Architecture Document

UPC 1.1

Design Specification

CONTENT

About this Guide

This document details the architectural aspect of Unified Product Catalogue (UPC) to help the product engineering team.

Support Need

For more info. refer to this websites:

Purpose of This Document

Clients can also call and send email at:

Phone: 8673213690

E-mail: routchinky2022@gmail.com

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Document Change Log

Date	Author	Version
11.12.2023	Lopita Darshini Rout	Initial Version 1.1

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Chapter

Introduction

The UPC is an application that holds a unified view of the products and offerings of a telecom operator from one or more source systems as well as those defined directly in it. The UPC presents a single view of the product definitions across one or more lines of business allowing a central place to create product offers to end customers. It therefore not only brings in the cost benefits of a reusable common platform infrastructure but also enables convergence of product offerings which were offered in silos before, thereby enabling the operator to create unique combinations of offers with up-sell, cross-sell specific to market conditions, geography, demographics of the subscriber and based on other subscriber analytics data.

For more information, see ref. *XXXXXXXXXXXXXXXXXXXXXXXXXXXXX*

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Purpose of This Document

The purposes of this document are as follows:

- Detail the architecture aspects of UPC primarily to help the construction by the product engineering team.
- Help in identifying the parts of the architecture that can be implemented in near-term and long-term.
- Ensure the NFRs affecting the functional aspects such as Multi-tenancy, Data model extensibility etc. are addressed by the architecture even if some of them are not implemented in the near-term.
- Identify reusable parts of the architecture that can be harvested into the Cross-Application Platform initiative.
- Enable product marketing to leverage relevant portions of the solution aspects for pre- sales or customer workshops on UPC.
- Define the boundary between UPC product and solution.

.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Audiences

The audiences of this document are:

- Product Engineering team
- Product
- Marketing team
- Architecture
- team
- Product Management team responsible for release
- Solution Architects

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Exclusions

Exclusions shows the features that doesn’t included in this document:

- This document does not deal with anything that falls into the solution integration aspects of UPC such as integrating with an external billing system or Order management. While this document does highlight the boundary with other systems, the API and interface details including configurations, it is recommended to prepare a separate configuration or user guide that details the UPC API, configuration and integration aspects for an end customer deployment.
- This document does not deal with the road map, prioritizing of the UPC feature implementation. Though it may help in identifying the implementation complexities involved in realizing certain features.
- Security as a NFR is addressed in a separate document.

Note:This document should not be directly released to external customers or field personnel interacting with customers.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Business View

Path: Business View

UPC allows product marketing / management of a telecom operator to create Product offerings by mixing products across different business lines. UPC becomes an aggregator of Product definitions from existing systems or 3rd party systems having product definitions that are specific to one or more lines of business into a common repository.

UPC will be used by Order Management, CRM, self-care and other systems as part of their core functions to look up a consistent and common method of access to a product catalogue. Thus users and applications belonging to different roles will have access to different parts of the UPC product catalogue for different operations on them. UPC brings the needed agility to define offerings that contain a mix of one or more products from different lines of business by way of offerings but also the flexibility in varying the commercial and eligibility aspects of those offerings in terms of a wide variety of factors such as price, geography, channel, subscriber preferences or others.

UPC Use Cases Summary

Path: Business View > Use Cases Summary

Use cases are another tool for capturing functional requirements of the system. They define a goal-oriented set of interactions between external actors and the system.

For more information, refer to the following sections:

Functional Aspects

Integration Aspects

Deployment Aspects

Functional Aspects

Path: Business View > Use Cases Summary > Functional Aspects

Functional aspects defines the functionalities of the product, version, data model, offering, relationships etc.

The following aspects are:

- Define Products that contain the logical, physical and technical resources necessary to realize the service offered on that product upon an end customer subscription.
- Define logical bundles of one or more Products to make offerings
- Define Offerings on Bundles with Commercial, SLA, Channel preferences that can be purchased by an end subscriber of the telecom operator.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

- Express complex relationships such as dependency, exclusivity, aggregation, substitution between products, conditions of eligibility for an offering, conditions on pricing using business rules.
- Manage the life cycle of a Product, Bundle, Offering from definition to launch
- Manage multiple versions of a Products, bundles, offering definitions
- Users belonging to different roles are entitled to perform different operations on the different parts of the UPC data model.
- Standards aligned (TMF-SID) data model.

Integration Aspects

Path: Business View > Use Cases Summary > Integration Aspects

UPC will be the single point of Product/Offering definitions across different lines of business. Towards this it should allow:

- Configurable/pluggable aggregators and listeners for sources containing product definitions.
- Configurable/pluggable Propagators to persist UPC product definitions to external systems.
- Configurable/pluggable Notifier on changes in UPC product definitions to consumers.
- API exposure of UPC functions for integration with external systems like OM, CLM.

Deployment Aspects

Path: Business View > Use Cases Summary > Deployment Aspects

It includes the analysis and planning of organizational change management measures to support the transition from old situation to the new one.

The purpose of this aspect is to deliver the updates to the user:

- Support multi-tenant deployments.
- Support extensible data models by configuration.
- Support i18N and l10N in application as well as the user interface.
- Support single node and cluster deployments.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Use Case View

Path: Use Case View

This visualizes the usage aspects of UPC as a target system from the point of view of the actors who are systems or people interacting with UPC.

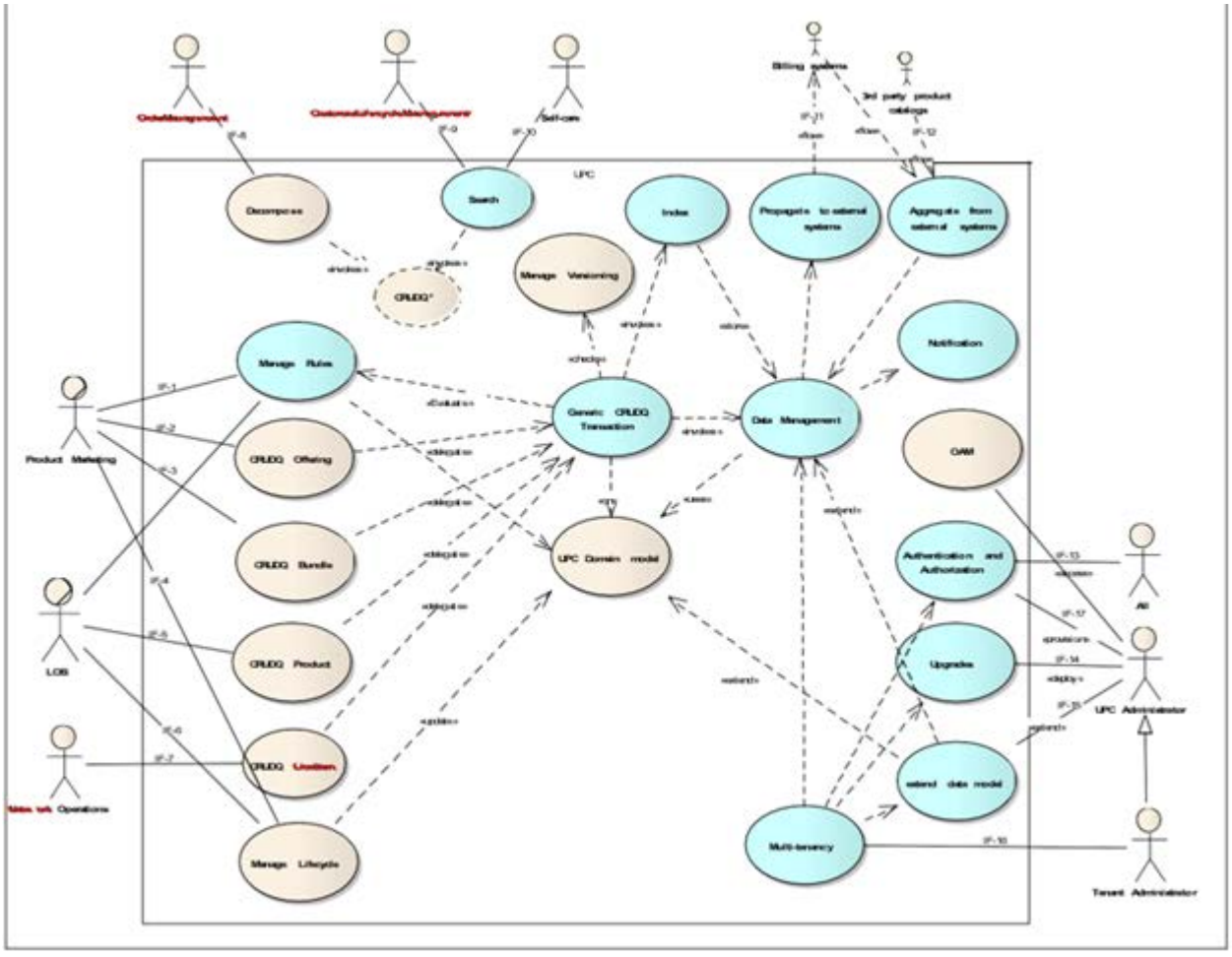
UPC by its definition allows product details to be stored and aggregated from several underlying systems.

Use Case Diagram and Description

Path: Use Case View > Use Case Diagram and Description

Use case diagrams help to capture the requirements of the system. It also identify the interaction between the system and its actors.

The Use-case diagram describes the functionality/feature as stated in the following screenshot:



©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Summary Description of the Use Cases

Path: Use Case View > Summary Description of the Use Cases

Use Case is a written description about the product, design and development, models etc. and how user will use this.

The purpose of the Use Case visualizes the usage aspects of UPC.

In the diagram above, the blue colored use cases are realized as:

- Reusable functions in the Cross-Application Platform
- And others are specific to UPC

The logical data model of UPC (UPC Domain model in figure) is visible to:

- The product owner
- Product marketing teams of an operator
- Mainly from the point of view of performing CRUDQ operations on it

The CRUDQ operations can be invoked via the UPC DS API which is currently based on SOAP/XML but can be RESTified in future releases. The CRUDQ operations can be invoked on different objects of the logical data model.

At present, the key objects in the domain model are:

- Offering
- Bundle
- Product
- Line Items

However, with the support for data model extensibility within UPC an administrator by means of configuration:

- Can add or remove domain model objects from the UPC schema
- Modify existing domain model objects in the schema thereby exposing different logical data models for different customer deployments or for different tenants in a multi-tenant deployment

The CRUDQ operations on the key objects of the UPC domain model in turn uses:

- The data management function for persistence
- And also may trigger one or more aggregators and propagators that are registered for retrieval
- Or propagation to external billing systems
- Or 3rd party product catalogues

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

The aggregators and propagators are configured within the Data management function and can be varied for different tenants in a multi-tenant deployment by configuration.

The authentication and authorization enables the actors to be classified into roles that are entitled to :

- Invoke any or all of the CRUDQ operation
- And allows fine-grained access at the level of the CRUDQ operations on the key domain model objects
- Thus, it is possible that a network operations personnel may be allowed to perform CRUDQ on only Line Items, while a Product owner / marketing personnel may have entitlements to perform CRUDQ on all parts of the logical data model

As a part of a CRUDQ transaction, UPC may implicitly perform versioning of the domain model objects and life cycle management or this can be made available explicitly as an option for the external users via API and UI.

- For example, a Product may be created in a 'In progress' state and while in that state may be updated any number of times. When the end user decides to get an approval, he may move the state to 'Pending' for approval and once approved, it may be automatically moved to a 'Inactive' state waiting for testing to be completed and ready to be 'Launched' and moved to 'Active' state. The state transitions can be a combination of automatic and manual means

Also, the presence of a Product definition in a certain state allows certain operations possible by certain roles.

- For example, a Product in an 'Inactive' state may not be made visible to a 'Search' done by a self-care or a CLM

Similarly, the Versioning is an aspect that allows:

- Different versions of a Product or Offering definition to co-exist and be simultaneously accessed by different tenants in a multi-tenant or a single tenant scenario
- Versioning is also an implicit function of UPC to check the Updates on a Product or Offering which is in an Active state and based on the differences can automatically result in creating a new version
- Again this is a function which has both manual and automatic controls

While creating the domain objects in UPC, there are situations where:

- The relationship among the UPC domain objects (say between two products)
- Or the pricing and eligibility aspects of consuming an Offering by an end subscriber may have to be expressed in a complex way involving predicate logic
- And parameters related to subscriber or Products

UPC as a system should allow a Product manager to create such Rules on the domain model objects.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

- For example, UPC may expose offerings for a subscriber who is browsing a product catalogue in a self-care portal to show only what is relevant based on certain factors like the subscriber's age or their other subscriptions
- Another example could say two products cannot be used as part of a Bundle for making an Offering or they can be bundled only when some conditions like geography or sales channel are met. There could be situations where the execution and enforcement of rules stored within UPC happens outside of UPC application, such as an Order Management application for example

UPC Actors and Interfaces

Path: Use a case view > UPC Actors and Interfaces

UPC Actors and Interfaces represents anything outside the system like: Marketing, Management, Billing, Model, Lifecycle Management and Versioning, Rules etc.

For more information refer to these section:

Actor- Product Marketing

Actor- Line of Business Owner

Actor- Network Operations

Actor- Order Management

Actor- Customer Life Cycle Management

Actor- Self-care

Actor- Billing Systems

Actor- 3rd Party System

Actor- UPC Administrator

Actor- Tenant Administrator

Use Case - UPC Logical Domain Model

Use Case - The Generic CRUDQ (Cross Application Platform)

Use Case - Data Management (Cross Application Platform)

Use Case - Manage Rules (Cross Application Platform)

Use case - Life Cycle Management

Use Case - Versioning

Use Case -Multi-tenancy and Data Model Extensibility (Cross Application Platform)

Use Case - OAM

Use Case - Authentication and Authorization

Use Case - Indexing

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Actor- Product Marketing

Path: Use Case View > UPC Actors and Interfaces > Actor-Product Marketing

The process of communicating a products value to the user/customers is Product Marketing.

This role is assumed to understand the subscribers and based on:

- Inputs from markets, geography
- Subscriber demographics
- And other analytics create the Offerings that can be presented to end subscribers of an operator for making a purchase by combining one or more products across lines of business

Actor- Line of Business Owner

Path: Use Case View > UPC Actors and Interfaces > Line of Business Owner

Line of Business is a general term that describes the related product or services a business or manufacturer offers. It serves a particular business need or a particular customer transaction.

The Line of Business is a role played by:

- A person within a telecom operator owns a specific line of business (like GSM voice, Broadband and so on)
- And who has intimate knowledge of telco products and their realization in the network which can be offered for sale to end customers

Actor- Network Operations

Path: Use Case View > UPC Actors and Interfaces > Network Operations

The actor-network theory states that each part of a system should be viewed as equally important and as belonging together in an interactive relationship.

The Network operations defines the Line items or technical articles that may be both:

- Customer facing
- And resource facing to activate a service in the OSS elements of the telco network for a specific product

Actor- Order Management

Path: Use Case View > UPC Actors and Interfaces > Order Management

Order management is a process of order capturing, tracking, and fulfilling customer orders. It begins when an order is placed and ends when the customer receives their package.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

When a new customer order is processed:

- The corresponding Product Offering
- Constituent products
- Line items are looked up by the Order management subsystem to provision the different systems involved to:
    - Activate the service
    - Bill the customer
    - And also update customer subscriptions in CLM

The OM will use the UPC DS API to decompose an Offering into its constituent parts and also obtain the rules defined for evaluation and enforcement during its transaction processing flow.

Actor- Customer Life Cycle Management

Path: Use Case View > UPC Actors and Interfaces > Customer Life Cycle Management

Customer care will have to show the offerings from the UPC for purchase by the customer. It will use the UPC wrapper API to:

- Allow search
- Read on offerings
- And show the different offerings from the lines of businesses
- The customer can place an order on the displayed offerings and this result in the CLM creating an order after obtaining the necessary document proofs from the customer and triggering the OM flow
- The Customer care should not be able to read details of the product like Line Items or others
- This brings the need to expose 'selective' parts of the UPC data to the different parties apart while allowing different operations on them
- The CLM will hold the active subscriptions of the user and manage the customer relationship aspects including new offers, trouble tickets and queries on billing etc

Actor- Self-care

Path: Use Case View > UPC Actors and Interfaces > Self-care

Used by end subscribers to browse product catalogues for offerings that can be purchased.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Actor- Billing Systems

Path: Use Case View > UPC Actors and Interfaces > Billing System

Generates bills for prepaid and post-paid on different services. Closely related to the Product definitions in terms of:

- The pricing aspects
- And rules related to pricing

The UPC has to synchronize with existing Billing system in terms of Product definitions and the OM has to provision the pricing related aspects when a customer purchases an Offering for:

- Billing the purchase
- Subscription

The Aggregation and Propagation functions of UPC are involved to perform this synchronization.

Actor- 3rd Party System

Path: Use Case View > UPC Actors and Interfaces > 3rd Party System

These can be product catalogues of 3rd party systems that are presented as part of a unified offering from UPC. Hence there needs to be continuous aggregation of changes in these systems into UPC.

Actor- UPC Administrator

Path: Use Case View > UPC Actors and Interfaces > UPC Administrator

System administrator is a person responsiblefor configuring and managing a company's entire infrastructure including all of the hardware, software, and operating systems.

The UPC system administrator is responsible for:

- OAM
- UPC application Upgrades
- And data model extensions (with the help of data modellers and product managers)

Actor- Tenant Administrator

Path: Use Case View > UPC Actors and Interfaces > Tenant Administrator

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Administrator specific to a tenant in a multi-tenant scenario. Handles tenant specific administration in terms of upgrades, OAM, data model extensions, Roles and entitlements etc.

Use Case - UPC Logical Domain Model

Path: Use Case View > UPC Actors and Interfaces > UPC Logical Domain Model

UPC allows CRUDQ operations on the UPC domain model objects which results in adding a new entity to the UPC database. This can be a creation of a Product or an Offering or a Line Item.

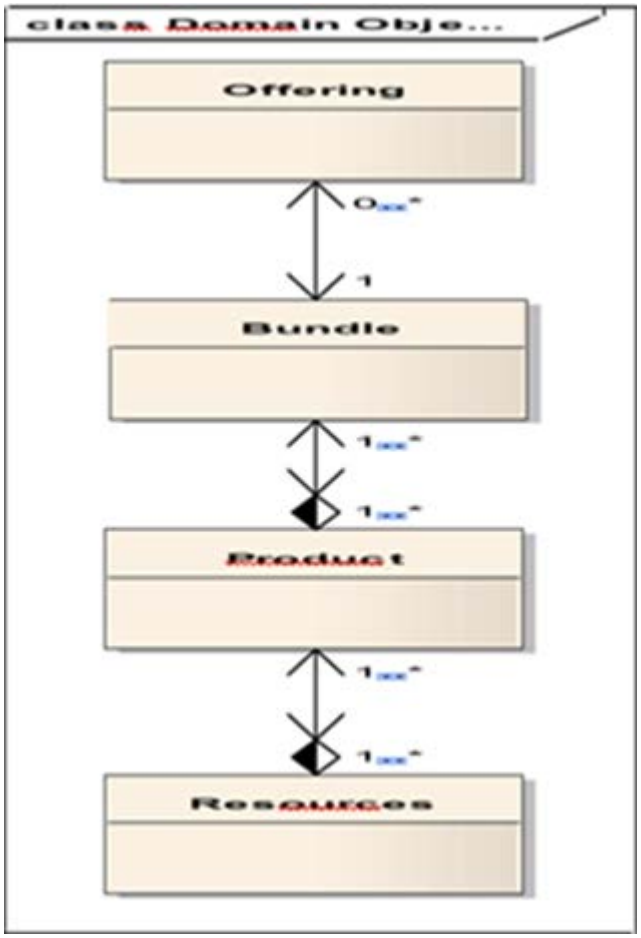
An example UPC domain model is shown below.:

- There are few key objects within the UPC domain model such as Offering, Bundle, Product, Resources. There may be other fringe objects such as
  - A Price
  - Channel
  - Geography
  - And so on which link to one or more of these key objects and not shown in this picture
- The key domain objects of UPC are linked up in a hierarchy.
- In this example, an Offering requires:
  - A Bundle to be available
  - A Bundle requires one or more Products to be available
  - And a Product requires one or more Resources to be available
- The aggregation relationship shows that entities can exist without belonging to its hierarchical parent. The bi-directional relationship enables traversal from both directions for a look up.
- The UPC domain model can be a variant of TMF SID domain model adapted to a specific operator's requirements. The logical domain model can be extended and customized in different deployments. Because of this, it is essential that the CRUDQ operations are domain model agnostic and the extensions to the logical domain model are handled by configuration.

Having a logical representation of a domain model like XML separate from the physical representation within databases allows the possibility to go beyond database technologies or specific persistent mechanisms.

- The logical data model also enables the capabilities like aggregation and propagation from external data sources as the physical models of each data source can be different and hence to transcend beyond the physical models, a layer of logical model based representation of UPC domain model objects are required.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template



Use Case - The Generic CRUDQ (Cross Application Platform)

Path: Use Case View > UPC Actors and Interfaces > Use Case - The Generic CRUDQ

The Generic CRUDQ brings in a data model agnostic transaction capability to handle the operations of:

- Create
- Read
- Update
- Delete
- And Query on the UPC domain model objects

  - **A Create operation** on a UPC domain model object results in persisting the object in the UPC database as its base function. Since the UPC domain model is hierarchical and includes all child objects at any level of the hierarchy, the Create takes in a hierarchical payload irrespective of what the payload actually

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

means. The transaction is synchronous, stateless and allows implicit linking of child objects as a part of creation of an object because of the hierarchy. Create returns the identifier of the hierarchical (top level of hierarchy) object.

- **Update and Delete** operations uses the unique identifiers or primary keys of the created objects to cause modifications or removal of an existing object.
- **The Read** returns an object and its linked child objects given its identifier. Because of hierarchical representation, the Read can be done iteratively by a client application to retrieve child objects of an object.
- **The Query** is broader than a read operation and it addresses the ability to retrieve attributes of one object, given a filter criteria on another object in the logical data model object hierarchy.

Use Case - Data Management (Cross Application Platform)

Path: Use Case View > UPC Actors and Interfaces > Use Case - Data Management

The data management supports the different forms of UPC persistence. Data management allows persistence of UPC data across not only the local UPC database, but also other external systems via propagation and aggregation.

- **Aggregation** is the method of collecting UPC domain objects like:
  - Product
  - Offering
  - Resource definitions in external systems like:

    Billing or Product catalogues

    And copying them into the UPC database in order to allow UPC as the central system of creating Offerings across different Products

This normally involves a translation of the data representation between the source system and UPC as well as a means to use the native protocol of the source system to collect the data from it. The retrieved data once converted to UPC logical data model are provisioned into UPC via the UPC data service (SOAP / XML).

Aggregation keeps the source system's data into UPC synchronized and therefore should be done periodically by polling. Since UPC data is about:

- The Products and Offerings
- The quantum and frequency of change is fairly minimal unless the aggregation involves 3rd party content systems

If the source system allows subscribing for change notifications, then aggregation can use that to pull the data whenever a data change happens on the source system. Aggregation may also involve a process flow before the retrieved objects are persisted into UPC.

Multiple legacy systems that are used for defining Products and creating Offerings are the target of Aggregation.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Aggregation initially may involve:

- Extract
- Transform
- And bulk load into UPC

If UPC becomes the central point of provisioning and provisioning on these source systems are halted, then UPC becomes the master and propagation to these systems are required further. However, if provisioning continues on these systems, then the aggregation should be done periodically to synch up into UPC. It is unlikely that provisioning should continue on these systems as the purpose of UPC is to centralize the creation of Products and Offerings across lines of business and not keep it fragmented in silos.

- **Propagation** is the means by which UPC synchronizes the changes to its domain objects stored in the UPC database into external systems. The change notifications in UPC can be subscribed to by external systems directly or by propagators which essentially transform UPC data models to target system data models and use the native protocols of the target systems to propagate. Propagation should be instantaneous as the change happens in UPC database in order to allow Order processing to be able to handle subscription to the new Offerings.

Use Case - Manage Rules (Cross Application Platform)

Path: Use Case View > UPC Actors and Interfaces > Use Case - Manage Rules

Rules allow expressing complex conditions and actions that keep changing due to business policy changes. The conditions of the rules are typically defined using the UPC domain model objects or data points in external systems such as analytics, subscriber data in CLM or a Billing system.

UPC specific rules can be defined against the attributes of a UPC transaction such as:

- Operation (CRUDQ)
- The state of the transaction
- The tenancy
- And the request or response payload which may refer to one or more UPC domain model objects

The rules get triggered when interactions occur between entities in the operator's Service Delivery Framework such as:

- An order fulfilment
- A self-care browsing
- A product catalogue
- Or recommendations for Product
- Or Offerings or even it can be rules triggered due to UPC specific operations like defining a Product

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

- Or an Offering
- Or even to handle UPC lifecycle events

The rule evaluation and enforcement that cuts across entities in a SDF is accomplished by means of distributed rule processing. The scope of UPC rules are limited to the definition of rules based on event triggers that arise within UPC. The evaluation of the rules and enforcement of the actions are only concerned with the UPC events.

Broadly, the following different types of rules can be defined on UPC domain model objects. These are limited categories of rules and in practice, there can be more categories:

- **Relationship rules**

  This addresses the complex relationships between the UPC domain model objects. These are normally classified as Exclusivity, Substitution, Dependency, Aggregation.

- **Eligibility rules**

  These rules define the eligibility based on geography or a market segment or other customer information.

- **Pricing rules**

These rules allow variations in pricing of the involving customer's existing subscriptions or usage or other data points like market segment, geography etc.

- **Recommendation rules**

These rules define the conditions that result in recommendations on the UPC products and offerings, given a context on the user demographics or other attributes.

UPC rules can also be triggered by external events and the action of a rule can influence external events. These external events can be from OM, CLM or Billing or other systems across the ESB.

Use case - Life Cycle Management

Path: Use Case View > UPC Actors and Interfaces > Use Case - Life Cycle Management

Life cycle management of UPC is about maintaining the concept of different states of the Products and Offerings (in general the UPC domain model objects) from the time a Product definition is created to its launch and retirement.

The states bring in certain rules about how it affects what is possible by whom in general on these domain model objects.

> For example, a Product in an Active state may not be removed unless it is approved by an authority. Or there could be a policy that a Product when removed will result in invalidating all the Bundles and Offerings on that Product automatically. The action on a life-cycle state change may thus vary depending on customer need.

Life-cycle states are affected when a CRUDQ operation on a UPC domain model object is performed. The life cycle state change can be automatic or manual.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Use Case - Versioning

Path: Use Case View > UPC Actors and Interfaces > Use Case - Versioning

Every key object in the UPC domain model carries a version to allow the possibility of evolving the definitions without having to create an altogether new definition.

When an Update is done on the following objects:

- On a UPC domain model object
- Depending on the object
- The attributes modified
- And the lifecycle state of that object
- The update may result in creating a new version of that object

This is done automatically as a part of processing a CRUDQ transaction. A manual override to create a new version can be done by client applications. In this case, the automatic versioning becomes disabled.

Use Case -Multi-tenancy and Data Model Extensibility (Cross Application Platform)

Path: Use Case View > UPC Actors and Interfaces > Use Case - Multi-tenancy and Data Extensibility

For more details about Multi-tenancy and Data Model extensibility, see ref.**Error! Reference source not found. Error! Reference source not found..**

Use Case - OAM

Path: Use Case View > UPC Actors and Interfaces > Use Case - OAM

The OAM allows administrators to monitor the state of UPC as a system in terms of the transaction statistics, faults and performance. On a fault, a procedure is followed by the administrator to bring back UPC to normal operation.

This can be sometimes:

- Manual
- Automatic or semi-automatic depending on the type of fault and the implementation

The OAM guide should detail method of procedures (MOP) for each fault scenario.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Use Case - Authentication and Authorization

Path: Use Case View > UPC Actors and Interfaces > Use Case - Authentication and Authorization

For more details about Authentication and Authorization, see ref. **Error! Reference source not found. Error! Reference source not found.**

From a UPC standpoint, clients accessing the UPC web service API is already authenticated and verified for authorization in terms of invoking a UPC web service. Also, a successful authentication results in creating a transaction context in which the authentication details, entitlements and tenant identifiers are made available for the rest of the UPC transaction processing.

Use Case - Indexing

Path: Use Case View > UPC Actors and Interfaces > Use Case - Indexing

Indexing is done on UPC data model attributes to allow a search on those attributes.

This will typically operate at the level of the UPC key objects like:

- Offering
- Bundle
- Product
- Resources etc

This means when a search is performed on these key objects with a filter criteria involving one or more attributes of the key objects, all matched objects in UPC database are returned based on the indexes.

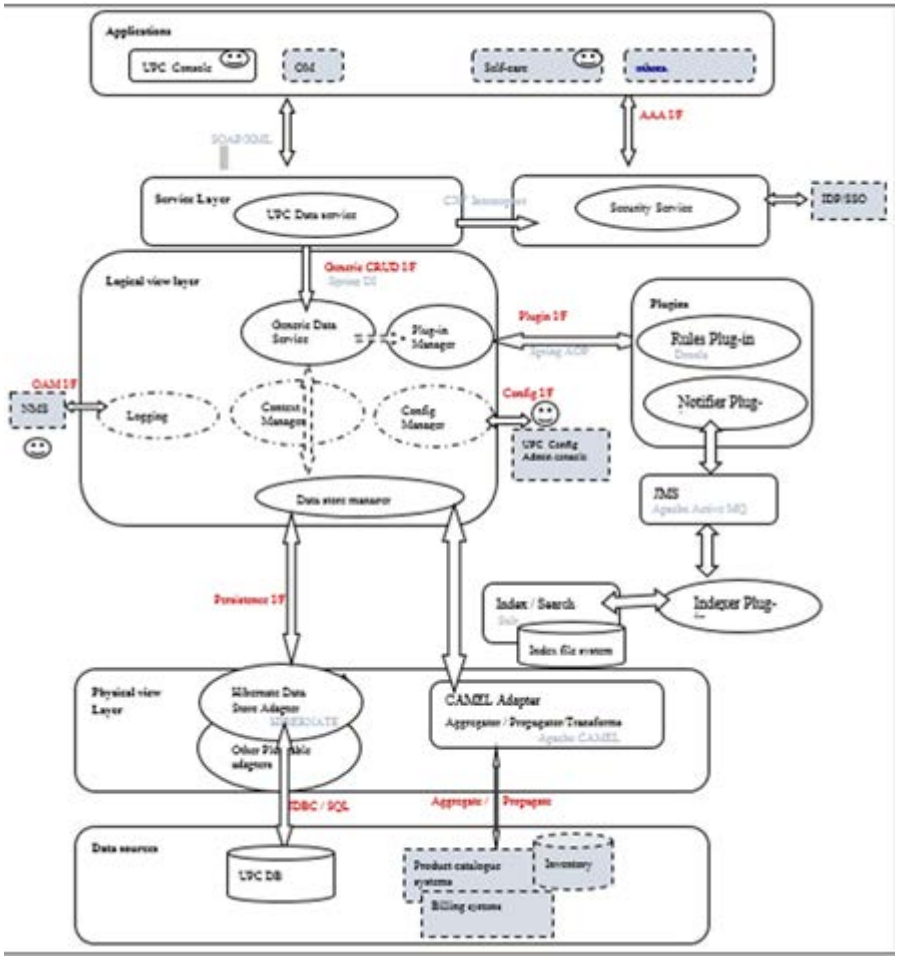
©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Functional View

Path: Functional View

Functional view defines the architectural elements that express/delivers the system's functionality. It defines processes to control and manage system behavior, such as monitoring, and other elements that are part of describing the functional behavior of the system.

The following screenshot describes the structure of the Functional View:



Architecture Description

Path: Functional View > Architecture Description

Architecture is the art and technique of designing and building, as distinguished from the skill associated with construction.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

The UPC architecture at its core is horizontally layered into 3 parts:

- Service layer - The CRUDQ based API exposure of UPC logical data model
- The Logical view layer- The aspects of handling generic CRUDQ transactions that results in persistence (Aggregation, Propagation, UPC persistence)
- The physical view layer - The adapters to UPC local database and to external sources

1. **The Service layer** allows external applications like OM, CLM, Self-care or a UPC console to be able to operate on UPC data. The API is exposed in two parts.
   - The externally visible part of the API is the UPC data service which has UPC specific operations like Create Product(), Update Product() and so on allowing UPC specific domain model objects to be exchanged as a part of a CRUDQ operations.
   - The internal part of this is a Generic CRUDQ transaction processing mechanism that is agnostic of the type of data that is passed in.

   This brings a key aspect of keeping the CRUDQ transaction processing independent of the data being handled and thus allowing extensions to the UPC data model. The externally visible part that is specific to data model is maintained as a very thin wrapper mainly to allow simplicity by being specific to UPC operations for a calling application.

   The API layer which is exposed as a web service (SOAP/XML) can be adapted to expose via REST in future. Every invocation to the UPC data service is intercepted to perform AAA functions using the security service mainly to validate the authentication tokens presented by clients and enforce access control policies like allow/deny the requestor for the operations invoked by client applications.

2. **The Logical view layer** allows handling the CRUDQ transactions on the logical data model objects using the Generic CRUDQ data service. The transactions ultimately result in persistence of different kind of UPC data model objects (Product, Offering) across one or more data sources including the UPC data store covering aspects of Aggregation and Propagation.

   Details of every transaction are maintained in a context by the context manager component. The context is created at the beginning of a CRUDQ transaction and is removed when the transaction is completed. During this period, every component in the transaction path, uses and updates transaction related details like the request payload, response payload, transaction state, authentication credentials etc. The context management reflects the single point in the architecture where the details of all running transactions can be obtained including the state of their processing. These are statistics for performance of the application that are logged periodically and monitored by external NMS.

   The transaction flow can be intercepted at different points of processing by Pluggable java logic that implement UPC specific aspects like lifecycle, versioning etc. The plug-in mechanism allows interception of a transaction by plug-ins that performs application specific processing. The context can be used by the plug-ins to affect a transaction and can even impact the control flow of a transaction. A plug-in may decide to abort a transaction based on certain conditions or may trigger workflows as a part of an interception or it may allow the transaction to continue with or without altering the context.

   At any point in time, the plug-ins that are interested in handling specific states of the transaction are loaded dynamically by the Plugin Manager component and the context is exposed to them apart from the control of the transaction itself. All the plug-ins implements a plug-in interface to route the transactions to the plug-ins.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Rule creation and evaluation feature are done as plug-ins in the transaction processing flow. Rules can be defined on different states of a CRUDQ transaction, on different attributes of the UPC transaction context and enforced via the plug-in invocation during the transaction processing. The action may result in continuing the transaction or aborting it and in the process may result in updating the context object. There can be cases where the outcome of the rule execution can also be published as messages into a message queue for a consumer.

UPC database persistence events are notified via Notifier plug-ins into message queues for consumption by propagators that push UPC data into external data sources.

UPC specific processing like Versioning, Lifecycle management are ideal candidates to be enforced via rules as they can be differently implemented for different customers.

Search on UPC data is issued on the UPC logical data model XML. Whenever a UPC data model object is persisted successfully, it is also indexed by Solr (Lucene) and linked into the appropriate bucket that contains its associated parent or child objects.The data model extensibility results in the ability to extend existing UPC data model of a customer with new attributes or objects or relationships by configuration. The UPC data model is implemented in the physical database (as a relational model in a relational store) and they are mapped via ORM and finally into the logical schema based on XSD which is exposed via the service layer. The changes to either the logical XML schemas or the physical tables may thus result in regeneration of the ORM configuration. This will also result in automatic regeneration of the UPC data service API to include the new operations and data models.

The data management component enables identifying the correct data store for persistence when there are more than one data store by configuration. This can happen in case there are multiple tenants whose persistent stores are different. The data management component also handles aspects like aggregation and propagation of data between UPC and external systems. Based on configuration, it can trigger the different external systems for synching specific UPC data model objects. It performs keep-alive pings of the configured external sources and also performs scheduling of the different aggregators based on configuration. It notifies into message queues configured for specific propagators when there are changes to UPC data and it also listens to notifications or polls external sources as part of an aggregation to populate the UPC database. The Config Manager component allows tenant specific configurations of the UPC schema, the UPC data stores, the aggregators, propagators etc. By keeping all the configurations managed from one place, multi-tenancy configurations can be localized to a single point and other components can be agnostic to it just by implementing the function they are meant for.

3. **The physical view layer** implements the adapters for persistence, propagation and aggregation.

   The Camel adapter can be configured for new aggregation or propagation schemes between UPC and external sources. The Camel adapter uses configuration to achieve the aggregation and propagation schemes with external data sources.

   The Hibernate adapter allows the incoming UPC XML payload via the Generic CRUDQ transaction processing to be automatically persisted into relational databases. The Hibernate adapter is generic enough to handle XSD of any kind for persistence.

   The physical layer thus allows databases of different kinds or even different technologies to be added to UPC.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Technologies Used

Path: Functional View > Technologies Used

Spring is used to wire the components of the logical view layer using dependency injection. Apache CXF is used to expose the SOAP/XML based

- Web service API to UPC client applications
- And its interceptors allow authentication
- And authorization of the requests into UPC to be handled by separate AAA service

Spring AOP is used to intercept the transaction flows for processing by plug-ins. Hibernate is used as the ORM mapping mechanism for persistence of UPC data. A pluggable hibernate adapter performs the conversion between the logical XML schemas and the object models. Apache Camel with Active MQ is used to handle the propagation and aggregation between UPC and external systems. Solr is used to index the attributes of the logical data model that are exposed for a search. JBOSS Drools is used for rule definition and evaluation.

Architecture - External Interfaces and Configurations

Path: Functional View > Architecture - External Interfaces and Configurations

External Interfaces are the connections between a software system and external entities. These entities can br other software systems. APIs, web services, hardware devices, databases etc.

For more information refer to these sections:

UPC CRUD I/F – SOAP/XML Based Web Service Interface

Generic CRUDQ Interface

UPC Data Management Component & Configurability

Persistence Interface for UPC Data Stores

Notifications from UPC into Message Queue

Configuration Administration Interface

OAM Interface

Plug-in Interface

UPC CRUD I/F – SOAP/XML Based Web Service Interface

Path: Functional View > Architecture - External Interfaces and Configurations > UPC CRUD I/F – SOAP/XML Based Web Service Interface

This is the northbound CRUDQ API exposure to client applications. The API is very specific to UPC.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

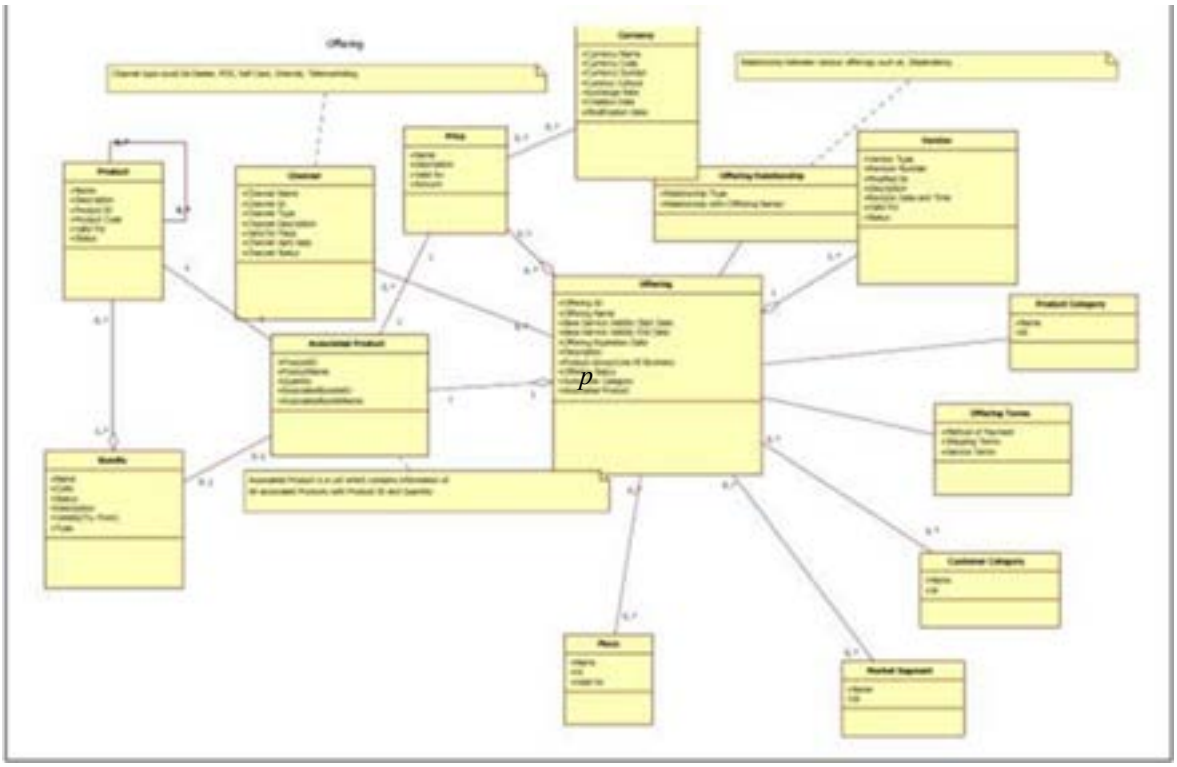
The APIs handle hierarchical XML structures as part of the CRUDQ operations. The APIs of the UPC DS exposes CRUDQ on key objects of the UPC logical data model. Each UPC Web service method accepts the XML payload having the data to be created as a part of the CRUDxxx operation and a version parameter that is passed in to identify the version of the UPC logical data model schema (XSD) that needs to be used by UPC to carry out the CRUDxxx operation.

UPC Logical Data Model

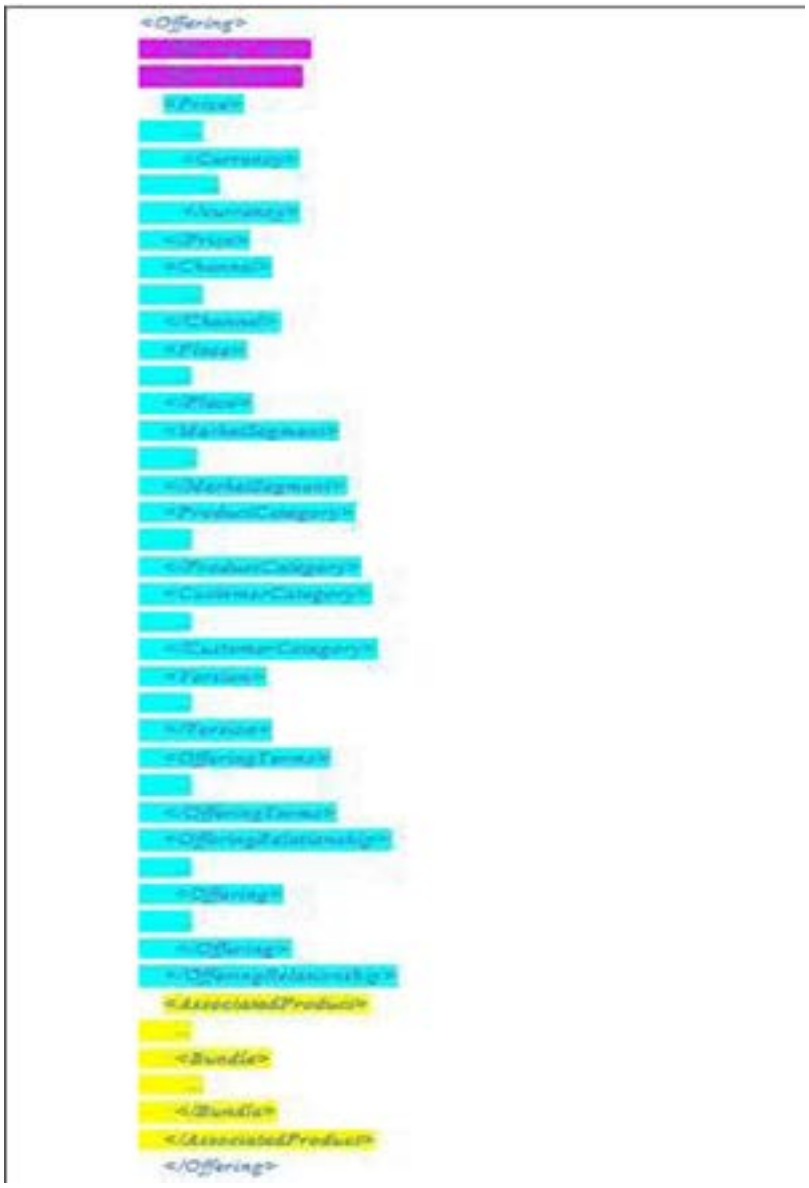
Path: Functional View > Architecture - External Interfaces and Configurations >UPC Logical Data Model

UPC Logical Data Model establishes the structure of the data, elements like price, market segment and the relationship between them.

The following screenshot describes the structure of the UPC Logical Data Model:



For example, the Figure 1 above shows an object model for Offering. This can be represented in XML as follows



The XML in Figure 2 contains child complex elements like Price, Market Segment etc. In this case, when a Create or Update is done on a Offering, the child element of Bundle will have only the identifiers which have been created earlier by a Create Bundle () while the XML content for the child elements like Price or Market Segment or Place have to be filled in as part of a Create of the Offering object.

Linkage between a parent and child object is achieved by using the object identifiers (or primary keys) in the payload of the child object as against specifying the content of the child object in a create or an update. For linking, the pre- requisite is that the child elements should be existing before the parent element is created. An Update operation can be used to add or remove linkages to child objects subsequently.

During a Create or Update, the mandatory elements are enforced by the UPC DS web service as it exposes specific XSD of the UPC data model and exceptions are thrown in case of mandatory fields or elements not being present in the payload. Beyond this, the UPC DS uses the Generic DS component to fulfil the CRUDQ operation.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

- In case of a Create operation, the XML payload passed will represent a hierarchy with the root of the XML starting from an Offering or a Product or a Bundle or a Resource or others. Thus an Offering may contain the child elements (simple) of the Offering like Offering Name or Offering Description as well as the direct child objects (complex) of the Offering such as Market Segment or Product Category and the linked objects such as the Bundle. In case of linked objects only the identifiers of the objects will be present, while in case of direct child objects, the entire content will be present. If child elements of the root element or direct child objects are missing because they are optional, then those fields will be set to NULL or the rows for the missing child objects will not be present in the physical database.
- In case of an Update operation, the XML payload passed will represent a hierarchy with the root of the XML starting from an Offering or a Product or a Bundle or a Resource or others. The direct child elements or the child objects of the object hierarchy being updated that are missing are ignored in the Update. The linked child objects that are missing in the XML payload will be unlinked from the relationship and treated as if an 'unlink' command is specified. If new linked child objects are present, new relationships will be added to the persistence database.
- A Read operation of a key object with an identifier will return the direct child elements, direct child objects and linked child objects of the key object being read.
- A Query operation can be either a simple query involving a specific object and its attributes as filter criteria or a complex query where the filter criteria spans across the objects in the UPC data model and not confined to a specific object.

Semantics of the XML Payload for CRUD Operations

Path: Functional View > Architecture - External Interfaces and Configurations > Semantics of the XML Payload for CRUD Operations

Generic CRUDQ Interface

Path: Functional View > Architecture - External Interfaces and Configurations > Generic CRUDQ Interface

This interface handles the CRUDQ transaction processing in a data model independent manner. It takes in the XML payload as Strings and triggers a CRUDQ transaction flow. This is at present an internal interface to UPC invoked by the UPC data service web service.

UPC Data Management Component & Configurability

Path: Functional View > Architecture - External Interfaces and Configurations > UPC Data Management Component & Configurability

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

The data management component in the architecture allows pluggability of adapters to one or more data sources for persistence, propagation and aggregation of all the CRUD requests entering UPC. The behaviour of this interface can be controlled by configuration having the following parameters.

The configuration below essentially represents the different data store adapters, (UPCDB in this example) registers for any of the C.R.U.D operation for specific objects in the UPC logical data model (such as Product, Offering) to be either synchronously persisted (Asynch flag is false) or asynchronously published to a message queue for consumption by the Propagators.

The data management interface within UPC acts like a switch.

It takes the above configuration and for a given CRUD invocation into UPC finds all the data stores (adapters) which have been configured to be invoked for the CRUD operations and UPC logical data model objects.

It performs the operation on the identified data store adapters by dynamically loading and invoking the methods on them. In case of notifications of successful persistence, the asynch flag is set to true.

In this case, the payload upon successful persistence, is placed on message queues. The tenant identifier allows differentiating the persistence and notifications on a per tenant basis.

Persistence Interface for UPC Data Stores

Path: Functional View > Architecture - External Interfaces and Configurations > Persistence Interface for UPC Data Stores

Persistence Interface for UPC Data Stores means for an application to persist and retrive information from a non-volatile storage system.

UPC data can be persisted in one or more data stores.

- In case of a single tenant, the UPC data could be all housed in a single relational database. In case of a multi-tenant model, there could be different databases for different tenants.
- This interface is expected to be implemented by the adapters that interface with the UPC data stores. The interface is essentially a CRUDQ API that is expected to be implemented by each data store that houses UPC data. The data exchanged in this API is always a stringified XML representing the key objects of the UPC logical data model such as Offering or Product for example.
- The data store adapter takes in this XML, converts it to the formats of the underlying data store and does the persistence in case of a Create and Update.
- A Read or Delete contains the same input XML payload with only the primary keys of the objects. The response of a Create contains the primary keys of the objects created while a Read contains all the data of the object which is retrieved by the primary keys. The response of the Update and Delete is void.
- UPC invokes the data store adapters methods by Java reflection synchronously. If there are multiple data stores registered for persistence of the same UPC data model object, then the persistence among them can be either serialized or parallelized depending on the configuration.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

- It is possible that data store adapters implementing this interface can themselves be generic and reusable. For example, a generic data store implementation for all relational databases can be accomplished by a Hibernate Data store adapter that translates the UPC logical data model in XML into Hibernate APIs after doing get/set on the Java classes of Hibernate using dynamic class loading in a generic fashion agnostic of the XML schema.

Notifications from UPC into Message Queue

Path: Functional View > Architecture - External Interfaces and Configurations > Notifications from UPC into Message Queue

On successful persistence, the data can be further propagated to one or more external data sources by UPC by placing the details of a request into a message queue.

Configuration Administration Interface

Path: Functional View > Architecture - External Interfaces and Configurations > Configuration Administration Interface

This interface allows the configurability to extend UPC logical data models together with or without multi-tenancy configurations. This interface is via a UI (or direct XML file modifications) which allows an administrator who has undergone a UPC configuration training to make modifications. The UPC user guide specifies the configuration details to be done for data model extensions and to set up multi-tenant transaction processing.

This is a single point of configuration for multi-tenancy triggered data model extensions and aggregator, propagator configurations.

OAM Interface

Path: Functional View > Architecture - External Interfaces and Configurations > OAM Interface

This is to monitor the performance of the UPC application and faults reported during the operations. The interface is via log files mainly. The external NMS or application monitoring systems should use the log files which contains the periodic statistics, alarms dumped by UPC. Corrective actions should be done by the administrator following the OAM guide which contains the method of procedures for every fault/alarm.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Plug-in Interface

Path: Functional View > Architecture - External Interfaces and Configurations > Plug-in Interface

The plug-in interface allows extensibility of functionality of UPC beyond the regular CRUD transaction processing. This is a Java Reflection API to load the plug-ins whenever UPC transaction states are updated and invoke the plug-in interface method implemented by the plug-ins.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

UPC Implementation View

Path: UPC Implementation View

The purpose of the implementation view is to capture architectural decision made for the implementation.

Implementation Decisions

Path: UPC Implementation View > Implementation Decisions

Implementing decisions deals with specific issues and address the key requirement fo UPC like - Product, Resources, Concept of Creating an Offering etc.

UPC Transactions and Logical Data Models

Path: UPC Implementation View > Implementation Decisions > UPC Transactions and Logical data Models

A Logical data model is like a graphical representation of the information requirements of a business area. It is independent of any physical data storage device.

At the heart of UPC is the UPC logical data model. This captures the key objects such as:

- Offering
- Bundle
- Product
- Resources etc. closely aligned to the TMF SID model

Though every deployment may result in customizing some aspect of this domain model, which is a key requirement for UPC, the key objects will be present in some form or the other capturing the concepts of creating an Offering or a Product or Resources across the different lines of businesses, the functionality being what UPC is meant to represent.

The application level transactions in UPC are primarily the CRUDQ operations on the UPC domain model. Clients use HTTP/SOAP to exchange XML data representations of the UPC domain model objects. The XML data passed in are single rooted, hierarchical structures representing the key objects of the UPC domain model.

The XML exhibits two types of relationships:

- Aggregation
- Composition

For example, Figure 1: UPC logical data model - High level, the Product and Resources are related by a n..n aggregation relationship.

As an example of a domain model, an Offering may include a Bundle which in turn may have one or more Products where each Product may contain one or more Resources. Now, if an Offering has to be created, it can be visualized as filling up a form containing all these hierarchical elements of Bundle, Products and Resources

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

However, in practice, the process of creation of an Offering is realized by two steps:

- First step involves defining the Offering elements such as Price, SLA and so on. Second step involves linking the Offering to a Bundle which is already created and available.
- The same two-step process applies to create any of the UPC domain model objects such as a Product or a Resource

A Product is created with all the details having Product elements and linked to one or more Resources. While it may appear that a single hierarchical payload rooted at:

- Offering would be sufficient to represent the XML that is exchanged with UPC
- It may not look correct to create a Product which can stand independent of an Offering (because of the aggregation relationship rather than composition)
- To be contained in a payload with a root of an Offering

It would be ideal to allow Products to be created with XML structures that are rooted with a Product element. Also, different roles can create different things.

For example, a network operations engineer would have the skill to define the Resources to realize a Product. It would be appropriate if only the Resources are allowed to be seen in the XMLs that this person is dealing with rather than exchanging Resources under the garb of an always uniform document rooted at Offering. Thus when multiple points in the data model can be created independently, the need to link them with their child objects which exist independently is allowed by having the child object identifiers. As an example, the creation of a Product will send a XML as follows:

The following screenshot describes the structure of the UPC Transactions and Logical Data Model:

```
<Product>
</ProductCode>
```

```
</ProductDesc>
<Price>
            </PriceType>
            ...
</Price>
<Resources>
            </ResourceID>
</Resources>
</Product>
```

The above XML describes both the simple and complex elements of a Product like ProductCode, ProductDesc and Price while it specifies the linkage elements of Resources with their identifiers.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

The key objects and their relationships are always seen as a single integral atomic representation. Thus the client applications when updating parts of a key object which is already created should ensure the changes submitted are with respect to the entire object, its child objects and all its linkages. Allowing separate Link / Unlink commands as a part of Update of a key object is seen to result in complicating the UPC application level transaction model involving handling failures and rollbacks making the transaction model stateful and more complex in terms of implementation. Alternatively, by defining the semantics of having an entire key object and its relationships as a single payload, the transaction processing is moved to the database level. This keeps the application level transactions stateless and simple.

The main point here is about simplicity of representation and stateless behaviour of the CRUDQ transaction model.

Absence of Session Context

- There is no notion of a session context maintained when a client invokes a CRUDQ operation within the UPC application. This keeps fail-over of requests from client from one UPC node to another in a cluster in a more seamless manner without the need to maintain distributed session context. Thus in this sense, every CRUDQ operation can be replayed by clients on a failure of a UPC node which will be handled by another UPC node that is active in the cluster. The UPC CRUDQ transaction model is thus simple and does not entail any complex orchestration that requires a roll-back and hence the notion of maintaining a state of execution.
- A fall out of the lack of session maintenance can be a case where a Product created by a client via the SOAP / XML CRUDQ API that result in persistence in UPC database upon failure of that node, results in a possibility of the client retrying and thus resulting in again creating a duplicate record. Had a session context been maintained across a distributed UPC cluster, the re-request from the same client would result in being handled on a replicated session context which can potentially have the identifier of the object created in the original request (though this may still have a very small window of vulnerability, wherein the node crashed before the identifier was checkpointed into the session context).
- The proposed solution to handle this is to do an MD5 checksum of the XML payload (or DB record) and persist it along with the newly created record. MD5 is a 32 byte hash and hence the possibility of two records colliding even though the contents are different will be very rare. While the same record will produce the same Hash key.
- If there is a collision on records of slightly different values, then a 'Record compare' shall be done to ensure they are indeed different records. This will help to keep the comparison to run on very few records as MD5 32 byte is a extremely large key space when compared to the number of 'Products' or Resources or Offerings that will be ever created in the life of UPC and the chances of collisions are extremely low or none.

Logical Data Models And Mapping to Physical Data Models

The logical data models in UPC are expressed as XSDs. Mapping each of them to underlying physical representation involving relational tables and columns are done in a way the UPC transaction processing or the persistence logic are not tied to processing specific XSDs or table/column representations.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

The mapping itself is configurable and handled as a part of hibernate configuration apart from UPC specific configuration. The hibernate adapter of UPC uses this configuration to map the XML elements to the Hibernate java classes and properties that in turn maps to the physical tables and columns in a relational database. This does not mean that the product is tied to Hibernate. Hibernate's ORM mapping and its ability to handle transactions to relational databases has been leveraged to reuse what is available with open source. However, the UPC product architecture is flexible to have any underlying.

Representation of physical structures as long as an 'adapter' that can process the UPC logical data model XMLs for the CRUD operations are made available. The Hibernate adapter does a generic transformation of a XML to a ORM representation which is persisted into relational tables on a Create, Update and Delete operation and vice-versa on a Read operation.

UPC Specific Behaviours and AOP

The core CRUD transaction processing within UPC is kept independent of data models. However, specific behaviours like versioning and handling lifecycle states of the UPC data model objects or application of Rules and workflows to the processing of the UPC transactions etc. are handled in specific logics that act on the CRUD transaction processing states by using interceptors based on Spring AOP. This model allows extensibility of specific behaviours to the Product by plugging components on a need basis rather than implementing a monolithic block of UPC specific logic.

A reusable AAA service isolated from UPC transaction processing

The authentication and authorization aspects of the requesting application have been completely isolated into an independent, reusable web service as a SOA entity. This function will handle the aspects of connecting to an IDP, issuing signed authentication tokens, retrieval and enforcement of policy decisions on a request from a client application, token expiry and re-authentication and so on. The requests hitting UPC are siphoned off via interceptors at the web service level to perform these tasks and keep UPC application to focus on CRUD transaction processing.

UPC Component Architecture Description

The description of UPC Component Architecture contains the structure of an information system. It focuses on the features, transactions, lifecycle management and versioning, rules, data management etc.

For more information:

UPC Transaction Processing Components

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Lifecycle Management and Versioning

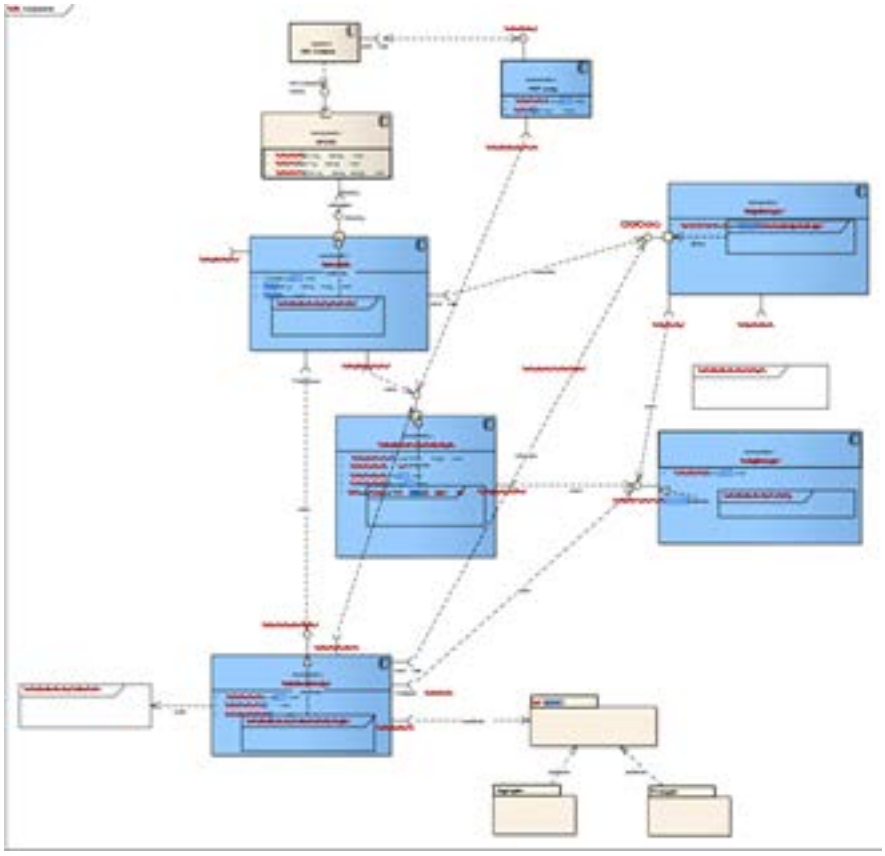
Plug-ins for Filtering Of Response Payload

Plug-ins for Auto-filling Of The Request Payload

UPC Rules Definition and Enforcement

UPC Data Management

UPC Search Function



UPC Transaction Processing Components

The following components handle the CRUDQ transactions within the UPC architecture.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Generic DS

Exposes generic create, read, update, delete and query methods to its caller. This component is invoked primarily by the UPC DS which implements the web service wrapper. This component is agnostic of specific data types and mainly takes in XML in the form of strings. It is more of an orchestrator in implementing a set of interactions for each transaction. It does the following:

- Create a Transaction context or update it if it is already made available. Fill up the context with the details of the request.
- Invoke configurable plug-in to process on every transaction state change. These include UPC specific plugins for Lifecycle, Versioning, Rules enforcement etc.
- Invoke DataStoreManager for persistence to one or more underlying data stores.
- On completion, invoke configured plug-ins for any post-processing
- Return the response to the caller.

Config. Manager

This component is a holder of all the configuration items for all other components. Every component maintains its configuration against the component name, with one or more configuration sections and configurable parameters under each section. The configurations are XML files. The DataStoreManager, DataStore adapters, PluginManager, Plug-ins maintain their configuration with this component.

The key function delivered by this component apart from the reusable common configuration is the abstraction of Multi-tenancy. The calling components automatically get the correct configuration based on the tenant identifiers associated with a transaction. This enables a single point of configuring multi-tenancy aspects.

Plug-in Manager

The Plugin Manager acts on the normal CRUD transaction flows by intercepting them and invokes one or more plug-ins that process on specific transaction states (pre or post- processing) , for any of the C.R.U.D transaction(s) and result in updating the transaction context and may control the flow by aborting or continuing the transaction.

Transaction Context Manager

The UPC application transaction context for every CRUDQ operation is maintained during the life of a transaction as a place for all the interacting components to share transaction specific information.

Lifecycle Management and Versioning

The states that a Product or an Offering or other key objects in the UPC logical data model may go through are listed in use case UPC 1.1.2.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

The diagram in Figure 4 states the default behaviour on lifecycle state change for any key object in the UPC logical data model as it is created, updated and frozen.

In_Progress

When a UPC logical data model object is created it enters the In_Progress state. Every object in the UPC logical data model is uniquely identified by a combination of version and an object identifier. The Creation of a new object always starts with a version 1.0. Further updates on the same object will not cause a version change unless the end user over-rides the version by specifying it via the SOAP API.

In case a version over-ride is specified during an Update, the UPC Plug-in increments the current version and modifies the Update command to a Create command.

A Create command with a flag specified as a 'clone' will result in the UPC plug-in incrementing the version for the given root element as well as the linked child elements., the linked elements will only contain the identifiers of the object and not other attributes.

Thus the 'clone' flag should be taken as a special form of Create which will result in creating incremented versions of the object hierarchy that automatically fills up the attributes of the linked objects of the object to be created. This will be controlled by an option which specifies if new versions of the linked objects need to be created or not. The clone flag is applicable only on objects which are in an Active state or a Retired state.

The idea is to allow a new evolution path for an Active or Retired Product or Offerings by replicating it entirely and re-starting the whole lifecycle process for it. The lifecycle state of a cloned object is set to 'In_Progress'. As a part of editing this cloned object, new versions can be created during the editing process.

To summarize the above functional behaviour:

- Versioning is done automatically with a starting version number for an object that is newly created.
- Update on an existing object will not result in a new verison. It just modifies the contents of the object of an existing version.
- Update on an existing object can be done with a version override which results in creating a new version of the object with the given changes as is. In this case, the UPC plug-in modifies the operation from Update to Create after incrementing the version.
- Clone of an existing object in Active or Retired state is possible and it will result in creating a new version of the object.
- The cloning can result in copying the link objects if specified by a flag or may result in creating a new version of the object with its existing linkages. In case the clone flag is set, the UPC plug-in may have to verify if the current state is Active or Retired and throw and error if not.

When an object is rejected based on an approval decision, it can re-enter the In_Progress state to rectify any flaws identified during the approval process. As a part of edit, new versions of this object can be created.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

When an object has reached the Active state, due to reasons that are technical or otherwise, the object and its linkages if it exists can be suspended from being offered and further may re- enter the In_Progress state to do edits and re-do the approval process before becoming Active. In this case, the primary purpose is to re-edit the existing version for any flaws and make it available as before. Hence, creation of new versions on this object is not allowed. The UPC plug-in on an Update with a version over-ride will check the current state of the object. If it is in 'suspend' state, then it will disallow version increments and set the state to In_Progress.

When an object is In_Progress state, it does not show up for any linkages with other Active objects. A Search on this type of object will only return objects in the Active state. If there are multiple Active versions of an object, all the versions will show up. This depends on the role of the requestor.

A requestor with a Product Manager role may be able to see all the objects in whatever state they are in. While a decomposition request from OM would be able to see only the Active objects in the system as a part of the decomposition. However, UPC as a system will not allow a Create or an Update having linked UPC objects which are not in an Active state.

The implementation of this can be handled by the post-processing interception of the QUERY transaction by the UPC plug-in by filtering out the results which contain objects in state other than 'Active'. Also, the same can be used to perform the role based checks and filtering based on the entitlement information in the transaction context.

Pending_Approval

- Once an edit on an object is complete, the state is changed to Pending_Approval. This may result in triggering any workflows for approvals. In this state, no edits are possible on an object until it is moved to an Approved or Rejected state.
- The implementation of triggering workflows for approval and modifying the state to Approved or Rejected automatically is very specific to a customer deployment. There is also a possibility of a manual state update to Approved or Rejected on the object via an Update operation.
- The UPC Plug-in can be configured to do an automatic or a manual approval.
- In case of automatic approvals based on workflows, the plug-in may have to listen for such events and trigger an update on the object to move the state to Approved or Rejected.

Approved

- An approval process may result in an 'Approved' message on the message bus in case of automatic approval process flows (or a manual SOAP API Update operation) which results in triggering UPC to modify the state of the object from Pending_Approval to Approved.
- Once in Approved state, any flaws identified post approval will result in returning the state to Pending_Approval which can Reject the approval and take it to In_Progress for fixing the flaws. All Approved objects if they are going to be launched will enter an Inactive state before making it Active or in some cases may be configured to enter an Active state automatically after an elapsed time.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

- In case, an end-to-end validation needs to be done, then the object can be moved to Validate_For_Launch state. In this state, it is possible to link this object with other key UPC objects which are also in the Validate_For_Launch state.

Rejected State

This state is entered when an Approval process rejected the changes which can be further moved to a Cancelled state in case of no changes would be pursued or to an In_Progress state for further edits.

Cancellation State

This state is entered when an object which was rejected by the approval process is no longer to be pursued and hence moved to a cancelled state equivalent to an abort.

Inactive State

This state is just an intermediate state before an Approved object is made Active. No edits are possible in this state.

Active State

- This state makes an object visible for linkages with other key UPC objects and also makes it visible for any search or read requests from OM as a part of decomposition or CLM or others.
- While in Active state, an object along with its linked hierarchy can be suspended for technical or other reasons.
- An object in this state can expire automatically, provided it is not linked to any other key UPC object. From Active state, an object can be manually retired provided it is not linked to any other key UPC object. An object can also be cloned which creates a new version of this and its linked objects if there is a need to evolve a new version of an object which is active. The cloned object starts with the In_Progress state again.
- Thus edits on the same object is possible by Suspension -> In_Progress route or edits on a new version of the object is possible by Cloning.

Retire State

The Retire state is done to retire an Active object which is no more linked to other key UPC objects and no more in use. An object in this state can be moved to Archived state. From a Retire state, an object can be cloned and a new version of it and linkages can be re-edited to evolve it.

Expiry State

An object comes to this state if there is an automatic expiry set on it when it was activated. No linkages should exist for an Active object to enter this state. From an Expiry state, an object can be cloned and a new version of it and linkages can be re-edited to evolve it.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Suspend State

In this state, the object which was Active is suspended to fix some critical flaws identified. From a Suspend state, the state is modified to In_Progress to again go through the approval process. Suspension of an object will cause all the linked UPC objects also to be suspended.

Validate_For_Launch State

This state is used to test an object before it can be launched. This state allows, other objects in this state to be linked with one another for test purposes. While a object in this state cannot be linked to other key UPC objects which are not in a Validate_For_Launch state. If corrections are needed during the testing, the state can be moved to Rejected from where it can be taken to In_Progress and edited to re-enter the Pending_Approval process.

If in this state, the testing is found satisfactory, the object can be moved to Active state.

Archive State

In this state, an object is archived for historical purposes.

Architecture Notes

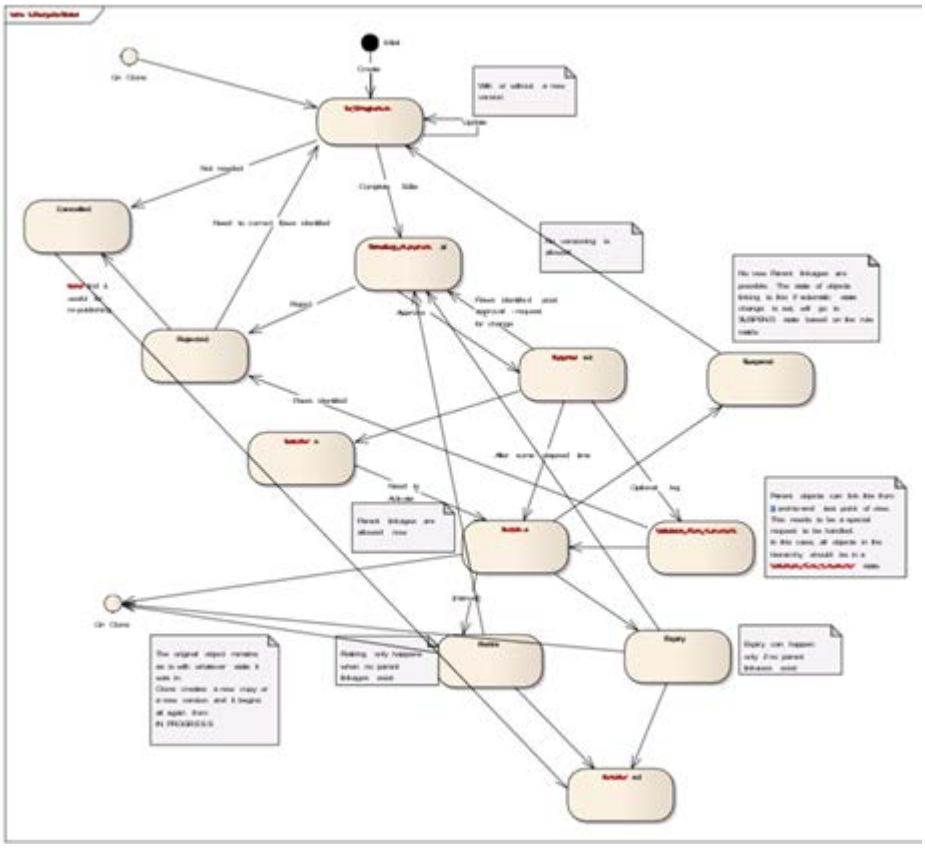
The UPC plug-in mainly validates the state transitions done by the end user via the SOAP API of UPC DS are correct in terms of the ingress and egress. The UPC plug-in is invoked via the PluginManager as a interception on the CRUD transactions some before the transaction begins (pre-process) and some after the transaction is completed (post-process). Invalid state transitions are returned with exceptions on the CRUD with UPC_INVALID_STATE exception.

The UPC plug-in may trigger workflows, may listen on events and in turn trigger updates or reads into the UPC database.

See figure 9.3.4 that shows the invocations of the UPC plug-in for the lifecycle and versioning management.

The recommended choice for implementing the UPC object lifecycle management and versioning is via the Rules to allow customizations in both single and multi-tenant models.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template



Plug-ins for Filtering Of Response Payload

In many situations it is required to filter out the elements in a response of a Read operation on UPC logical data model objects. This may be to limit the exposure of data based on the roles of the requesting application (which is available from the authentication information in the transaction context) or it could be a case of some fields like lifecycle management or rules that are attached to objects which are used by the UPC from being exposed to an end user. In these cases, a post-processing plug-in may operate on the TRANS_COMPLETE states of a transaction and result in filtering out the elements based on configuration and the roles of the requestor.

Plug-ins for Auto-filling Of The Request Payload

Similar to filtering plug-ins on response payload as described above, there is a need to automatically fill up fields which are not appropriate to be filled by end users. This can be fields that denote the requestors who created an object, the date and timestamp etc.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

UPC Rules Definition and Enforcement

Rules are needed in many situations within UPC. Broadly they fall into two categories

Rules that Influence UPC Functions

These are configurable rules that are set mostly one time in a deployment. These types of rules are defined by a system administrator based on engineering's advice and it will be triggered every time on a UPC CRUD transaction states. The context for the rule evaluation comes from the UPC data model objects or the request payload in the transaction context.

Examples of these rules are:

- A rule which says " If the Lifecycle state is Pending_Approval, then if there are certain properties in the Offering or Product definition then it will be automatically rejected".
- A rule which says "Whenever aggregation is done from source X for a 'Product' definition, if the Product definition involves Line of business Y, then get it approved by person A before persisting into UPC.

Rules that are Tagged on the Events of UPC or External to UPC

In this case, when a Product owner is defining a Product or Offering or Resource, explicit rules can be attached to UPC events like a CRUD that causes these rules to be triggered whenever a CRUD happens or it can be specific to events like a CreateOffering or UpdateResource etc.

There can also be rules set on events that happen outside of UPC.

Example of a rule which is triggered on a UPC transaction

> "When creating a Bundle, there can be dependency, exclusivity, aggregation, substitution relationships between Products linked to that Bundle. These relationships can be simple in terms of product A depends on product B or can tend to have a more complex expression that involves product A with certain characteristics can be aggregated with (B AND C) while Product A with another set of characteristics can be aggregated with only B".

The above rule is defined on the 'schema' definition of Bundle and not specific logical data model. It will be triggered when a Create or an Update of a Bundle containing specific Product descriptions are linked to it.

Example of a rule which is triggered on events outside of UPC

> "An Offering on Product A will get 30 minutes of additional usage per month if the user purchases Product B within a certain time period"

In this case, the rule is defined by a product owner in UPC and tagged on the Offering object having Product B which is a logical data model instance within UPC. This is done by the Product manager who created the Offering on the Bundle having Product B.

The trigger for the rule is set to an external event which in this case is the Process flow within Order Management. This process flow is kicked in whenever a user purchases an Offering and OM sends a decomposition request for this Offering into UPC which results in retrieving this tagged rule. The context of the rule execution will be passed by OM to the rule engine. In this case, the context will involve the

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

- Product B' purchased by the end user
- Time of purchase
- Identity of the end user

The rule engine goes through its conditions which are:

- Assert 'Product code' is B
- Assert 'TimeOfPurchase' is between X and Y dates
- Assert User with UserIdentity has an Active subscription for Product A.

The first two items are part of the request processed by the OM flow while the last item is derived from the subscriptions in CLM for the given user identity.

Note that the Rule engine should have access to all the UPC transaction context and that is used in the conditions of the rules along with the knowledge base. Almost all of the context attributes are available in the transaction context object as part of a CRUD transaction while there may be some attributes which are derived by querying external systems or a cached version of the data within the system where the rule is getting processed.

The above rule also has an action to fulfil if all the assertions evaluate to true. In this case, the user with the given user identity is provisioned with an allowance of extra 30 minutes of usage for Product A within the billing system. This is the consequence of executing a rule and this can affect objects within the system executing the rule or outside of it. There can also be cases where a workflow may be triggered as a consequence. Also, it is possible the execution of a rule triggers a consequence which results in execution of other rules because of chained rules.

To summarize the requirements on the architecture for rule processing:

- Rules apply as a cross-cutting horizontal feature across UPC, OM, CLM, Billing, Provisioning systems.
- The trigger for a rule can be based on one or more events that occurred within any of the applications above. (In the above example, the trigger is an Order Management process flow for a new Order. While in the context of a CLM it could be handling a new user registration or in case of UPC, it could be a Product manager creating an offering or an end user browsing the product catalogue)
- The rule definition can happen on a application different from the application that ultimately executes the rule. (In the above example, the rule was defined in UPC as a part of creating an Offering, but the execution is done in the context of a purchase of that offering by a customer which is handled by OM)
- Execution of rule in one application can trigger the execution of rule in one or more other applications because of the events that were caused by the outcome of the execution rule in the first system. (The provisioning of an allowance within a Billing system in the example above could be extended to trigger a rule further which in turn checks for the 'credit history' of the user).
- It is required to have a single interface for rule definition that can be potentially reached by traversal from the system specific administrative GUI (for example, when a rule has to be tagged to a UPC Offering, clicking a URL allows migration to the Rule Authoring page).
- The rules of all systems are stored in a single repository which can be a database and can be versioned.
- The roles of the users allowed to define rules are validated against the Single sign-on AAA web service.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

- The list of trigger points across the applications (OM, CLM, and UPC, Billing ...) are named and are configured to be used in the rule definitions.
- The relevant domain objects (Schema and instances), context variables for all the applications and the events within them are made available during the definition of the rules.
- The relevant context or the knowledge for execution of the rules is made available during the execution of the rule.
- The events on which the rules are triggered are like states of operation within each application involved (OM, CLM, UPC CRUD states...). The interception to invoke rules meant for those states or events should be done by the concerned application. While this works well for new applications that are under construction and process flows that can be defined on the fly, legacy applications will have an issue of benefitting from it. If there are policies to be executed based on the events from a legacy application or influencing data into a legacy application because of the rule execution, it will be handled by specific solution integration.

The rules are defined from a single interface in the BRMS and stored in a repository. The events within the different applications as they process transactions or process flows are made available during the definition stage within the BRMS IDE where the rules are defined. The event definitions collectively represent the trigger points in each application on which one or more rules may be executed. If only a single application like UPC is deployed, then the events represent only that of the UPC. There can be a collection of Rules defined against each event. For example, with UPC, there can be rules for every event such as a Create, Read, Update, Delete or it can be specifically CreateOffering, UpdateProduct and so on. The rules are evolved and versioned within the BRMS rules repository. It is possible to search the repository for rules of a given name. BRMS uses Lucene to allow a search.

The BRMS rule engine is integrated with the applications (including UPC) via "Interception" as a plug-in. Thus whenever an application processes a state where the event occurs, the rules corresponding to the event are retrieved from the repository. In order to avoid transaction delays in loading rules, every application may periodically search the repository for rules that relate to events processed within that application and refresh them in an in-memory cache.

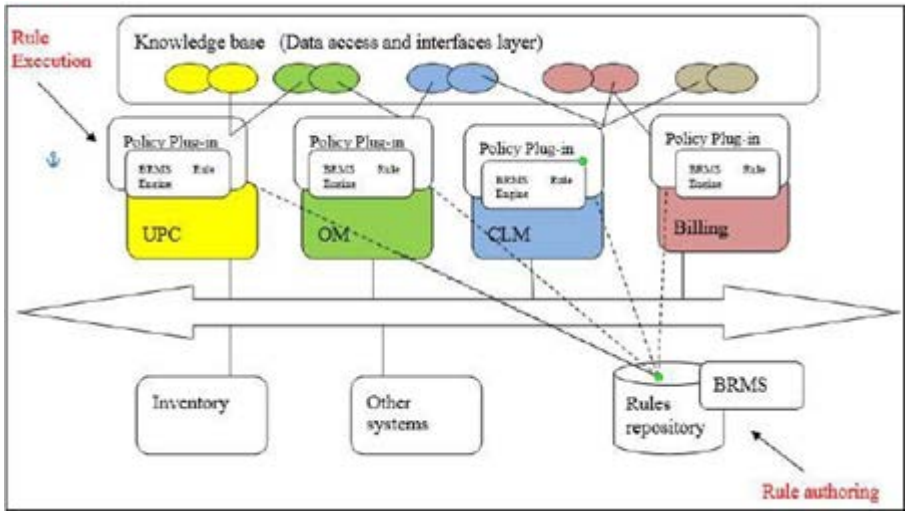
A 'Policy plug-in' component infuses the rule engine with the 'context' or ' knowledge base' required for processing the rules corresponding to that event. The context variables are purely determined by the conditional expressions in the rules that are defined. The policy plug-in component which infuses the context into the rule processing is able to interface locally with the application's domain objects and database and the external systems based on need. In order to make this part of the code less prone to changes, a list of standard 'interfaces' to the different source systems and the corresponding objects that are available as a 'knowledge' out of it for the rules that use them are made available. This should ideally cater to a large majority of the rules that can be defined and hence may not need any additional 'coding' to create a rule.

For example, if the rules don't use context beyond UPC, only the UPC related domain model objects are made available.

During run-time, whenever an application (OM, UPC and CLM) reaches a state where the trigger point event is meaningful, it invokes the rule engine and passes the context. The rules corresponding to the context execute and it may affect one or more applications in the process depending on the 'actions' specified in the rule. These may affect the context of the local application or external applications. These are again routed via the policy plug- in. There can also be additional rules triggered because of further events that are triggered by these rule processing.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

The Figure 4 shows the entities involved in the eco system of UPC, OM, CLM that are required to process the rules.



RedHat BRMS is recommended as the tool of choice for rule authoring and rule execution. The BRMS allows storing the rule files in a repository that can be version controlled.

- Logical Data model (Schema)
- Object instances corresponding to the logical data model (Data)
- Transaction context

Request

- Operation name
- Input payload object for the CRUDQ,
- the tenant identifier,
- Authorization information

Response

- Output payload object (response)
- The tenant identifier
- Authorization information

Rule:

""An Offering on Product A will get 30 minutes of additional usage per month if the user purchases Product B within a certain time period"

Authoring the rule

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

In order to create the above rule in a BRMS IDE, the facts and the range of possible values to select for the rule conditions need to be supplied. In this example, the condition part of the rule will be as follows

WHEN

- OMContext.OMrderFlowAction = 'Purchase Offering '
- Offering contains Product A
- Purchase date is between X and Y dates.

User has subscriptions for Product B

The attributes that are needed to author the above facts in a BRMS IDE should correspond to the objects used in the respective systems.

For example, the name of the workflow within Order Management is part of the condition of the rule. This requires the variable that denotes the name of the workflow within Order.

Management to be mapped with the enumeration of all the values like 'Purchase Offering'… to be made available within the IDE during the crafting of the rule.

The other attributes available for the conditions are as follows:

- OMContext.PurchaseOffering.OfferingID has a value
- UPCContext.PurchaseOffering.PurchaseDate is after mm/dd/yyyy
- UPCContext.Offering.OfferingID is same as
- OMContext.PurchaseOffering.OfferingID UPCContext.Offering.Product*1..n+.ProductName has "A"
- CLMContext.Customer.mobileNumber is same as OMContext.PurchaseOffering.mobileNumber
- CLMContext.Customer.Subscription[1..n].Product*1..n+.ProductName has "B"

As seen above, the BRMS IDE is pre-provisioned with the Java objects from the different systems and these facts are organized into the different packages that can be retrieved by the corresponding system like OM or CLM or UPC. In this case, the e rule logically falls within the ambit of Order Management. Hence, it will be part of the OM knowledge base package which is periodically scanned and retrieved by the OM.

The Order Management workflow which handles the operation 'purchase offering' will have to be intercepted to trigger this rule. In this case, the local rule engine within OM will have the above facts in the knowledge base.

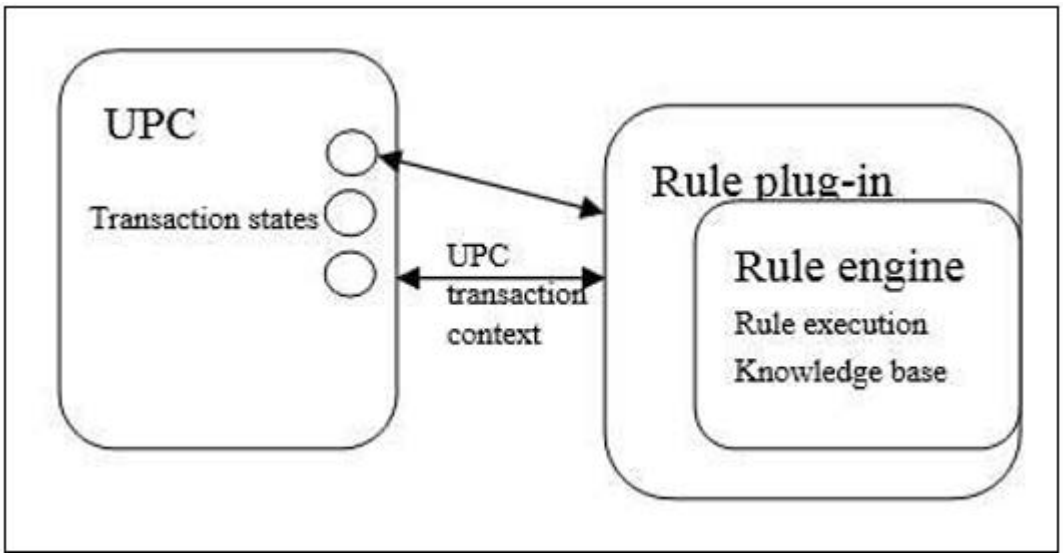
All Offering instances which are linked to Product A - This can be accomplished via a 'Search' command into UPC. BRMS offers a method to have a Java class that can be invoked to populate the enumerated list for selection as part of the rule authoring. Similarly a Java class to get the subscriptions of the user from CLM can populate the enumerated fields of the CLMContext that will be used.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

The objects and fields names used as part of defining the rules will be the same as the objects and field names during run-time execution of the rules. This mapping needs to be maintained. The rules can be segregated into packages specific to different systems which trigger those rules. The packages can be versioned and evolved. These systems periodically scan the rule authoring system for updates and load the respective packages. Within UPC or OM or CLM, the rule engine loads these packages into the knowledge base and the rule plug-in which is invoked by the UPC as part of AOP interception during processing of a CRUDQ transaction will inject the necessary objects containing the values for these rules. The rule execution results in further producing messages on the message queue or affecting the database or the transaction context or others.

Realizing Rule processing Within the UPC Architecture

This chart shows the rule processing within the UPC Architecture:



UPC Rule plug-in encapsulates a rule engine like Apache Drools. It is triggered whenever there is a state change on a UPC CRUDQ transaction. In each state, the transaction context is passed to the rule plug-in to verify if the context can trigger any rule and if so, the context is set in the working memory of the rules engine which triggers all the rules that matches this context.

> For example, a set of rules for recommendation of products could look for a special flag in a read request which specifies 'recommend' and then it may run through all the rules that belong to the category of 'recommendation'. The action part of the rule execution may affect the transaction context attributes and / or publish to a message queue about the result of execution and / or query the UPC database for more attributes.

The rules will be segregated based on the category they address to avoid unwanted rule execution on a condition. The categorization can bring in a hierarchical ordering wherein the top-level category could be based on tenant identifier, followed by the UPC operation, followed by the transaction state and the UPC data model objects impacted and so on.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

It is possible that the UPC UI may have to enforce some rules. This is again a interception from the UI and a call into the Rule plug-in.

UPC Data Management

The data management component in the architecture addresses the following features:

- Offers CRUDQ functions on the objects in the UPC domain model interfacing with one or more underlying data store technologies (JDBC, LDAP, Web service) for persistence.
- Flexibility in adding data store adapters to external sources for Propagation by configuration as detailed in section 3
- Flexibility in adding data store adapters to external sources for Aggregation by configuration as detailed in section 2
- Flexibility in adding data store adapters to external sources for Notification by configuration
- Handling multi-tenancy requirements by configuration as detailed in section 4
- Handling data model extensions that happen in underlying data stores by configuration as detailed in section

UPC Search Function

UPC search has the following requirements:

- Search will be issued on the logical data model of UPC exposed via SOAP/XML used by the UPC UI or client applications.
- The search is similar to an ad-hoc query and will be on the UPC data model key objects like Offerings, Products, Resources etc. that matches a certain criteria based on some of the attributes of those objects. For example, find all Offerings where Offering. Channel = "retail" or find all Products with 'incoming call' as a Resource.
- The search can be asked to return attributes of interest in the matched objects or linked objects of the matched objects.

The response of the search will align to the UPC logical data model XSD The solution involves the following:

- When a persistence happens on a object into the UPC data base due to a C.U.D operation and the persistence is successful, the DataStoreManager will set the state of the transaction to "Persistence Success".
- On a transaction state change, an interception happens to the PluginManager which in turn calls the notifier Plug-in that have registered for this state change.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

- This publishes the details of the transaction on a JMS queue. The indexing process which listens on this queue is notified with the details of the transaction.
- If it is a Creation of a new object, the request XML that was used to create along with the identifiers of the newly created object in the response XML are present in the queue.
- If it is an Update or Delete, the request XML payload of the updated/deleted object is available in the queue. The index process uses the XML content to perform the indexing into an index database using Lucene index APIs.
- The indexing is done on the values of the attributes of that object.

It is required to handle queries that may have conditionals involving attributes across the linked hierarchy of the UPC data model. an example could be "get all Resources which are linked to Products that were created after 'Jun 10 2012"., When an object like Product or Resource gets created for the first time, if it has one or more linked children objects (for example, a Product can have Resources), then, a new Lucene index document is created for the parent object along with other linked child object identifiers. As this object or any of its child objects get updated, then a search is done using the object identifier of the updated object. All the index documents that contain this object identifier will have to be re-indexed to take in the new update. The search can be done using the XPATH corresponding to this object identifier as represented in the UPC data model. For example, if the Product that is updated has an object identifier of 1234, then a search is done for '/Offering/ Bundle/Product/id = '1234''.

The search expression will be of the form SELECT

1.more XPATH in UPC data model WHERE

Lucene search expression constructed with the UPC data model attributes as XPATHs. For example:

SELECT

Offering WHERE

Product.code: "xxx" AND "Product.createdDate:>1/6/2012

When a search expression of the nature of Lucene is given in the UPC web service exposure via SOAP/XML, the DataStoreManager's search method applies the WHERE part of the search expression on the index database and retrieves all the matched records. The SELECT part of the search expression lists the objects of interest which may be the same as the matched object or it could be the linked object of the matched object.

In case of former, then the object identifiers retrieved from the Lucene documents that matched based on the search criteria are used to iteratively read the UPC database by calling the read on Generic CRUD component.

In case of latter, the identifiers of the linked objects in the matched document is used to perform a read of the linked object iteratively and the collection of XMLs are returned as a response to the Search transaction. The figure below shows the flow across the architecture components for index creation.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Search Across a UPC Cluster and Multi-tenancy

The index file system is common to all the nodes of UPC in a cluster and hence a indexing done because of a database record change in one node of a UPC cluster can be made available for a search from other nodes of the UPC cluster.

The Indexer process and the message queue run in every UPC node in an all active configuration. The dispatch of the messages produced by every UPC application on a persistence will be distributed based on a 'mod' on the 'identifier' of the object that got updated and the number of nodes and using the outcome of the 'mod' operation as the identifier of the UPC node.

> For example, if an object that got updated is a Product with an object identifier of 5 and a Resource linkage with a identifier of 2, the mod operation will be done on 5 with the number of nodes (say 3 in this example). This will return 2 and hence the message will be sent to the message broker in node 2. This is done to ensure that all persistence operations (Update, Delete) on the same UPC object is processed by the same node in the cluster to preserve the integrity of the indexes stored in the shared file system.

If Lucene APIs are used for indexing and searching, a shared file system having the indices across the UPC cluster can be maintained using NFS. Alternatively, Solr allows replication of file system based indices across nodes of the cluster. In a multi-tenant configuration, tenant specific directories in the file system will be made available via NFS mount or via replication.

Alternate Solution for a Configurable Search using Hibernate HQL

This approach examines a more simpler alternate solution to the search which uses Hibernate HQL. However, this addresses a limited scope because of the constraints on the search expression in XML and the need to parse them. The request XML will contain:

- <ObjectOfSearch> Offering </ObjectOfSearch>
- <FilterCriteria>attribute="<value>"</FilterCriteria>
- <FilterCriteria>attribute="<value>"</FilterCriteria>
- <FilterCriteria>attribute="<value>"</FilterCriteria>

This returns a collection of the objects specified in the ObjectOfSearch. The FilterCriteria allows specifying one or more attributes whose value is matched with the selected object attributes for filtering the collection.

In the Hibernate DataStore adapter, the configuration allows specifying a generic HQL and applying it on the POJO and substituting the FilterCriteria values into the HQL.

Because of the complexity of expressing the rich Search expressions with custom syntax and implementation, this approach is limited in scope in terms of what can be achieved.

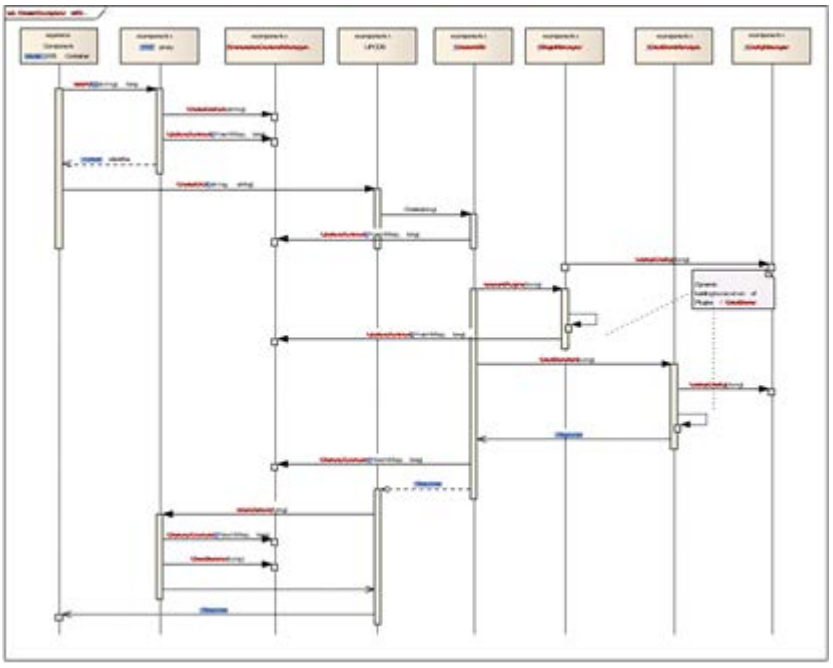
©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

UPC Component Level Call Flows

It is a Visual representation of the sequence of steps and interaction involved in a telephone or communication system.

Create Flow

This diagram shows the components level of call flows:



Receiving the Request and doing the Authentication and Policy Enforcement

When a request to Create a UPC logical data model object is received by UPC such as a Create Product(), Create Offering(), CreateResource()...etc., the web service container that has the endpoint for the UPCCRUDQ WSDL deployed, invokes the appropriate CreateXXX() method on the UPCDS component passing the SOAP payload that contains the following parameters.

- Request XML payload conforming to the UPC logical data model object XSD (Product, Offering).
- Version that tells the version of UPC logical data model schema to be used.
- Optional version of the object being created if the automatic versioning is to be over- ridden.
- SAML Payload that represents the authentication credentials (or a token) of the requestor.
- Every CRUDQ request into UPC is intercepted by a call to the PEP proxy::doAAA(). The SAMLPayload that came as part of the SOAP request is passed to it. The PEP proxy does the following.
- Decryption of the SOAP/XML payload.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

- Signature validation of the token issuer in case the SAML payload contains token issued by an IDP.
- If token is not present, but credentials (login/password) are present, then authenticate the credentials with the IDP and get back a valid token.
- Obtain the entitlements (in terms of the methods allowed for this requestor) from the Authorization or Policy repository.
- Enforce the policies for this request.

Persisting AAA Data into Transient Transaction Context

Based on the outcome of the policy enforcement, if allowed to proceed further, create a transaction context by calling the Create Context() passing in the type of operation (CRUDQ) and subsequently an Update Context() passing in the attributes (from the authentication and authorization) to be stored in the context.

The context attributes stored into a transaction context are:

- Token
- Tenant identifier
- Operation entitlements (methods which are allowed for this requestor)
- The UPC logical data model objects which are allowed for this requestor for the allowed operations

The PEP Proxy returns the context identifier of the created transaction context back to the web service container.

Call to the CRUD Operation on the Web Service Exposed by UPC Data Service

Post AAA, the web container call is handled by the CreateXXX() web service implemented by the UPC data service component. The transaction context identifier received from the PEP Proxy is also available apart from the SOAP payload containing the XML to be created. See

The UPC DS validates the XML payload for well formedness and delegates the transaction handling aspects to the GenericDS component.

Call to GenericDS Component for a Create Operation

The GenericDS updates the Transaction context with the request payload , begins the transaction processing flow with the transaction state updated to TRANS_BEGIN. The GenericDS is not specific to the XML payload or its structure. It treats it like a XML string in a type agnostic manner.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Interception on Transaction State Change to Call PluginManager

The PluginManager is invoked when a transaction begins by a call to the ExecutePlugins() method. This is done to allow pre-processing handlers to run before the transaction is continued further. The PluginManager looks up the configuration manager to dynamically load and execute all the plugins that have been configured for processing in this state. As a part of a plugin processing, the transaction context could be modified.

Handle Persistence, Aggregation, Propagation and Indexing.

- The GenericDS on completion of plugin execution, calls DataStoreAdd() method on the DataStoreManager component. A configuration lookup is done by this component and based on that, the stores that should be persisted are identified.
- The XML payload is passed on further to the Data store adapters which are configured for persistence for this specific UPC logical data model object.
- The adapters are dynamically loaded and invoked. The adapters are expected to implement standard interface methods. Each adapter translates the given XML payload to the native format and persists using the native protocol (JDBC, web service etc.).
- It is possible that every XML payload may result in a multi-table write within a database and transaction aspects like ACID, transaction failure, rollback are handled within the data store adapter as applicable to the characteristics of the data store the adapter is interfacing with.
- Further on success with UPC persistence, the DataStoreManager may invoke one or more propagators by depositing the messages for propagation based on configuration lookup to a message queue asynchronously. Success or failure of propagation transactions are handled appropriately.

Return Response for CreateXXX()

The Data store adapter returns a response XML which aligns to the XSD of the object being read. The XML contains all child elements, child objects and linked objects for the identifiers. See 8.3.1.2 for examples on the response. The GenericDS component may clean up the transaction context of all except the AAA related context attributes.

Interception of the Transaction State to PluginManager

The transaction state is TRANS_COMPLETE. Any plugins which are configured to process this transaction state from a post-processing aspect can implement this.

Interception to PEP Proxy to do Post-processing on Transaction Context.

The response is returned to the UPCDS which is intercepted to handle any post-processing on the transaction context created by the PEP proxy. In this case, the PEP proxy may remove the context or retain it if the token has not expired to avoid setting up transaction context for further requests with the same token.

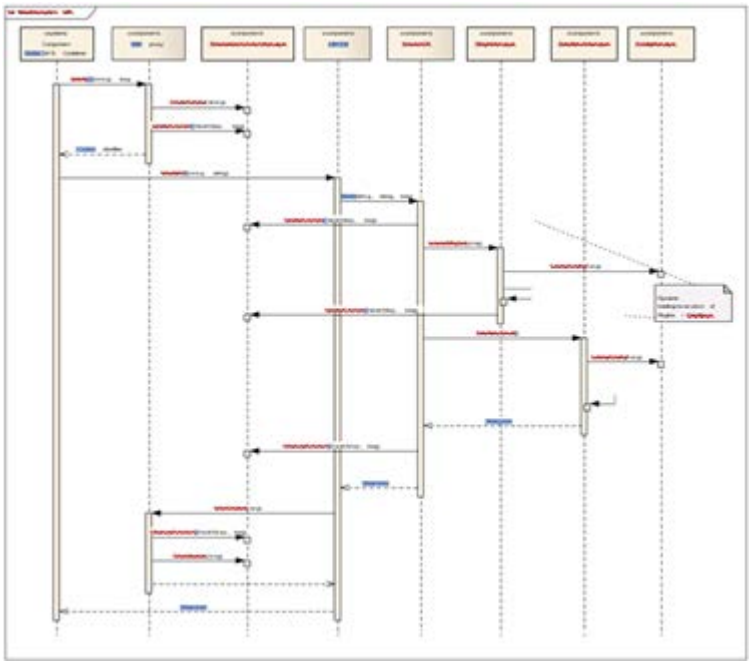
©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Returning the SOAP Response

Further, the XML response is returned as a response to the SOAP message request that was processed by the UPCDS.

Read Flow

This diagram shows the functions of read flow:



Receiving the Request and doing the Authentication and Policy Enforcement

When a request to Create a UPC logical data model object is received by UPC such as a CreateProduct(), CreateOffering(), CreateResource()...etc., the web service container that has the endpoint for the UPCCRUDQ WSDL deployed, invokes the appropriate CreateXXX() method on the UPCDS component passing the SOAP payload that contains the following parameters:

- Request XML payload conforming to the UPC logical data model object XSD (Product, Offering).
- Version that tells the version of UPC logical data model schema to be used.
- Optional version of the object being created if the automatic versioning is to be over- ridden
- SAML Payload that represents the authentication credentials (or a token) of the requestor
- Every CRUDQ request into UPC is intercepted by a call to the PEP proxy::doAAA(). The SAMLPayload that came as part of the SOAP request is passed to it. The PEP proxy does the following.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

- Decryption of the SOAP/XML payload
- Signature validation of the token issuer in case the SAML payload contains token issued by an IDP
- If token is not present, but credentials (login/password) are present, then authenticate the credentials with the IDP and get back a valid token.
- Obtain the entitlements (in terms of the methods allowed for this requestor) from the Authorization or Policy repository.
- Enforce the policies for this request.

Persisting AAA Data into Transient Transaction Context

Based on the outcome of the policy enforcement, if allowed to proceed further, create a transaction context by calling the CreateContext() passing in the type of operation (CRUDQ) and subsequently an UpdateContext() passing in the attributes (from the authentication and authorization) to be stored in the context.

The context attributes stored into a transaction context are:

- Token
- Tenant identifier
- Operation entitlements (methods which are allowed for this requestor)
- The UPC logical data model objects which are allowed for this requestor for the allowed operations

The PEP Proxy returns the context identifier of the created transaction context back to the web service container.

Call to the CRUD Operation on the Web Service Exposed by UPC Data Service

Post AAA, the web container call is handled by the CreateXXX() web service implemented by the UPC data service component. The transaction context identifier received from the PEP Proxy is also available apart from the SOAP payload containing the XML to be created. See

The UPC DS validates the XML payload for well formedness and delegates the transaction handling aspects to the GenericDS component.

Call to GenericDS Component for a Create Operation

The GenericDS updates the Transaction context with the request payload , begins the transaction processing flow with the transaction state updated to TRANS_BEGIN. The GenericDS is not specific to the XML payload or its structure. It treats it like a XML string in a type agnostic manner.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Interception on Transaction State Change to Call PluginManager

The PluginManager is invoked when a transaction begins by a call to the ExecutePlugins() method. This is done to allow pre-processing handlers to run before the transaction is continued further. The PluginManager looks up the configuration manager to dynamically load and execute all the plugins that have been configured for processing in this state. As a part of a plugin processing, the transaction context could be modified.

Handle Persistence, Aggregation, Propagation and Indexing.

- The GenericDS on completion of plugin execution, calls DataStoreAdd() method on the DataStoreManager component.
- A configuration lookup is done by this component and based on that, the stores that should be persisted are identified.
- The XML payload is passed on further to the Data store adapters which are configured for persistence for this specific UPC logical data model object.
- The adapters are dynamically loaded and invoked. The adapters are expected to implement standard interface methods. Each adapter translates the given XML payload to the native format and persists using the native protocol (JDBC, web service etc.).
- It is possible that every XML payload may result in a multi-table write within a database and transaction aspects like ACID, transaction failure, rollback are handled within the data store adapter as applicable to the characteristics of the data store the adapter is interfacing with.
- Further on success with UPC persistence, the DataStoreManager may invoke one or more propagators by depositing the messages for propagation based on configuration lookup to a message queue asynchronously. Success or failure of propagation transactions are handled appropriately.

Return Response for ReadXXX()

The Data store adapter returns a response XML which aligns to the XSD of the object being read. The XML contains all child elements, child objects and linked objects for the identifiers. The GenericDS component may clean up the transaction context of all except the AAA related context attributes.

Interception of the Transaction State to PluginManager

The transaction state is TRANS_COMPLETE. Any plugins which are configured to process this transaction state from a post-processing aspect can implement this.

Interception to PEP Proxy to do Post-processing on Transaction Context.

The response is returned to the UPCDS which is intercepted to handle any post-processing on the transaction context created by the PEP proxy. In this case, the PEP proxy may remove the context or retain it if the token has not expired to avoid setting up transaction context for further requests with the same token.

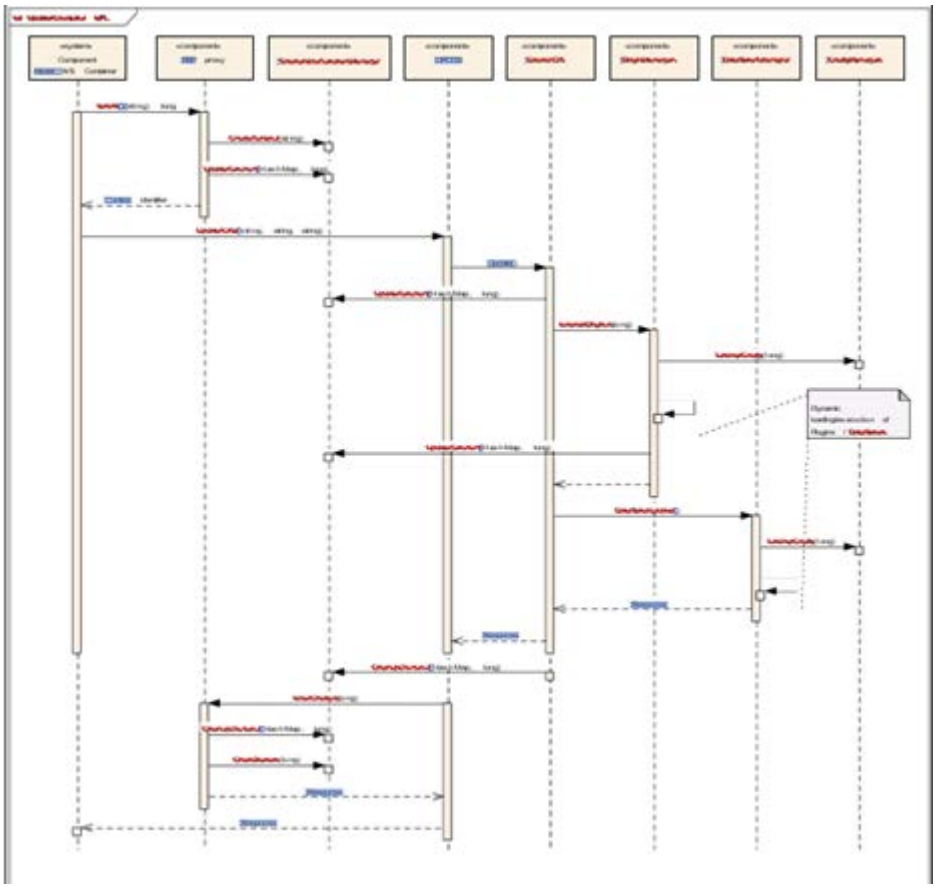
©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Returning the SOAP Response

Further, the XML response is returned as a response to the SOAP message request that was processed by the UPCDS.

Update Flow

This diagram shows the functions of Update Flow:



Receiving the Request and doing the Authentication and Policy Enforcement

When a request to Create a UPC logical data model object is received by UPC such as a CreateProduct(), CreateOffering(), CreateResource()...etc., the web service container that has the endpoint for the UPCCRUDQ WSDL deployed, invokes the appropriate CreateXXX() method on the UPCDS component passing the SOAP payload that contains the following parameters

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

1. Request XML payload conforming to the UPC logical data model object XSD (Product, Offering).
2. Version that tells the version of UPC logical data model schema to be used.
3. Optional version of the object being created if the automatic versioning is to be over- ridden
4. SAML Payload that represents the authentication credentials (or a token) of the requestor
5. Every CRUDQ request into UPC is intercepted by a call to the PEP proxy::doAAA(). The SAMLPayload that came as part of the SOAP request is passed to it. The PEP proxy does the following.
6. Decryption of the SOAP/XML payload
7. Signature validation of the token issuer in case the SAML payload contains token issued by an IDP
8. If token is not present, but credentials (login/password) are present, then authenticate the credentials with the IDP and get back a valid token.
9. Obtain the entitlements (in terms of the methods allowed for this requestor) from the Authorization or Policy repository.
10. Enforce the policies for this request.

Persisting AAA Data into Transient Transaction Context

Based on the outcome of the policy enforcement, if allowed to proceed further, create a transaction context by calling the CreateContext() passing in the type of operation (CRUDQ) and subsequently an UpdateContext() passing in the attributes (from the authentication and authorization) to be stored in the context.

The context attributes stored into a transaction context are:

- Token
- Tenant identifier
- Operation entitlements (methods which are allowed for this (requestor).

The UPC logical data model objects which are allowed for this requestor for the allowed operations

The PEP Proxy returns the context identifier of the created transaction context back to the web service container.

Call to the CRUD Operation on the Web Service Exposed by UPC Data Service

Post AAA, the web container call is handled by the CreateXXX() web service implemented by the UPC data service component. The transaction context identifier received from the PEP Proxy is also available apart from the SOAP payload containing the XML to be created. See

The UPC DS validates the XML payload for well formedness and delegates the transaction handling aspects to the GenericDS component.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Call to GenericDS Component for a Create Operation

The GenericDS updates the Transaction context with the request payload , begins the transaction processing flow with the transaction state updated to TRANS_BEGIN. The GenericDS is not specific to the XML payload or its structure. It treats it like a XML string in a type agnostic manner.

Handle Persistence, Aggregation, Propagation and Indexing.

- The GenericDS on completion of plugin execution, calls DataStoreAdd() method on the DataStoreManager component. A configuration lookup is done by this component and based on that, the stores that should be persisted are identified.
- The XML payload is passed on further to the Data store adapters which are configured for persistence for this specific UPC logical data model object. The adapters are dynamically loaded and invoked. The adapters are expected to implement standard interface methods. Each adapter translates the given XML payload to the native format and persists using the native protocol (JDBC, web service etc.).
- It is possible that every XML payload may result in a multi-table write within a database and transaction aspects like ACID, transaction failure, rollback are handled within the data store adapter as applicable to the characteristics of the data store the adapter is interfacing with.
- Further on success with UPC persistence, the DataStoreManager may invoke one or more propagators by depositing the messages for propagation based on configuration lookup to a message queue asynchronously. Success or failure of propagation transactions are handled appropriately.

Return Response for UpdateXXX()

The Data store adapter returns a response XML which aligns to the XSD of the object being read. The XML contains all child elements, child objects and linked objects for the identifiers. The GenericDS component may clean up the transaction context of all except the AAA related context attributes.

Interception of the Transaction State to PluginManager

The transaction state is TRANS_COMPLETE. Any plugins which are configured to process this transaction state from a post-processing aspect can implement this.

Interception to PEP Proxy to do Post-processing on Transaction Context.

The response is returned to the UPCDS which is intercepted to handle any post-processing on the transaction context created by the PEP proxy. In this case, the PEP proxy may remove the context or retain it if the token has not expired to avoid setting up transaction context for further requests with the same token.

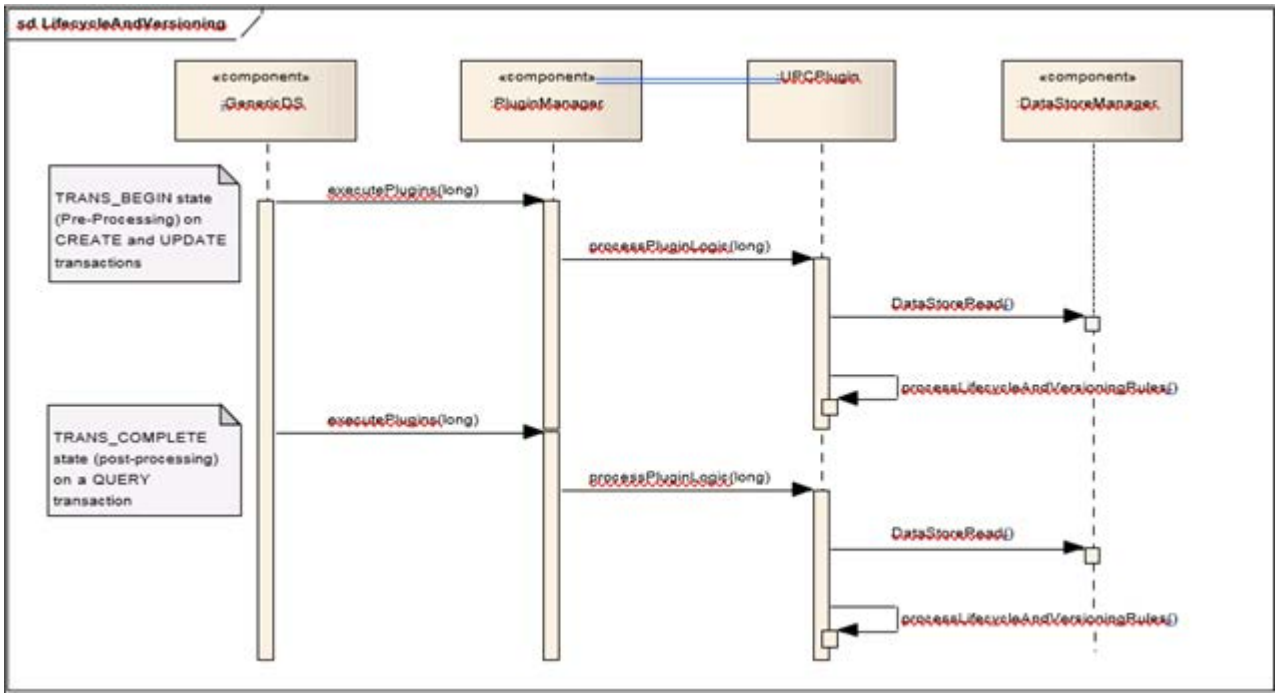
Returning the SOAP Response

Further, the XML response is returned as a response to the SOAP message request that was processed by the UPCDS.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Lifecycle and Versioning

This chart shows the chain of Life cycle and Versioning:



©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

UPC Non-functional View

The following sections deal with aspects of non-functionals that influence the functional architecture in a large way.

Multi-tenancy

Multi-tenancy adds the ability for a single instance of a UPC application to support different data model and custom logic for each tenant. A tenant of UPC can be a telecom operator who can have one or more operating companies (OpCo) in different geographies.

For detailed requirements on multi-tenancy, see ref. not found.**Error! Reference source not found. Error! Reference source** .

Multi-tenancy Scope Addressed by Architecture

- There can be a single UPC Database holding all tenant information as shown above.
- There can be separate logical database (though on same physical server cluster) for each tenant to allow evolution of tenant's data growth and performance independently of other.
- There can be a centralized database holding all the tenants data
- There can be site specific database holding specific tenants data completely
- There can be a hybrid configuration of site specific and centralized database to hold the data from different tenants.
- There can be site specific database holding partial data of the tenant while the centralized database stores the remaining part of the data
- There can be aggregators, propagators specific to a tenant
- There can be same UPC schema for all tenants and any changes will apply to all of them. All evolve with the same schema.
- There can be separate evolution of UPC schema for the different tenants
- There can be same or different lifecycle management or other validation specific to UPC for the different tenants
- There can be different rules of exclusivity, dependency or others for each tenant. Multi- tenancy is applicable during both rule definition as well as run-time enforcement of the rules.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Architecture Details

The problem of multi-tenancy is centrally addressed within a single component which is the ConfigManager in the architecture. No other component has knowledge of multi-tenancy as they just carry on with their functionality irrespective of the tenancy.

The variants of data, schema, logics required by different tenants are brought in by configuration. The configuration required by each component is managed centrally by the ConfigManager on behalf of all other components.

The ConfigManager uses the transaction context to get the tenantID and in turn use the correct tenant specific configuration when other components processing a transaction flow requires a configuration item that are UPC schema related or database related or pluggable logic related. See the diagram in Figure 11: Multi-tenant model for the points in the architecture that requires enablement of multi-tenancy.

Points in the Architecture that Requires Multi-tenancy Enablement

UPC Data Model Extensibility

UPC data model extensibility needs are triggered by:

- Requirements from multiple tenants requiring different schema (logical or physical)
- Specific feature extensions or data model / schema extensions
- Different schema versions caused by Aggregation, Propagation to underlying sources
- Evolution of UPC schema over time.

The data model extensibility enables UPC deployments at different customer locations to start with a base domain model and evolve independently into multiple versions of the domain model as new sources of aggregation, propagation, consuming application come into play. It is also the case where a Customer has their own interpretation of the TMF SID or their own common data model existing which needs to be supported by UPC.

The architecture addresses data model extensions in UPC by abstracting the CRUDQ transaction processing flows in a way, the logic is independent of UPC data model specific processing. The UPC specific processing like Lifecycle, version etc. are handled in specific pluggable modules.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Configurable Points in the Architecture for Data Model Extensibility

The extensibility configurations field provides additional settings and some advanced setting are available only (for example, adding custom mobile plugin).

Data Model Agnostic Behaviour at the Persistence Layer

Throughout the UPC transaction processing flow, domain objects are handled in a type agnostic manner as XML strings. Before they are persisted into the database, they are translated into appropriate domain model objects as required by the persistence mechanism using mapping configurations that map the logical XML elements to the physical world that can be a relational or a LDAP or other database structures.

> For example, if Hibernate is used to manage the persistence to relational stores, this mapping configuration file will contain the mapping between the XML elements of the UPC logical data model and the Hibernate generated POJO classes that represent the Object form of the relational tables underneath in the database. The Hibernate data store adapter processing this configuration invokes the CRUD API of Hibernate on the POJO classes of Hibernate using Java Reflection and thus avoids implementation having specific references to domain model objects.

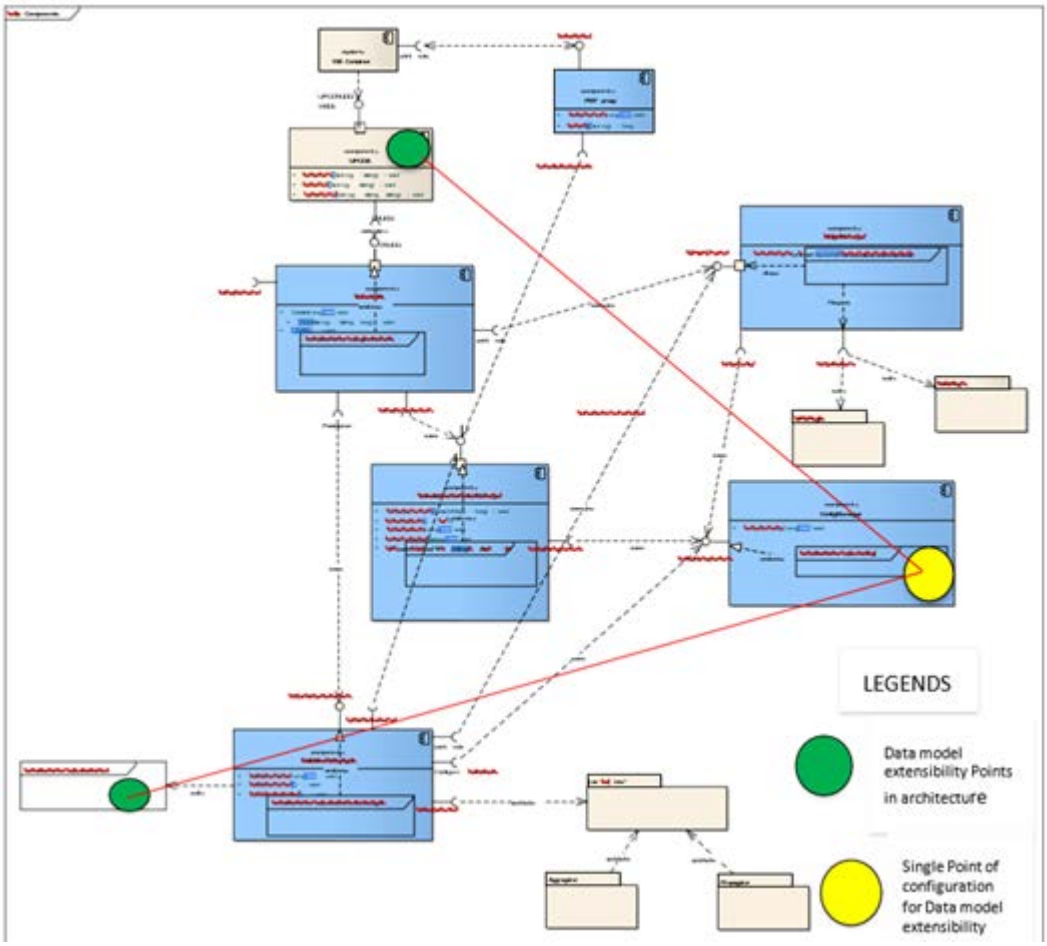
When a table in a database undergoes a change with new fields or type changes to existing fields, the corresponding Hibernate configuration also undergoes a change, which results in re- generation of the Hibernate Java classes. Further to process a CRUD operation that has XML as the payload (coming from the UPC DS / Generic DS component invoked by clients of the Web service API of UPC), by way of dynamic loading, the new Hibernate Java class getter/setter are filled up to form the object to be persisted. The same applies when an association is modified between objects or new objects are introduced which results in updating the database and re- defining the Hibernate configuration and metadata configuration.

Data Model Agnostic Behaviour at the UPC Data Service Exposure Layer

The UPCDS component which exposes specific data types like Product, Offering for client applications via WSDL are developed in a way, new additions of logical data model objects does not cause the UPC DS to be redeveloped.

Since the UPC DS component mainly does validation of the incoming payload per the XSD and delegates the transaction context to the Generic data service component, a generic way of handling a XML payload using un-typed interface allows the component to handle any types. The WSDL thus becomes a method of exposing the different tenant specific API and logical data models to clients rather than the need to develop implementations specific to each WSDL.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template



Multiple simultaneous versions of the UPC logical data model can be present in a single application instance of UPC because of multi-tenancy or because of single tenant evolving the data model resulting in multiple versions of the schema to be supported. The API exposed by the UPCDS component takes in a version parameter that allows the transaction processing to pick up the correct version of the schema when a metadata mapping is done before doing the persistence.

Scenarios for Data Model Extensions

In general, all the above changes would require configuration updates to be done to the following four components in the architecture.

- Database
- DataStore
- DataStoreManager
- UPCDS

Example 1: A new column PRODUCT_RANK in the table PRODUCT is added to represent the rank of a Product in terms of its subscriptions.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

- In this case, the Hibernate XML configuration will have a new property element called 'ProductRank' for the Class 'Product'.
- This Hibernate configuration file can be used to generate the Java classes which will have the setter 'setProductRank' and getter 'getProductRank'.
- The HibernateDataStore class which implements the interface

Example 2: An existing column PRODUCT_CATEGORY is removed and instead a association of

- Many is created between PRODUCT and PRODUCTCATEGORY with PRODUCTCATEGORY being a new Table.

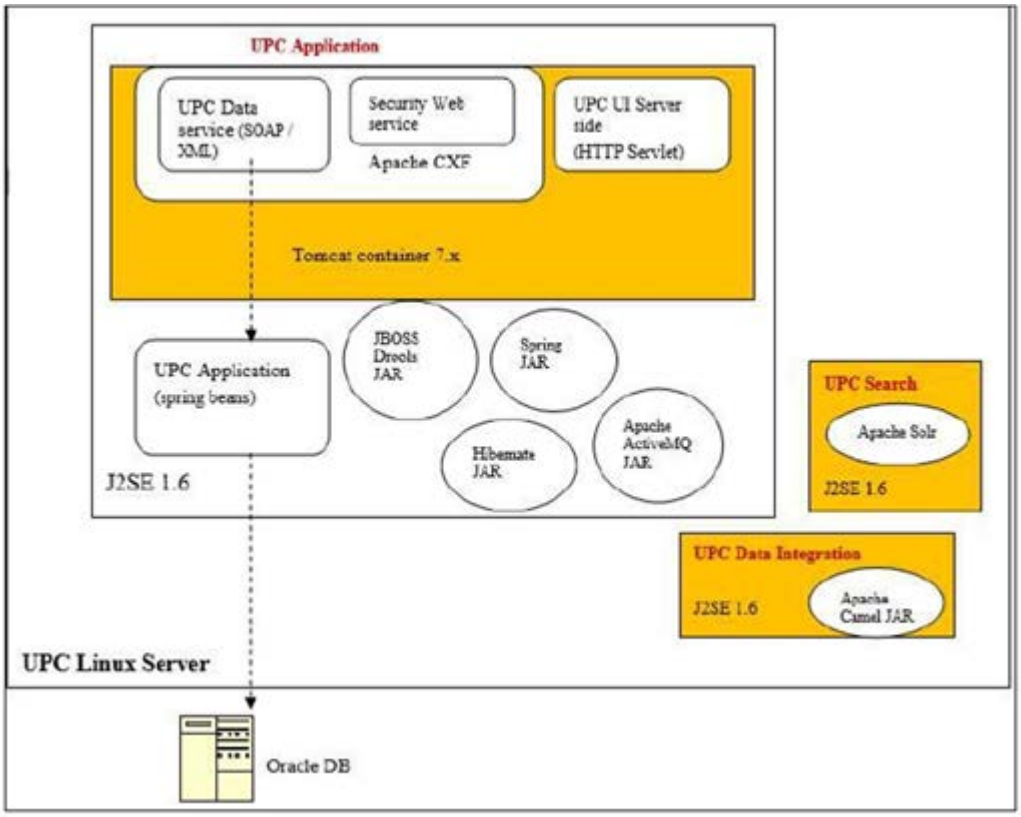
Example 3: A new relationship is created between OFFERING and PRODUCT table with an OFFERING_PROD_RELATIONSHIP

Example 4: The relationship between a PRODUCT and LINE_ITEM tables via the PROD_LITEM_RELNSHIP is removed and instead 2 new columns called LINE_ITEM1 and LINE_ITEM2 are added directly into the PRODUCT table.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

UPC Deployment View

This view describes the environment within which the system is executed. It used to visualize the hardware/ processors/nodes of a system and the link of communication between them.



Single Node Deployments

In the above architecture, there are three Java applications that appear as three processes in Linux.

- The UPC application that includes the UPC UI server side and the Security Service
- UPC Search
- UPC Data Integration

The UPC application handles uses Apache CXF to expose the UPC data service API. The UPC UI server side runs within a tomcat web container. The security web service that does authentication and authorization can be deployed in its own container. However to keep things simple, it is deployed within the same Apache CXF as the UPC application. The backend of UPC that implements the generic CRUDQ function runs in a Spring container. It uses other JARs to realize the functions of UPC like rule processing, persistence and message queues.

The UPC search responsible for indexing the UPC data and allowing search on the indexes is handled by the Apache Solr which runs within a JVM instance of its own.

The UPC Data integration responsible for aggregation and propagation behaviours runs in its own JVM instance.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

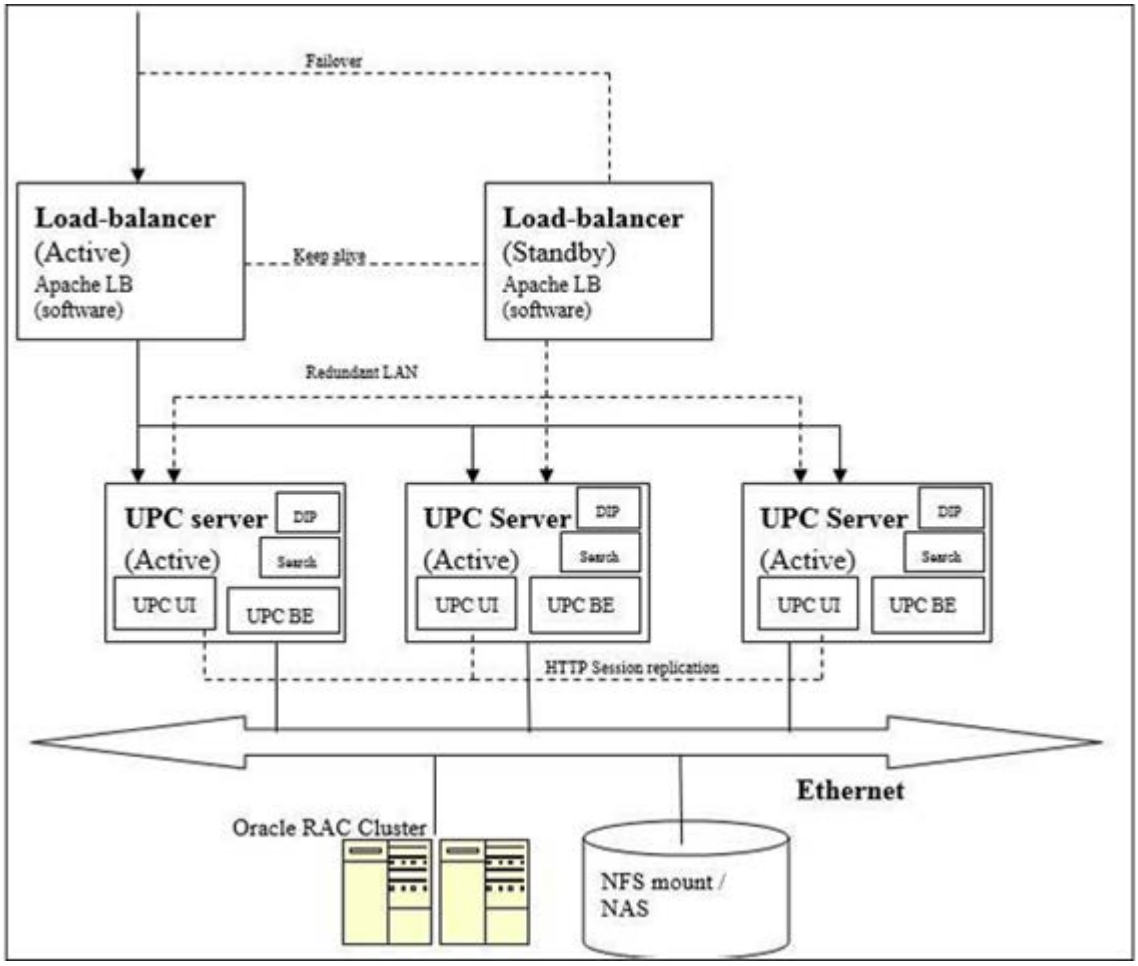
HA Configuration

- The UPC data service API does not maintain any client session context as every CRUDQ operation is simple and stateless.
- However, the UPC UI server side maintains client session context. This could have a partial request that is being constructed by a user and can potentially get destroyed if a UPC node fails (or Tomcat or JVM crashes).
- Further UI actions from the client if handled by another UPC node in the cluster (because of fail-over as routed by the load balancer), the session context that has been established with the client should be made available in the other UPC node to which the request has been routed in order to continue the client UI actions without a break.
- On the other hand, as mentioned before, the UPC data service is quite stateless and hence clients can retry requests that have failed that can result in another UPC node processing the request.
- The UPC UI server side invokes the UPC data service via SOAP/HTTP at present. Thus co- locating the UPC UI server side and the UPC data service in the same Tomcat container as in a single node configuration will result in bringing down a UPC data service along with the UPC UI server.
- Thus in a cluster deployment of UPC, the backend of UPC (which is the UPC DS) and the front- end of UPC (which is the UPC UI Server) are deployed in separate Tomcat web containers on every node of the cluster. The invocation of the backend by the front-end can be routed by the same load-balancer which also does the routing across the front-end when clients invoke the UPC UI on the browser.
- Thus a node failure will cause the load-balancer to redirect all the traffic destined for that node either into the front-end or the backend to other active nodes in the UPC cluster.
- When a UPC node in a cluster fails, the session context cached in the UPC UI server side (front-end) will be lost. If the user was creating a Product or Offering, this could result in intermediate data that has been modified before a final request for submitting the form is done will be lost. For all the products or Offerings for which the data has been created/modified and submitted into the UPC UI server side, this would mean the request is lost and the user has to re-enter them again. Tomcat cluster deployment allows preservation of session context using shared disk, RDBMS or in-memory session replication across the cluster. This means that the session context in every node can be seen by other nodes, resulting in processing further request against a session context that was not created in a node.
- Since UPC Backend transaction processing is stateless, all the requests that are in progress within the UPC application, will be retried by the clients on timeouts of their requests. The requests that are retried will be re-routed to the other UPC Active nodes in the cluster by the FE load balancer. The load balancer uses keep-alive pings to the UPC nodes (to the UPC URL) and monitors the availability of a UPC application. When a request times out, the load balancer automatically starts routing the SOAP requests to other UPC nodes.
- Apart from the UPC backend, there are other processes like the UPC search and the UPC data integration. If the UPC search process fails in a node, the query function of UPC is affected on that node. The UPC application on the local node depends on the UPC search application to index the data persisted in UPC. When there is a failure of the UPC search on one node, the UPC application can index the same with the UPC search process on another UPC node. This is still possible and can be easily achieved by the load-

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

balancer as the invocation of the UPC search for indexing is done via REST/SOAP. The UPC query API which is used by client applications to perform ad-hoc queries on the UPC data also depends on the UPC search SOAP API. Thus this can as well be routed to other UPC nodes. The monitoring of the failure of the UPC search process can be done and if it is not running, it can be restarted by means of a cron job on Linux.

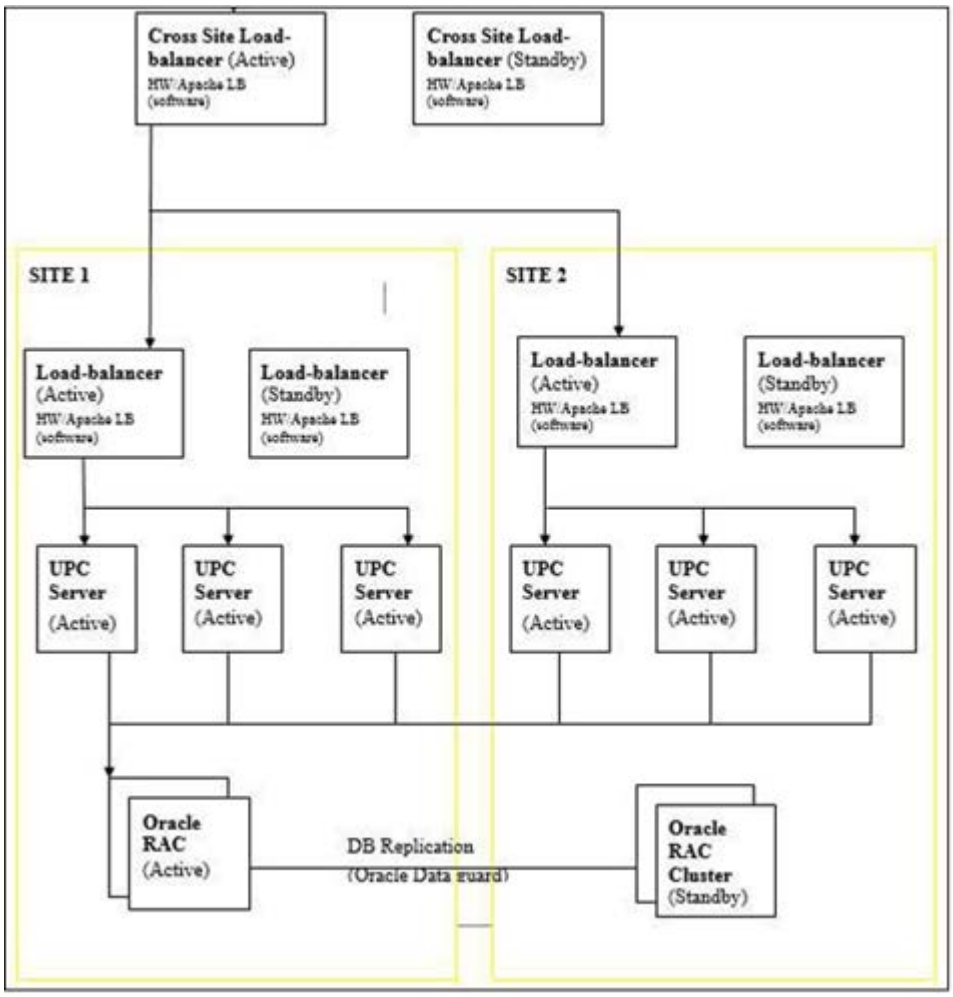
- The Data Integration Process (DIP) does the aggregation and propagation to and from UPC with other external systems. The dependency between UPC and this process is loosely coupled by means of a message queue. Thus, if either UPC or DIP is down on a node, the messages get persisted in the queue until the application is restarted on that node. Both UPC and DIP processes will be monitored and restarted if it goes down on a node by a Cron job.
- The front-end load balancer is maintained in an active-standby configuration to handle single point of failure and switchover to standby on a failure. Persistence is done by the UPC cluster nodes onto an Oracle RAC. Figure 18 shows the UPC HA deployment architecture.



Multi-site Cluster Deployment

In case of multi-site deployment, the first level load-balancers route the incoming traffic to the UPC nodes across the sites in a round-robin fashion.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template



Further, traffic entering a site will be load-balanced across the nodes within that site. The UPC database is shared across sites by means of Oracle DB replication by Oracle Data Guard. Thus the UPC nodes in both the sites initially persist and read data off the Active Oracle RAC. If there is a failure of Oracle RAC in one site, all UPC nodes start using the Oracle RAC in the other site. The minimal configuration requires two UPC nodes per site. Failure of both the nodes will render the site as failed and result in re-direction of the incoming traffic to the other site until the site that failed comes back up.

Performance and Scalability

UPC can be horizontally scaled by adding nodes to a UPC cluster. The database tier can be independently scaled by adding nodes to the database. Vertical scalability within UPC can be achieved by upgrading the servers with more cores and also by adding more processes of the UPC application, UPC search or Data Integration Process within a node.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

The generic CRUD transaction handling may bring in additional objects and calls. However, at present, they do very less processing. If the XML conversions to objects and vice-versa are done using SAX this will add to the performance. Objects should be reused to reduce new allocations and a subsequent GC runs. Once a performance goal in terms of TPS and latency are arrived at, the following interface points should be verified in terms of load testing and measurements should be recorded. For this, the definition of a transaction within UPC should be arrived at.

- The SOAP interface of the UPC DS calling the generic CRUD.
- The Rule interface that executes the rules.
- The Persistence interface with Hibernate.
- The messaging interface with Active MQ.
- The end user latency at the UPC UI.
- Indexing and querying.

For horizontal scalability, every UPC node may be dimensioned with a capacity as follows: Max capacity of a node - (Max capacity / number of nodes).

Thus if the max capacity of a node is 50 TPS and 200 TPS is the total capacity to be handled, then 5 nodes will be commissioned with each of them processing 40 TPS. Thus if there is a failure of one of the nodes, the 40 TPS handled by that node will be distributed equally on the other four nodes making them handle a max capacity until the failed node comes back up.

OAM

The UPC transaction context allows a single place to record the different statistics counters using JMX and logging.

- Running average of each of the CRUDQ transactions in the last 'X' minutes
- Average latency of each of the CRUDQ transactions in the last 'X' minutes
- The count of failures of each of the CRUDQ transaction across different categories since the application started.

The counters are updated for every transaction. But the logging is done based on a pre- configured time interval to avoid file I/O during transaction processing.

The different failure scenarios of UPC will be analyzed including the error scenarios during transaction processing and a table of alarms of different severity level will be published via JMX and logging. An NMS in the customer's network will be able to monitor these alarms and take appropriate action based on a Method of Procedure manual

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Security

The security architecture is handled separately and outside the scope of this document.

Localization and Globalization

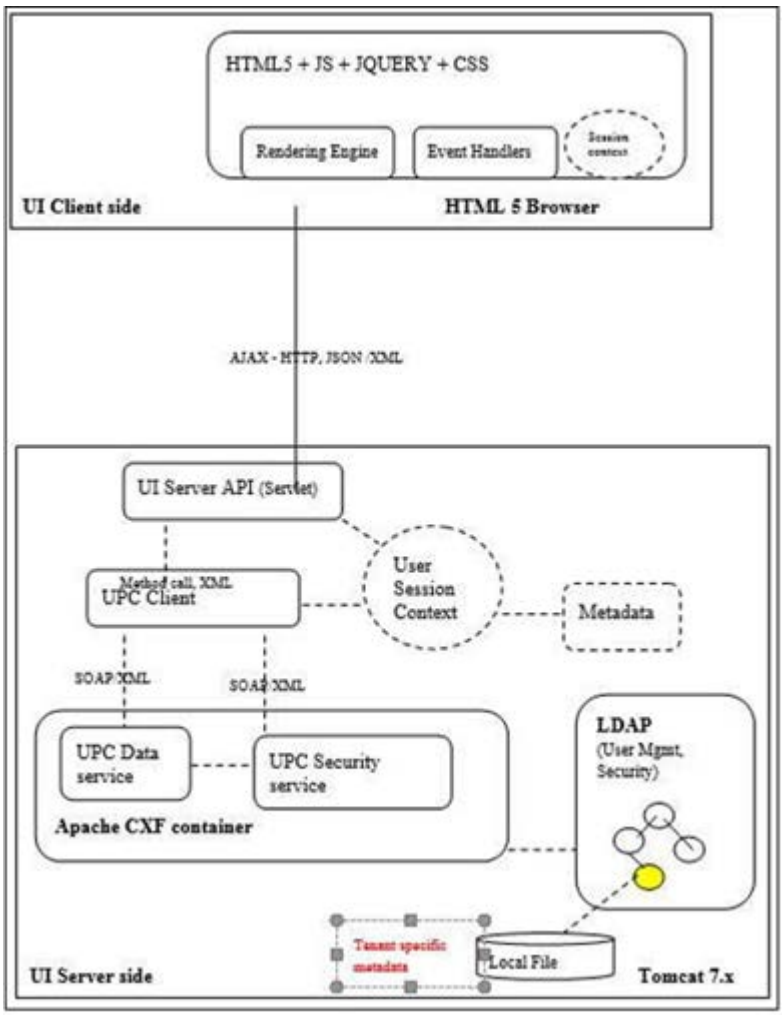
The UPC implementation will support i18N and l10N aspects. The XML payload entering UPC for the CRUDQ operations can have multi-byte character sets. The implementation should follow the Java coding practices on i18N while doing string operations on the request and response payload during CRUDQ transaction processing.

The XML schema, the dates, the enumerations will all be in English. This leaves a small portion of the payload involving text description to be in multi-byte. Thus, the UPC rules will be in English and allowed to use the attributes that are expressed in English. The UPC database will be configured to store multi-byte character set on specific columns while the dates and column names and enumerations will remain in English.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

UPC UI Architecture

It shows the both client side and server side of the UI Architecture:



- The UI architecture has a client side and a server side. The UI client side is a web browser which can render HTML5 content. (Support for different browser versions and types). The HTML, JavaScript, CSS based pages are deployed as part of the WAR on the UI server side within Tomcat 7.x and loaded by the browser when the URL for the UPC UI is accessed by a user.

  The UI Server side API handles all the Ajax calls (HTTP/JSON) whenever the user selects or submits form data displayed in the UI. This in turn invokes the UPC client module which is a facade for the UPC Data service and the Security service that are running as web services within the Apache CXF container. The UPC client uses SOAP/XML to invoke both these web services. Note that for higher performance this could be just Spring based DI calls in future.

  When a user logs in, the security service authenticates the credentials and returns a SAML payload that contains the authorization claims in terms of the operations that can be invoked by the user. This is based on the roles of the user as well as the tenancy. The SAML context returned is stored in the session

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

context until the user logs out. The session context is destroyed on a logout or it can expire automatically if no user action on a session is detected for a configurable time period.

- The UI client side maintains a local session context where it tracks the current state of the display, the metadata corresponding to the state. As user actions trigger state changes, the context is updated and the corresponding state specific metadata is fetched from the server.

  The initial pages that are rendered, be it a login page or the operational page because of user actions result in Ajax calls from the client browser (the Event handlers) to the UI Server side API. The data sent by the client browser are either URL encoded or as XML or JSON format.

  The UI Server side acts as a facade class for the corresponding methods for these Ajax calls. Based on the operation and the attributes, the payload is converted into XML and passed on further to the UPC client. The UPC client implements the client side of the UPC DS WSDL. The UPC DS returns the response in XML. This is further passed back to the UI Server side API. The UI Server side API converts it to JSON or XML and returns it to the UI client side.

  The UI client side event handlers on a response invoke a rendering engine with the input payload in XML or JSON. The rendering engine uses metadata in the session context which contains tenant specific, i18N, l10N, and other control aspects (show/hide) on the UI components displayed for a specific state of the UI.

UI Requirement Analysis

Requirement analysis is the process of determining user expectation for a product.

Page Display

The UI components displayed in a page can be restricted based on user authorizations as present in the session context after a login based on tenancy. Each tenant can begin with a layout which is different from the other. As an example, based on the role + tenancy, a particular user may not be shown the Products tab or Offerings tab in the display.

The text and images statically displayed on a page can be different for each tenant.

Controlled Input

The form fill up by the user for some of the fields could be based on enumeration. This can be influenced by the locale settings. Since the values input by the user for these fields are based on enumerations, they are known in advance and the locale mapping for them will be defined based on tenancy. These field names and values will remain in English in terms of transmission to the UPC DS and further during persistence in database. This means presentation will be done in native locale while the persistence will use the equivalent English terms.

Un-controlled Input

In some cases, the user input to the forms such as a free form text or fields for which the values are not pre-defined by enumerations will be in the native language as chosen in the browser preference settings. The forms when submitted will result in transmitting this data in native language format to the server side. No translation to English can be done on this as there are no enumerations. The data in this case will be persisted in the native language in the database.

Schema Versus Data and i18n, l10n Aspects

With respect to a locale setting it is possible that there are differences in the way a schema and data are handled.

For example, given the display of this name value pair.

{Colour: Red, Size: 10, Date Of Creation: 10/10/2012}

Colour, Size and Date Of Creation describes the schema , the values corresponding to them are Red, 10 and 10/10/2012. To display the above in UI, both the schema and the data values need to be applied with the specific locale and will appear in the native language. This is true also for the input fields, where the schema and the data values (input by the user) are in the specific locale.

However, since the schema is controlled by the UPC application, there is a possibility to translate this at the UI server side to 'English' and keep only the values in the XML payload part of each schema element that is transmitted to the UPC web service and persisted. This means that the database tables and columns can be in English with the character set of the values stored in the table corresponding to the specific locales. This brings in reusability of table definitions across multiple language settings. And hence there is no need to maintain separate tables for each language in a multi-tenant model. This will also help to keep the Java code that performs the validation of the XML passed in and the response assembling code which returns a XML response on reads to be language neutral.

This is true for 'enumerations' of values of a particular schema element also.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Data Types

In case of a input of special data types such as Dates, numbers, calendar, it is important to understand that the UI widgets that are used as part of JQuery or any other tool is capable of supporting i18n / l10n or allows extensibility to do the same.

Appearance and Themes of the UI Based on Tenancy

Different tenants can have different possible color schemes applied to the page, the styling of fonts, the size of fonts, the layout, and the brand logo and so on. The architecture / design should allow this flexibility and enable this in a manner that is easy to do. For the scope of this architecture, these are expected to be given as guidelines as a part of the user guide of the UPC application and from the design, the HTML pages used by the UPC UI will be clearly segmented and identified to show how these impacts can be brought in.

Rule UI

The Rule authoring environment is part of BRMS. This has its own UI. This is different from the UPC UI. The compliance to i18n needs to be verified with BRMS. The rule definitions involve feeding the knowledge base with specific UPC schema and enumerated values to be used as part of the Rule condition definitions. In case the values of attributes defined as a part of the rules cannot be 'controlled' or cannot be enumerated and is supplied by the user, there is a need to maintain multi-byte locale specific character set in the rules file and use them for comparison during runtime when XML payload as above represent the attributes used in the rules. In this case, a comparison is required to be done in terms of multi-byte locale specific character set during the run-time execution of the rule. The support for this need to be verified within BRMS / Drools rule engine.

Error Messages, Statistics, Alarm and Audit Logs

All error messages, statistics, alarm and audit log dumps into log files by UPC application need to be enabled to support configurable locale specific messages/errors etc. If JConsole is used to render the above, the support for i18N in JConsole should be verified as well.

Multi-tenancy Triggered Extensibility

Extensibility involves extensions to UPC data model for different tenants that reflect in the UI when new Products or Offerings are created. Extensions to data model can be simple ones with just additional name-value pairs to an Object and can be more complex where new objects are added to the model to cater to tenant specific needs. The UI layer should allow rendering of the differences caused by the different data models.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Display of Large Payloads

In case of display of a large number of records in a paginated fashion by the UI client, there is a need to buffer the payload. This will result in a lower computing overhead on the browser.

UI Functional Architecture Elements

The functional architecture involves:

UI Display States

Metadata for Rendering

UI Client Side Rendering Engine

UI Client Side Processing Pattern

UI Server REST API

UI Server Session Context

UPC UI Client

UI Display States

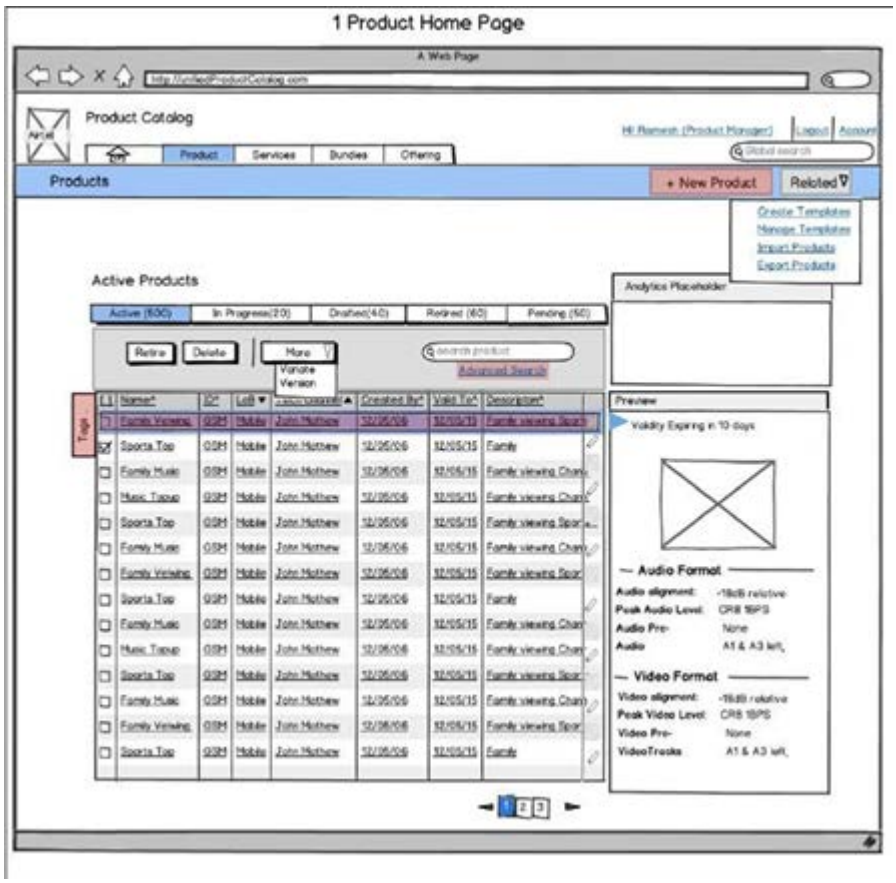
The UI display starts with an initial page and the contents of the page can contain the different HTML elements or JQuery/JavaScript based widgets. Further as a user performs an action on the UI elements that result in event handling, the outcome of the event handling results in the UI display entering a new state.

As shown in a sample state diagram below, every action by the user on a UI element in a particular state (a new page display) result in event handling to be done. The event handler on a successful response, causes a state transition.

> For example, when the UI client enters a Process_ List Products, it triggers the event handler for List Products. Upon success, the UI client state enters List Products. In this state, the display is rendered by the rendering engine based on the metadata.

Metadata for Rendering

The following UI page from UPC is taken as an example for describing the metadata structure and semantics. This page is reached after a login from the UPC login page.



Metadata for Simple UI Elements

When a user logs in successfully, the above screen shows a Header area where the brand logo and few operation links like logout, Account and the greeting text of 'Hi Ramesh' are shown. Apart from this, there is a tab area where the tab headers are displayed with 'Product', 'Services'...etc. The Grid is displayed only when selection of a Product. So it is empty on a successful login state.

The metadata that represents this state of 'login'.

The metadata is comprised of 'State' keyword followed by one or more UI component (widgets or elements) and one or more attributes for each UI component. The metadata primarily describe the Name of the state (Login), the identifiers as per the HTML markup of the UI elements that have to be shown to make them visible, the UI identifiers as per the HTML markup of the UI elements that need to be hidden to make them disappear and the specific data that is obtained from the event handling that need to be rendered into the particular UI element in case of a 'show'.

> For example, in the metadata shown above, the name of the state in which this metadata is applicable is 'Login'. The Show has the different UI elements to be shown.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

- UICompBrandAreaGreeting is the id of the UI element which has to be shown and which represents the 'Hi Ramesh' text area in the above UI. The 'Ramesh' text comes from the Login event handler and is available as part of the data cached by the login event handler on the same UI element (UICompBrandAreaGreeting) in a key called 'displayName'. The $(displayName) indicates to the renderer that this is a variable and it has to read the cache corresponding to the UI element. The concat:true attribute tells the renderer that append the displayName value to existing text which is already in the HTML markup as 'Hi'.
- The metadata above can be easily extended to add additional controls such as the CSS styling by adding additional attributes which JQuery readily offers on display elements.
- Thus adding a css attribute as below can be used to modify the font, color, size of the element dynamically.
- when a user logs in, there is a need to display tabs showing the 'Product', 'Services', 'Bundle', 'Offerings'. Depending on the tenant specific policy, we may need additional control on the display of these tabs in terms of number of tabs as well as the text displayed on each of them.
- The JQuery UI tabs plugin expects the tabs to be specified as an unordered list as follows.
- In order to specify the text of the tabs dynamically with 'text' for all the 'a' elements in the the metadata specifies this aspect which is used by the rendering engine to display them.
- In the above, the metadata specifies there are two tabs with names 'Product' and 'Offering' for this user and the special type tabs tell that this is for a widget whose name is tabs and hence the renderer has to handle this based on the 'Jquery UI tabs' plugin.

Metadata for Widgets

In Figure 14, when a user selects a tab such as a 'product', the corresponding event handler retrieves all the Products in XML and displays it in a Grid. If we use flexi grid jquery widget to display the Grid, then the metadata will be as follows for the Grid element which is identified as UI Comp Grid.

Since this widget is a JQUERY plugin and offers the controls via the following name value pairs, the metadata for this can directly include the relevant controls. For example, the columns of the grid and its characteristics can be specified via the col Model attribute and the URL sources data directly from a script (which can be a java script) that produces the data in XML or JSON. Thus, if a query on Products returns a XML with Product records that is aligned to the col Model columns, the data is readily seen in the Grid. The metadata in this case will exactly have the attributes of the flexi grid as a part of it.

The renderer on seeing the type as flexi grid for the UI element UI Comp Grid displays the flexi grid with data.

Thus to a large extent, the metadata for widgets in the UI will align with the parameters of the specific JQUERY widget.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

UI Client Side Rendering Engine

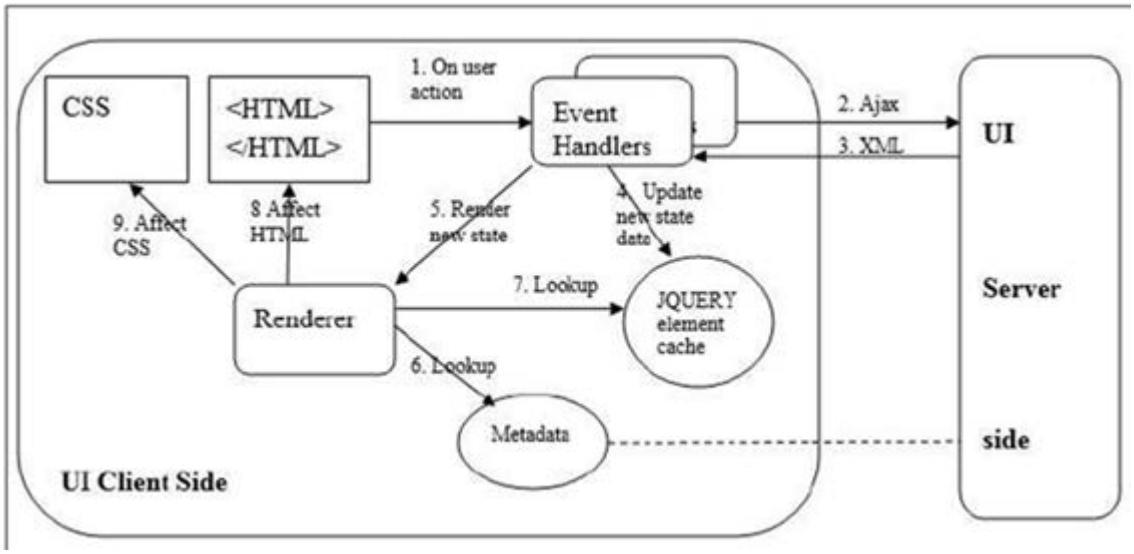
1. The rendering engine does the following
2. Read the metadata for the specific state

For each UI element specified in the show, do a JQuery show() with the following:

- **a.** Apply any global attribute in metadata to the selected element.
- **b.** If the type of the UI element corresponds to a widget, do widget specific attribute settings.Else, for each attribute, select the UI element and set the attribute as in metadata
- **c.** For each UI element specified in the hide, do a JQuery hide().

UI Client Side Processing Pattern

The following pattern will be in general followed at the UI client side.



The HTML and CSS for the initial page is retrieved from the UI Server side. All the UI elements are present in the HTML, but only those that are needed for the current state are shown. Rest are kept hidden.

When a user action happens on the UI, the event handler which is setup in Java script on the specific UI element sends one or more Ajax requests to UI server side. The response will be in JSON or XML. The event handler caches the responses into the JQUERY data cache of the appropriate UI element which will be shown in the new state. The event handler calls the renderer.

The renderer uses metadata for the new state and the data corresponding to the UI element in the JQUERY element cache and then result in affecting the CSS and HTML of the chosen elements of the new state.

UI Server REST API

This will be a very thin layer of HTTP Servlet methods running in Tomcat that handles GET and POST actions as a result of event handling in the UI client side. The REST API will invoke the UPC UI client component via Spring DI and obtains the response. The UPC UI client component implements the client side of the WSDL exposed by the UPC DS and Security Web service. It uses the session context to store the credentials and the metadata of the sessions corresponding to the user who has logged in based on tenancy and roles. It passes back the XML obtained from the UPC UI client as response to the Ajax calls.

UI Server Session Context

The session context will hold name-value pairs corresponding to a specific user session. In general it will hold the:

- User credentials (login name, display name)
- Tenancy (domain name)
- UI Metadata (JSON string) specific to this use
- State of the UI client (screen state)

The session context will come to life when a user successfully authenticates and it is freed up when a user logs out. A copy of metadata corresponding to a group will be maintained in a file in every node of the UPC cluster. The file name and location will be maintained in the LDAP along with the user/role/tenant. At this point, the hierarchy in the LDAP tree will have tenants and groups and users within a group.

To keep it simple, the metadata will be maintained per group within a tenant. Thus all users of a group will have the same metadata. Upon a user authenticating successfully, the metadata location file specified in the LDAP will be made available as part of the SAML context resulting in the UPC client loading it from the file system and keeping it as part of the session context. As the UI client changes its states, the corresponding metadata for that state will be passed onto the UI client. This will be used by the UI client renderer until there is a state change again.

The Context component defined in the UPC backend side can be reused here.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

UPC UI Client

This exposes an API interface that is invoked via Spring DI by the UI Server API that handles the Ajax calls from the UI client. This primarily interfaces with UPC Data service and the Security service and updates the session context and/or returns the response to the UI Server API. This is a thin facade for the underlying services.

UI Functional Flows

Flow for Create Product and Query Product to be added from UI client to UPC Data Service.

Non-functional Flows

The metadata used by the client side UI can allow variations for different tenants the following aspects of the UI. This will be more detailed in the high level design document of UPC.

- I18N and l10N
- Policy based rendering
- Data model extensibility
- Appearance and Styling

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

Definitions

NGM Next Generation Messaging

UPC Unified Product Catalogue

CAP Common Application Platform

AIS Application Integration Service

CLM Customer Lifecycle Management

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template

APPENDIX 2: APPENDIX

Feedback

xxxxxx endeavours to provide accurate and useful documentation for all products. To achieve this goal, the documentation group welcomes your comments and suggestions regarding any aspect of user documentation.

Send your comments by e-mail to: xxxxxx@xxxxxx.com.

Trademarks and Registered Trademarks

Products and product names mentioned in this document may be trademarks or registered trademarks of their respective owners.

Trademark

Copyright

Copyright ©Xxxxxx Corporation 2015. All rights reserved. No part of this document may be reproduced, distributed, stored in a retrieval system or translated into any language, in any form or by any means, electronic, mechanical, magnetic, optical, photocopying, manual or otherwise, without the prior written permission. For additional copies of the document, please contact by e-mail: xxxxxxx@xxxxx.com.

Disclaimer

Xxxxxx makes no representations or a warranty with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, xxxxxx reserves the right to revise this publication and to make changes from time to time in the contents hereof without obligation to notify any person of such revision or changes.

©<Default ¶ DocEmpo LLC Font>Copyright, October 2023 - FrameMaker Template