U-creator
U dream, we create.

컴포넌트 아키텍처 설계를 위한 체계적인 접근

백창현 책임
SA Team
Architecture Center
Samsung SDS
October 2005

삼성SDS SAMSUNG

컴포넌트 아키텍처

❑컴포넌트 아키텍처

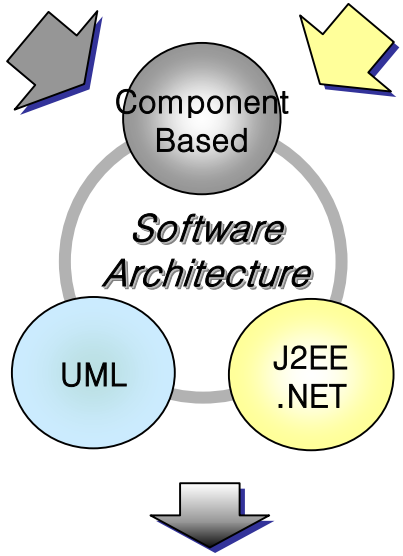
▶ 어플리케이션 레벨 소프트웨어 컴포넌트들의 구성을 위한 전략

▶ 컴포넌트 사이의 구조적 관계와 행위간 의존성에 대한 전략과 방향성을 제공함.

컴포넌트 아키텍처의 구성

컴포넌트 인프라 스트럭처

- 컴포넌트가 동작하는 컨테이너와 컴포넌트의 런타임 환경을 제공
- 컴포넌트의 구성, 연계, 관리 및 트랜잭션 처리를 위한 인프라 제공
- 컴포넌트의 기술 기반으로 Tier 중심의 시스템적인 체계화된 기본 구조를 가지고 있음.
- J2EE 및 .Net을 주요 인프라로 활용하고 있음.



컴포넌트 아키텍처

- 컴포넌트 기반의 소프트웨어 아키텍처를 구성하는 주요 컴포넌트들의 아키텍처
- 컴포넌트의 Static / Dynamic의 관점에서 구성됨
  - ✓컴포넌트간의 포함관계 (Statics)의 방향성 제공
  - ✓ 컴포넌트간의 상호 이용 관계 (Dynamics)의 방향성 제공
- Layer기반으로 주로 정의되며, CBD의

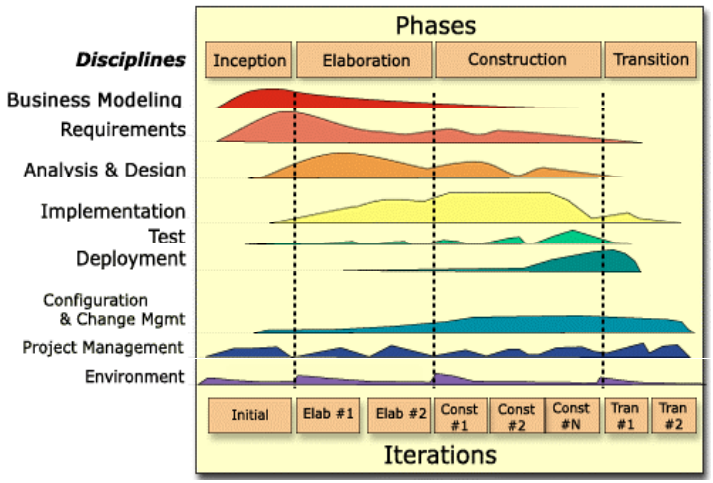
- 컴포넌트 아키텍처는 컴포넌트 인프라 스트럭처 상에서 구현되는 소프트웨어 아키텍처
- 컴포넌트 기반 개발(CBD)의 성공을 위한 필수적인 요소

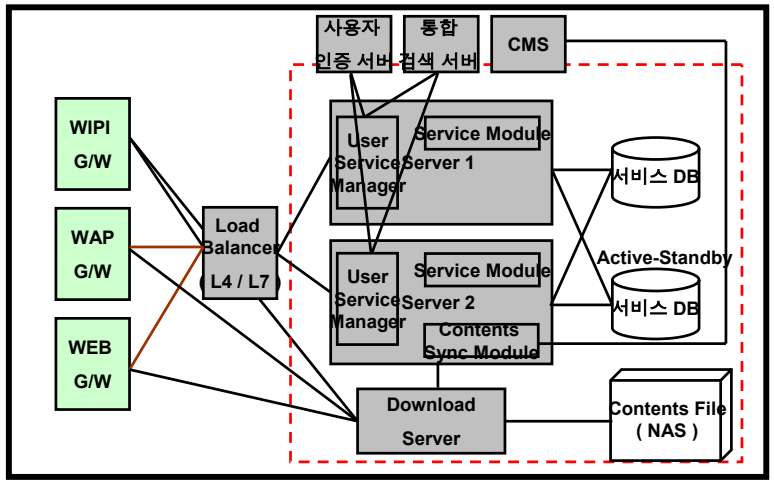
목 차

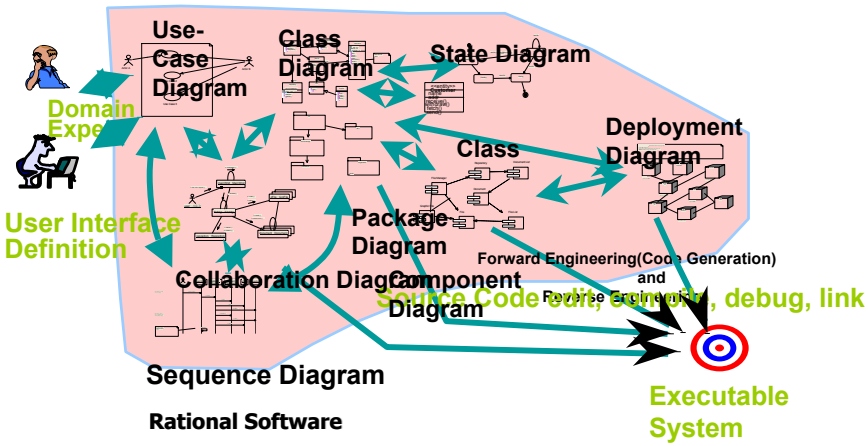
삼성SDS SAMSUNG

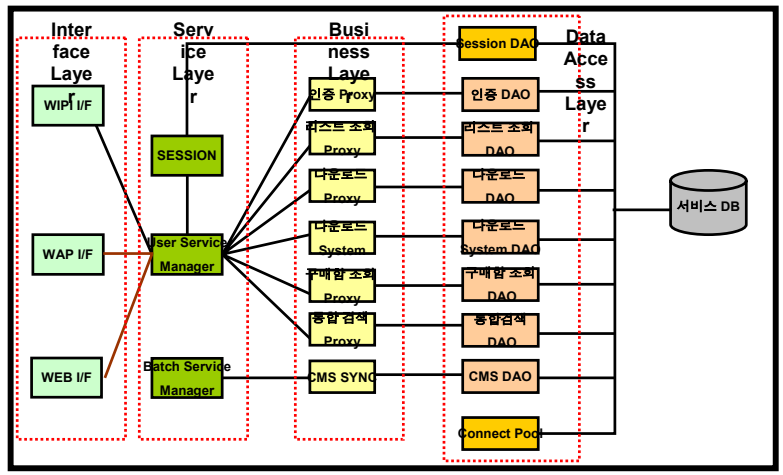
컴포넌트 기반 개발방법론과 아키텍처

- CBD와 CBA는 상호보완적인 관계
  - CBD : 컴포넌트 중심으로 개발을 진행하기 위한 체계와 절차
  - CBA : 컴포넌트 중심 개발을 위한 시스템의 구조와 컴포넌트들의 체계 및 연관 구조









체계적인 컴포넌트 아키텍처 설계의 필요성

- 컴포넌트 아키텍처의 궁극적인 목표
  - 어떻게 하면
  - 구축 비용을 절감되고
  - 구축 기간을 줄이면서
  - 재활용 율이 높으면서도
  - 보다 좋은 시스템을
  - 효과적으로 설계할 수 있을까?

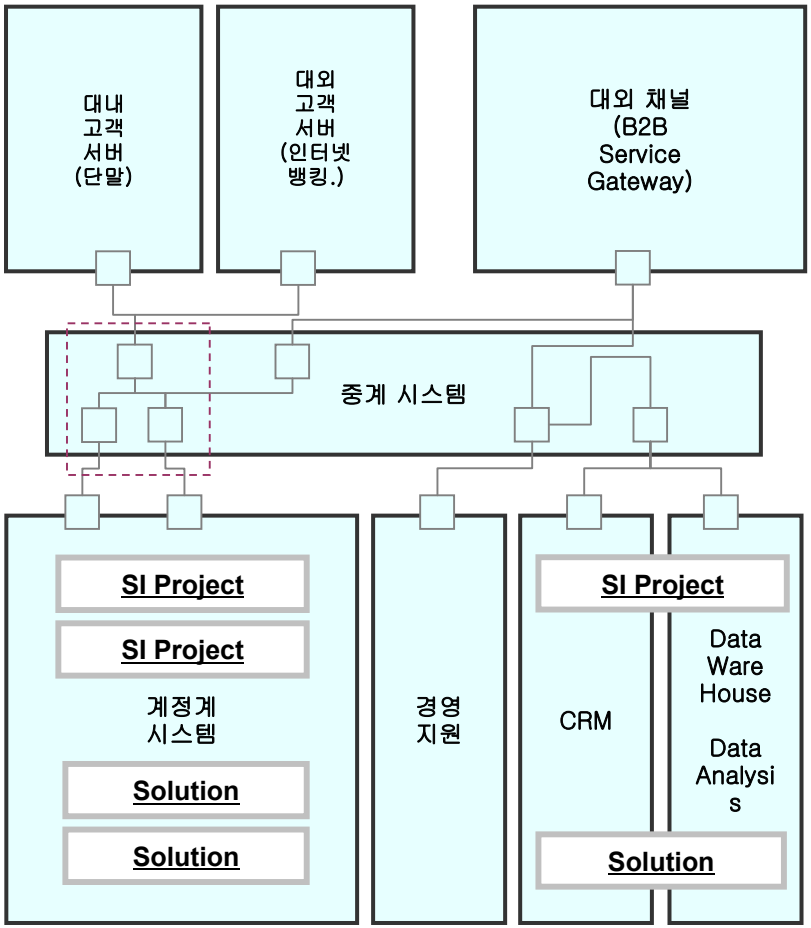
➔ 체계적인 컴포넌트 아키텍처 설계의 접근
  - 컴포넌트 기반의 아키텍처 설계
  - 품질속성과 레이어 기반의 설계에 기반한 체계적인 아키텍처 설계의 접근
  - 기술 기반 (컴포넌트 인프라 스트럭처)의 특성을 반영함.
  - 도메인의 특성을 참조 아키텍처 (Reference Architecture)로 반영
  - 패턴과 프레임워크 중심으로 설계 및 구현과 직접적인 매핑이 되는 아키텍처 설계
  - 핵심 설계 요소가 필요한 부분은 집중적으로 정의하여 마이크로 아키텍처로 해결

- **체계적인 설계 기법은 소프트웨어 아키텍처 문제의 한 부분을 해결함.**
- 도메인 중심의 재활용을 위한 프레임워크와 표준 설계 체계가 병행되어야 함. ←→ Product Line

System Integration vs Solution Development

❑ System Integration Vs Solution Development

- ▶ System Integration : 컴포넌트 중심의 대단위 아키텍처 설계 (CBD 기반이 다수임)
- ▶ Solution Development : 객체 및 메커니즘 중심의 복잡한 아키텍처 설계



	System Integration	Solution Development
영역	시스템들의 통합	시스템내의 개발
형태	금융권,공공기관, 국방부의 SI구축 프로젝트	룰 엔진 /빌링시스템/프레임웍
품질속성	시스템들 사이에서 주로 존재 (성능, 유지 보수성 강조)	솔루션 내의 속성 강조 (재활용성, 범용성 강조)
요구사항/목적	상대적으로 명확.	불명확, 함께 개발.
요구사항중점	범위	깊이
우선 순위	일정, 기능 구현	완성도, 품질.
설계결정	고객과 동의. 의사결정 프로세스 중요	자체 정의.
설계중심	시스템. 컴포넌트 인터페이스	패턴. 컴포넌트. 모델
PJ 착수	수주 후	내부 결정 후.
PJ 완료	일정 종료시	충분한 완성도 달성시.
Pattern	프레임워크에서 대부분 사용되며, 신규로 패턴 도입할 필요는 거의 없음.	신규 개발

System Integration – Lesson Learned

- SI의 목적은 전체적인 기능들의 조화에 있음
  - 필요한 부분은 외부 솔루션 / 프레임워크를 도입

- 재활용은 필요한 경우에 고려해야 함.
  - 재활용을 강조한 복잡한 설계 (ex: 브리지 패턴의 사용, 데코레이터 패턴의 사용) 를 수행하면, 함께 작업하는 엔지니어들이 따라오지 못한다.

- 하드웨어와 소프트웨어를 함께 생각해야 함.
  - 많은 비기능적 요소들이 하드웨어와 맞물려 구현된다. 보안은 DMZ의 도입과 침입 탐지 시스템의 도입으로, 성능은 H/W의 추가와 L4 스위치 장비의 도입에 의해 구현가능하다.

- 제약 사항과 고객 측의 문서 표준화 / 설계 표준화를 고려하여 작업을 해야함
  - 프로젝트에서 만든 SI결과물은 유일무이한 존재
  - 유지 보수하기 위해서 문서를 사전에 준비해야 함.

- 레이어 /티어의 구성에 중점을 맞춤
  - 블랙보드 스타일, 파이프&필터 스타일등의 아키텍처 스타일은 SI프로젝트 수행에 적용이 까다로움
  - 레이어 구성와 티어의 구성으로 아키텍처를 구조를 우선 정의함.

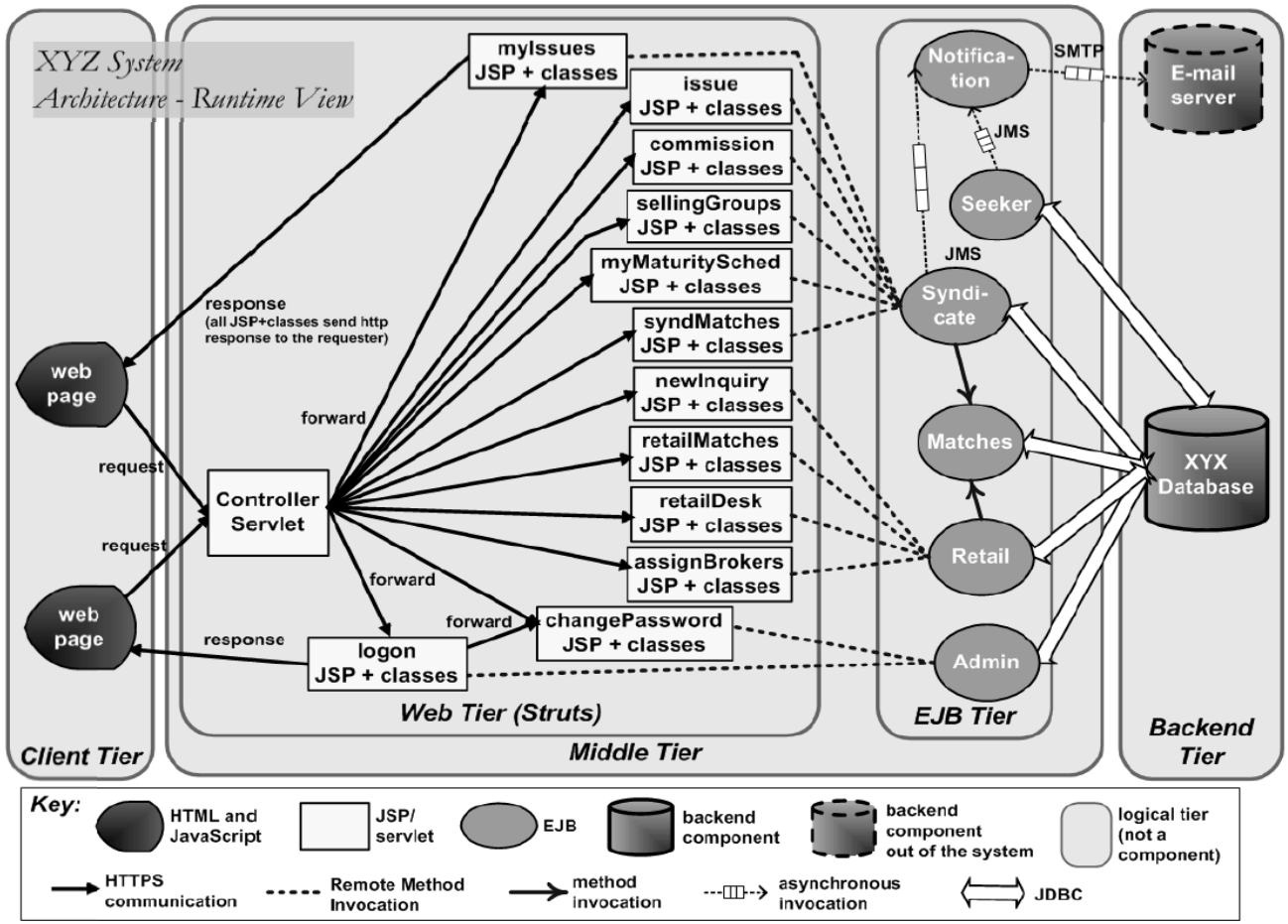
➔ 소프트웨어를 구축하는 것이 SI 프로젝트와 솔루션 개발로 이원화되지는 않음.
  ➔ 하지만, 위의 2개의 비교를 통해서, 자신의 프로젝트에서
  어떤 방향성을 택할 것인가를 다시 한번 가늠해 볼 수 있다.

전형적인 J2EE 기반의 컴포넌트 기반 시스템 구조

❑ J2EE 금융 어플리케이션 구조 예제

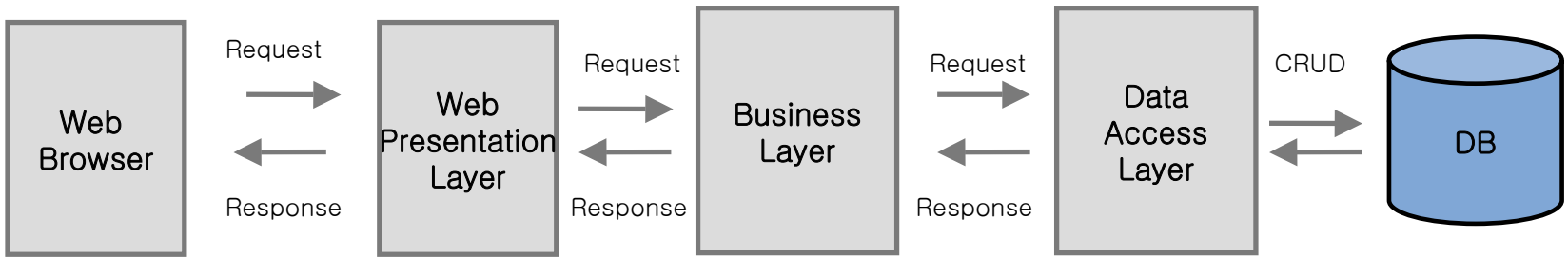
▶ Tier 구분에 의한 전형적인 구조

▶ 컴포넌트 인프라 (J2EE)에 의해 컴포넌트들 간의 수행 역할 및 체계가 구분되어 있음. .



J2EE 애플리케이션 3 Tier 아키텍처

- J2EE 애플리케이션은 Multi-Layer 의 구조의 아키텍처를 가져간다.
  - 각 Layer 별의 기본적인 역할은 구분되어 있다.
  - J2EE의 일반적인 구조를 시스템 구성을 위한 참조 아키텍처로 활용한다.
  - 이 참조 아키텍처 상에 기능적/비기능적 요구 사항과 제약 사항을 반영한다.

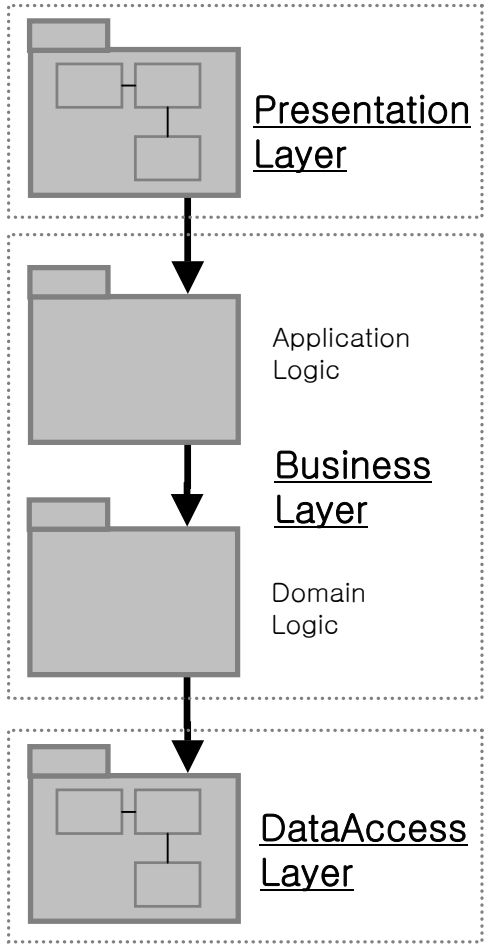


Category	Layer Role	Issue Description
Presentation Layer	웹 클라이언트와 어플리케이션 비즈니스 로직간에 상호작용을 담당하는 레이어	프리젠테이션 로직과 비즈니스 로직을 어떻게 분리할 것인가?
Business Layer - Application Logic	비즈니스 인터페이스를 제공하고, 비즈니스 서비스들의 흐름을 제어하는 레이어	상세한 수준의 세부 인터페이스를 노출할 것인가? 서비스를 위한 최소한의 상위 인터페이스 만을 노출할 것인가?
Business Layer - Domain Logic	도메인 모델을 구성하고, 도메인에 연관된 데이터를 제어하는 로직을 포함하는 레이어	도메인 모델을 만들 것인가? DB중심의 도메인 모델을 구성할 것인가? 객체지향의 도메인 모델을 구성할 것인가?
DataAccess Layer	데이터에 접근하는 부분 담당하는 레이어	도메인 로직과 DB의 연계를 위한 데이터 접근 방식은 어느 수준을 선택할 것인가?

소프트웨어 설계의 주요 이슈

- 소프트웨어 설계의 이슈는 유사하게 나타남.
- 이를 어떻게 체계적으로 적용하여 나아갈 것인가? 라는 사실이 중요함.

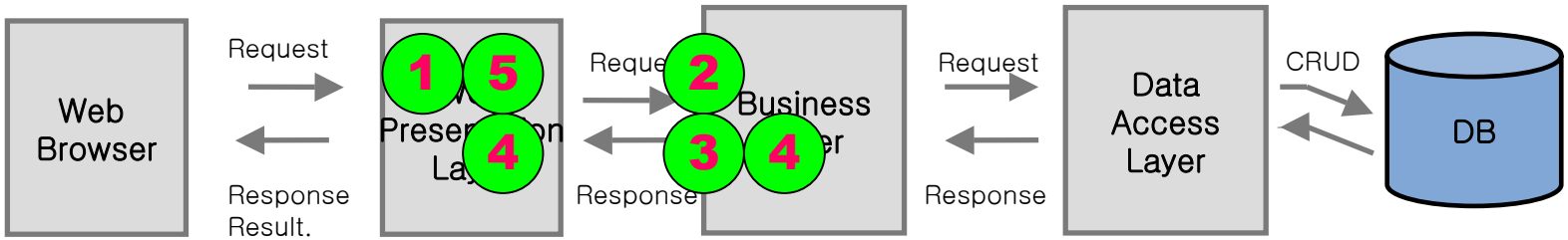
설계 이슈	이슈 정의
Concurrency (동시성)	시스템을 어떻게 프로세스, 태스크와 쓰레드로 나누고 이에 관련된 효율성, atomicity, synchronization과 scheduling 이슈를 어떻게 다룰 건지에 대한 이슈
Control & Handling Events	데이터의 흐름과 컨트롤의 흐름을 어떻게 구성할 지와 반응적이고 일시적인 이벤트를 다양한 메커니즘을 통해서 어떻게 처리할 건지에 대한 이슈 (예. implicit invocation and call-backs)
Distribution of components	소프트웨어가 어떻게 하드웨어에 분산되어 있는지, 컴포넌트들간에 어떻게 통신을 하는지, 미들 웨어가 이 기종 시스템간에 어떻게 사용되는지에 대한 이슈
Error & Exception Handling & Fault tolerant	에러와 예외 처리 및 장애방지에 관한 이슈
Data Persistence (영속성)	영속성이 있는 데이터를 어떻게 처리할지 에 대한 이슈
Interaction & Presentation	사용자가 system과 어떻게 상호작용할 지에 관한 이슈



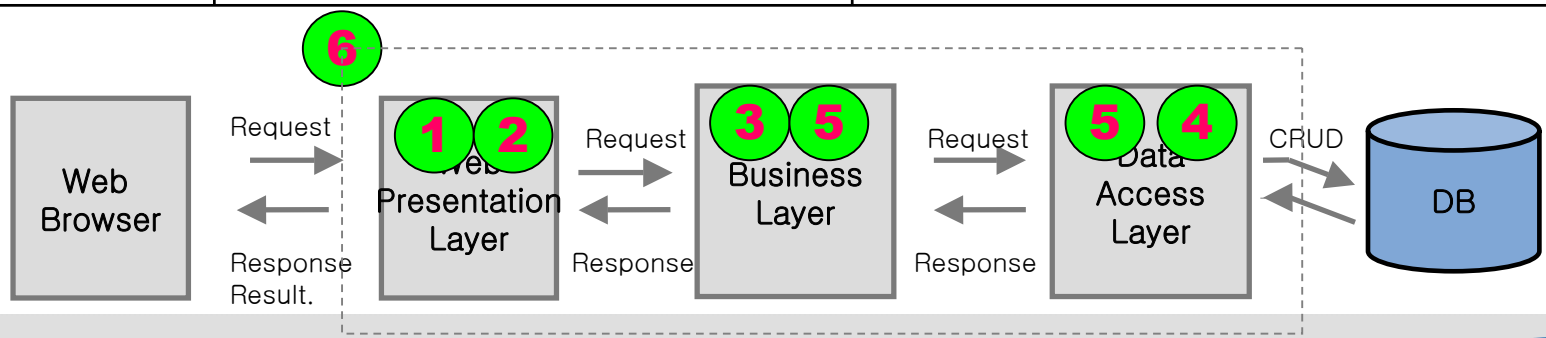
Source : Software Engineering Body of Knowledge,2004

이벤트 제어(Control/Handing Events) 및 동시성(Concurrency) 이슈의 발생

설계 이슈	이슈 정의	웹 시스템 주요 이슈
Control & Handling Events	데이터의 흐름과 컨트롤의 흐름을 어떻게 구성할 지와 반응적이고 일시적인 이벤트를 다양한 메커니즘을 통해서 어떻게 처리할 건지에 대한 이슈 (예. implicit invocation and call-backs)	1. 페이지의 단순 또는 복잡한 흐름 제어 2. 비즈니스 로직 (어플리케이션 로직, 도메인 로직)의 제어, 구성 및 배치 3. 인터페이스 구성 4. 접근 권한 제어 (Authorization) 5. 사용자 인증 제어 (Authentification)



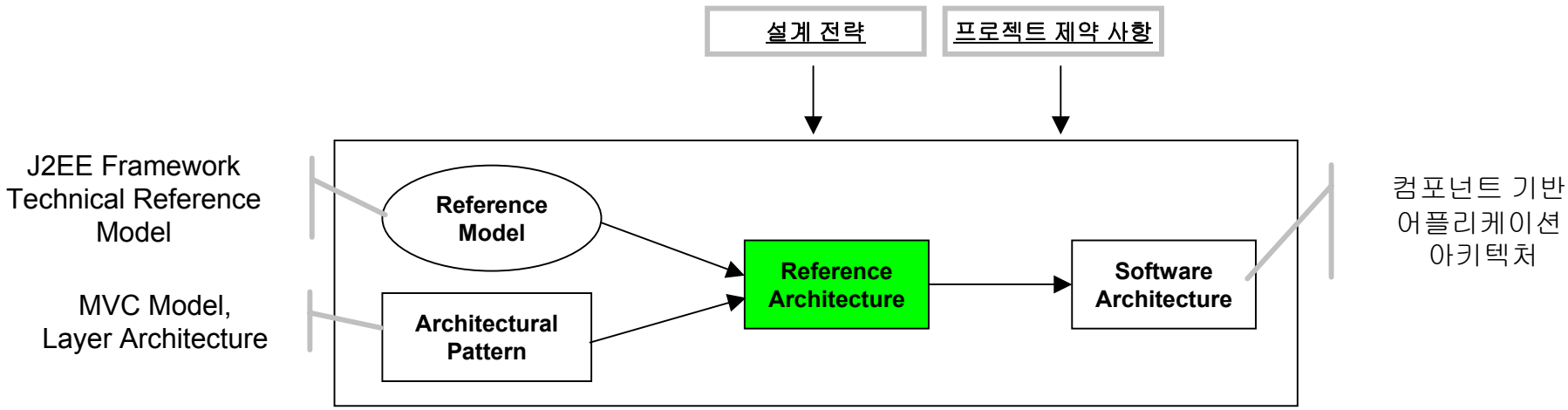
설계 이슈	이슈 정의	웹 시스템 주요 이슈
Concurrency	시스템을 어떻게 프로세스, 태스크와 쓰레드로 나누고 이에 관련된 효율성, atomicity, synchronization과 scheduling 이슈를 어떻게 다룰 건지에 대한 이슈	1. 다중 사용자 HTTP요청에 처리 2. 다중 사용자 세션관리에 처리 3. 비즈니스 로직 호출에 처리 4. 데이터 억세스에 대한 DB 연결 처리 5. 트랜잭션에 대한 처리 6. 시스템 자원에 대한 처리



J2EE의 아키텍처 설계 이슈와 레이어 아키텍처

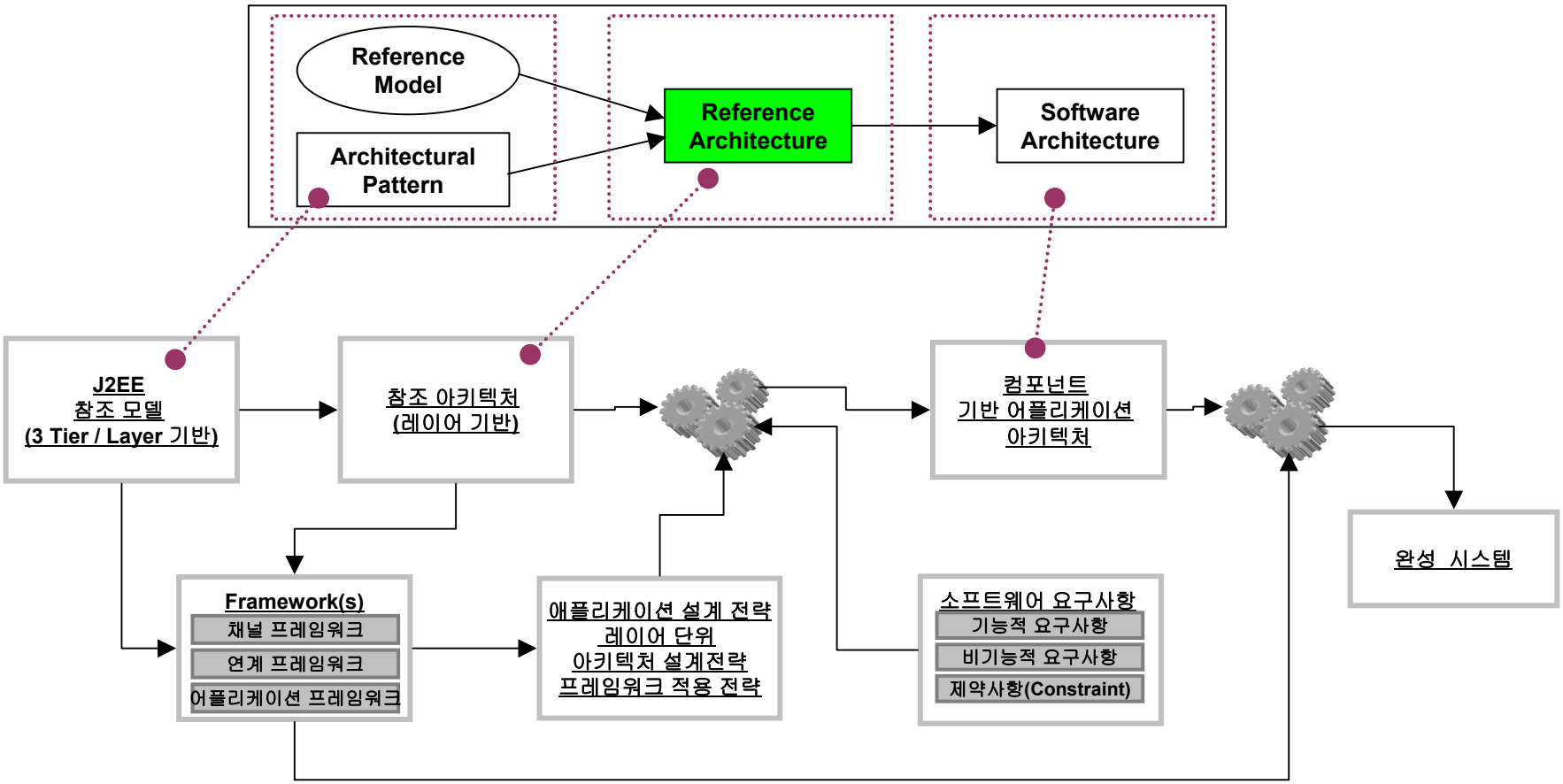
- 레이어 아키텍처 스타일
  - 레이어 스타일은 각 레이어 별로 주요 역할이 구분되어 있다.
  - J2EE는 소프트웨어 설계 이슈를 해결하기 위한 잘 정의된 레이어 구조를 가지고 있다.
  - 때문에 J2EE의 소프트웨어 설계 이슈는 레이어 단위로 발생한다.

- 레이어 기반 J2EE 아키텍처 설계 전략
  - 레이어 기반으로 아키텍처 설계 전략을 구성하며,
  - 아키텍처 설계 전략을 기반으로 완성하고자 하는 시스템의 참조 아키텍처를 도출한다.



❑ J2EE 애플리케이션 후보 아키텍처의 도출

▶ 소프트웨어 요구 사항을 기반으로 품질 속성을 정의하여 후보 아키텍처를 선택

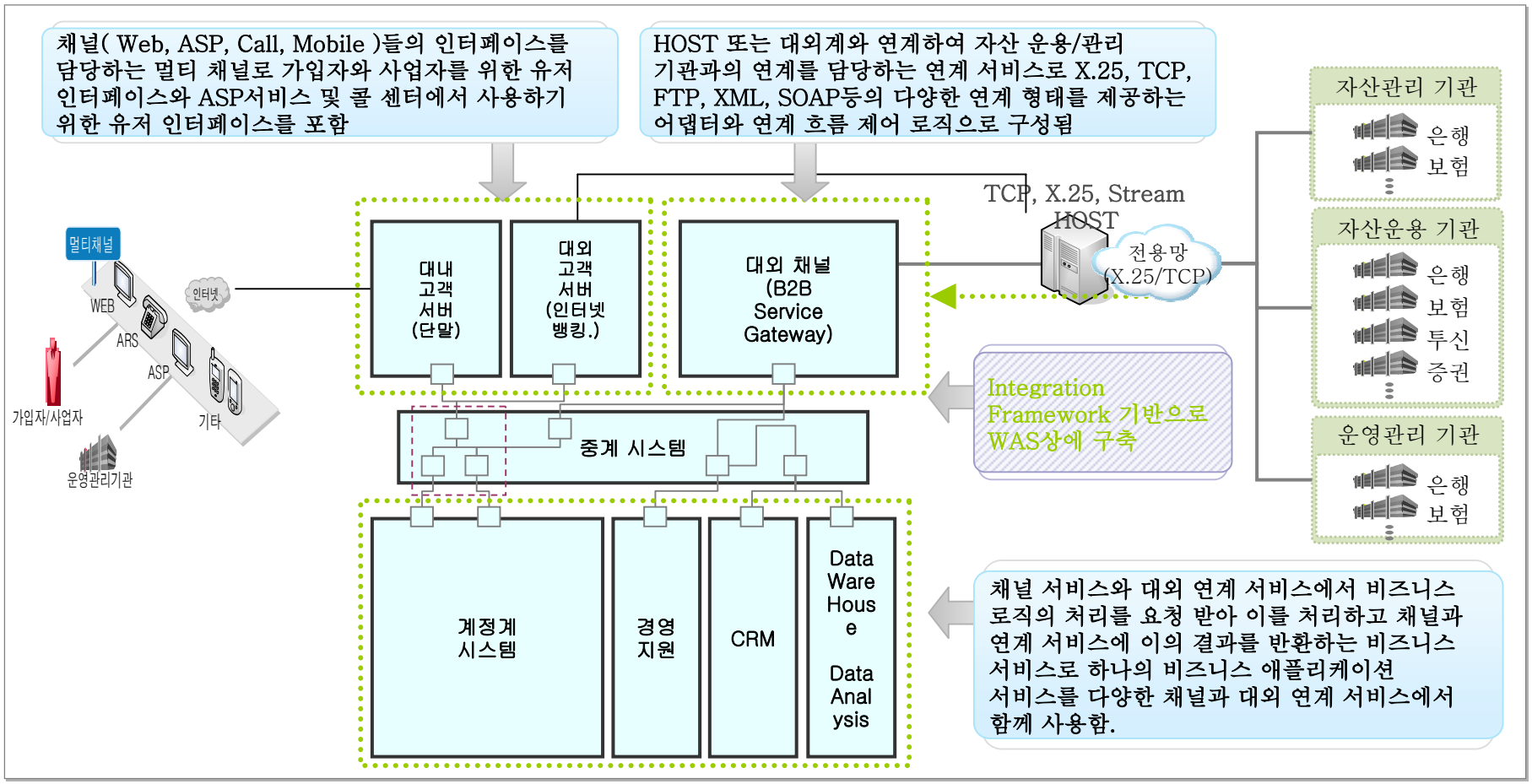


목 차

삼성SDS SAMSUNG

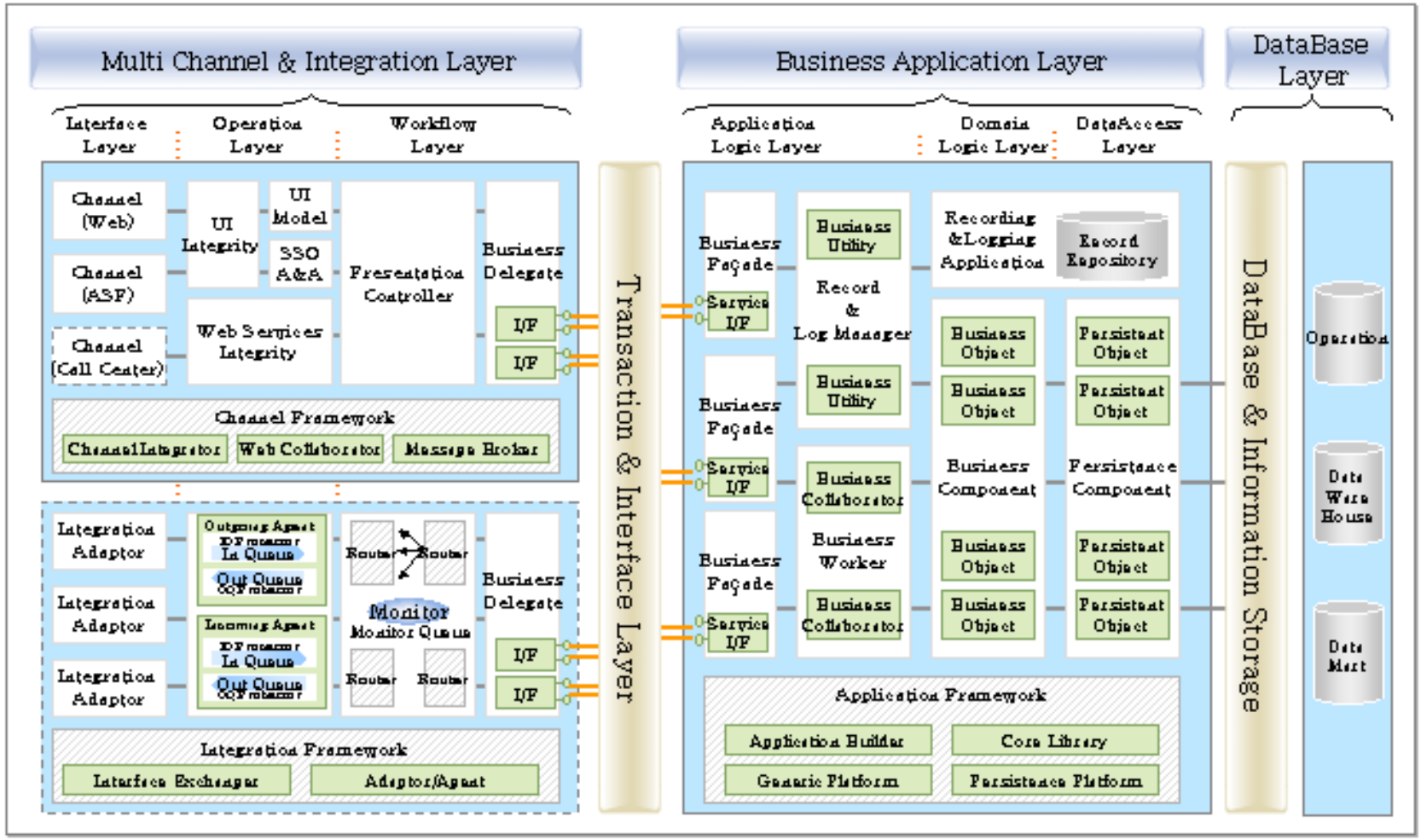
Business Centric Layered Architecture

- 멀티 채널 서비스와 대외 연계 서비스에 수반되는 비즈니스 로직이 통합된 애플리케이션 아키텍처를 구성하고, 단일화된 서비스 인터페이스를 제공하여 채널과 연계 서비스에서 함께 사용하는 비즈니스 중심의 분산 레이어 구조의 서비스 아키텍처를 구축함.



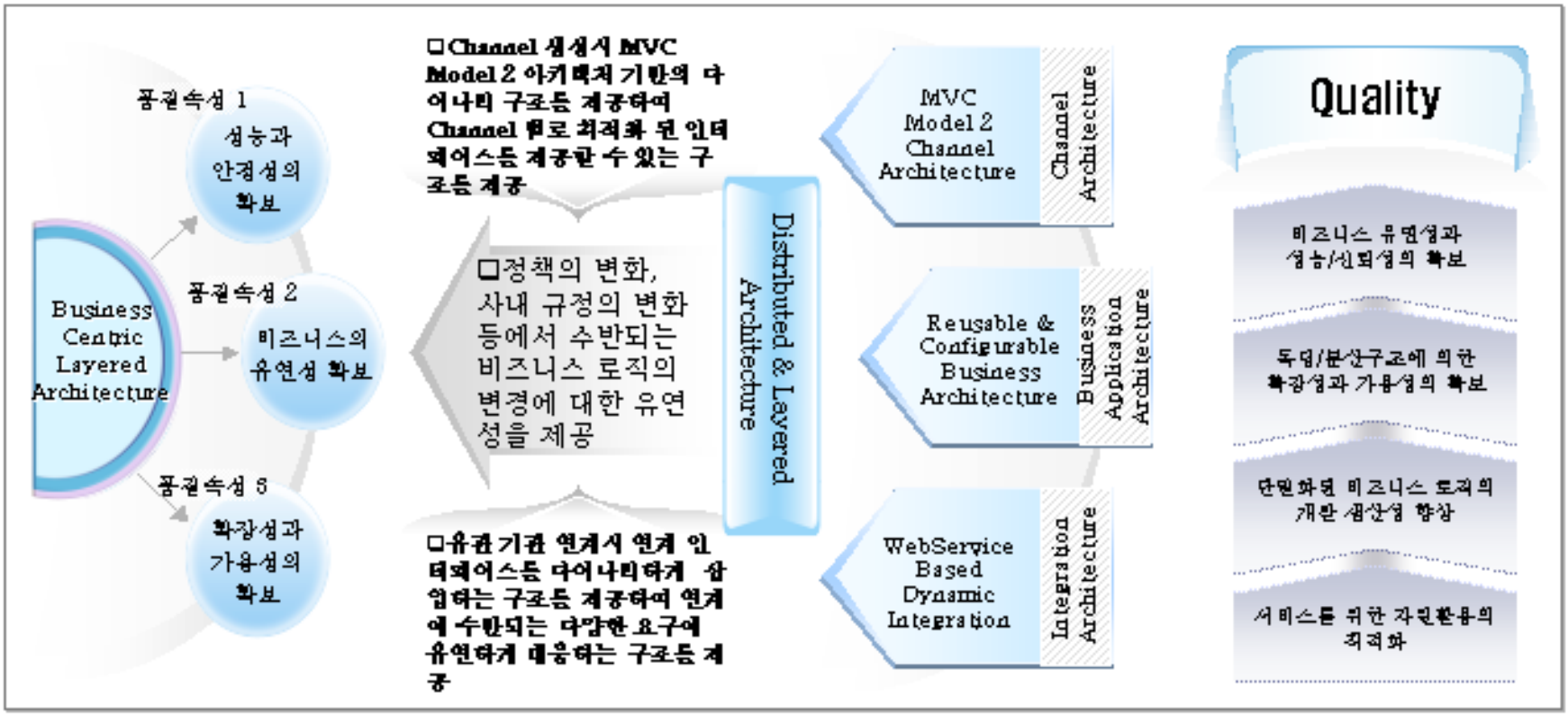
Business Centric Layered Architecture in Detail

- Reference Architecture for Financial Domain (J2EE Model Based)



Business Centric Layered Architecture 특징

- 다이나믹한 비즈니스 서비스 제공에 근거한 비즈니스 중심의 분산 비즈니스 애플리케이션 서비스 (Distributed Business Application Service)와 서로 다른 채널 지원을 위한 가변성 확보를 위한 채널 서비스 및 유관 기관들의 가변적이고 안정적인 연계를 지원하기 위한 연계 서비스로 분리되어 구축됨으로써 비즈니스의 유연성, 성능/신뢰성, 확장성/가용성의 3가지 측면을 고려할 수 있도록 설계되었음.



목 차

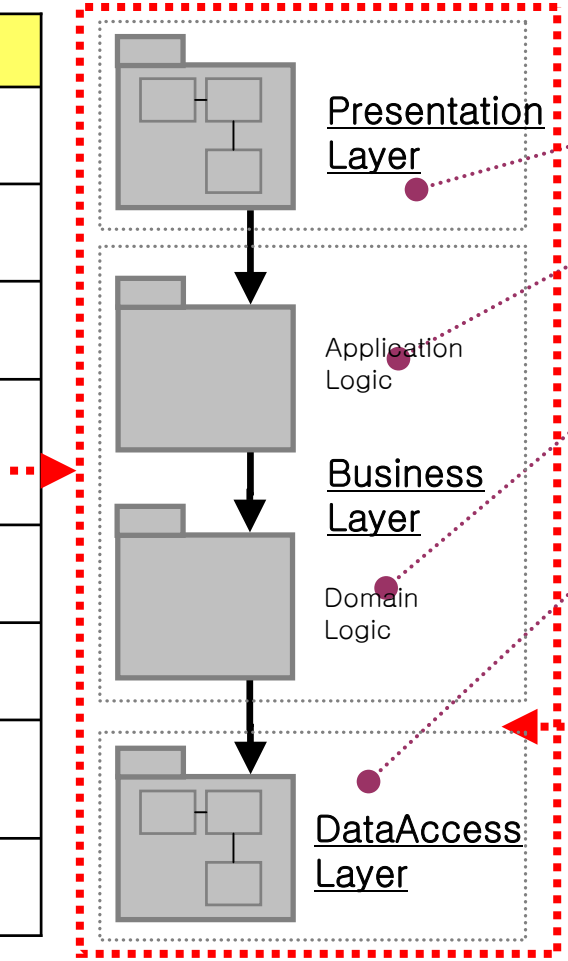
삼성SDS SAMSUNG

분산 아키텍처 전략의 이슈 영역

❑ 추가적인 비기능적 요구 사항은 참조 아키텍처를 바탕으로 확장 전개 가능함.

▶ 비기능적인 요소를 위한 컴포넌트 기반설계는 레이어 중심으로 반영을 할 수 있음

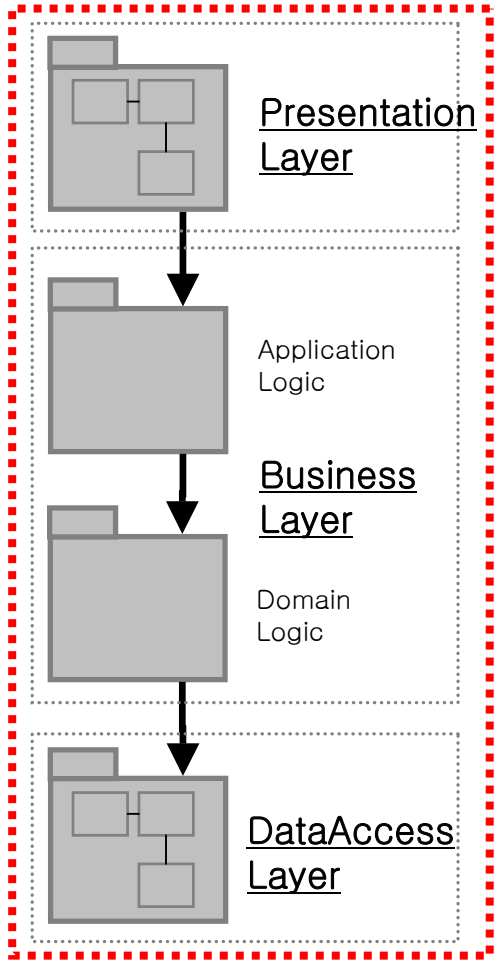
관점	품질 속성
운영	Availability
	Performance
	Usability
	Security
설계	Modifiability
	Testability
프로젝트 환경	Schedule &Resource
	Cost



분류	설계 전략
Presentation Layer	MVC Model 1
	MVC Model 2
Business Layer – Application Logic	Coarse grained service interface
	Fine grained service interface
Business Layer – Domain Logic	Transaction Script
	Table Module
	Domain Model
DataAccess Layer	Pure Relational
	Light Object Mapping
	Full Object Mapping
Common Layer	Layer/Tier Data Transfer
	Session State Control
	Session Usage
	(Data) Caching
	Access Control
	Internationalization
	Messaging
	Error & Exception Handling
	Logging

비즈니스 로직 - 애플리케이션 로직

- 애플리케이션 로직의 정의
  - 시스템이 제공하는 서비스의 바운드리(Boundary)로 프리젠테이션 레이어와의 인터페이스를 위한 일련의 오퍼레이션으로 구성된다.
  - 비즈니스 로직을 캡슐화하고 서비스의 트랜잭션을 제어하며 비즈니스 로직의 수행 결과들을 **Coordinate** 한다.
- 애플리케이션 로직의 역할
  - 소프트웨어가 해야 하는 일을 정의하고 해당 기능을 수행하는데 필요한 도메인 객체로 이끄는 역할.
  - 다른 시스템의 어플리케이션 로직과 상호작용이 필요할 경우에 이를 위한 인터페이스 제공
  - 도메인 로직의 도메인 객체에게 **delegate**를 하거나 **collaborate**를 시키는 역할을 담당.
  - 비즈니스 규칙이나 업무 지식은 도메인 로직에 포함됨.
  - 비즈니스 상황을 반영하는 상태를 가지지 않으나 사용자나 프로그램의 진행상태를 반영할 수 있음.
- 애플리케이션 로직의 설계
  - 시스템의 규모와 비즈니스 및 기술 구조의 변경으로 인한 시스템의 영향 등을 고려하여 판단 해야 함.



❑ 인터페이스 수준의 선정

Category	Strategy	구현 기술
Business Layer – Application Logic	Coarse grained service Interface	Remote Session Façade (RMI) [POJO, Session Bean] Command Pattern [POJO, Session Bean]
	Fine grained service Interface	[POJO, Session Bean],

❑ 어플리케이션 로직의 레이어 구성 전략

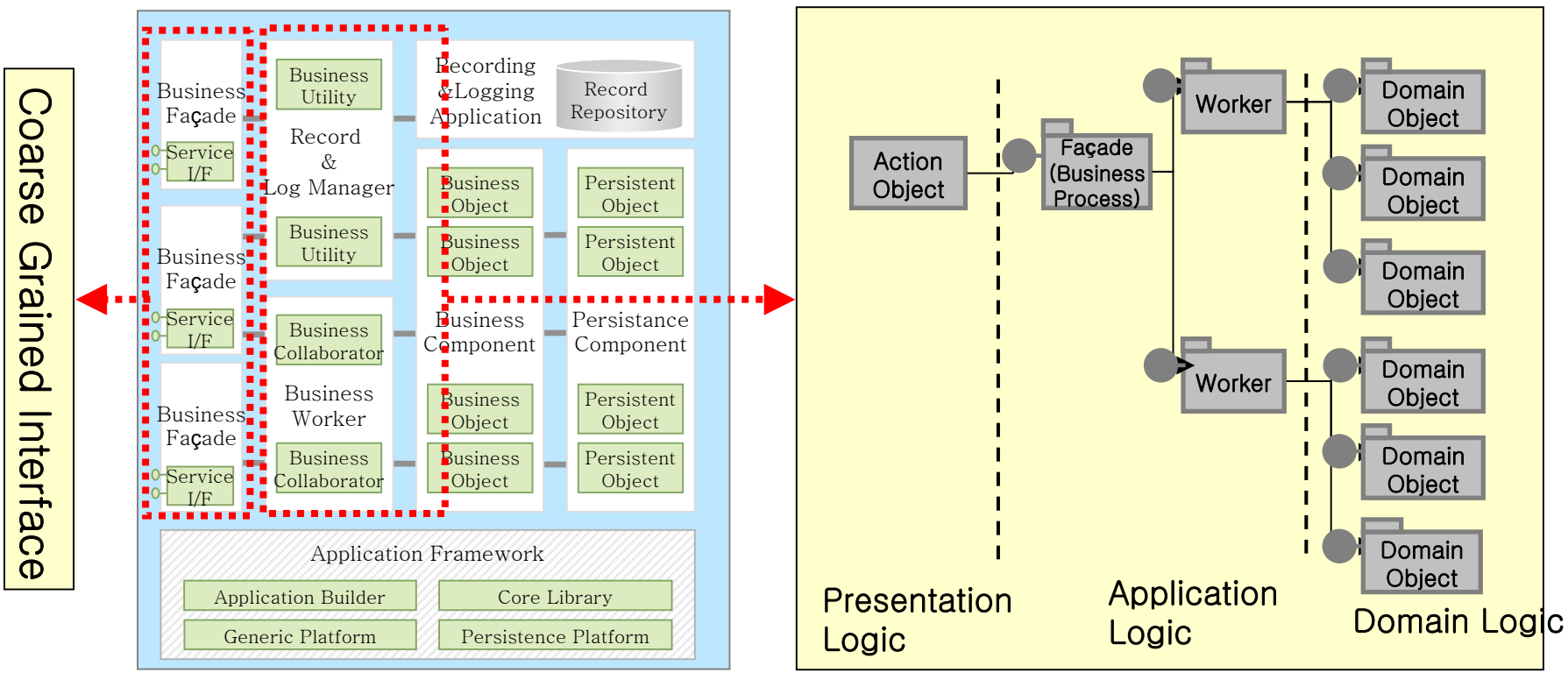
- ▶ 인터페이스와 비즈니스 프로세스 제어 부분의 구성에 따른 레이어 분리
- ▶ **위임형** : 비즈니스 로직을 DB 혹은 도메인 로직에 위임하고 최소한만을 처리함.
- ▶ **중앙 집중형** : 비즈니스 로직을 인터페이스와 통합하여 처리
- ▶ **복합형 : 비즈니스 로직을 인터페이스와 분리하여 구성**

비즈니스 레이어 구성 전략– Coarse Grained Interface & 복합형

❑ 복합형

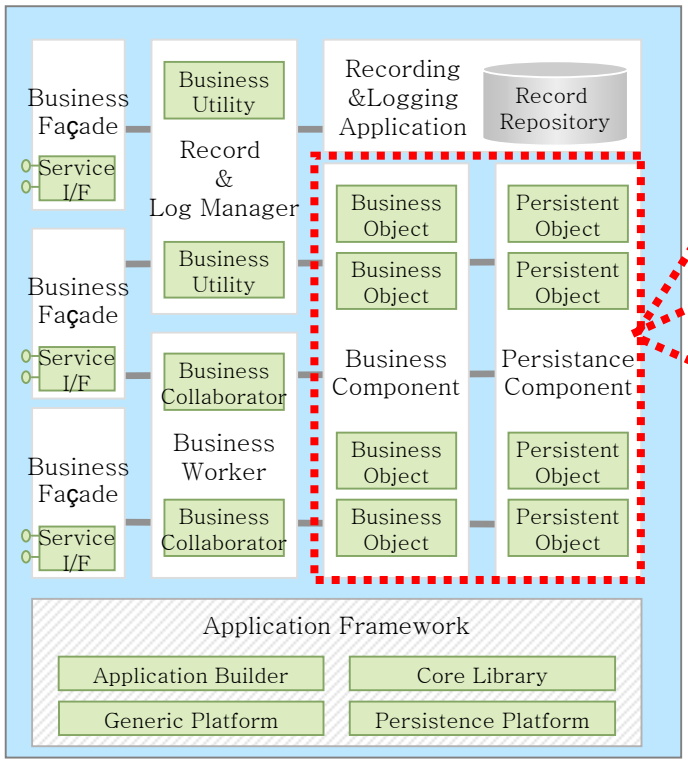
- ▶ 비즈니스 인터페이스의 기능은 워커 객체에 위임
- ▶ 워커(Worker)의 분리로 적절한 기능들의 집합으로 묶어 Cohesion을 향상 (長)
- ▶ 지나친 중앙 집중으로 인한 로직의 복잡성 문제 해결(長)
- ▶ 레이어의 추가로 작업 량 증가(短)

➔ Reference Architecture를 기반으로 Layer를 추가함.



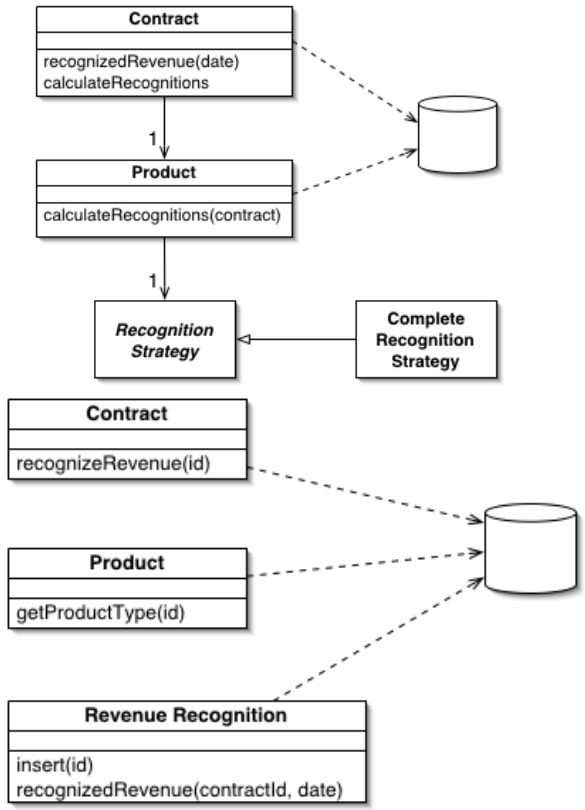
도메인 로직 구성 전략

- 도메인 로직이란?
  - 도메인에 관련된 비즈니스 로직을 처리하는 부분이며
  - 도메인 모델을 구성하고 도메인에 연관된 데이터를 제어하는 로직을 포함한다.



recognizedRevenue(contractNumber: long, asOf: Date) : Money
calculateRevenueRecognitions(contractNumber long) : void

- **Transaction Script**
- **Domain Model**
- **Table Module**



도메인 로직 구성의 주요 이슈

- 도메인 로직을 위한 모델의 구성 방법
  - 도메인 로직을 위한 별도의 엔티티 클래스 모델(도메인 모델)을 구성 할 것인가?
  - 혹은 데이터 모델(ER 모델)만을 구성할 것인가?
- 도메인 모델의 구성 시에, 데이터 모델과 도메인 모델의 연계 방법
  - 데이터 모델의 도출 시점과 도출 방법.
  - 데이터 모델과 도메인 모델과의 관계
- 애플리케이션 로직과 도메인 로직의 연계 방법
  - 애플리케이션 로직과 도메인 로직은 어떠한 방식으로 연계할 것인가?
- 기존 데이터 모델의 활용
  - 기존의 데이터 모델(ERD)이 존재하는 경우, 이들을 어떻게 활용하여 개발할 것인가?
- 데이터 베이스 기능의 활용
  - 데이터 베이스 기능을 어느 수준까지 활용하여 도메인 로직을 구성할 것인가?

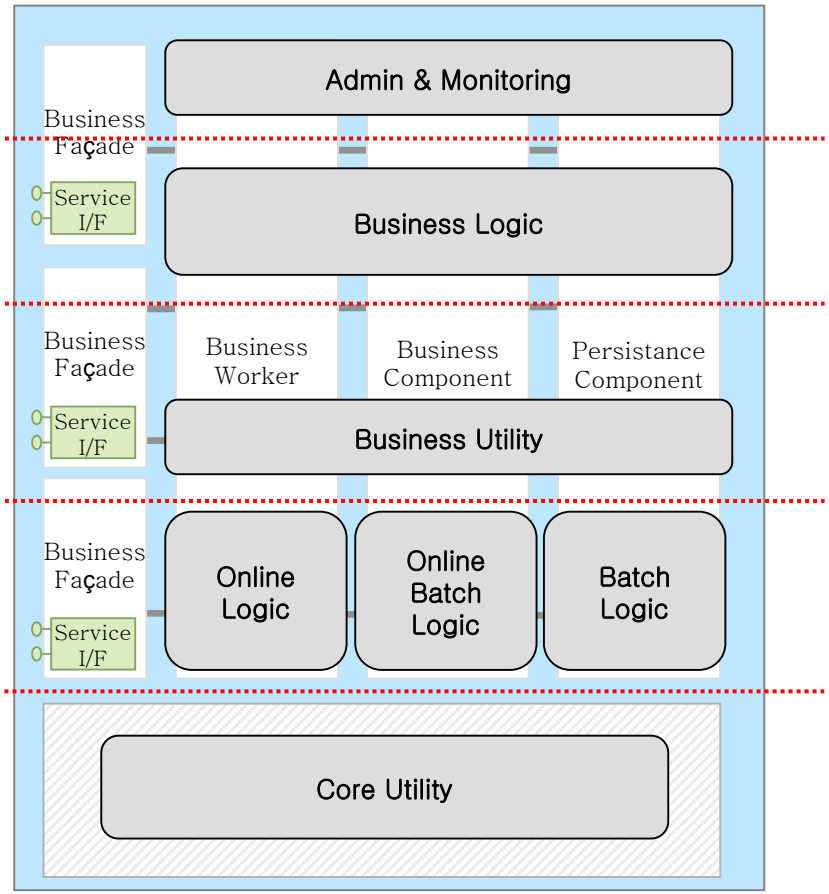
Category	Strategy	구현 기술
Business Layer - Domain Logic	**Transaction Script**	**[Session Bean, POJO]**
	Table Module	**[POJO]**
	Domain Model	**[Entity Bean, POJO]**

수평적 레이어 확장 방안 (Horizontal)

❑ 모듈의 공통화 및 사용빈도에 따른 수평적 레이어의 구성

❑ 레이어 구성 기준
- ▶ 모듈의 공통화
- ▶ 사용 빈도
- ▶ 재활용 가능성
- ▶ 프레임워크 지원 여부

❑ 레이어 분류 유형
- ▶ 매우 일반적으로 사용되는 모듈 (Core Utility)
- ▶ 업무 유형에 따라 구분되는 모듈 (Batch Logic, Online Batch Logic, Online Logic)
- ▶ 업무에서 공통적으로 사용되는 비즈니스적인 유틸리티 (Business Utility)
- ▶ 업무 로직 (Business Logic)
- ▶ 타 컴포넌트/시스템과의 연계를 위한 연계 로직 (Component Communication Logic)
- ▶ 관리 모듈 / 모니터링 모듈 (Admin, Monitoring Logic)

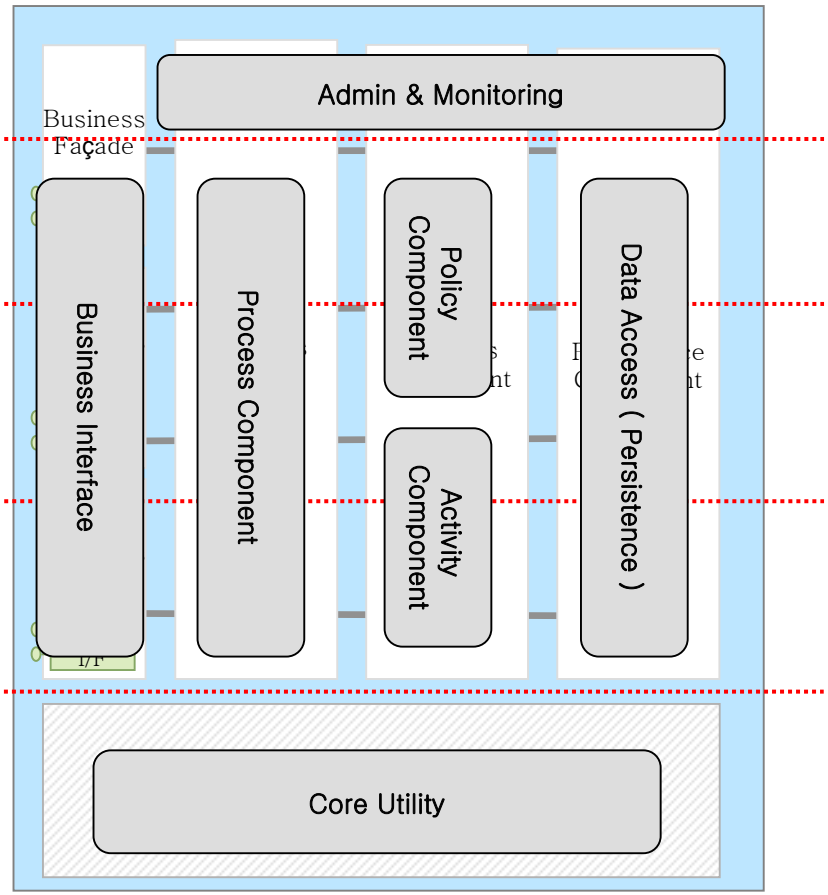


수직적 레이어 확장 방안

- Enterprise System은 기업의 업무 수행이라는 관점에서 접근할 수 있음.

- 업무 수행을 위해서는 다음의 4가지를 필요함
  - A. 업무 수행 절차 (Process)
  - B. 업무 수행 정책 (Policy)
  - C. 업무 수행 필요 정보 (Data)
  - D. 업무 수행 활동 (Activity)

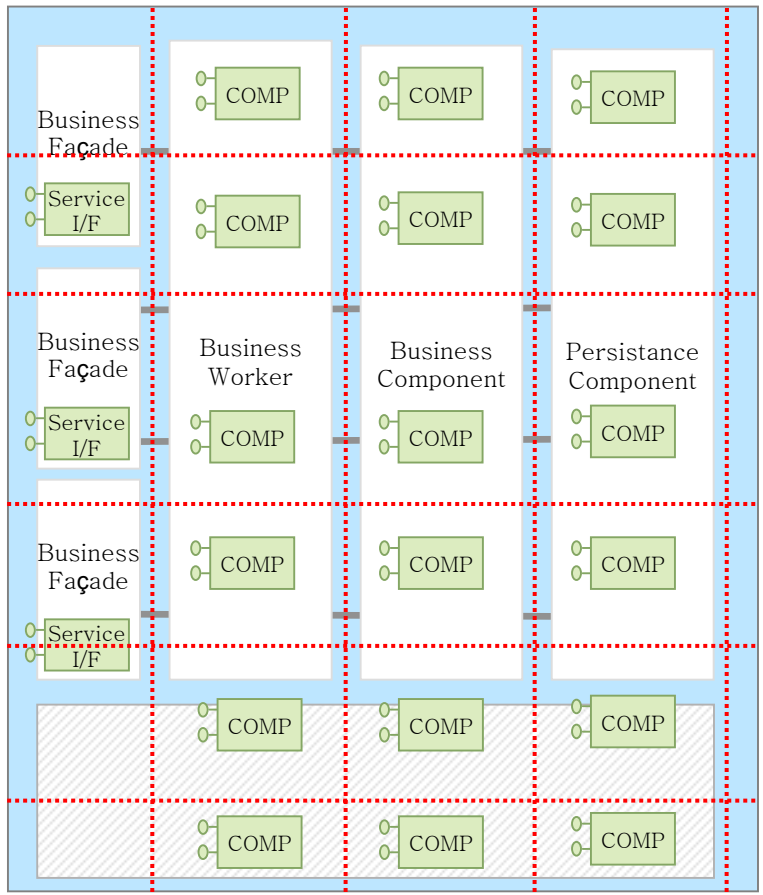
- 이에 따른 레이어 구성을 수행함.
  - Data : 비즈니스 랜잭션의 처리를 위한 데이터의 확보
  - Policy : 비즈니스 수행을 위한 정책의 결정 (해당 비즈니스 트랜잭션의 처리에 대한 정책적인 방향의 결정)
  - Activity : 비즈니스 수행을 위한 단위 활동의 수행
  - Process : Data, Policy ,Activity를 묶어 내기 위한 프로세스
- 위의 4가지의 구성을 통해 비즈니스 트랜잭션의 수행을 완성함.
- 수직적 레이어의 구분은 위의 4가지 요소들에 대해서 누가 무엇을, 언제 수행할 것인가? 에 대한 전략을 수립하는 것에 해당함.



아키텍처 설계 전략의 추가 영향 요소 목록

❑ 추가적인 고려 사항은 참조 아키텍처를 바탕으로 반영 가능함.

항목	설명	세부 사항
프로젝트 상황	프로젝트가 속한 상황과 환경 및 관리적인 요인들	프로젝트 범위
		시간(일정)
		비용(예산)
		인적자원
		커뮤니케이션 (의사결정 과정)
		위험요소(Risk)
		고객 (기술 수준, 참여도)
		요구 사항의 변동율/명확성
		조달(관련 자원의 조달)
품질 속성	프로젝트에서 구현하고자 하는 시스템의 품질 속성	가용성(Availability)
		성능(Performance)
		보안성(Security)
		사용성(Usaility)
		변경성(Modifiability)
		테스트성(Testability)
기술/설계 제약 사항	시스템의 구현 이전에 기정의된 결정 사항, 관리규정, 지침	기술 참고 모델(TRM)
		설계/기술 표준
		연동 시스템
		도입 예정 솔루션
		소프트웨어 인프라
		하드웨어 장비
		기존 시스템
목표 시스템 특징	구현하는 목표 시스템의 특징과 성질	비즈니스 흐름 중심
		데이터 중심
		서비스 중심
		프리젠테이션 중심
		신규 개발 중심
		변경 유지 보수 중심
		시스템 통합 중심



목 차

삼성SDS SAMSUNG

프레임워크의 역할

- 체계적인 컴포넌트 설계를 지원.
  - 품질 속성 구현 전략의 지원.
  - Reference 아키텍처 구현을 위한 레이어
- 프레임워크의 구성
  - Reference 아키텍처와 연계되어 함께 구성되어야 함.

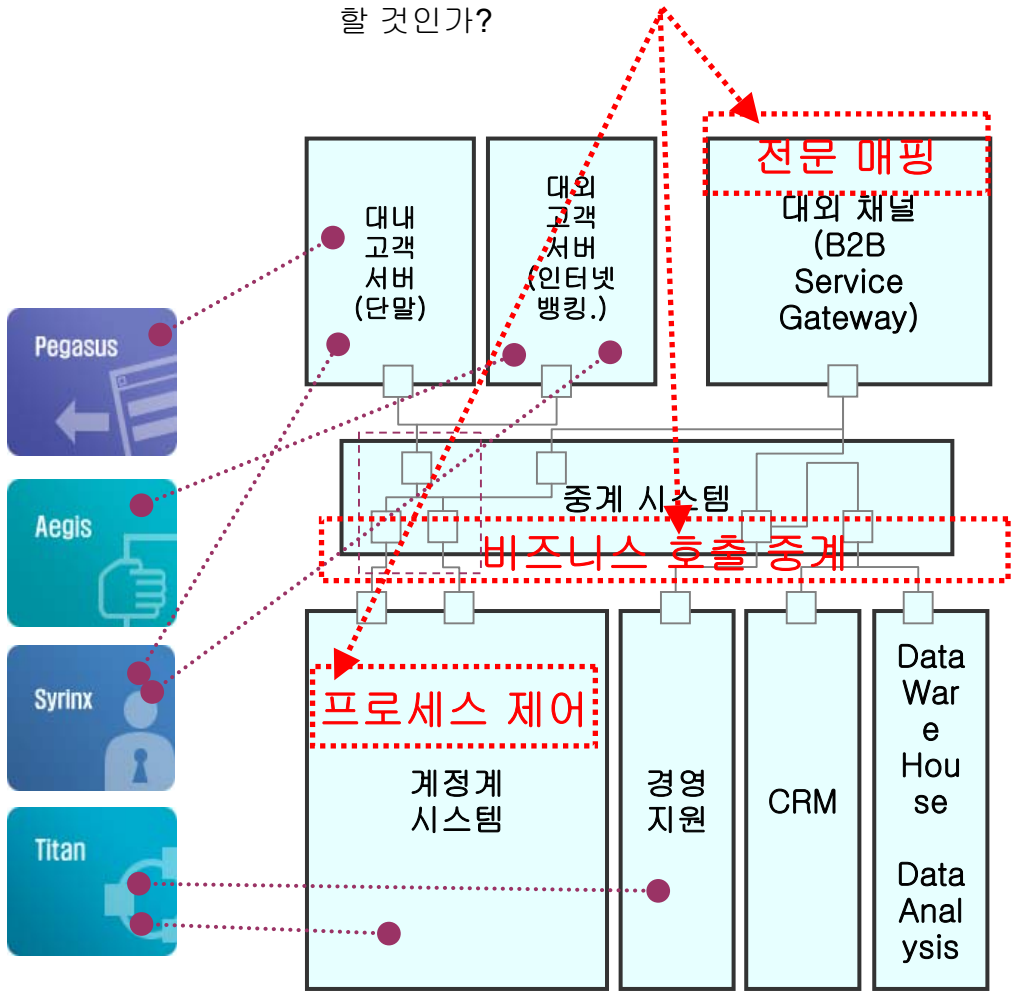
- 포함되지 않는 서비스 시스템의 설계는 어떻게 할 것인가?

Pegasus Framework (Portal Framework)
Personalization 정보에 따라 포틀릿 기반 하에 다양한 컨텐츠를 통합하여 제공

Aegis Framework (Web Application Framework)
Web 기반의 어플리케이션을 MVC 모델에 기반하여 체계적인 개발이 가능하도록 지원

Syrinx Framework (Security Framework)
Identity Federation 기반 SSO, 사용자 정보 관리, Role 기반 접근 권한 제한 기능 제공

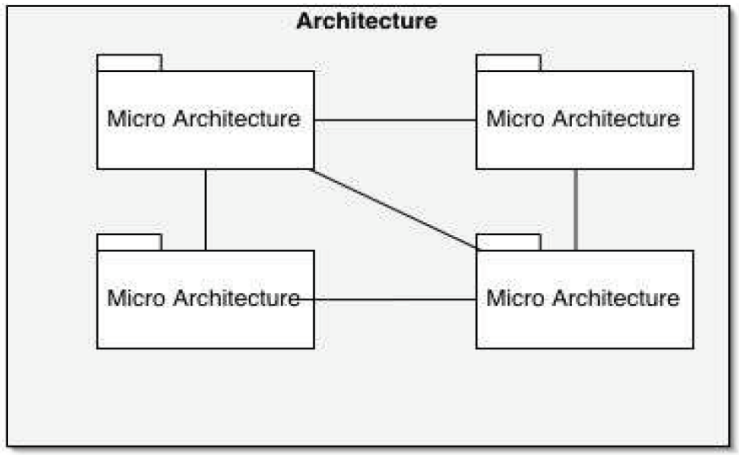
Titan Framework (Core Framework, Core Services)
모든 어플리케이션에서 사용 가능한 공통 서비스를 관리하고 공급하는 역할 담당



마이크로 아키텍처의 적용

- 마이크로 아키텍처의 정의
  - 마이크로 아키텍처는 어플리케이션을 구성하는 빌딩 블록에 해당함.
  - 패턴(Façade, Command, Etc)보다 좀더 상위 레밸의 추상화된 개념이며, 문제를 해결하기 위한 패턴들의 집합으로 구성함.
  - 마이크로 아키텍처는 패턴을 활용하여 서브 시스템의 디자인과 같은 복합적인 문제를 풀기 위한 아키텍처를 기술하고 있음
- 마이크로 아키텍처 예제
  - 메시지 전문 매핑 마이크로 아키텍처
  - 메시징 엔진 마이크로 아키텍처
  - 로그 분석 엔진 마이크로 아키텍처
  - Etc …
- 마이크로 아키텍처 & 아키텍처
  - 아키텍처는 일련의 마이크로 아키텍처를 포함하고 있음.

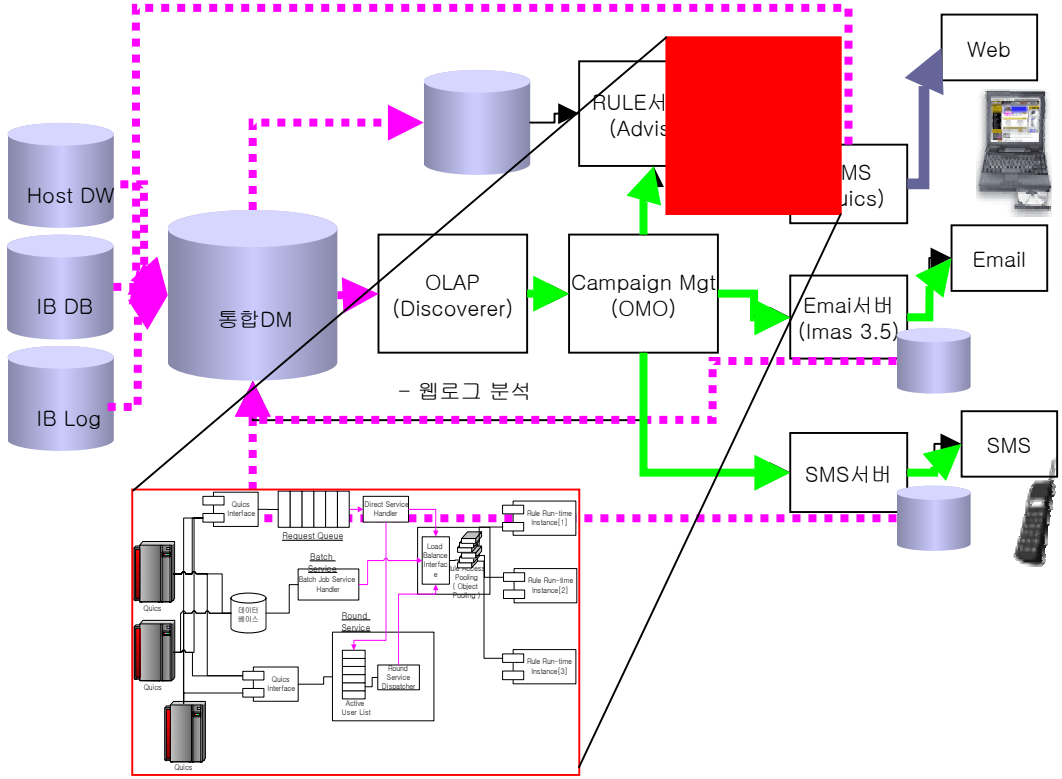
%Reference – Core J2EE Patterns, 2nd Edition



마이크로 아키텍처 적용 사례

❑ K은행 eCRM 시스템 구축의 비동기화 룰엔진 제어 모듈 구성

- ▶ 룰 기반 개인화 시스템을 실 서비스에 적용시키기 위한 상황
- ▶ CRM 개인화 서비스 엔진의 실 사이트 반영
- ▶ 뱅킹 및 기존 웹서비스에 영향을 주지 않아야 함.



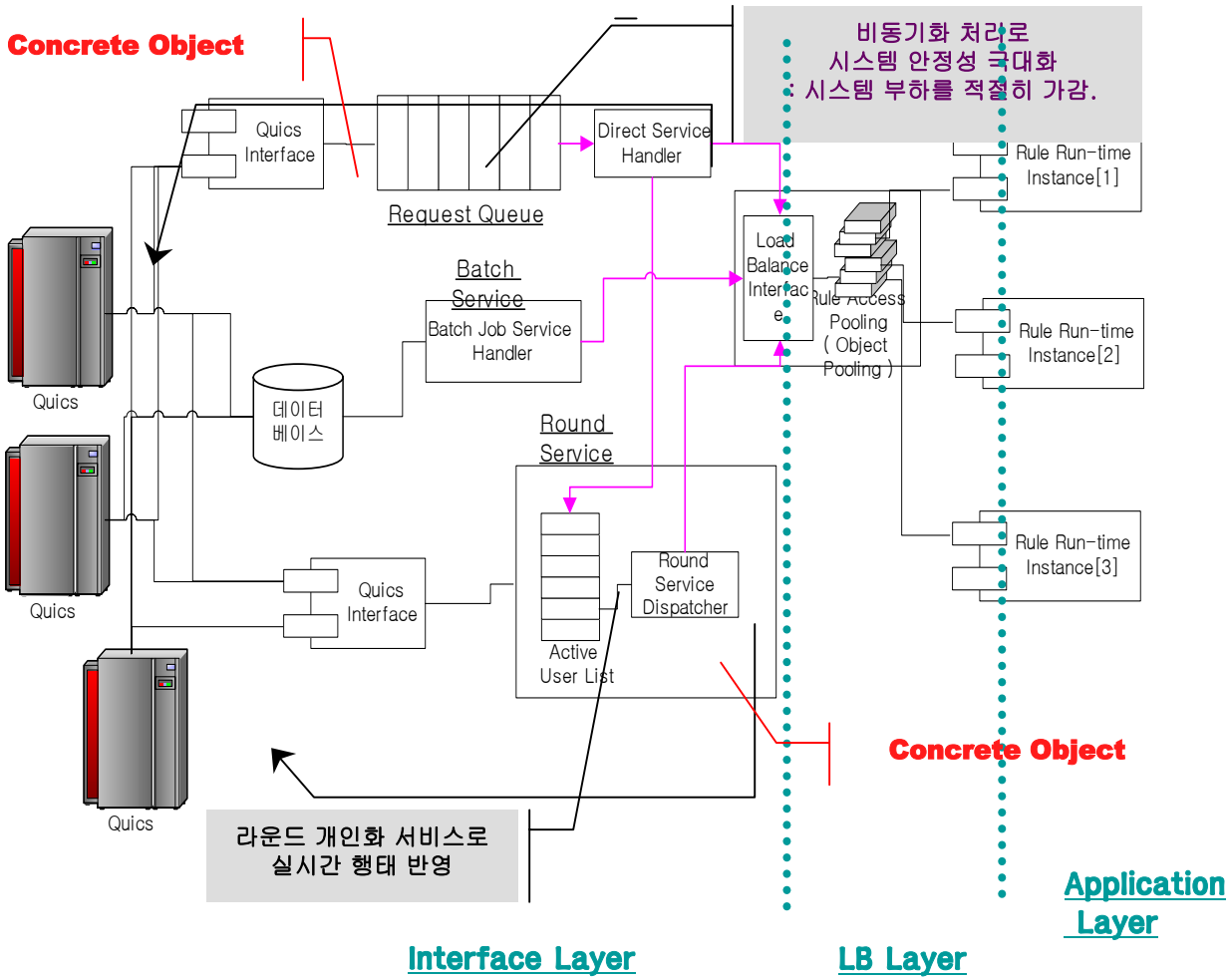
❑ 마이크로 아키텍처 요구 사항

- ▶ 2대의 개인화 서버에 적절히 분산해서 처리해야한다.
- ▶ 10분당 3만 명을 감당할 수 있어야 한다.
- ▶ 일부 감당못하는 인원은 절절히 내부적으로 Handling할 수 있어야 한다.
- ▶ 감당 못하더라도 웹 서비스에 영향을 주면 안됨
- ▶ 2대의 서버 혹은 DB가 죽더라도 웹 서비스에 영향을 주면 안된다.

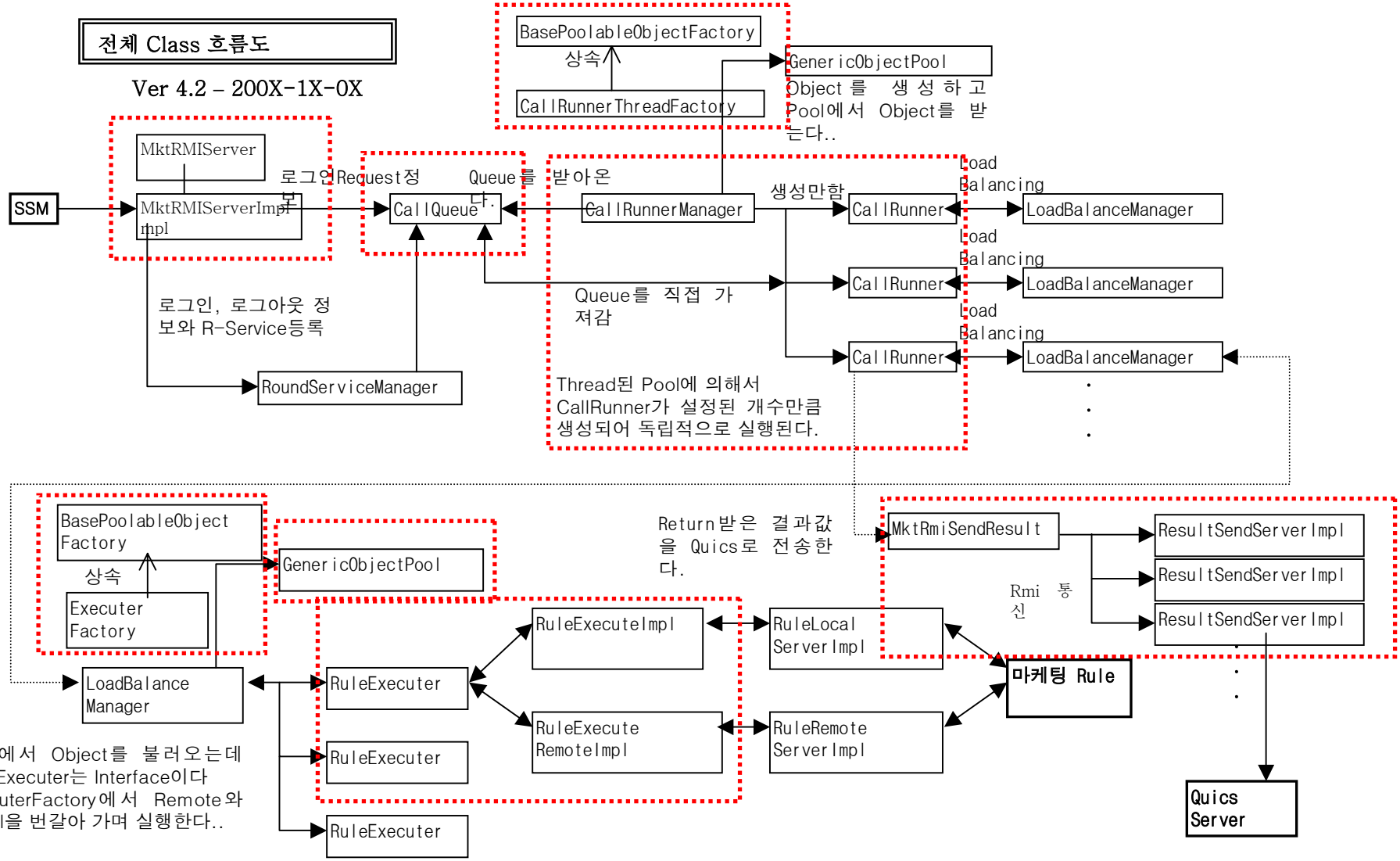
➔ 비동기화 처리

비동기화 연계 모듈의 마이크로 아키텍처 설계

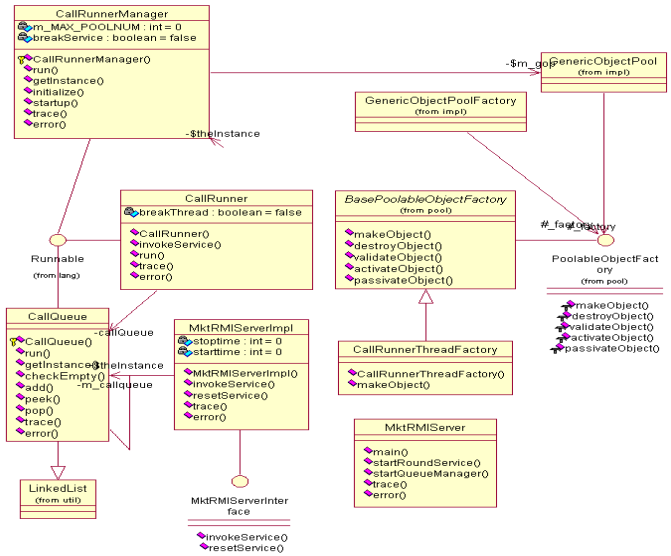
- Asynch
  - Call Receiver / Result Sender의 분리
- Requester Module
  - Request Sender
  - Request Result Receiver
- Processing Module
  - Request Receiver
  - Request Result Sender
- 보내는 쪽 및 받는 쪽 양쪽에서 자체 구현

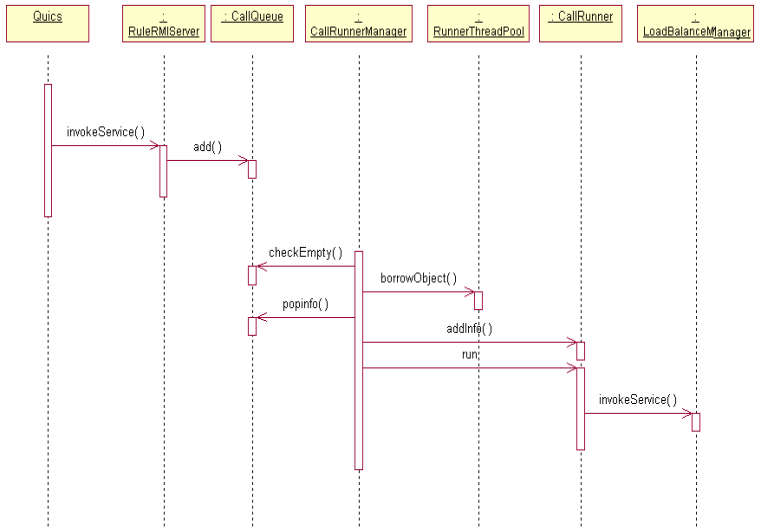


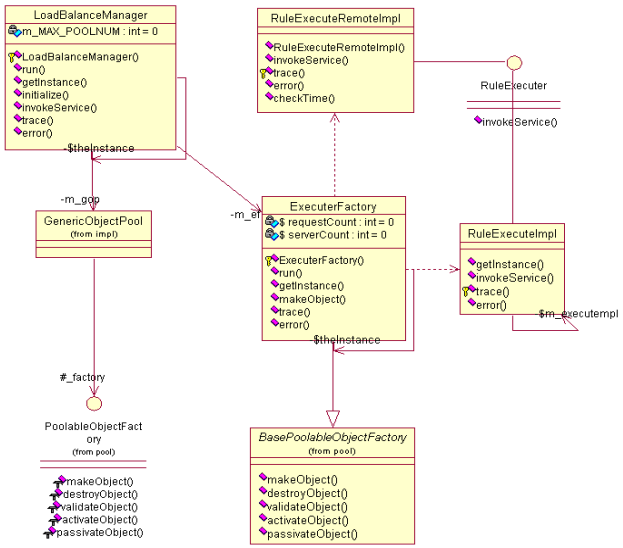
마이크로 아키텍처 – 상세 설계 1/2

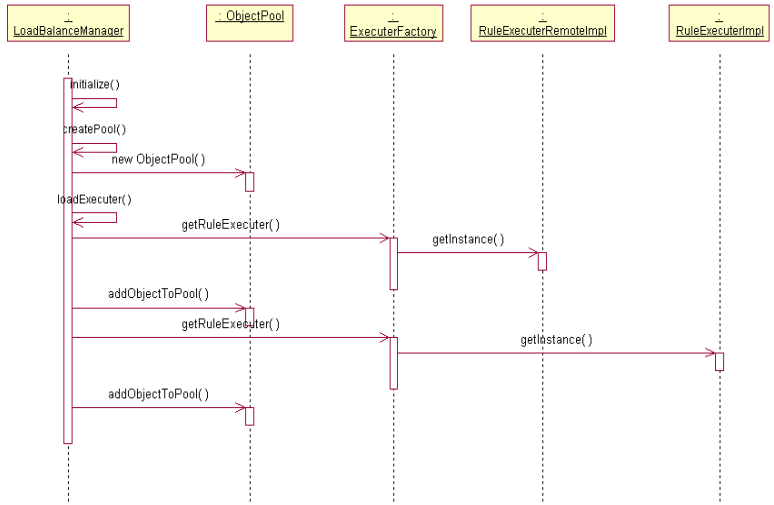


마이크로 아키텍처 – 상세 설계 2/2







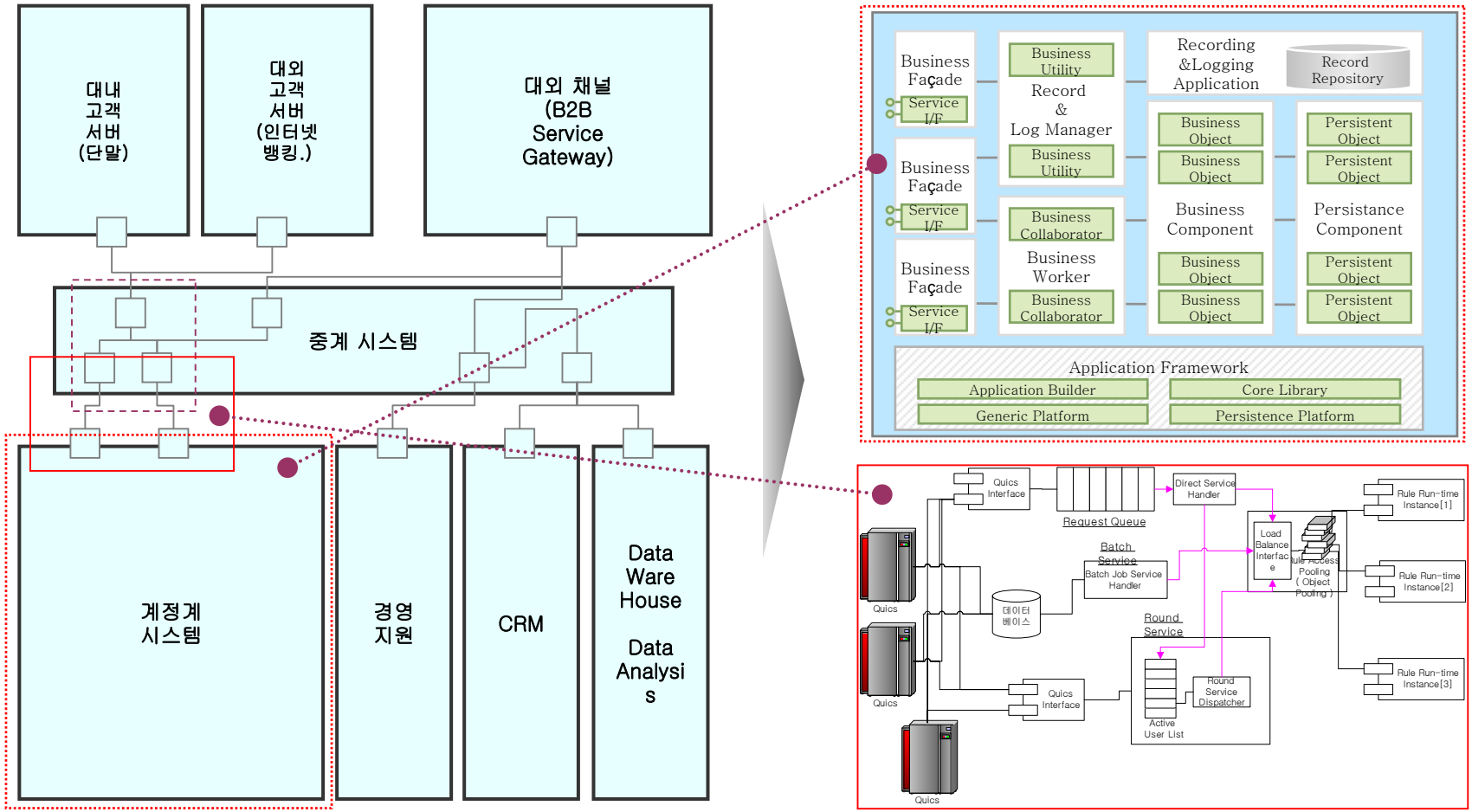


목 차

삼성SDS SAMSUNG

❑ Project Architecture for SI Project & CBD

- ▶ Component Structure : 컴포넌트 중심의 대단위 아키텍처 설계 (CBD 기반이 다수임)
- ▶ Micro Architecture : 메커니즘이 복잡한 특정 영역의 아키텍처는 Sub System으로 구성



결론

- 체계적인 컴포넌트 아키텍처 설계의 접근
  - 컴포넌트 기반의 아키텍처 설계
  - 품질속성과 레이어 기반의 설계에 기반한 체계적인 아키텍처 설계의 접근
  - 기술 기반 (컴포넌트 인프라 스트럭처)의 특성을 반영함.
  - 도메인의 특성을 참조 아키텍처 (Reference Architecture)로 반영
  - 패턴과 프레임워크 중심으로 설계 및 구현과 직접적인 매핑이 되는 아키텍처 설계
  - 핵심 설계 요소가 필요한 부분은 집중적으로 정의하여 마이크로 아키텍처로 해결

- 체계적인 설계를 통해 소프트웨어 아키텍처 문제의 **일부분을 해결** 할 수 있음.
  - 추가적인 문제들
    - 비즈니스 단위의 컴포넌트 분할
    - 기존 시스템의 재활용
    - ...

- 도메인 중심의 재활용을 위한 프레임워크와 표준 설계 체계가 병행되어야 함.
  - Product Line
  - Reference Architecture
  - Architectural Asset
    - Security Architecture Guide
    - Component Design Guide
    - etc