**Industries Order Management Implementation Best Practices**

**Introducing Industries Order Management**

Order management (OM) is a system that brokers, or exchanges, data between the front office customer request and the back office fulfillment systems to deliver accurate products and services to the customer.

**Industries Order Management, often referred to as Order Management or IOM, is the Salesforce Industries (SFI) solution for OM.**

IOM’s job is to ensure that all fulfillment systems receive the data that they need, and in the correct sequence, in order to successfully deliver telecommunications services.

IOM delivers best-in-class, next-generation OM capabilities to communications, media, and energy companies.

IOM is based on a framework that provides a seamless user experience across all OM-related modeling and runtime functions and uses a single centralized catalog with supporting tools to allow users to collaborate in real time.

Order fulfillment via IOM starts with order decomposition. This process creates a set of purpose-built technical requests and maps the commercial order to them. Fulfillment of the decomposed order is then orchestrated across relevant systems and parties.

As IOM and Industries CPQ are on the same platform, customer service representatives (CSRs), fulfillment operators, and fulfillment managers have full visibility across all stages of the order. In-flight order revisions allow for making changes to an order while the fulfillment is in progress. MACD (Move, Add, Change, or Delete) allows for changes to be made to an existing active service (Asset). IOM and Industries CPQ are seamlessly integrated to allow for important functions, such as in-flight amends, to be used out-of-the-box.

IOM provides a wide array of both next-generation and traditional order management capabilities on this architecture, including:

* Support for Commercial IOM
* Commercial order decomposition
* Order orchestration based on dynamically generated workflows
* Integration with Fulfillment Systems
* Automated, manual, and fallout management capabilities
* Order handling and lifecycle management
* Visualization of fulfillment flows by CSRs, fulfillment operators, and other users without having to swivel-chair outside of the Salesforce Platform.

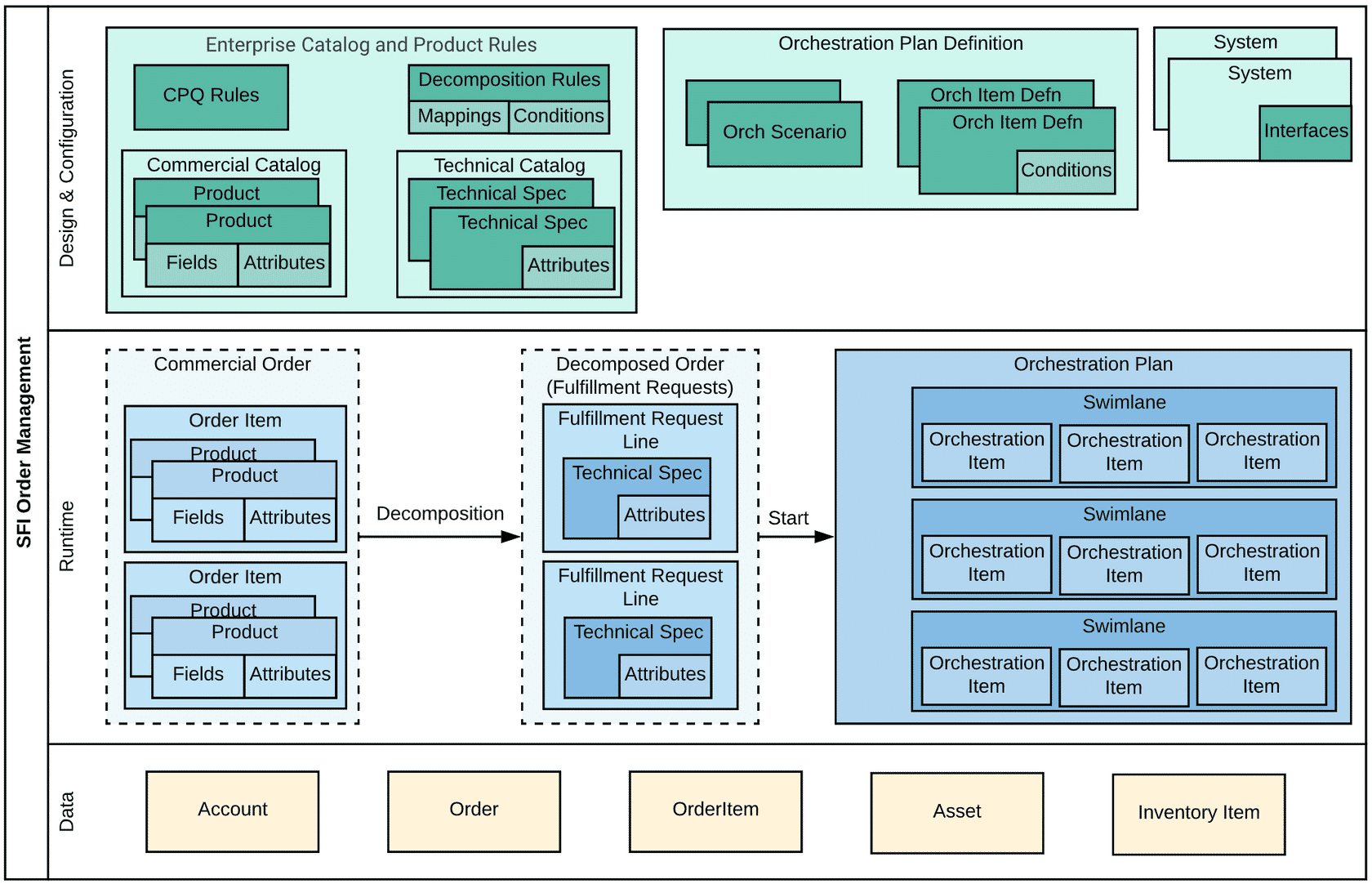
**Order Management Overview**

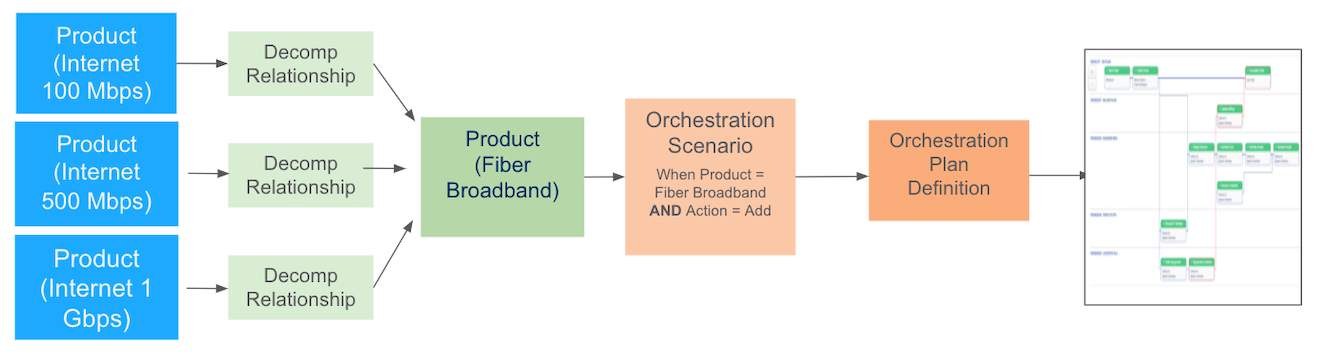
There are three main perspectives for the Industries Order Management:

**1. Design and Configuration:**IOM is a next generation Order Management system that is Catalog Driven. Naturally, a big part of the design and configuration of the IOM is performed via EPC and the Shared Catalog. This starts the Commercial Catalog which configures the Products and Services that are referenced by Order Items. It also includes the Technical Catalog which configures the CFS/RFS layers. In addition to the Product configuration, IOM allows the configuration of the Decomposition Rules/Relationships which drive the decomposition process. Orders are built in CPQ and they must be valid therefore they must conform with the CPQ Rules. Additionally, IOM provides facilities for configuring the orchestration and integration components.

**2. Runtime:** Users create Orders and submit them for decomposition and Orchestration. Upon order submit, IOM uses the configuration to perform the order decomposition and generates a set of Fulfillment Requests that contain Fulfillment Request Lines. The latter is the technical counterparts of the Order Item which are commercial in nature. Once the order is decomposed, IOM generates a bespoke orchestration plan which is specific to the order requests. The Orchestration Plan contains Orchestration Items organized in swimlanes and linked with dependencies according to the configuration.

**3. Data:**the data perspective in the diagram illustrates a number of Salesforce Objects (standard or custom objects provided by IOM) that are created/read and updated during the process.



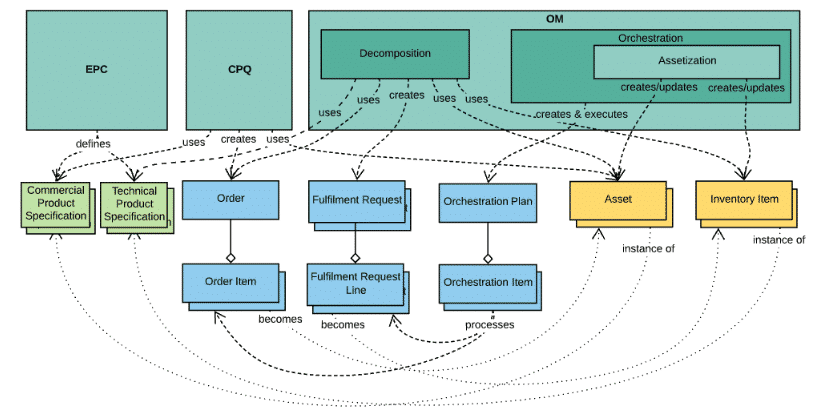


**Order Management Data Lifecycle**

**The IOM Data Lifecycle diagram shows the relationship between SFI components and the entities (that is, Objects) they manage.**

SFI Enterprise Product Catalog (EPC) is used to define the products, services, and offers that form the Commercial Catalog as well as the products and services that form the Technical Catalog. Using IOM, fulfillment designers can define the relationships between commercial and technical products. These relationships ultimately drive the decomposition process.

SFI CPQ is used to drive the sales process based on the product definitions and rules configured in EPC. At the end of the sales process, CPQ will create, validate, and submit an Order containing one or more Order Items to IOM. IOM decomposes the Order into a set of Fulfillment Requests (FRs). Each FR contains one or more Fulfillment Request Lines (FRLs). There are similarities between the FRL and Order Item, and the FRL can be considered a technical product.



Once the order has been decomposed, IOM will dynamically generate an Orchestration Plan based on predefined design time configurations. A Plan typically contains multiple Orchestration Items organized into multiple swimlanes according to the IOM design time specifications. A special Orchestration Item is used for the process of assetization.

**Order Management System**

The goal of IOM is to revolutionize the order management space by providing the first truly enterprise catalog-driven IOM application that:

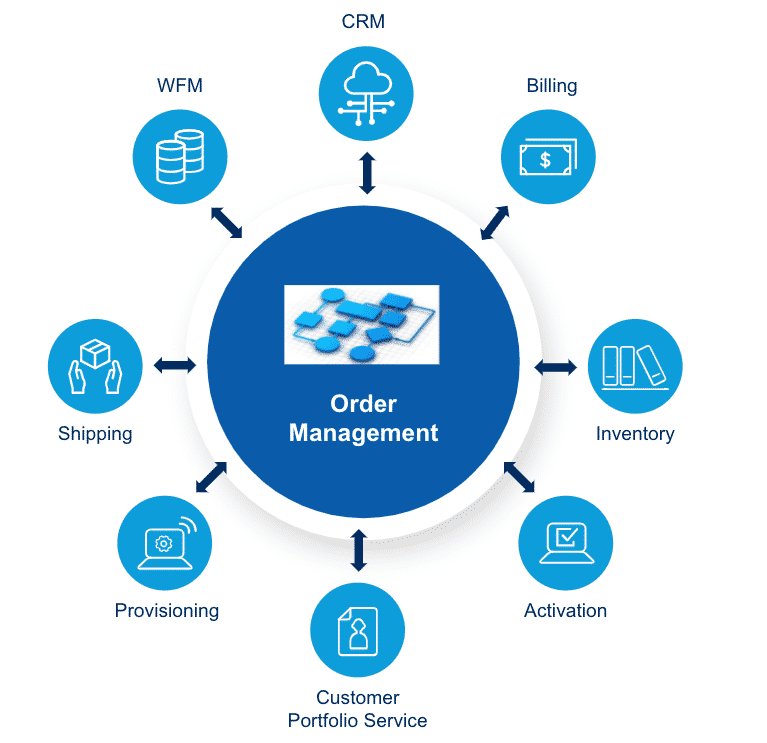
* Integrates into your end-to-end ecosystem using a catalog-driven approach for Sales, CPQ, Ordering, and Fulfillment processes.
* Provides a user interface (UI) supported by a set of SFI OmniStudio platform tools that allow your teams to collaboratively and rapidly create, manage, and deliver offerings to market.
* Allows for an evolutionary approach to transform both front-office and back-office systems to become catalog-driven over time. This supports the elimination of data silos and the mastering of commercial configurations into a single model.

SFI presents a unified solution that addresses both service provider and subscriber needs on an integrated omnichannel communications platform that allows design, order capture, order management, and customer management scenarios to work cohesively. This brings agility to service providers in rapidly delivering new products, and it offers customers a seamless experience, from purchase through to delivery of the services, on any channel or device.

**Next Generation Order Management**

Next-generation enterprise catalog-driven architectures aim to blur the line between front-office and back-office applications by elevating commercial order management applications into the front office, where they can be an equal partner to other functions in the ecosystem such as CPQ. A commercial order management application acts as a broker of data to the systems required to fulfill the orders generated by the front office regardless of whether that data is commercial or technical in nature. It’s important to note that IOM is not intended to fully replace complex Service Order Management (SOM) level orchestration systems that provide service inventory and often support more complicated provisioning rules and technical fulfillment system interfaces. As a best practice, only the commercial data and technical data needed to support selling processes should be modeled in the EPC.

SOM level orchestration is typically a back office function that communicates with the lowest level fulfillment system in the architecture. It orchestrates the fulfillment of entities that have no commercial relevance and aren't needed for the selling processes done in the front office.



The above diagram shows a sample of the types of fulfillment systems that an order management application may interact with during the fulfillment process. For example, order management may need to access an inventory system to reserve the resources needed to fulfill the service, it may need to trigger the shipping of physical goods to the customer, initiate activation of the service in the network, and initiate billing of the service once it has been activated.

# Discovery and Design Best Practices

**Discovery**

In an IOM project, fulfillment designers start with a discovery phase.

Discovery sessions are very important to design a better solution. Discovery should include discussions aimed at understanding the business processes, requirements for fulfilling an order, and performance threats.

**Business Processes**

* Document/diagram the existing processes if any–identify steps that can be automated or optimized.
* Whiteboard (LucidChart/Visio) the optimal business processes.

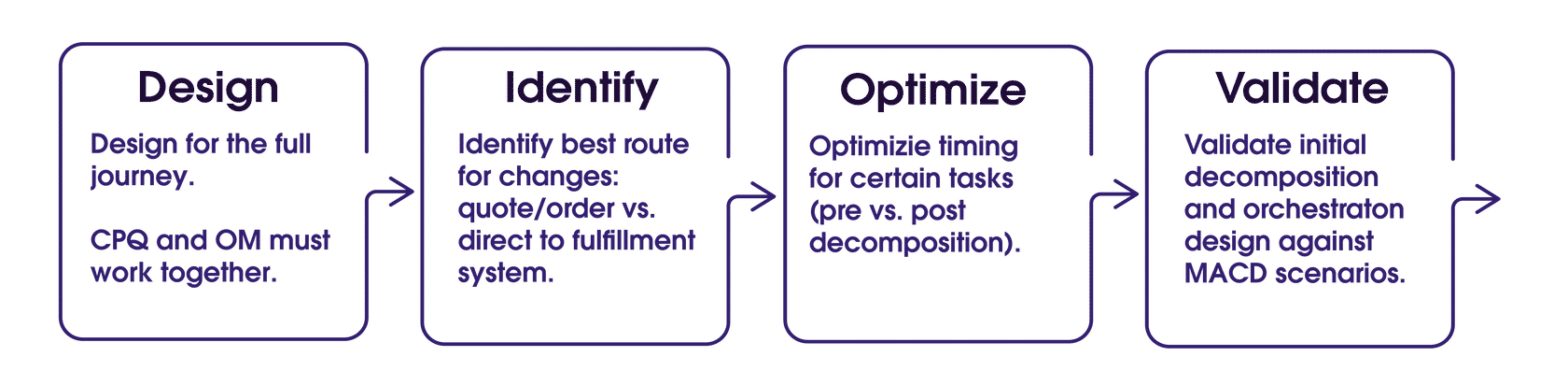
**Fulfillment Systems**

Information regarding the ecosystem (***that is, the Fulfillment Systems that must be integrated during orchestration***) should be gathered, including:

* **Complete list of Fulfillment Systems and their functions**
  + **Categorization of types of interactions with the Fulfillment Systems:**
* Automated or manual
* Synchronous or asynchronous
* **Communication with automated systems should also consider:**
* Communication protocols, endpoints
  + **Recommendation:**prefer JavaScript Object Notation (JSON)-based Representational State Transfer (REST) endpoints
* Callout payloads—Understanding the payloads exchanged with external systems allows for a better design of decomposition.
  + **Recommendation:** Ask for yaml/xsd and request/response examples, and identify abnormally large payloads as these will impact performance and scalability.
* Error conditions—Understanding the error scenarios in integration with the external systems provides input that will drive Retry Policy and fallout management discussions.  See Overview of Industry Callouts for a list of HTTP response codes.
* Integration patterns with the Fulfillment System with respect to the action performed (add+delete versus modify).
  + **Recommendation:** This information will be an input to decomposition and orchestration design
* Identify performance capabilities for Fulfillment Systems, that is, timeouts, number of requests, and throttling if applicable.
* Integration complexity
  + **Recommendation:** For complex integrations or if multiple callouts are required in order to submit single-action (micro service application programming interfaces or APIs), consider a middleware solution for integrating with Legacy Fulfillment Systems.

**Design Process**

The design process will be iterative, and it’s recommended to review and validate the design before implementation. This may help avoid unnecessary rework and reduce the occurrence of unused configuration or complex conditions.



* Plan for an iterative approach rather than a long big-bang cycle. Iterations may split delivery by Product Families or Products, but it also typically means splitting the main type of orders: new provide (refers to an order that adds new assets to the account for the first time), MACD (orders that move, add, change or delete an existing service), and in-flight amends (orders that change other orders that are currently fulfilling in the system).
* Assets are created from the order items in the new provide order. During the assetization process, one must consider the fields/attributes that must or must not be persisted. Since MACD orders start from previously created Assets, the characteristics previously persisted will play an important role in the decomposition of the MACD order. For example, one-time characteristics that are only required during the fulfillment process may not need to be persisted, as they may not be relevant and impact the processing of the subsequent MACD order.
* Even though the plan is to go live with new provide orders first, the design should validate the product model, decomposition, and orchestration against MACD processes. When not validating against MACD, it’s possible to create a model and process that generates assets that are not compatible with the MACD process. Remember that decomposing a MACD order will consider the incoming order and the existing commercial and technical assets.
  + Identify the best route for MACD orders:
  + Asset → Quote → Order → IOM. This route is longer and more time consuming, but it may be necessary in business-to-business (B2B) implementations where changes must be negotiated with the customer.
  + Asset → Order → IOM. This is the typical MACD route taken for changes.
  + Direct changes on Fulfillment Systems. This should be considered in cases when small, high-volume changes must be processed and they do not need to be recorded on the Asset records. Often, these type of orders are not commercial and better routed directly to the SIOM layer.
* Optimize the submit process. Remember, the decomposition process is a very complex process that requires loading and processing a large number of records. Sometimes, the requirements call for additional operations to be performed during the order submission process. In this case, one must consider the processing duration and number of queries executed within the transaction so that governor limits are not reached.

**Performance and Scalability Considerations**

Timely understanding of the order sizes and volumes that must be processed is critical to make correct design decisions that future-proof the solutions.

**Here are a few points to consider:**

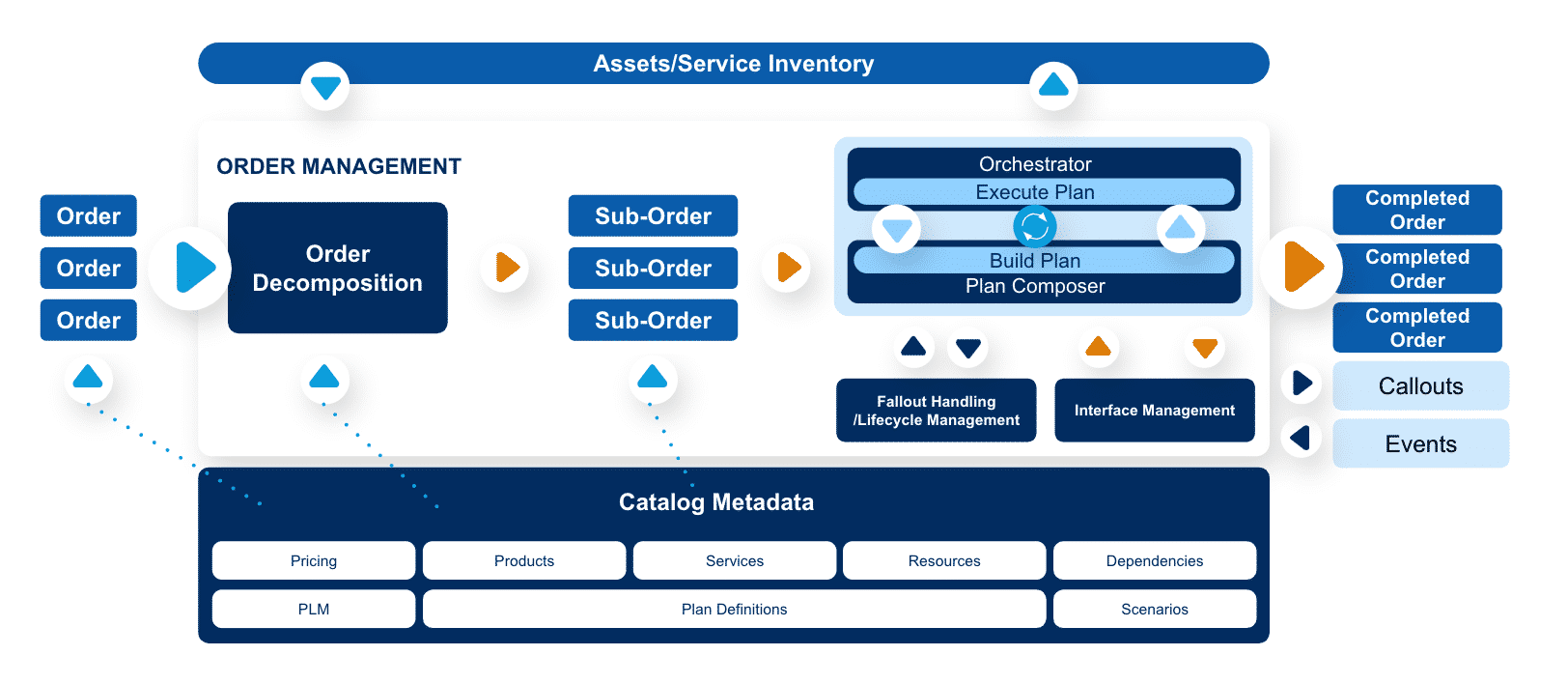
* **Large Orders may hit governor limits during processing**
  + There are several factors that contribute to an order being considered large including: the number of line items, number of attributes per line item, number of picklist attributes, depth of hierarchy, number of decomposition relationships and their conditions, number of fulfillment request lines items generated and the number of attributes per fulfillment request line.
  + **Recommendation:** Multi-site orders should be split by site into multiple sub-orders (component parts), thus reducing the order size processed per transaction. ESM has out-of-the-box capability to assist with this.
* **Order Volumes and existing Org Loads**
  + **Recommendation:**Correct sizing of volumes avoids surprises down the line when processing on the org is slowed down.

# Decomposition Best Practices

**Decomposition Design**

The main purpose of order decomposition is to decompose an order into sub-orders, or component parts.

The IOM decomposition process is responsible for enriching an order with the technical information required to fulfill the order during orchestration. This may include generic, reusable technical products or attribute values that are not provided by the customer relationship management (CRM) system during order capture. The SFI shared catalog provides the product configuration details required at runtime in order to operate in an enterprise catalog-driven manner, as shown in the following diagram.



**Consider the following during design:**

* Consider **TMForum SID model**in design.
* Pure academic designs may not be the most optimal ones. Whiteboarding detailed and deep hierarchical product models may look good on a diagram but may be inefficient in processing. Reducing hierarchical depth, and the number of layers of decomposition can improve processing time and memory usage.
  + For examples of multi-level decomposition
  + Design decomposition to support MACD and Integration needs of underlying fulfillment systems.
  + Mapping transient information in decomposition may result in unexpected results in subsequent MACD operation. An example of transient information is the Order Item Action—the value of the Action field makes sense for the current order, and, if mapped to a persisted technical attribute or used in a decomposition condition, it may impact the decomposition results.
* Identify information that needs to be persisted in technical inventory versus data that is transient and can be discarded after fulfillment.
* Remember that decomposition is not a strict requirement for integration with fulfillment systems, and in some cases, and for some fulfillment systems, there may be no decomposition required.
* Identify repeated patterns and consider introducing reusable building blocks such as decomposition classes and CFS specifications for future scalability.

**Product Modeling for Decomposition**

EPC/Shared Catalog is where you model and define the technical products to use in the decomposition and fulfillment steps.

Specifications defined in the Technical Catalog are defined like other products, meaning that most of the fields you configure for the Commercial Catalog also apply. However, there are some differences: The technical specifications have no need for pricing, and the product should never be orderable. There are likely no attachments as the product is never exposed in CPQ.

Conversely, there are configurations that are only used by IOM, such as the Scope field. The Scope field is important in controlling the decomposition behavior. You can also manage product attributes through EPC and use appropriately designed layouts to simplify product creation.

You model the Commercial Catalog from the perspective of the order capture and assets created for a customer. You model the Technical Catalog, however, to meet the technical requirements of fulfillment and to make the integration with fulfillment systems easier.

There are two main approaches when it comes to modeling the Technical Catalog:

1. **Flat Modeling**
   1. The model is only concerned with the resource-facing services (RFS) layer and is modeled to facilitate the integration with fulfillment systems. Flat modeling typically has less complexity because there are fewer decomposition relationships and mappings.
2. **Hierarchical Modeling**
   1. The model is composed of the customer-facing services (CFS) layer and the RFS layer. The CFS layer confers an abstraction of the RFS layer that’s less technical and more understandable from the commercial side. It is advised though to avoid multilevel decompositions. Multilevel decompositions complicate decomposition configuration and generally require more time for maintenance and testing.

To fulfill orders containing commercial products, a fulfillment designer must create corresponding decomposition rules. This may result in a very large technical catalog as well as a considerable amount of repetitive, potentially error-prone rules that are difficult to maintain.

For example, a TV service may have a large number of channels or channel packages that can be defined that all share the same fulfillment process. Another example is a variety of commercial broadband offerings, such as bronze, silver, and gold, that are all realized through the same Broadband fulfillment process.

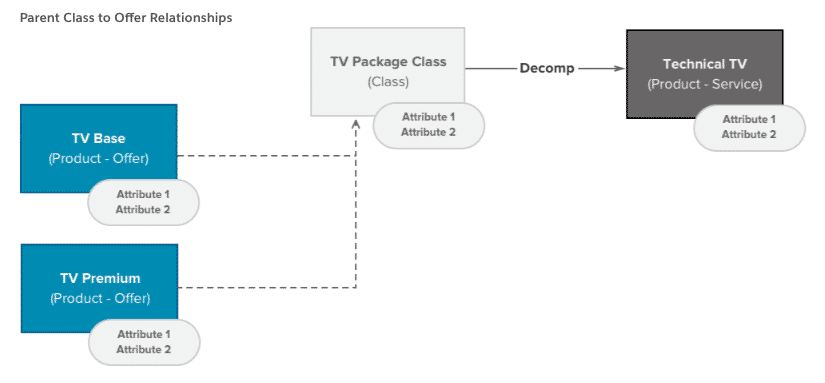
It is strongly recommended that fulfillment designers reuse the same entities and the same configuration as much as possible.

IOM generates actions associated with the decomposed technical products automatically, as a result of an analysis of the actions and products on the incoming request, against any existing technical inventory held by IOM. For example, if no assets are located by IOM for a new order, then all decomposed actions generated by IOM will be Add, since IOM assumes it’s a new customer.

As a fulfillment designer, you can create a technical product with a type of “Class” that can be shared by all products that are related.  This allows you to define a single decomposition relationship once for the class instead of for each commercial product.

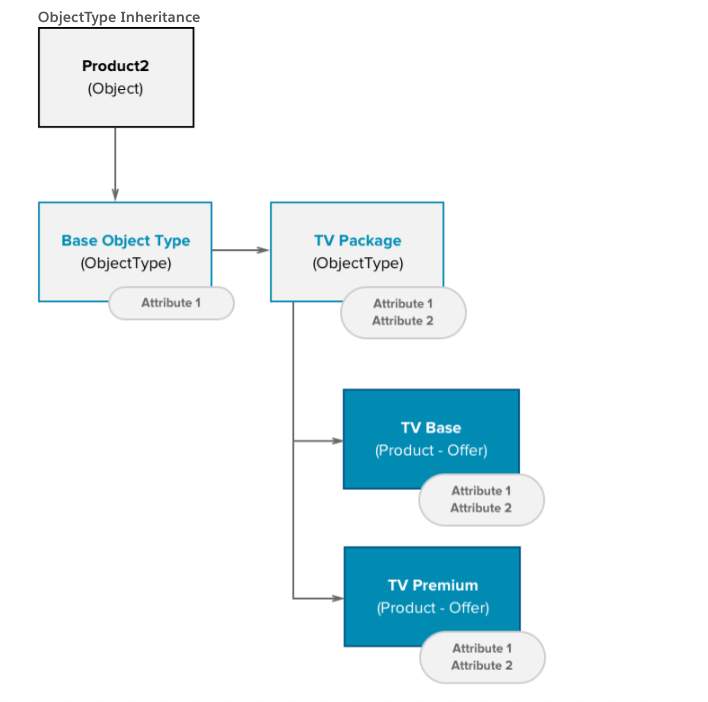
**To share decomposition configurations across multiple products:**

* 1. Create a product with a record type “Class” and define the decomposition against this product.
  2. Create new offers that reference this new class by populating the Parent Class field with the product that you created in step one.



Catalog designers can use ObjectType to model products that share the same traits. Use of ObjectType affords inheritance among products declared of the same ObjectType. It also enables inheritance for ObjectTypes themselves. The inheritance means that an attribute defined (bound) to an object type is automatically bound to all products of that specific ObjectType. This is a very powerful concept for product designers.

The following diagram shows how attributes are inherited across a hierarchy of ObjectTypes. For more information, see **Create an Object Type.**

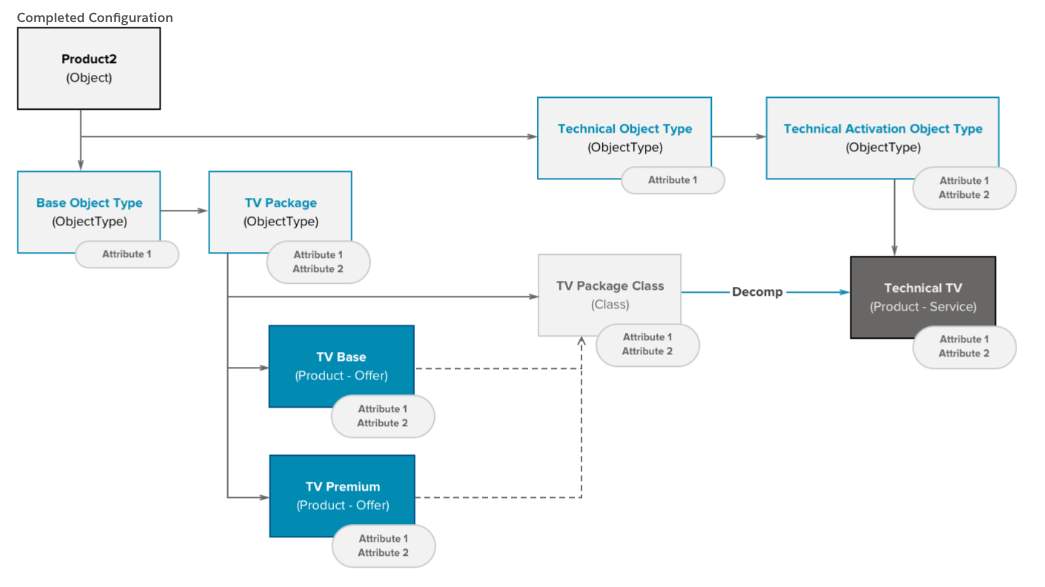


However, IOM does not currently use ObjectTypes but does use the parent class concept. Catalog designers can take advantage of both concepts at the same time to optimize the way decomposition works.

You can use ObjectType with both commercial and technical products for more effective modeling. As previously mentioned, to allow for class-based decomposition, the fulfillment designer creates a class and defines the decomposition rules against the class, then updates the offers to set the Parent Class field to point to the class.

Since the decomposition function maps attributes and fields to technical products, the class must include the same attributes and fields in its configuration. Use the ObjectType in the Class definition and thus inherit fields and attributes. This can greatly reduce the amount of work and provides better control of attribute definitions.

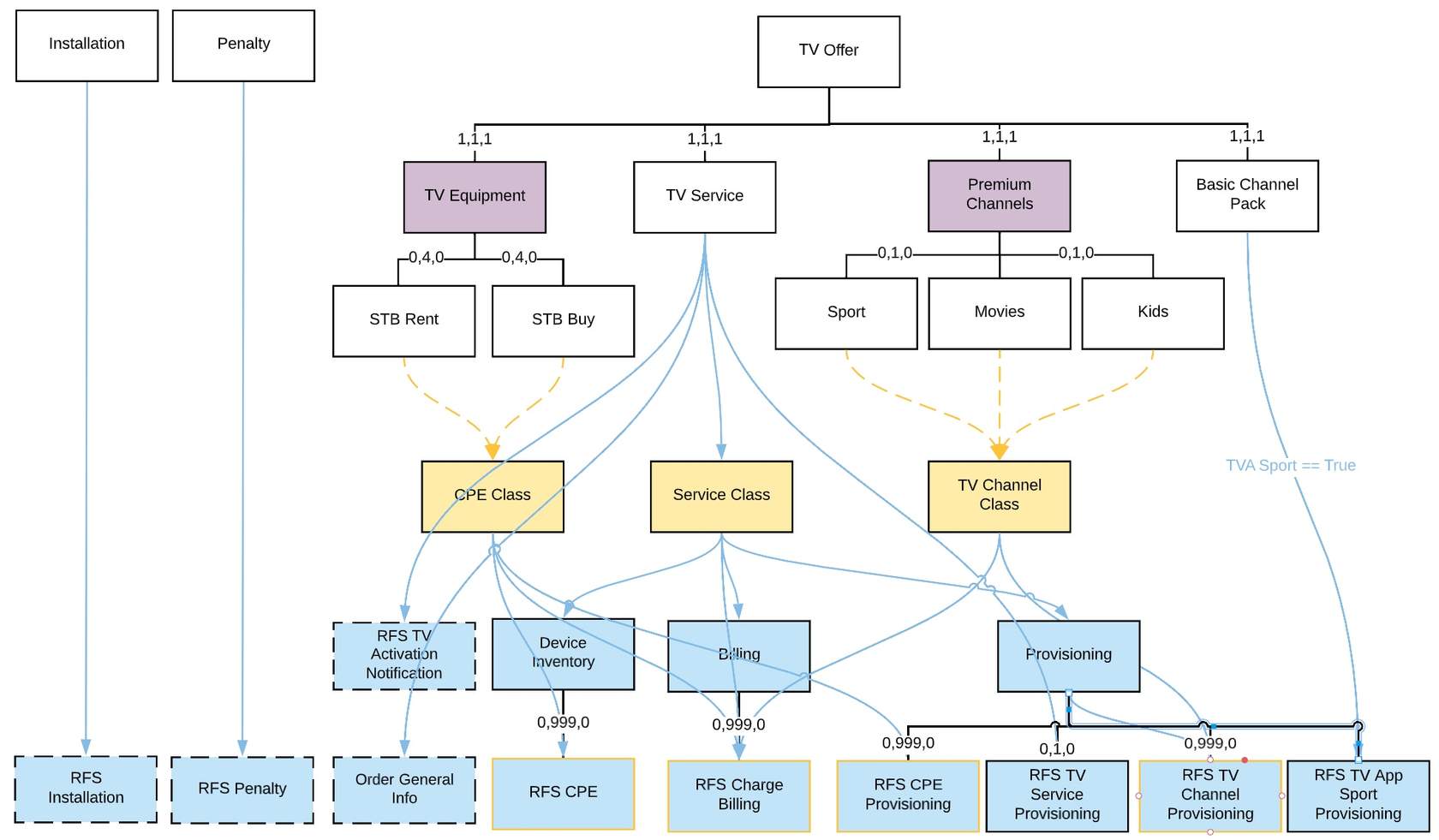
The following diagram shows a sample configuration using both ObjectType and Class.  Imagine you have 10 or 20 similar TV offerings as in the diagram below. Rather than creating and maintaining 10 or 20 separate decomposition relationships, you only have to create and maintain one. A single decomposition relationship between the product class (TV Package Class) and the technical product (Technical TV) can achieve this.



The product configuration is mostly performed through the Product Console. However, there are exceptions. Use the Product Console or Product Designer to create product classes and configure them.  You may need to change the record type after creation.

**Visual Decomposition Design**

Use any Unified Modeling Language (UML) tool, like Visio or Lucidchart, to design the decomposition visually. Always model the decomposition within a diagram first before starting configuration. Visualizing all the structures and end-to-end processes will help identify unnecessary entities and allow for process optimization. It’s much more effective to identify these during the design phase than to refactor post-implementation.



In the diagram, colors represent the following:

* **White background:** Commercial product or offer
* **Purple background:** Virtual product
* **Yellow background:** Product class
* **Blue background:** Technical product
* **Blue lines:** Decomposition relationships
* **Yellow dashed lines:**Parent product ID, set by the product-to-product class relationship
* **Straight black lines:** Parent-child relationship
* **Blue text on the decomposition relationship:** Decomposition condition

**Product Class-Based Decomposition**

* Reduce the work of configuring decomposition relationships by using class-based decomposition.
* Remember that it’s easier to associate a product to a class than to maintain decomposition relationships and mappings for it.
* Ensure use of object types to reduce maintenance work when attributes are added/removed.
* Maintain separation between Commercial Model (Offers, Products, and Services) and Technical Model (CFS, RFS, and Resources). Commercial Model is driven by Product structure as presented to the customer, and Technical Model is driven by fulfillment system requirements.

**Product Modeling for IOM**

When working with IOM, consider the technical catalog model as early as possible in design. Modeling in IOM begins with design-time configurations that result in the running of dynamic workflows to fulfill the products, services, and resources defined on orders.

The process of IOM leads to creating the assets for the customer. This occurs when the product is being provisioned and external systems must be contacted to book or provision the product.

**The assetization process creates, updates, or deletes assets and technical Inventory Items.** Technical items, also called Inventory Items, are the realization of CFS and RFS, just like assets are the realization of commercial products.

**Modeling Product as an Attribute**

To reduce the catalog size, sometimes a product is modeled as an attribute. This translates to an attribute that has a true/false value, symbolizing whether or not the customer has the product. Using attribute-based pricing can also impact the pricing of the product by changing the values of an attribute.

Setting the attribute value to true should have the effect of activating the product in provisioning. Conversely, setting the attribute to false should result in de-activating the product in provisioning.

This behavior is easily modeled via decomposition conditions.

**Mapping to a Flat-Model Fulfillment System**

When integrating with a Fulfillment System that accepts a fully flat model (aka SKU model), one must consider the potential volume of records if the mapping is a result of conditional decomposition modeling. A classic example would be integration with a billing system that requires a distinct billing code for each combination of characteristics of a product or service. For example, an internet offer can be sold in multiple combinations of download and upload speeds, as well as different terms. Each distinct combination is mapped to a distinct billing code.

One approach is to implement the mapping using distinct technical products for each billing code and use conditional decomposition for mapping the commercial offer to the technical products. While this solution would achieve the out-of-the-box goal and is certainly making use of IOM’s features, the result is a large catalog that is difficult to maintain as well as a large amount of computation during decomposition of the order as IOM must evaluate all conditions to assess whether the decomposition relationship is applicable to the order.

A more creative approach is to map to a generic technical product that has an attribute for billing code, which is populated during orchestration rather than decomposition. One could create a task that uses a calculation procedure and calculation matrix to retrieve the billing code based on multiple characteristics configured in the offer.

This approach is similar to the **Attribute-Based Pricing** method employed by CPQ.

There are a few advantages of this approach:

* Simpler and more compact Technical Catalog which is easier to maintain
* Vastly improved run-time decomposition performance
* Separating mapping of commercial products to technical products from mapping of different characteristics to a billing code in different transactions; results in significant performance and scalability improvements

**Large Order Considerations**

Sometimes, orders can get quite large in terms of the number of line items they contain. As mentioned earlier, decomposition must process the entire order (that is, all line items) and the commercial and technical assets that the line items reference. As such, the decomposition process may result in a situation where processing a large number of items exceeds the resources allowed by the Salesforce Platform for one transaction:

* Apex central processing unit (CPU) limit
* Heap size limit
* Salesforce Object Query Language (SOQL) limit

A classic example of a large order is an order created for a large business account that wants to deploy services and products to multiple sites. These are typically B2B orders and they may have a large business value.

The recommendation, in this case, is to split the order into multiple smaller orders—typically one order per site—and submit each order individually. This will reduce the processing per transaction and allow processing and monitoring of each site order independently of the others.

# Orchestration Design Best Practices

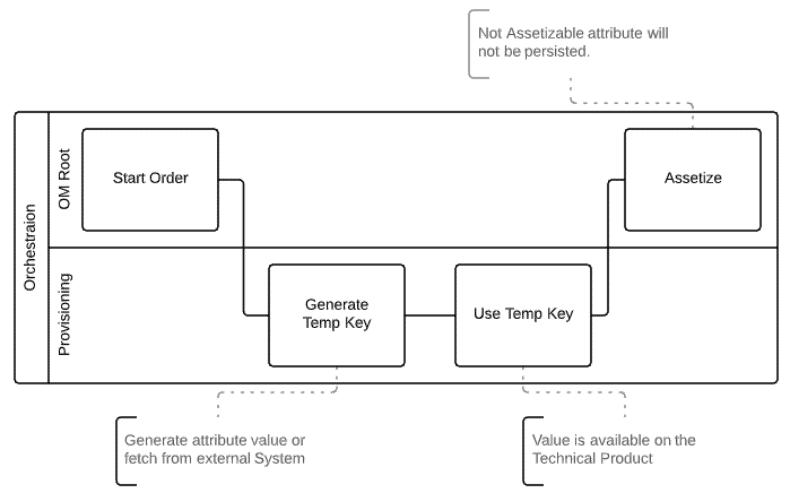
**Process Variables**

Sometimes, information is generated and obtained during fulfillment and passed from one orchestration item to another, but then is not persisted. This is referred to as a need for a process variable–data generated within the orchestration but not needed after the orchestration completes.

It’s possible to achieve the same results using previous releases without using the Not Assetizable Attribute feature. Control the value of the attribute using an auto-task. Essentially, before assetization, ensure that the value of the attribute is emptied so that it does not get assetized.

**Follow these steps:**

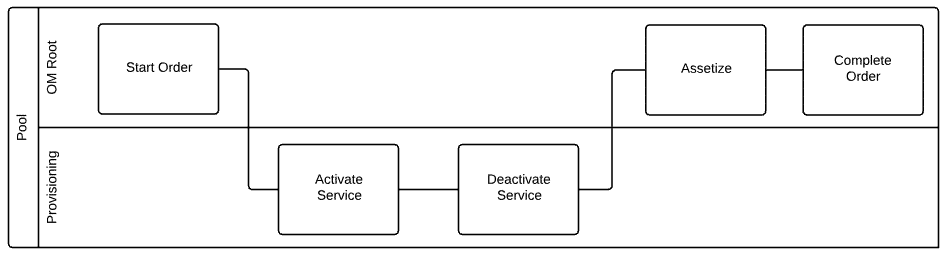
1. Define the attribute on Technical Product.
2. Mark the attribute as Not Assetizable.
3. The attribute should only be set in fulfillment.
4. Make sure you do not map the attributes in decomposition!5
5. If the orchestration item retrieving the value and the orchestration item that needs to use the value work with different Technical Products:
   1. Copy the value from one Tech Prod to the other via a custom Auto Task, or
   2. Define the attribute to a Tech Prod that is available to both tasks.



**Conditional Tasks vs. Independent Orchestration Plans**

At times, different callouts are made for the same RFS to the same fulfillment system depending on the action.

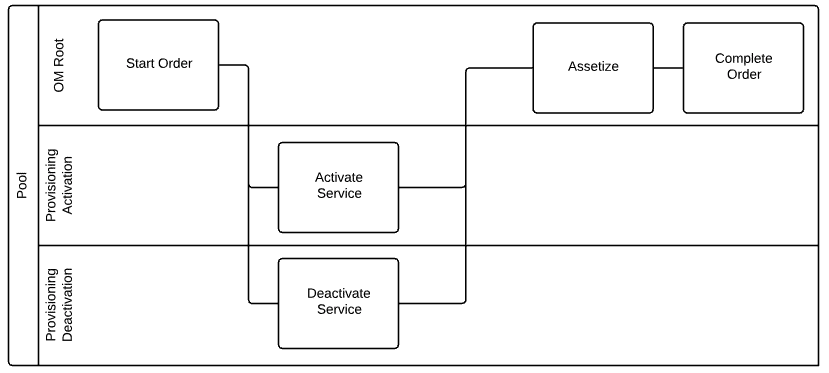
**There are two ways to model this:**



**1. Conditional Tasks on the same swim lane (that is, orchestration plan definition)**

If you’re modeling one orchestration plan definition for the fulfillment system, the scenario would be RFS Provisioning with both Add, Disconnect actions. In this case, you would need to add conditions to both tasks to ensure that Activate Service is only executed for Add, and Deactivate Service only executes for Disconnect.

The result of this modeling is a reduced number of swimlanes. But at runtime, you’ll always see both tasks in the orchestration plan, except one of them will be grayed out (that is, skipped) according to the action of the fulfillment request.



**2. Define individual orchestration plan definitions and map them to the action via scenarios.**

We can model one orchestration plan for each action. In this case, you end up with two orchestration plan definitions: Provisioning Activation and Provisioning Deactivation. Each has a scenario defined on the RFS Provisioning, but the action will be “Add” and “Disconnect” respectively. There’s no need to add conditions for the tasks in this case, as the condition is implemented by the scenario.

The result is a clearer orchestration plan where only one of the swimlanes is added depending on the action of the fulfillment request action.

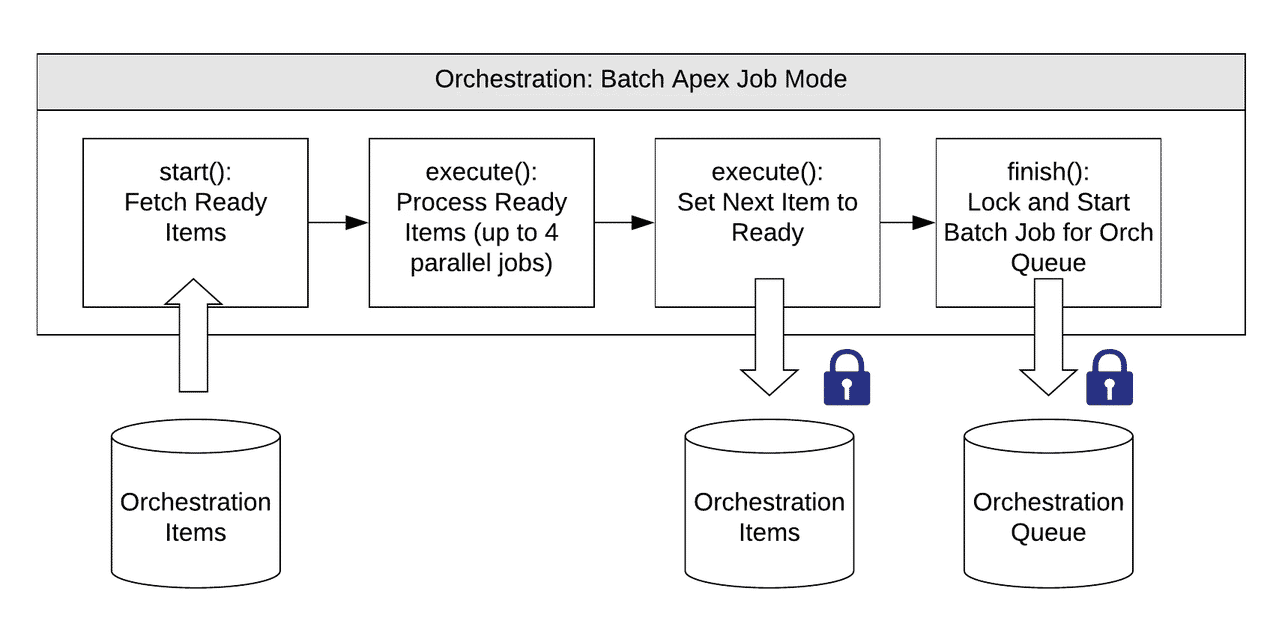
**Recommendations for using Platform Events Architecture to manage Orchestration**

Initially, Orchestration Items processing was implemented by using custom Orchestration Queue object (Queueing) and Batch Apex Jobs (Processing). Orchestration Queue subjects are used in Batch Apex orchestration mode to support parallel execution of Orchestration Items and limit the number of simultaneous jobs to four.

**This approach has a number of bottlenecks:**

* Orchestration Queue object is used as a ***synchronization***mechanism to ensure that ***only up to four Batch Jobs are running simultaneously*** since there’s a hard limit of five batch jobs running in parallel (Salesforce).
* The mechanism described above has an additional ***lock on the Orchestration******Queue*** object when Items become ready on initial Plan Creation and at the end of Apex Job run (Item execution). So, if a significant number of Orders are submitted, it leads to a ***lock contention problem*** related to Orchestration Queue.
* Ready Orchestration Items for a particular Queue are fetched by an Apex Batch Job from a database in the beginning of Queue processing. Query to fetch items puts an ***additional load on the database.***

**Here’s a high-level diagram of Batch Apex Processing:**



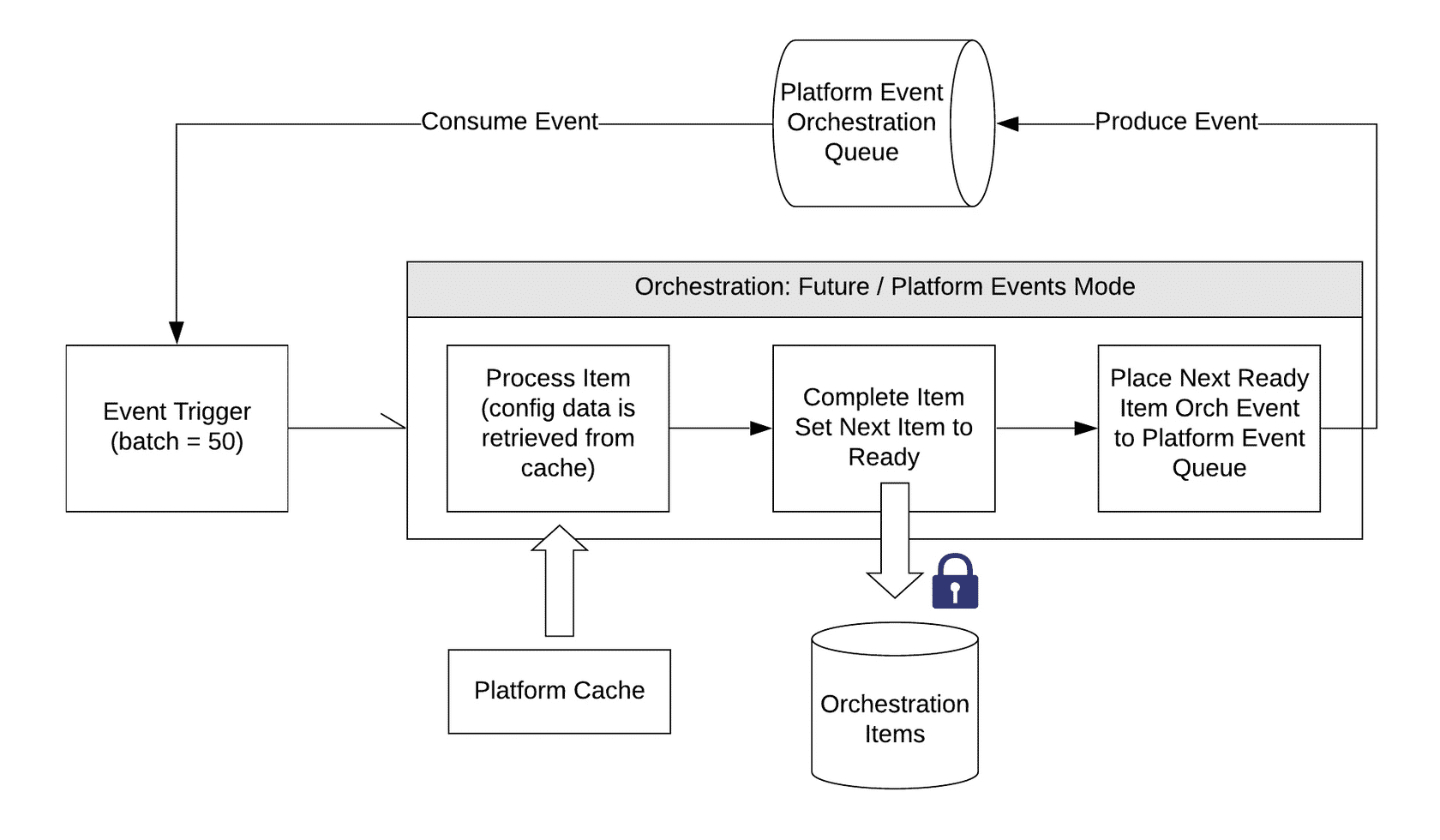
To avoid these bottlenecks, ***Platform Event-based Queue***was introduced. It allowed moving to a producer-consumer design pattern, avoiding resource locking contention problems and reducing the load on the database.

Once the Orchestration Item becomes ready, it’s placed in the Platform Event orchestration queue. Then, it’s consumed by an orchestration trigger and executed asynchronously via a future-based orchestration processor.

Future-based engines have a greater level of parallel processing (avoiding five transaction limit in batch jobs) and also reduce the number of async calls consumed (one call by future versus three calls for batch job start → execute → finish).

**Orchestration Flow**

**Here’s a high-level diagram of Platform Event-based processing:**



**Summary**

In summary, Platform Events mode is more performant and more reliable than Batch Apex mode." Therefore, Platform Events should be enabled as a best practice for B2C order processing with volumes that go beyond hundreds of orders per hour.

Please be aware, however, that when platform events is enabled (particularly with high volumes of order processing such as thousands of orders per hour), it is important to ensure an adequate number of platform events and async Apex callouts are available in the environment. These are shared resources across all of the applications and customizations installed in the environment and need to be considered when deciding how many of these resources are required.

# Integration Best Practices

**Building Callout Payloads**

Integration with external Fulfillment Systems typically requires an exchange of data via Hypertext Transfer Protocol (HTTP) requests. IOM prefers a REST approach for this type of integration and it provides the Callout orchestration item for this activity.

A Callout orchestration item makes an HTTP request to a preconfigured System Interface and expects a response back. The request will build a payload, usually in JSON format.

**There are a few ways to build the payload for the request:**

* Using the out-of-the-box System Interface Implementation. This will perform a simple serialization of the line items processed by the Callout into a JSON format. However, this may not be satisfactory when integrating with Fulfillment Systems that expect the payload in a particular format.
* Build the payload in a declarative manner using DataRaptor technology. IOM allows the fulfillment designer to configure multiple DataRaptors to be invoked at runtime to format the request payload for a Callout orchestration item. Similarly, response processing DataRaptors can be configured to process the response from the Fulfillment System.
* Create custom integration adapters. If the logic required for the Callout orchestration item to integrate with a Fulfillment System is more complex than what DataRaptors can achieve, a custom system interface implementation can be developed using Apex. Using this method, the developer is in control of all aspects of the Callout including request payload creation, integration protocol (for example, adding additional header parameters to the HTTP request), response processing, and error handling.

**Using DataRaptors for processing Callout payloads should always be considered due to their declarative manner (clicks versus code), and only resort to custom code if more complex logic must be implemented.**

**Integration with Chatty Fulfillment Systems**

Some Fulfillment Systems may have more chatty interfaces. This means that it takes multiple HTTP requests to achieve one business goal. For example, a Provisioning System may require an account to be created before the service can be provisioned.

**There are a few ways to implement this integration:**

## *Multiple Callout tasks orchestrated by IOM*

This will use the IOM platform to orchestrate the integration. It may be acceptable in some cases, but one must be cautious of the final complexity of the orchestration plan. A large and complex orchestration plan can be more difficult to manage in case of in-flight changes.

## *Single AutoTask invoking Integration Procedures to sequence the HTTP requests*

Integration Procedures provide a declarative way of configuring a simple workflow of activities that combine different steps of processing. Using Integration Procedures, one can combine custom Apex, DataRaptor, Callouts, and other actions into one transaction that’s invoked via REST or Apex. The benefit of this approach is in its declarative approach. It can also result in a more readable and more manageable orchestration plan in IOM.

## *Single Callout to a middleware API which implements the sequence of HTTP requests*

Complex integration logic is sometimes better suited for being implemented in a middleware layer. These typically use more advanced programming languages and expose REST APIs that can be used by multiple clients in the customer’s ecosystem. Additional considerations of security, performance, and scalability may make this approach more advantageous in the long term.

**Synchronous vs. Asynchronous Interactions**

During the discovery phase, the fulfillment designer must identify how IOM will interact with the external Fulfillment Systems.

1. Synchronous Interaction
   1. A synchronous interaction is blocking, meaning that IOM will post an HTTP request which is expected to be processed quickly by the Fulfillment System and receive the response within seconds. HTTP connections do not stay open too long; therefore, if the Fulfillment System fails to respond in time, the Callout will be considered failed.
2. Asynchronous Interaction

An asynchronous interaction can be implemented in two ways:

* Using the Asynchronous Callout. In this case, SFI IOM posts the request out and expects a quick acknowledgement from the external system. The HTTP connection and the transaction is closed, and SFI IOM will wait for the response on a specified endpoint. The Uniform Resource Locator (URL) of the endpoint is communicated to the external system, and it’s expected that the system will call into SFI IOM at the aforementioned URL when the request has been fully processed.
* Using a combination of Synchronous Callout and Push Event (tasks that pause orchestration until a condition is found to be true) orchestration items. The process is similar to the one described for the Asynchronous Callout, but the difference is that the fulfillment designer has the opportunity to configure the orchestration items in a way that allows other activities to execute in between the original request and the final response.

**An assessment of the performance and scalability of the external Fulfillment Systems must be considered and the appropriate type of interaction should be modeled in the orchestration definition.**

# Asset Migration Best Practices

**Asset Migration Overview**

Asset migration is the process of updating the asset database with items from an external database or assets that were not created via the new provide IOM flow. Asset migration is needed when we are moving customer assets/records over from a legacy system that is to be deprecated and into the new Salesforce system. IOM stores the asset records in the Asset standard object and the technical inventory in a custom object called Inventory Items.

The main goal of the migration process is to bring the artifacts that belong to the account (that is, commercial and technical product instances–aka assets and Inventory Items) to look and behave the same as the ones obtained via the new provide process.

Migrating the entire asset and Inventory Items database typically requires custom code that will create the records manually. This solution is fully customized to the needs of the project as it will need to implement the same logic designed for decomposition. Typically, this solution involves a staging database used for processing, some level of extract, transform, and load (ETL) tools, and a large upload into Salesforce / Amazon Web Services (AWS) tables at the end.

**There are a few aspects to consider:**

**Assets**

The goal of the process is to ensure that the assets in the database have the correct field/attribute values as if they were created via the SFI CPQ/IOM.

* **Creation:**When importing assets from an external database, Asset objects need to be created in Salesforce and all the fields populated with correct values.
* **Update:** SFI Communications Cloud products are enriching the standard Salesforce Asset object with additional fields. Also, the Project solution may add new custom fields. When assets already exist in Salesforce from a previous solution, but they haven’t been used with SFI, they need to be updated to set the values for fields as per the product needs or solution’s design. Asset update may need to be considered again when the solution is changed, a new field is added/removed, or the behavior of an existing field is changed.

**Inventory Items**

These include the technical counterparts of the Assets with similar considerations.

* **Creation**: Create the Inventory Items that would otherwise be generated via the new provision process. This process may require creating them from scratch or importing them from a previous solution.
* **Update:** This is needed when the Inventory Items exist but the solution has been changed and attributes on the technical specification are changed. In this case, you may need to run a batch job that updates the attributes on the affected Inventory Items to bring them up to date.

**Creating Inventory Items for Migrated Assets**

Typically, Inventory Items are created during the assetization phase of order fulfillment. For assets that are not created via IOM, you need to create the Inventory Items during asset migration.

When the commercial assets have been migrated and are ready to use in SFI CPQ, but the Inventory Items have not been migrated, one solution to consider is the “No Change Order”.

Running a MACD order without having previously generated the Inventory Items will result in decomposition that creates FRLs with Add action, as the decomposition component will fail to find the Inventory Items that are specified in the decomposition rules. The Inventory Items are generated by default by the decomposition process and persisted during the assetization step of the fulfillment.

**High-level description:**

## Step 1

1. Create a special Orchestration Plan Definition that contains one single Orchestration Item Definition–namely, the Assetize task. While the scenario of this plan will be on the offer (same as the normal end-to-end scenario used for normal orders), we need to ensure that this plan is only invoked for asset migration purposes. This can be done in two different ways, both having the same effect:
   1. Define an order level field (that is, Order Type) that has a pre-set value used for migration. Set the scenario to activate the plan only when the preset value is found on the order.
   2. Create a custom SubAction value (that is, Migration) which is then used in the scenario condition.

## Step 2

Create a MACD order. Ensure that the Order object is configured to trigger the scenario defined in step 1.

## Step 3

Add assets to cart but don’t change any of their structure or value. This will result in an order that has all order items with the action of “NoChange”.

## Step 4

## Submit the order to IOM using the standard APIs.

1. Order decomposition will result in FRLs with the action of “Add”.

## Step 5

The Orchestration Plan should have only the special plan created in step 1. Assetizing the order will create the Inventory Items in the database.

The process described above must be applied to all top-level assets (offers) that require migration.

**Benefits**

* Reuse the existing definition of the technical products and decomposition relationships.
* Use the built-in logic for decomposition and technical products creation.
* When properly controlled during the order creation process, it allows the Project to go live earlier as the assets could be migrated on-demand as well as in the background.

**Caveats**

* Using this process involves an end-to-end utilization of the IOM. The order may take a few seconds for each top-level asset that needs to be migrated. The exact duration will need to be captured in the development process as it depends on the number of order items and the complexity of the decomposition.
* There isn’t much room for performance tuning. This process must be executed for each top-level asset/account individually (that is, cannot use batching due to the way information is cached in IOM).
* Since the process is performed in individual transactions, the database will contain for a period of time both fully migrated assets as well as yet to be migrated assets. Using a flag to identify the state of the asset is recommended.
* Given the state of the database with both migrated and non-migrated assets, certain reports may not be possible.
* When creating an order on a non-migrated asset, an on-demand process must be executed behind the scenes to migrate the asset before the order is created. As such, the order capture process will incur a slight delay which needs to be mitigated with UI solutions (loading masks, messages, etc.).

**Asset Migration Scale Considerations**

Migrating a large number of assets will be time and resource intensive. As such, scalability considerations must be considered early.

Consider the customer needs to migrate hundreds of millions of assets. This will put a considerable amount of stress on the database. Trying to execute the migration during offline hours may take a very long time. During this time, the customer is unable to perform MACD operations on the assets that have not yet been migrated.

***Consider On-Demand Migration to complement Batch Migration***

To mitigate this situation, the designers may consider extending the background batch jobs with on-demand asset migration. If a user intends to create a MACD order for a non-migrated asset, the system can attempt an on-demand migration of those assets. This may result in a slight user experience degradation, as the user must wait a few seconds for the assets in question to be migrated before being able to proceed with the order creation. But it may be a small price to pay when considering the uninterrupted mode of operation that it affords.

***Consider partitioning the Assets***

Large volumes of assets are better handled in chunks. For instance, partitioning geographically or by Product Family gives the project better control over the migration process.

**Asset Migration Testing**

Issues pertaining to migrated assets are not evident until the migrated assets go through End to End network fulfillment and billing activation. Hence a representative subset of the regression suite should be executed covering all the MACD scenarios for the Migrated Assets.

# Performance and Scalability Considerations

**Salesforce Limits**

Industries Order Management is an application built on the Salesforce platform. As such, it runs in the cloud on a shared platform and must conform with all the Salesforce governor limits.

There are several aspects of performance and scalability with respect to Salesforce limits. For instance, many of the APIs invoked on IOM read and write data to the platform. A developer could combine the IOM APIs with other custom operations within the same transaction. Developers may also use other Salesforce platform technology for custom logic (trigger, Process Builder, Flow) which performs operations that are included in the same transaction. Therefore, one must consider the Salesforce governor limits including, but not limited to:

* number of SOQL per transaction
* number of DMLs
* number of records retrieved
* APEX CPU limit
* heap memory usage, and more.

Additionally, IOM uses Platform Events and Asynchronous execution to improve scalability. These resources are limited and governed by customer-purchased licenses.

**Here are some details on these limits:**

**SFDC limits for Platform Events (HV, no add-ons):**

* Publish: 250,000 per hour (add-on +25,000)
* Delivery: 50,000 events for all clients per 24 hours (add-on + 100,000). 1.5 million monthly
* The event retention period is 72 hours

**SFDC limits for async execution (futures):**

* 250,000 or the number of user licenses in your org multiplied by 200, whichever is greater
* Async requests are of lower priority over sync requests; thus, job processing is delayed by SFDC if CPU/DB resources usage reaches a threshold (minutes to hours delay).

**Orchestration mode**

IOM can be configured to orchestrate tasks using a BatchApex or PlatformEvents mode. The former is based on batch jobs and the latter is based on Platform Events and it has proven to be more performant. When using the PlatformEvents mode, there will be one platform event and one asynchronous future execution used for each AutoTask or Callout orchestration item. Since these are shared org resources, the implementation must consider the processing volumes in conjunction with other activities that may be present on the org to ensure sufficient resources are available at runtime.

**Event-based orchestration benefits over custom Queue**

* Remove four queues-batch jobs limitation by using @future (not limited by five parallel txns)
* No lock contention due to lock on orchestration item object (counters and status updates)
* Async transactions are now shorter (saving time on locks), less time to hold resources
* Remove deadlock conditions and lock timeouts related to Queue object
* Reduce load on DB (SOQL to fetch items data is removed)
* Reduce the number of async requests (batch Apex job takes three async requests, futures have one)