **guaranteed message delivery**. From a queue message producer side, this guaranteed delivery requirement is met by ensuring that the messages that are placed on the queues are persisted. This is usually provided, out of the box, by the Messaging Layer.

The next requirement of guaranteed delivery is to ensure that the consumer (receiver) side is implemented to ensure that there is no message loss. There are two sub-requirements that need to be adhered to, to ensure that the above requirement is satisfied. These include:

1. The acknowledgement of the queue message, i.e. the point at which it is removed from the persistent layer, must be carried out at the end of the execution thread. This is generally achieved via the configuration of the **acknowledgement mode**.
2. The exception handling of the queue receivers must have the ability to differentiate between **business and technical exceptions** and react accordingly. The queue receiver processing has the capability to configure **redeliveries** in the event of an exception. A **technical** exception is defined as an exception where the target system is not available or there is a network glitch where it makes sense to attempt redeliveries. A **business** exception is data related and each redelivery will likely result in the same exception, so it does not make sense to attempt redeliveries. Based on these descriptions the following steps will be carried out in the exception handling routines:
    - In the event of a **technical exception and if redeliveries are NOT exhausted**:
        - Log Error
        - Re-try the transaction
    - In the event of a **technical exception and if redeliveries are exhausted**:
        - Log Error
        - Place the original message in a **DLQ**
        - Acknowledge the original message
    - In the event of a **business exception** (schema validation or other):
        - Log Error
        - Place the original message in a **Poison** queue
        - Acknowledge the original message