Packaging for Organization and Reuse

Overview

Packaging for **organization** only provides the ability to compartmentalize your metadata. This helps to manage and maintain a package[1]. Packaging for **reuse**, adds another dimension which allows components to be extended (not modified). Modifying a package component may negate the reuse component if it requires the package developer to make changes.

This document follows the architectural decomposition document, ***CRM – Architectural Decomposition***, guiding principles and steps outline from a Top-down approach. In addition, this document explores the reuse attribute of packaging with a specific Point-Of-View (POV). This POV assumes the designer is familiar with Object-Oriented (OO) Principles as covered by Robert Martin and/or Martin Fowler.

Package Reuse

Context

Designing a Package for Reuse will depend on how one dissects the problem statement. If the problem statement is addressing constant or static values (i.e., custom metadata, fields, permission-sets, profiles, etc.) then reuse is the values themselves. There are areas or data protection, etc. but this POV addresses the behavioral side. That is, more attune to a polymorphic behavior.

The sections that follow will define the problem statement and then decompose our problem statement into personas, roles, responsibilities, and layers to help manage the problem. Please note, the document is not showing all aspects, such as,

- User Stories and various scenarios,
- UML Behavioral Diagrams,
- UML Static Diagrams,
- CRUD Matrix and CRC Cards
- TDD (Test Driven Design)

Problem Statement

A Bank Representative would like to have the ability to search against different *sinks* for various attributes. These *sinks* could be a Salesforce Org, external host, etc. In addition, the Bank Representative would like to provide different elements to search for, such as Salesforce Org and/or external systems to locate customers/prospects with multiple search criteria.

[1] It is assumed you follow what is considered a package component (see Appendix)

Personas

The personas are rather vague[2] in the context of the problem statement. This will require us to consider a Bank Representative in multiple Business Units (i.e., Wealth Management, Home Lending Customer Care, etc.). In addition, the context will change the meaning of what a customer or prospect means.

These personas classify our roles and responsibilities. And each role and responsibility will utilize different resources (i.e., objects, permissions, profiles, configurations, etc.). A CRUD Matrix, CRC Cards and TDD come to mind in the analyses.

Extending Reuse

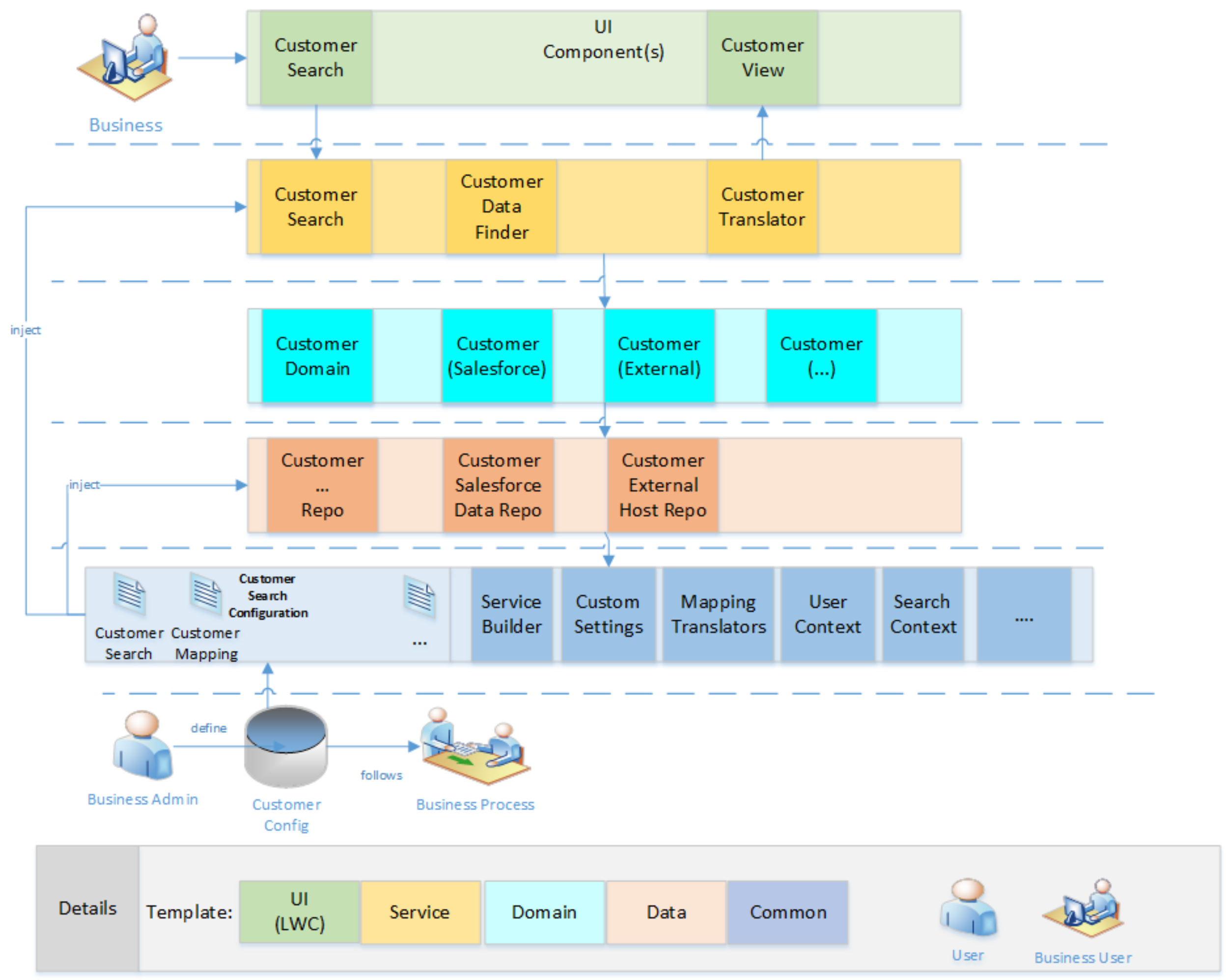
Given the above ***Problem Statement***, we begin to break the scenario further into layers for manageability,

- User Interface (UI)
- Business (Service) Logic
- Domain (Business Objects)
- Data Logic (Repository/Selector)
- Common Logic (Cross-Cutting Concerns, Utilities, etc.)

The above is a boilerplate layered design. Each of the above components may be a package itself. For dependency injection, one can use Force-DI[3] or other Creational Design Patterns.

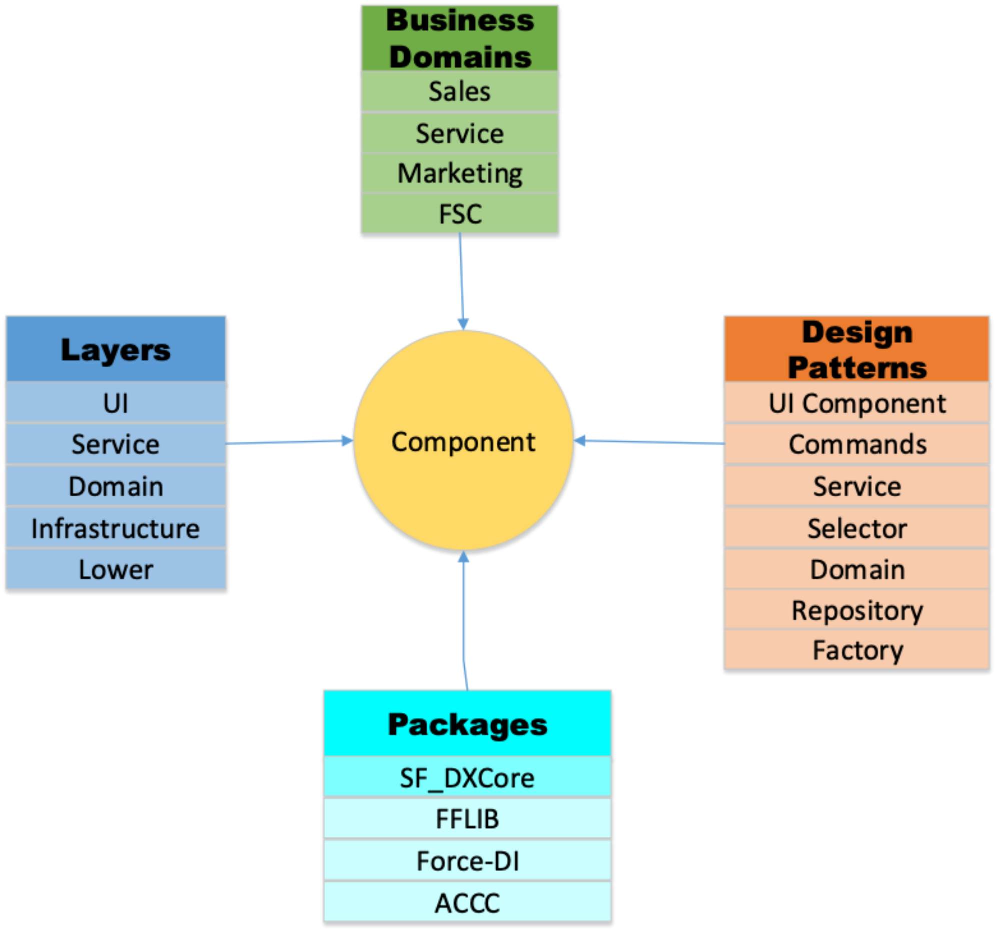
[2] This problem statement is vague on purpose to ensure one can identify volatile areas

[3] Substitute your own DI with Factories, Builders, Product Traders along w/ constructor or method injection

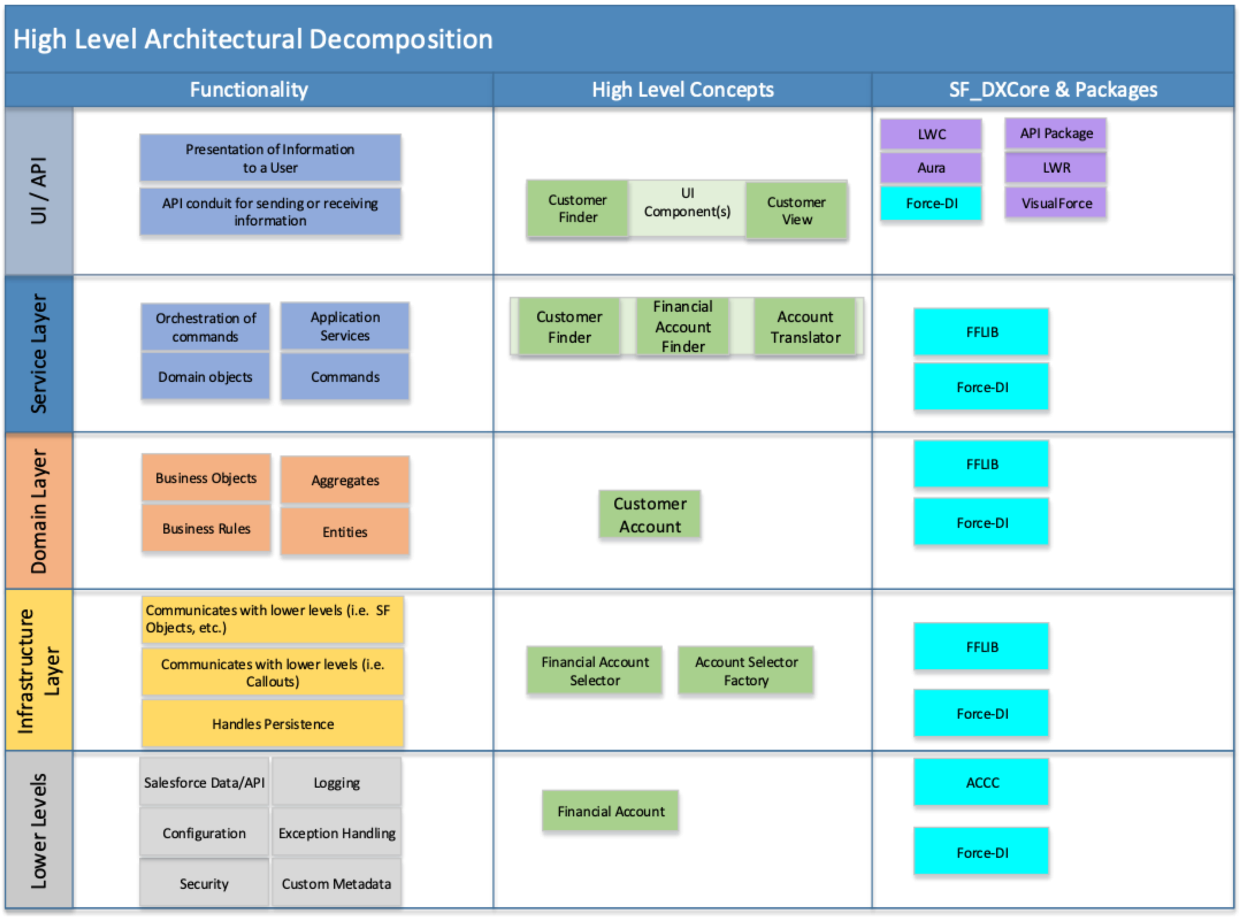


The next discussions analyze the problem statement by layer (i.e., UI, Service, Domain, etc.)

Appendix: Aspects of a Component/Package



Appendix: Layered Design



High Level Architectural Decomposition

	Functionality	High Level Concepts	SF_DXCore & Packages
UI / API	Presentation of Information to a User; API conduit for sending or receiving information	Customer Finder; UI Component(s); Customer View	LWC; API Package; Aura; LWR; Force-DI; VisualForce
Service Layer	Orchestration of commands; Application Services; Domain objects; Commands	Customer Finder; Financial Account Finder; Account Translator	FFLIB; Force-DI
Domain Layer	Business Objects; Aggregates; Business Rules; Entities	Customer Account	FFLIB; Force-DI
Infrastructure Layer	Communicates with lower levels (i.e. SF Objects, etc.); Communicates with lower levels (i.e. Callouts); Handles Persistence	Financial Account Selector; Account Selector Factory	FFLIB; Force-DI
Lower Levels	Salesforce Data/API; Logging; Configuration; Exception Handling; Security; Custom Metadata	Financial Account	ACCC; Force-DI