



210823 상품팩토리 & 아키텍처1

🕒 생성일	@2021년 8월 23일 오전 9:09
▼ 성함	나한주
🔗 속성	
▼ 수업 유형	이론
🕒 수정일	@2021년 8월 24일 오전 2:28
👤 작성자	현동빈

오전 : BX-CBP_기본교육_PF제품소개

상품팩토리(PF)

상품 정보 범위

Application Map

상품팩토리 Requirement

상품조건의 유형

아키텍처

CBP의 특징

어플리케이션 컴포넌트 & 레이어 Overview

서비스 컴포넌트와 베이스 컴포넌트

서비스 & 베이스 컴포넌트의 분리

어플리케이션 맵 > L0

어플리케이션 맵 > L1

어플리케이션 맵 > L2

어플리케이션 맵 > L2

어플리케이션 계층

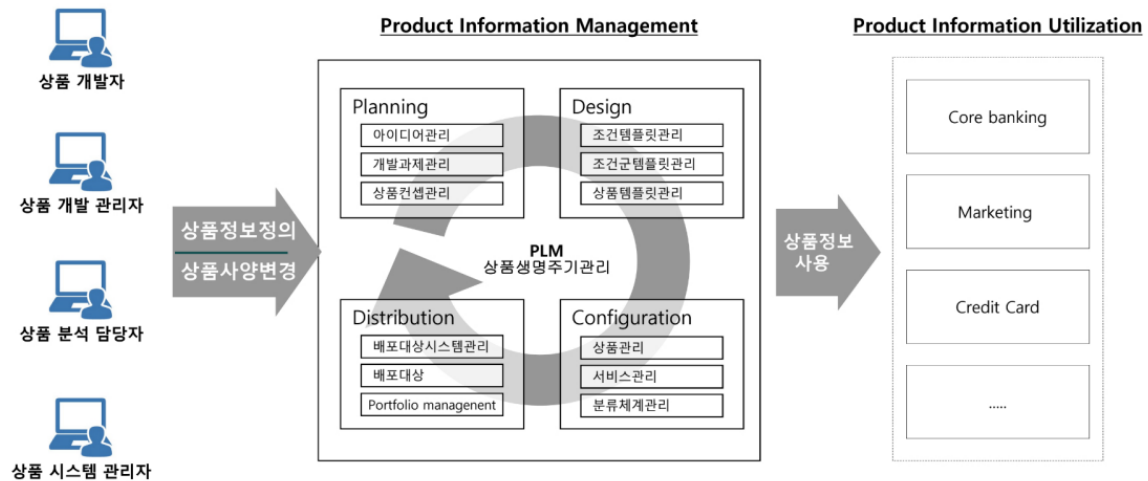
서비스

업무 로직 레이어

도메인 레이어

오전 : BX-CBP_기본교육_PF제품소개

상품팩토리(PF)



PLM : 상품생명주기관리

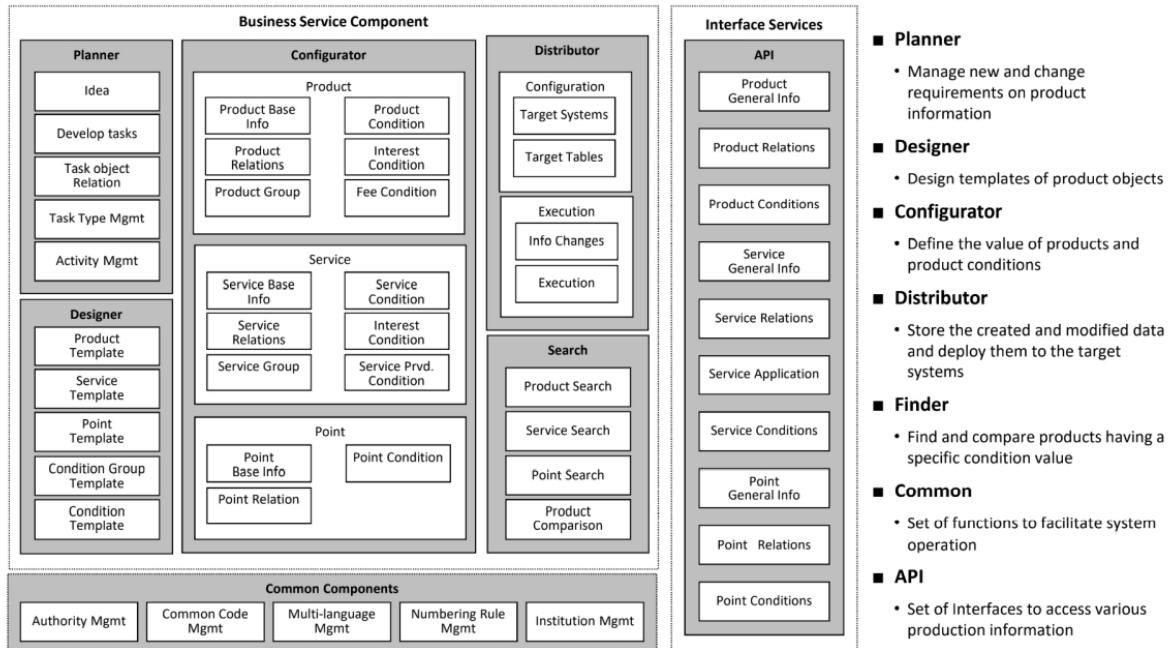
- 상품생명주기관리 과정 속지

상품 정보 범위

상품정보의 유형	설명
상품 기본 정보	• 상품명, 상품 판매 기간, 상품 판매 가능여부(상품 상태)
상품 조건 정보	• 상품의 특성을 상품조건이라고 명명하여 관리 예시) 가입대상, 가입채널, 가입기간, 납입금액, 이자지급방식, 적용이율, 만기후이자율
상품 관계 정보	• 상품간 관계 • 상품과 부점과의 관계, 상품과 문서와의 관계,...
금리체계정보	• 상품에서 사용하는 금리 체계 정보 • 금리종류별로 금리를 계산하기 위한 구조를 관리 예시) 적금중도해지금리체계 : 중도해지이율 적용 예금담보대출 금리체계 : 수신금리 + 가산금리 가계일반자금대출 금리체계 : 약정이율 - 우대금리 신용대출 금리체계 : 기준금리 + 유동성프리미엄 + 업무원가 + 리스크프리미엄
상품분류정보	• 상품을 다양한 기준으로 분류하는 경우, 어떤 분류에 속하는지의 정보 예시) 대출자금융도에 따른 대출 상품의 분류 판매대상 고객군에 따른 수신 상품의 분류
상품 템플릿 정보	• 상품 템플릿은 상품을 정의하기 위한 틀을 이야기함 • 상품 템플릿은 상품을 정의하기 위해 어떤 상품 특성을 정의해야 하는지의 목록을 가지고 있음 예) 보통예금 상품은 상품특성중에서 예금원가결산관련 특성을 정의해야함 대출상품은 상환특성 정보를 정의해야함

상품분류정보 ≠ 상품템플릿정보

Application Map



- Planner
- Designer
- Configurator
- Distributor
- Search

⇒ PF에서 확인가능

상품팩토리 Requirement

PF 요구사항

- 다양한 신규요건을 테이블 변경 없이 적용할 수 있어야 함.
 - 다양한 요건 수용시 잦은 테이블 변경으로 신속한 상품개발이 용이하지 않음.
- 모든 상품정보를 수용할 수 있어야 함.
 - 데이터 구조 제약으로 업무팀에서 관리하는 경우 존재함.
 - 하드코딩으로 반영함에 따라 유지보수가 용이하지 않음
- 상품정보 통합 관리가 용이해야 함.
 - 모든 상품정보를 동일한 구조로 정의하기 어려움
- 상품정보 정의가 용이해야 함.
 - 관련 지식 부족한 담당자로 인한 정보 등록 오류 발생함.
- 상품정보의 정확성을 제공 해야 함.
 - 시스템들이 대고객 기준 동일한 정보 제공할 수 있어야 함.



BX 상품팩토리

- 1 업무와 독립적인 데이터구조**
 - 추상화된 데이터 모델 기반
 - 다양한 신상품 및 서비스출시 요구에 테이블 및 프로그램 영향 없이 대응 가능함.
 - 처리 업무 어플리케이션 변경 최소화
- 2 템플릿 기반의 상품개발 체계**
 - 오퍼레이션 에러 최소화
 - 기능별 역할 분리
- 3 상품정의 용이**
 - 상품복사 기능
 - 상품템플릿 정의 용이한 구조
- 4 상품정보 일관성 유지**
 - 전행적으로 동일 상품정보 유지할 수 있는 배포 기능 제공



상품조건의 유형

- 목록조건
- 범위조건
- 금리조건
- 수수료조건

⇒ 각각의 상품조건 유형 파악

아키텍처

CBP의 특징

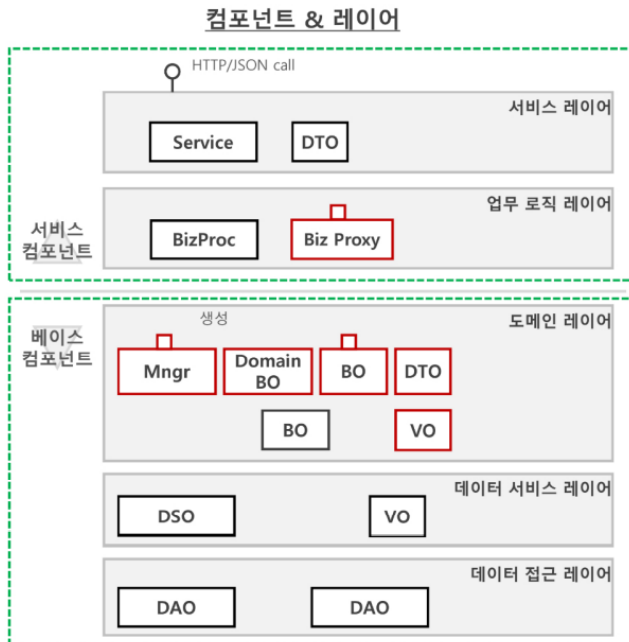
솔루션 명	솔루션의 특/장점
	<ol style="list-style-type: none">1. 독립적인 상품정보 관리 업무처리 로직과 상품정보의 완벽한 분리화2. 제약 없는 상품(조건) 확장 어떠한 상품의 조건도 단일 모델로 모델변경 없이 반영3. 신속한 상품 출시 상품정의 기준으로 기존 템플릿 사용시 10분 이내 정의 가능함4. 검증된 국내/외 사례 국내: 기업은행, 현대카드, 케이뱅크 / 중국: 공상은행, 마이뱅크
	<ol style="list-style-type: none">1. 조립식 서비스 신규 (Assemblable) 광범위한 업무공통 API를 조립하는 방법의 서비스 신규2. 데이터 정의에 의한 용건 대응 (Configurable) 광범위한 업무 요건이 프로그램 변경없이 대응 가능3. Plug-In 방식의 신규요건 대응 (Pluggable) 기존 프로그램에 변경 영향이 없는 Plug-In 방식의 신규 기능 대응4. 표준화된 개발/운영 방식 (Standardized) 자체 메타모델이 탑재된 표준 사전을 기반으로 모든 자원의 표준화 담보

- Assemblable
- Congigurable
- Pluggable
- Standardized

⇒ CBP의 특징을 구분하고 각각의 차이점을 명확히 알기

어플리케이션 컴포넌트 & 레이어 Overview

어플리케이션은 크게 2개의 컴포넌트(Service, Base)로 구분되며, 5개의 레이어(Service, BizProc, BO, DSO, DAO) 구조를 가지고 있다.



Componet

Service Component

- 화면을 가지며 화면제어 또는 서비스를 제공
- 베이스 컴포넌트를 이용한 코어 서비스 제공

Base Component

- 데이터 관리 기능을 가지는 BO
- 은행 공통의 기본 기능을 제공
- 각 베이스 컴포넌트는 각각의 주제 영역에 해당하는 정보의 생성 및 제공 등의 책임을 가진다.

Layer

Service Layer

- 독립적인 단위의 기능 제공.

BizProc Layer

- 공통 또는 복잡한 로직, 필수 아님.
- 일부를 BizProxy를 통해 외부에 제공.

Domain Layer

- 핵심 업무 기능 제공, Base API.
- CoreBO, Manager, BO

DS Layer

- BO와 DAO 사이에서 다중DBMS의 지원.

DA Layer

- Java Interface와 SQL을 포함하는Mapper로 구성.

컴포넌트의 종류

5개의 Layer

업무도메인과 서비스 구분하기

BizProc을 Biz Proxy로 감싸고 컴파일하면 다른 BO에서 사용 가능

일부 Biz Proxy를 외부(다른 BO)에 제공 ⇒ 서비스 중에 일부를 떼어내서 메소드처럼 나중
에 다른 곳에 재사용 가능

BO끼리는 호출을 바로 해서 사용하면 X

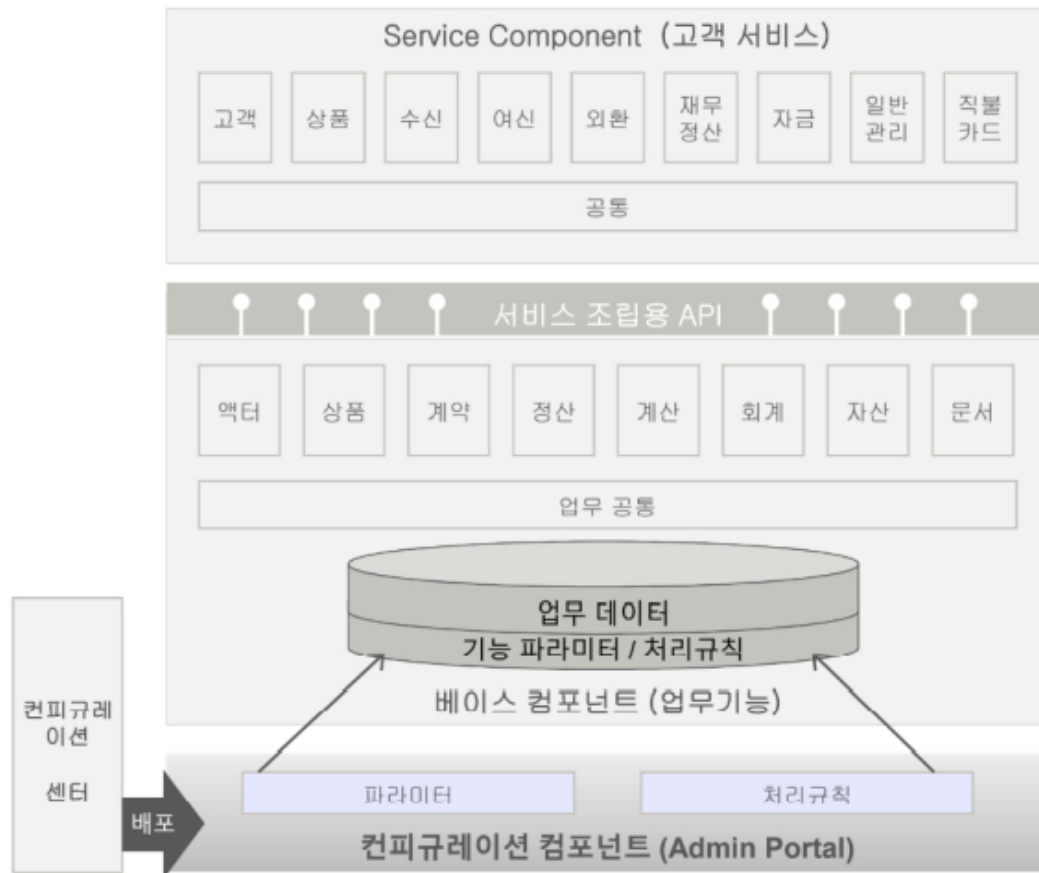
Ex) 액터와 회계를 서로 호출하여 사용 X

→ BO는 서비스 컴포넌트로 올라감

→ 서비스 컴포넌트 내에 BizProc을 이용하여 외부(다른 BO)에서 사용할 수 있도록 함

→ 회계에서 서비스컴포넌트의 BizProc에 접근하여 액터의 일부 메서드 호출 가능

서비스 컴포넌트와 베이스 컴포넌트



컴포넌트

- CBP 어플리케이션 맵은 다수의 컴포넌트로 구성됨.
- BXM 프로젝트는 독립적인 컴포넌트 CBP구현체
- BXM 기반의 Application 관점에서 컴포넌트는 하나의 BXM 프로젝트와 매칭됨
- BXM에서 ~Svc : 서비스컴포넌트
- ~BAT : 배치
- CORE-Intrfc : 코어인터페이스

서비스 컴포넌트(업무단위)

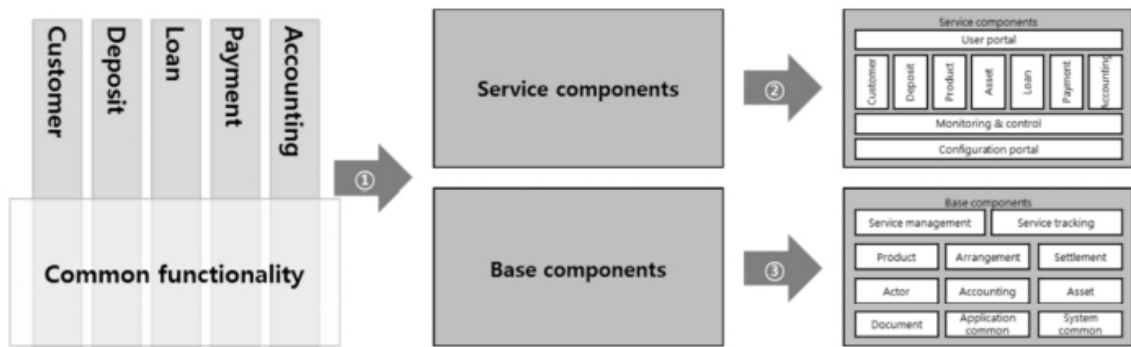
베이스 컴포넌트(단위기능)

⇒ BO가 아닌 것 구분 할 수 있어야 함

⇒ 액터, 상품, 계약....은 BO, but 수신은 BO가 아님

서비스 & 베이스 컴포넌트의 분리

- 업무 기능에서 처리 흐름 분리
- 서비스 컴포넌트는 각각 **비즈니스 라인**으로 구분(LOB)
- **재사용** 가능한 업무기능은 업무 객체로 분류하여 컴포넌트화



서비스프로젝트

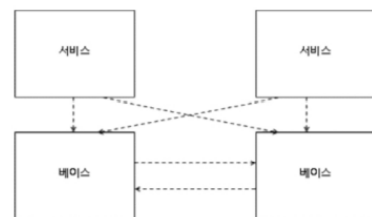
- 서비스프로젝트는 어플리케이션 계층에서 서비스 계층과 업무로직 계층으로 구성하는 프로젝트이다.
- 서비스프로젝트는 온라인프로젝트와 배치프로젝트를 포함한다.
- 서비스프로젝트는 데이터모델을 보유하지 않고, API를 제공하는 베이스프로젝트를 참조(Reference)한다.
- 서비스프로젝트는 LoB(Line Of Business)를 중심으로 식별된다.
예)수신서비스, 여신서비스, 고객센터

베이스프로젝트

- 베이스프로젝트는 어플리케이션계층에서 도메인 계층과 데이터서비스 계층, 데이터접근 계층으로 구성된 프로젝트이다.
- 베이스프로젝트는 온라인, 배치와 무관하게 구성된다. 즉 온라인이나 배치나 동일한 API를 호출할 수 있다.
- 베이스는 재사용 가능한 업무기능(Reusable Business Function)을 컴포넌트화하였다.
- 베이스프로젝트는 데이터모델을 보유하고, 컴포넌트 내의 데이터 정합성에 대한 책임을 진다.
- 베이스프로젝트는 **보유 데이터를 중심으로** 식별된다.
예)계약, 정산, 회계, 상품

프로젝트 간 참조관계

서비스프로젝트는 API를 사용할 베이스프로젝트를 참조하고, 베이스프로젝트는 상호간에 참조된다.
서비스프로젝트는 연동거래 시 대상 서비스프로젝트를 참조한다.

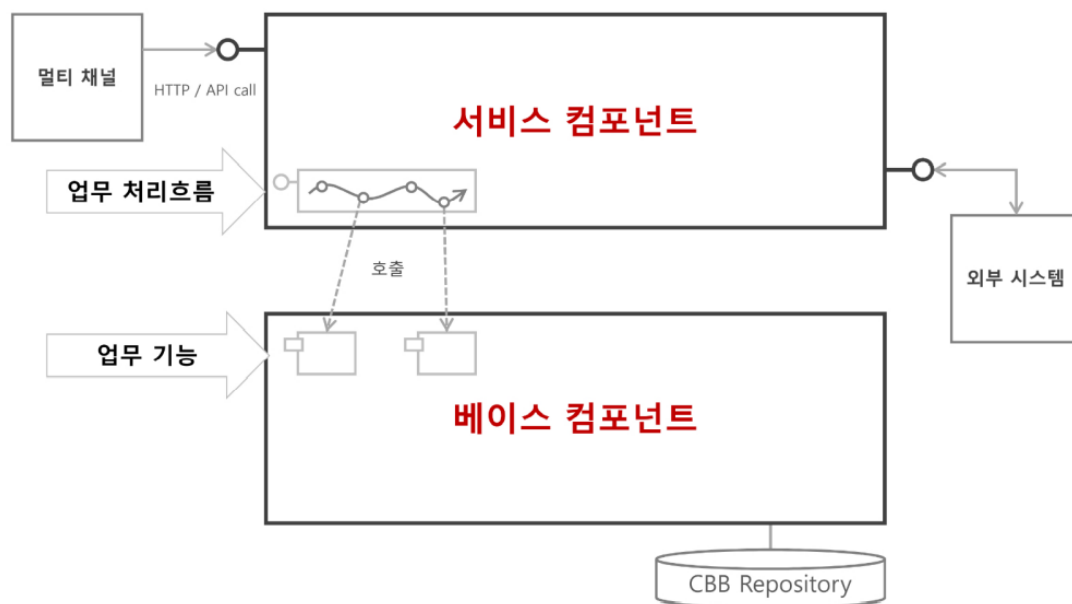


참고) 베이스프로젝트 간에는 순환참조가 발생한다

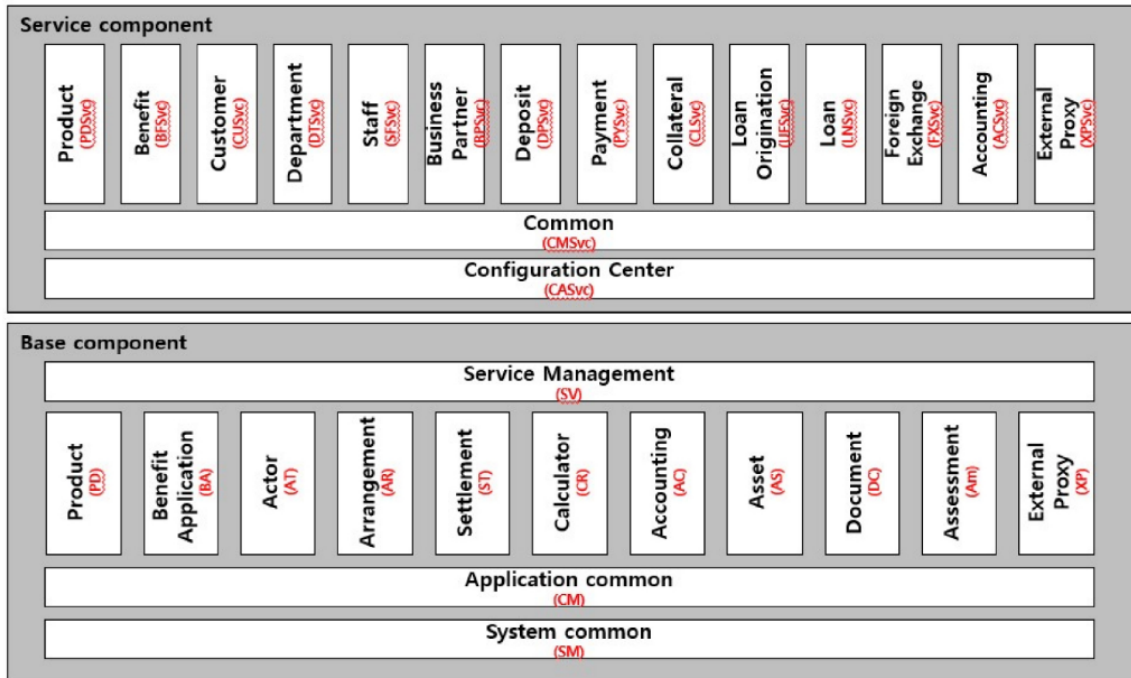
- 온라인프로젝트 : 실시간
- 배치프로젝트 : 일정기간 일괄해서...
- 서비스프로젝트는 데이터모델을 보유하지 않고 API를 제공하는 베이스프로젝트를 참조함(서비스프로젝트에서는 로직만)
- 서비스프로젝트는 LOB 중심 식별

- 베이스프로젝트는 온라인이나 배치나 동일한 API를 호출할 수 있음
- 베이스프로젝트는 재사용 가능한 업무기능을 컴포넌트화
- 베이스프로젝트는 데이터모델을 보유하고, 컴포넌트 내의 데이터 정합성에 대한 책임을 짐
- 베이스프로젝트는 보유 데이터를 중심으로 식별
- 서비스와 서비스 간의 호출 : 연동거래

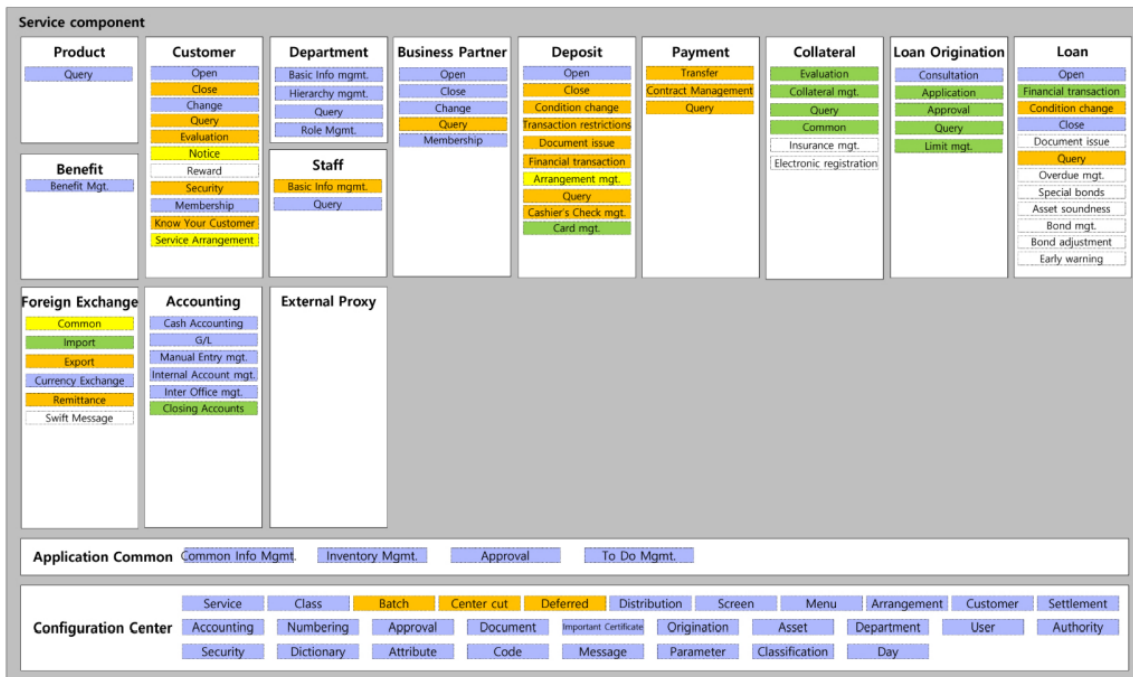
어플리케이션 맵 > L0



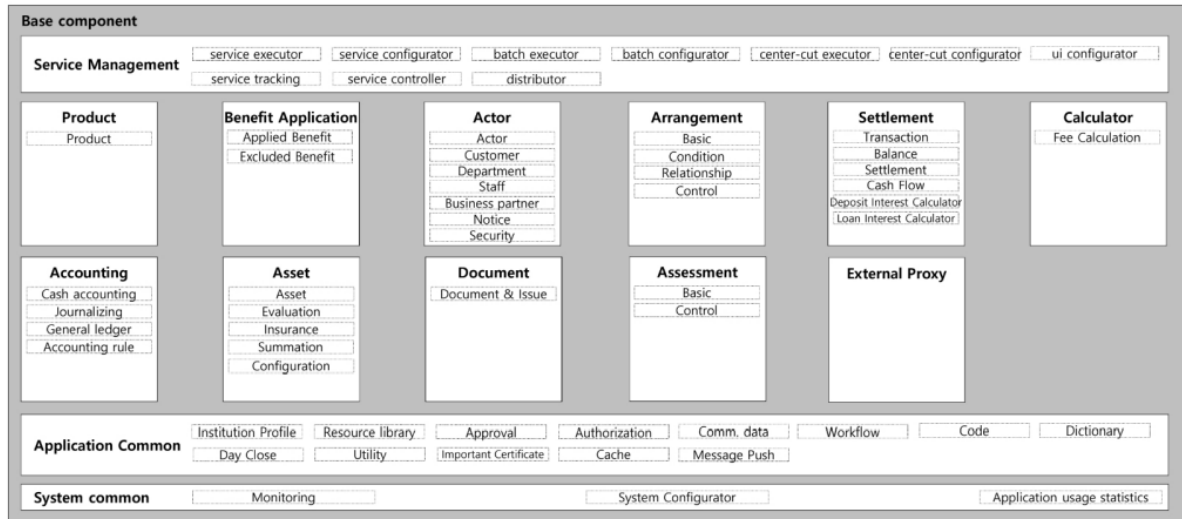
어플리케이션 맵 > L1



어플리케이션 맵 > L2



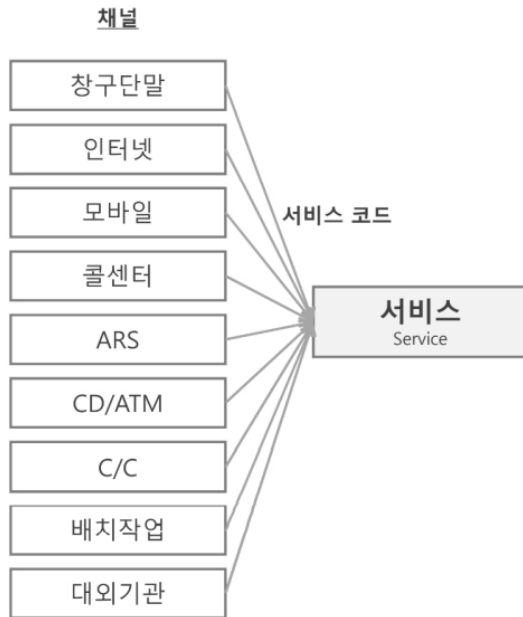
어플리케이션 맵 > L2



어플리케이션 계층

- 서비스
- 업무 로직(Optional)
- 도메인
- 데이터 서비스
- 데이터 접근

서비스



정의

서비스(Service) 계층은 CBP외부에 의미 있는 독립적인 단위의 기능을 제공하는 계층이다. CBP외부란 채널을 의미한다.

서비스(Service)는 필수계층이다. CBP가 외부에 기능(Function)을 제공하기 위해서는 그 기능을 반드시 서비스로 구성해야 한다.

서비스코드

- 채널에 제공되는 독립적인 단위 1개가 서비스코드가 된다. 채널에서는 서비스코드를 지정하여 호출하는 방식으로 CBP시스템에 독립적인 단위의 기능처리를 요구하게 된다.
- 서비스코드 1개는 서비스Class의 Operation 1개와 반드시 1:1로 매칭된다.

서비스의 채널독립성

- 서비스는 기본적으로 채널독립적으로 구성된다. 서비스 채널에 따라 사용하는 기술이 TCP소켓, http 등과 같이 상호 다르다고 하더라도 이것은 어플리케이션 프레임워크에서 적절한 ServiceEndpoint를 제공하여 대응하도록 하므로, 동일한 기능이라면, 중복되는 서비스를 만들지 않도록 한다.
- 채널독립성의 대외기관 예외
- 다만, 내부채널이 아닌 대외기관에서 서비스 호출에 대해서는 동일 기능이라고 하더라도 서비스를 분리한다. CBP에서는 대외기관으로부터의 서비스요청은 중계로그의 기록 등의 추가적인 처리를 하기 위해서 서비스코드를 분리해서 식별하도록 하고 있다.

BO 호출

- 서비스가 채널에 제공하는 기능(Function)은 서비스가 다양한 베이스프로젝트의 해당 단위기능을 제공하는 BO에 위임(delegation)을 함으로써 처리된다. 따라서 서비스는 기본적으로 적절한 BO를 처리흐름에 따라 호출하는 것으로 구성된다고 볼 수 있다.
- 공통적인 BO 호출 흐름을 별도의 업무로직(BizProc) 클래스로 추출할 수 있다.(Extract)

서비스입력검증

- 서비스의 입력검증은 기본적으로 업무선처리에서 서비스입력검증 컨피규레이션 정보를 이용하여 처리한다. 자세한 사항은 [서비스입력검증을 참조한다](#).
- 따라서, 서비스입력검증과 관련된 코드는 서비스클래스에 작성하지 않도록 한다.

서비스출력조립

- BO의 호출 결과에 따라 서비스의 정상 출력을 정확히 조립하여야 한다.

서비스연동

- 서비스 연동은 서비스에서 타 서비스를 F/W연동메커니즘을 이용하여 호출하는 것을 의미한다. 즉, 필요에 따라 이미 만들어진 채널제공용 독립적인 기능을 서비스에서 재사용할 수 있다는 것을 의미한다. A계좌에서 B계좌로 이체하는 서비스는 A계좌출금서비스와 B계좌입금서비스를 연동하는 방식으로 구성한다.
- 서비스를 연동하는 처리기능은 서비스계층 또는 업무로직 계층의 역할이다.

업무 로깅을 위한 조립

- 서비스는 서비스처리과정에서 발생한 업무처리내용을 아키텍처에서 정한 레이아웃으로 로그를 조립하여야 하는 책임을 지닌다.

구현 필수 제약사항

- 입출력을 OMM객체로 구성하여야 함
- 반드시 입출력을 가져야 함. 업무적으로 출력이 별도로 필요 없다고 하더라도 강제로 출력OMM을 만들
- BxmService로 구현하여야 함
- 입출력OMM은 동일 서비스클래스 내에서만 공유될 수 있음.

⇒ 각각의 설명을 잘 읽고 이해

업무 로직 레이어

정의

- 서비스업무로직(BizProc, Business Procedure) 계층은 다음의 두가지를 구현하는 계층이다.
- 1) 개별 서비스의 처리흐름 중 여러 서비스에서 공유되어야 하는 공통의 처리흐름과 2) 서비스의 업무처리흐름이 복잡하여 별도의 독립적인 클래스로 분리하고자 할 때 구성하는 계층이다.
- 서비스업무로직(BizProc)은 필수계층이 아니다. 필요에 따라서 식별된다. 즉, 서비스에서 BizProc없이 BO를 바로 호출할 수 있다.

처리 기능

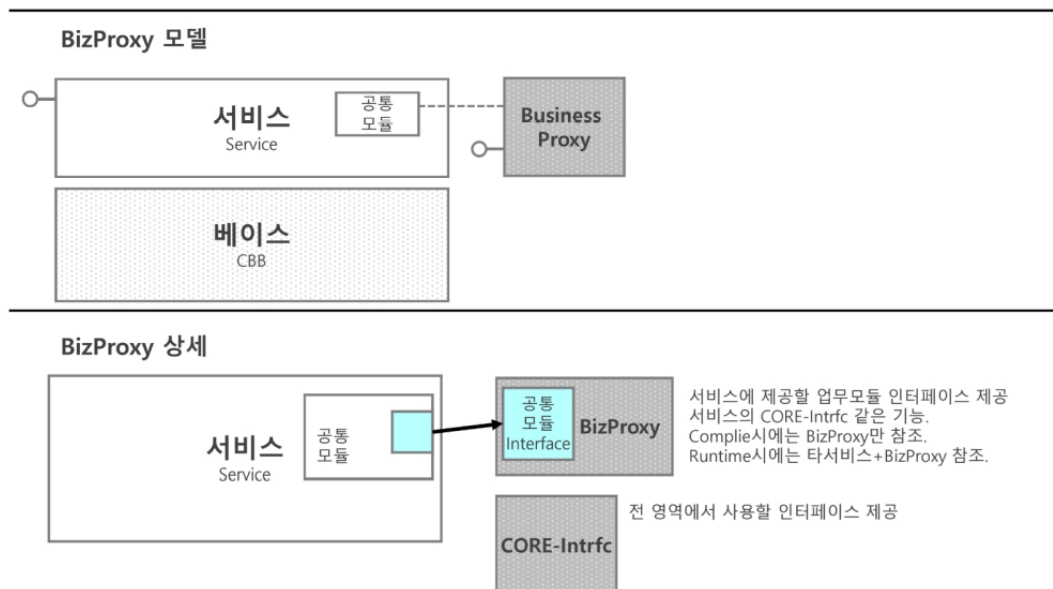
- BO호출이라는 측면에서 서비스와 동일하다.

구현 필수 제약사항

- BxmBean으로 구현하여야 함
- 동일 서비스프로젝트 내에서만 공유되며, 서비스프로젝트 간 공유는 되지 않음

연동거래는 아니며, util처럼 타 업무 서비스를 호출해야 하는 경우 Business proxy를 통해 타서비스에 업무모듈을 제공하도록 한다.

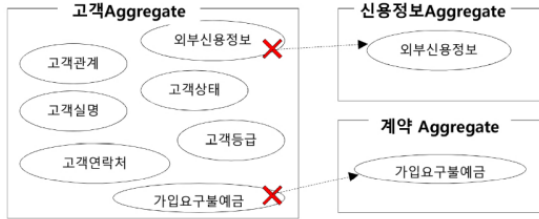
개발에서 소스상에 참조를 BizProxy 하나로 통일하며 Runtime 시의 서비스간의 참조는 연동서비스와 동일



- BizProc의 역할 이해
- BO → 서비스 컴포넌트 BizProc → 외부 BO(일부 기능 호출)
- CORE-Intrfc(코어 인터페이스)
- 서비스는 통째로 다른 서비스를 호출

도메인 레이어

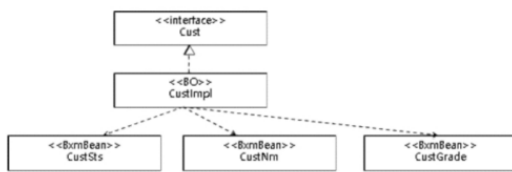
Aggregate 형상



참고)

Aggregate is a pattern in Domain-Driven Design. A DDD aggregate is a cluster of domain object that can be treated as a single unit.

BO 구현 형상



정의

- 비즈니스오브젝트(BO, Business Object)는 CBB(Corebanking Base)의 중심개념(Concept)을 제공하며, 모든 비즈니스로직을 구현하는 계층이다.
- 일반적으로 CBB API라고 하면, BO를 의미한다.

Aggregate의 이해 (BO = Aggregate)

- Aggregate란, 단일오브젝트로 취급되었을 때, 업무요건을 더 효과적으로 반영할 수 있는 도메인오브젝트(비즈니스로직의 구현체)의 집합을 의미한다. CBB는 Aggregate의 개념을 기본 설계개념으로 수용하고 있다.
- 예를 들면, 고객관계, 고객상태, 고객연락처는 모두 별도의 비즈니스로직을 갖는 오브젝트이지만, 컴포넌트 외부에서는 고객이라는 Aggregate로 취급되고, 고객이라는 단일 BO로 처리된다. 반면, 고객의 외부신용정보나, 고객이 가입한 요구불예금정보는 별도의 Aggregate로 설정되어, 신용정보BO나 계약BO에서 담당하도록 된다.
- CBP에서는 이 Aggregate를 BO(Business Object)라고 줄여서 부른다.

BO의 구현 및 호출

- BO(또는 Aggregate)는 컴포넌트 외부에 API로서 제공되는 단위이다. 외부에 API를 제공하기 위해서 모든 BO는 외부로 노출된 Interface를 구현한다.
- 동일 프로젝트(컴포넌트) 내(內)에서라도 BO간에는 컴포넌트 외부에 준하여 Interface를 통하여 접근한다.

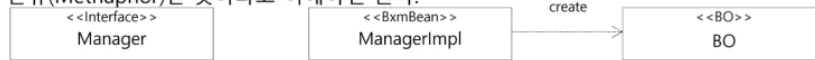
BO의 식별성(Identification) 결정

- BO의 Aggregate범위 외에 BO 대한 중요한 결정은 BO가 식별성을 갖도록 할 것인가의 결정이다. 식별성이 부여된 BO와 식별성이 없는 BO는 오브젝트의 생성 방식과 사용하는 방식이 다르다.
- 식별성의 필요성이 없거나 적은 BO
 - 식별성이 없는 BO는 어떤 상황에서도 입력에 대하여 정해진 동작만을 하는 비즈니스로직의 특징이 있다.
 - 이러한 BO는 CBP에서 Singleton으로 작동하고, 멤버변수를 갖지 않는다.
 - CBP에서는 BxmBean으로 구현한다.
 - 참고) EJB Stateless Session Bean
 - 예) 책임자승인BO, 통지BO, 채번BO
- 식별성 요건이 강한 BO (Domain BO)
 - 해당 BO가 나타내는 개념(Concept)이 식별자를 갖는다.
 - 해당 BO가 멤버변수로 식별자를 갖는다.
 - 오브젝트를 생성할 때마다 멤버변수를 적절하게 Initialize시켜야 한다.
 - 오브젝트를 생성하는 Factory의 역할을 하는 Manager객체를 만들어야 한다.
 - Transaction내에서 동일한(식별자가 같은) BO는 반드시 1개만 만들어져야 하고, Transaction내에서 일단 BO가 만들어지면 Transaction종료시까지 유지되어야 한다.
 - 참고)EJB Entity Bean
 - 예) 고객BO, 계약BO, 계약조건BO, 부서BO, 계약거래BO, 상품BO, 상품조건BO

식별성이 있는 BO는 다음의 아키텍처를 준수해야 한다.

Manager

- 식별성이 있는 BO는 BxmBean이 아닌 POJO로 개발해야 하며, POJO는 new키워드를 통해 객체를 명시적으로 생성해야 한다. Manager클래스는 식별성이 있는 BO를 new 키워드로 생성하고, BO의 멤버변수를 적절하게 초기화(Initialize)하는 책임을 전담하는 클래스이다.
- OOO Manager라고 Naming한 것은 Manager가 실제로 그 일을 하기 보다는 그 일을 수행할 담당자(BO)를 데려온다는 현실을 은유(Methaphor)한 것이라고 이해하면 된다.



- Manager 자체는 Singleton으로서 BxmBean으로 구현된다.(식별성없는 BO)
- Manager는 Aggregate(BO) 외부에 Interface를 제공하여야 한다.
- Manager의 또 다른 역할은 식별성이 있는 BO에게 책임을 할당하기 곤란한 기능을 담당한다.
 - 검색: 특정 고객의 보유계좌 목록조회와 같은 검색기능을 계약BO에 할당할 수 없기 때문에 계약Manager에서 담당한다.
 - 최초등록: 고객정보등록은 아직 고객이 만들어지지 않았기 때문에 특정 고객을 나타내는 고객BO에게 할당하기 어렵다. 이 경우, 고객Manager가 담당한다.

→ 식별성 요건이 강한 BO = Domain BO = Aggregate = 대장VO

→ 대장BO는 새끼BO를 가지고 있음

→ 매니저클래스

→ 새끼BO를 호출하기 위해 대장BO를 호출하는데 대장BO를 호출하기 위해서는 매니저클래스를 사용함(Manager 안에 호출하는 메서드 존재)

⇒ 식별성의 필요성이 없거나 적은 BO와 식별성 요건이 강한 BO의 특징 이해

⇒ Manager 클래스의 역할 파악