Part 5.

SOA 서비스 지향 아키텍처

마이크로 서비스의 근간이 되는 SOA 아키텍처를 통해서, 아키텍처의 발전과, 실패 원인을 분석을 통하여 아키텍처 설계 방법을 알아본다.

SOA 아키텍처를 공부하는 이유?

SOA는 2000년대 초반에 부각된 아키텍처 스타일

세부 구현 기술은 변했지만, 현대의 MSA등 분산 시스템의 아키텍쳐 사상으로 그대로 반영되고 있음

SOA의 본질적인 개념을 파악하여 대용량 분산 시스템 구현에 활용

"역사는 단순한 과거의 기록이 아니다. 그것은 우리가 배우고 고쳐나가야 할 살아있는 교훈이다."

(History is not just the record of the past; it is a living lesson for us to learn and correct.)

- Robert Penn Warren (미국 작가/문학 평론가)

1. SOA 기본 개념

엔터프라이즈 시스템의 발전

IT 시스템 아키텍처의 변화

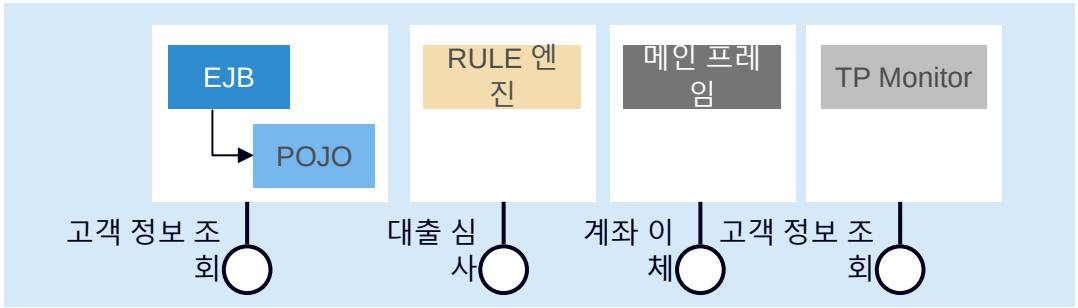
Approach	Timeframe	Development Model	Business Motivations
Mainframe	1960s – 1980s	Procedural (COBOL)	Automated Business
Client-server	1980s – 1990s	Database(SQL), Fat Client (Visual Basic)	Computing On the Desktop
N-Tier model and the Web	1990s – 2000s	Component or Object-Oriented (EJB, COM)	Internet and e-Business
Service Oriented	2000s – for the time being	Service-Oriented	Business Agility

SOA 아키텍처란?

- 기존의 애플리케이션의 기능들을 비즈니스적인 의미를 가지는 기능 단위로 묶어서 표준화된 호출 인터페이스를 통해서 서비스로 구현하고, 이 서비스들을 기업의 업무에 따라 조합하여 애플리케이션을 구성하는 소프트웨어 개발 아키텍쳐
- 현대의 Domain Driven Development와, REST API 기반의 MSA와 개념은 거의 동일함

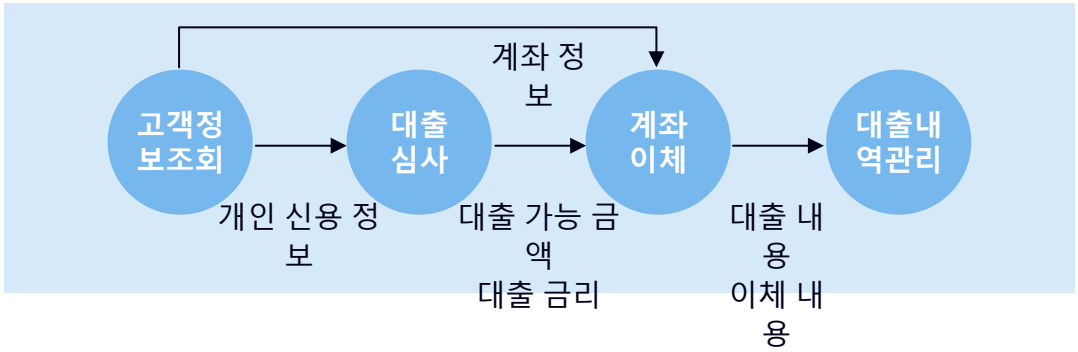
서비스화

업무 시스템을 업무적인 의미를 갖는 컴포넌트로 묶은 후
업무 기능을 표준 인터페이스로
제공하는 서비스 구현



서비스 조합

서비스를 조합하여
업무를 구현함



SOA가 주목 받았던 이유

웹서비스의 등장 (CORBA,DCOM 비표준화,복잡성)
→ **분산 기술의 표준화 필요성**

점점 확장되어가는 독립된 업무 시스템
→ **통합에 대한 요구**

기업의 비즈니스 속도가 빨라져 감에 따라 IT 시스템의 업무 변화에 대한 민첩한 대응력이 필요하게 됨
→ **민첩성에 대한 요구**

2. 서비스의 개념

서비스의 개념

- 플랫폼에 종속되지 않는 표준 인터페이스를 통해서 **기업의 업무**를 표현한 Loosely Coupled 하고 상호 조합 가능한 **소프트웨어 컴포넌트**

서비스로 적절한 것

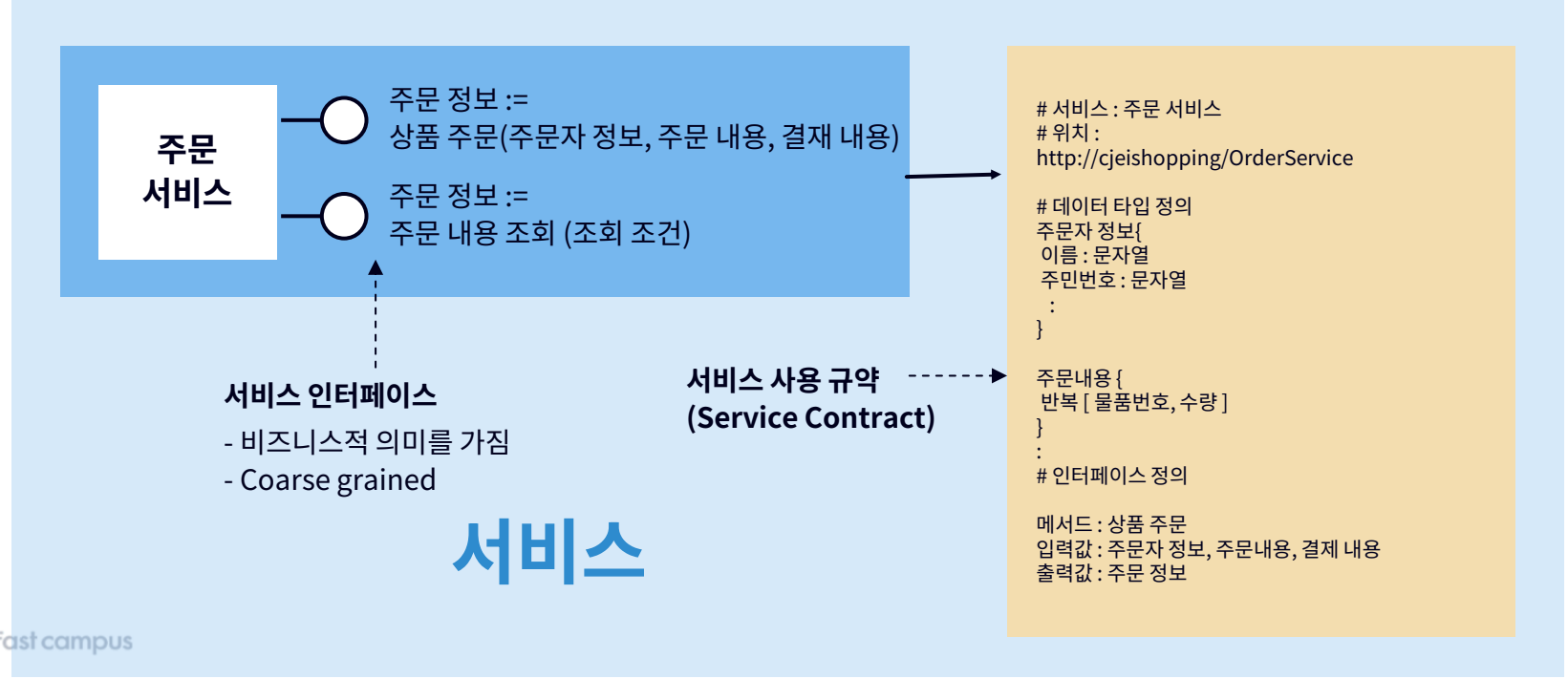
- 임직원정보 서비스
- 계좌이체 서비스
- 상품 주문 서비스

서비스로 적절하지 않은것

- 오토스케일링
- 컨테이너 배포

서비스 구성

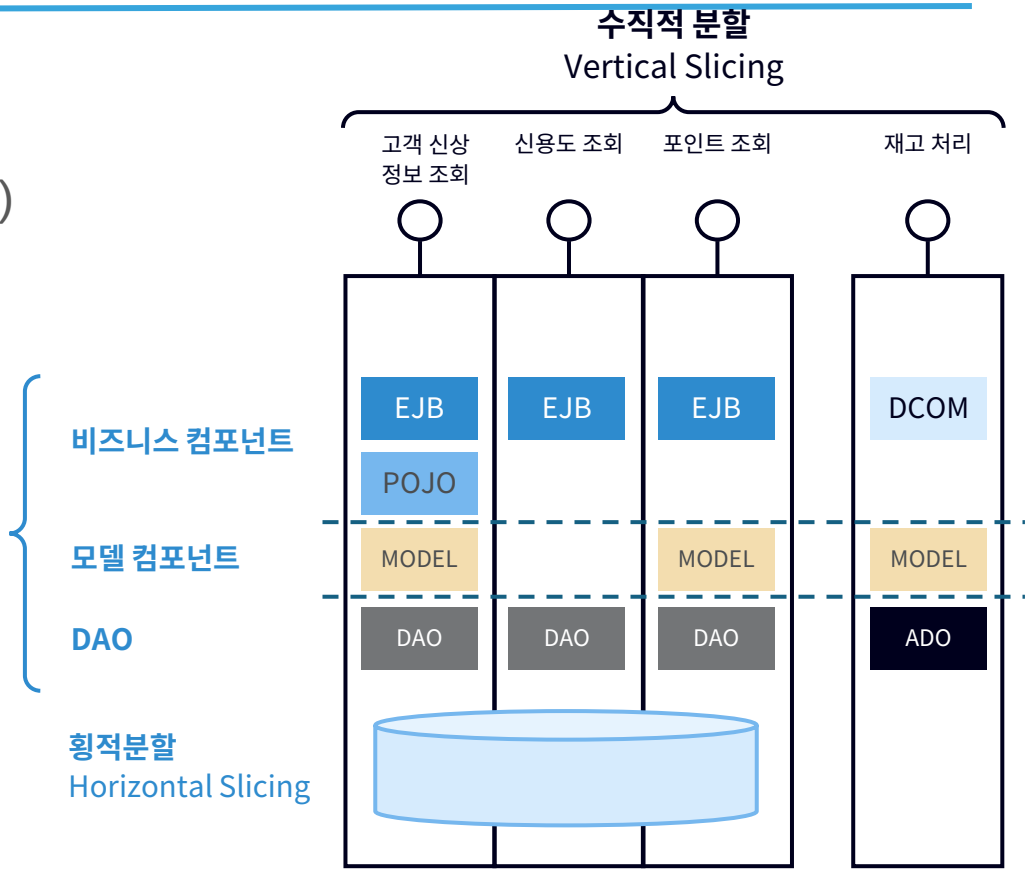
비지니스 의미를 가지는 기능을 API를 통해서 제공하는 소프트웨어 컴포넌트



서비스 특징

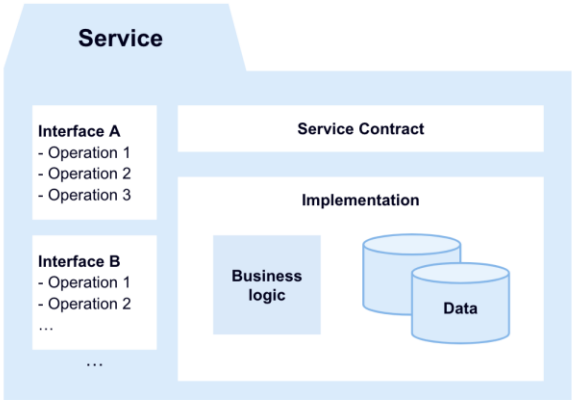
- 수직 분할 (Vertical Slicing)
- 스탠다드 인터페이스 (XML/WS)
- Loosely coupled
- 조합 가능 (Composable)
- Coarse grained
- Discoverable

현대의 마이크로서비스에서 서비스의 특징과 동일함



웹서비스 기반의 서비스 구성

- **서비스 규약 (WSDL)**: 서비스의 목록, 호출 인자, 데이타 타입등을 정의. (비교 REST API SPEC)
- **서비스 인터페이스 (WS):** 서비스 인터페이스 (비교. REST)
- **서비스 구현체** : 데이터와 비지니스 로직이 포함된 서비스



WSDL 예시

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:tns="http://www.example.com/calculator"
:

 <!-- 데이터 타입 정의 -->
 <types>
  <xs:schema targetNamespace="http://www.example.com/calculator">
   <xs:element name="AddRequest">
    <xs:complexType>
:

 <!-- 메시지 정의 -->
 <message name="AddRequest">
  <part name="parameters" element="tns:AddRequest" />
 </message>
 <message name="AddResponse">
  <part name="parameters" element="tns:AddResponse" />
 </message>

 <!-- 포트 타입 정의 -->
 <portType name="CalculatorPortType">
  <operation name="Add">
   <input message="tns:AddRequest" />
   <output message="tns:AddResponse" />
  </operation>
 </portType>

 <!-- 바인딩 정의 -->
 <binding name="CalculatorBinding" type="tns:CalculatorPortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="Add">
```

서비스의 분류

비즈니스 서비스

- 일반적으로 정의하는 서비스
- **태스크 지향 서비스 (Task Centric Service)** : 비즈니스 로직을 제공함
- **데이터 지향 서비스 (Data Centric Service)** : 데이터 조회나 변경을 담당
- 마이크로 서비스에서 DDD로 서비스를 나눌 경우 정의되는 서비스 타입

서비스 분류

Intermediary 서비스

- 아키텍처에 유연성을 더해 주는 서비스
- 기존 서비스 변경 없이 비지니스 로직을 변경할 수 있음

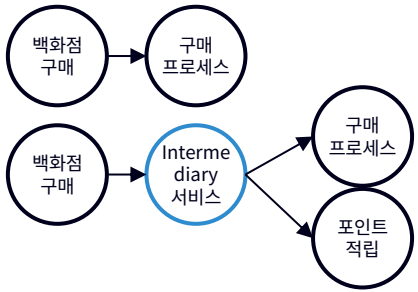
라우팅

메시지 변환

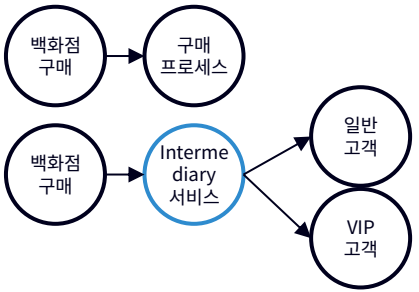
기능 추가

퍼사드 패턴. *채널시스템

Functional Adding

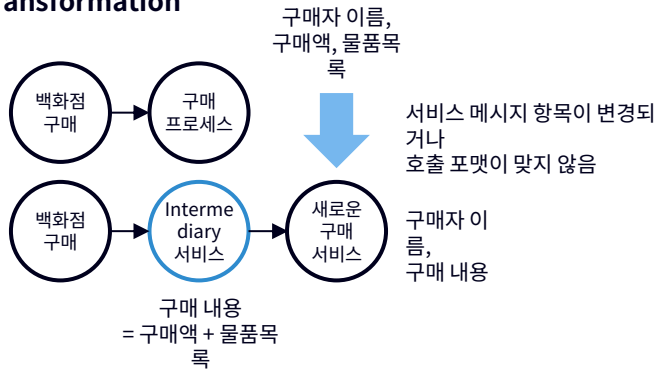


Routing



총 구매 금액 > 1000만 원

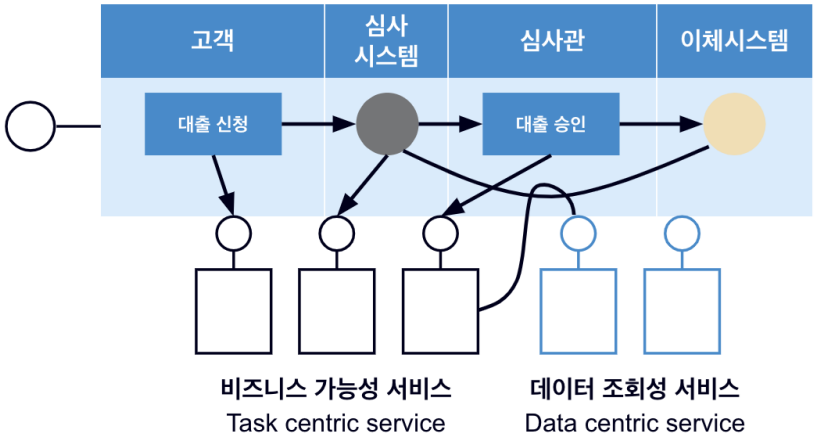
Transformation



서비스의 종류

프로세서 센트릭 서비스
(Process centric service)

- 기존 비지니스 서비스를 조합(Orchestration)하여, 비지니스 시나리오를 구현
- 일종의 워크플로우 엔진 같은 역할을 함
- 주로 긴 업무 프로세스 구현
- 상태 정보가 있을 수 있음



서비스 종류

비즈니스 로직을 서비스 하는 것이외에도, 예외적으로 기술 기능 제공등의 비즈니스 도메인 외부의 서비스가 필요할 수 있다.

애플리케이션 서비스 (Application Service)

- 비지니스 기능이 아니라 테크니컬 한 기능을 구현
- **예시** : 오토스케일링 API, 모니터링 API

연동 서비스 (Integration Service)

- 타시스템이나 외부 시스템과 연동할때 사용됨
- **예시** : 금융권 대외계 업무, 금융권 마이데이터

서비스 종류 요약

	Application Service	Business Service	Intermediary Service	Process centric Service	Integration Service
Description	Express technology-specific functionality	**Contains business logic and data.**	Bridge the gap from concept or design	Encapsulate business process and orchestrate other services	Export business logic to other enterprise or organization
Sub category	Wrapper svc Utility svc	**Task centric ,Entity centric**	Façade,Function Adding, Routing, Transforming	Process centric	
State mgmt	No	**No**	No	Yes	No
Reusability	??	**High**	Low	Very Low	High
Mandatory in SOA	No	**Yes**	No	No	No
Note		**Most Important Service**	It needs when new service or new service connection is made		

3. SOA 아키텍쳐 발전 모델

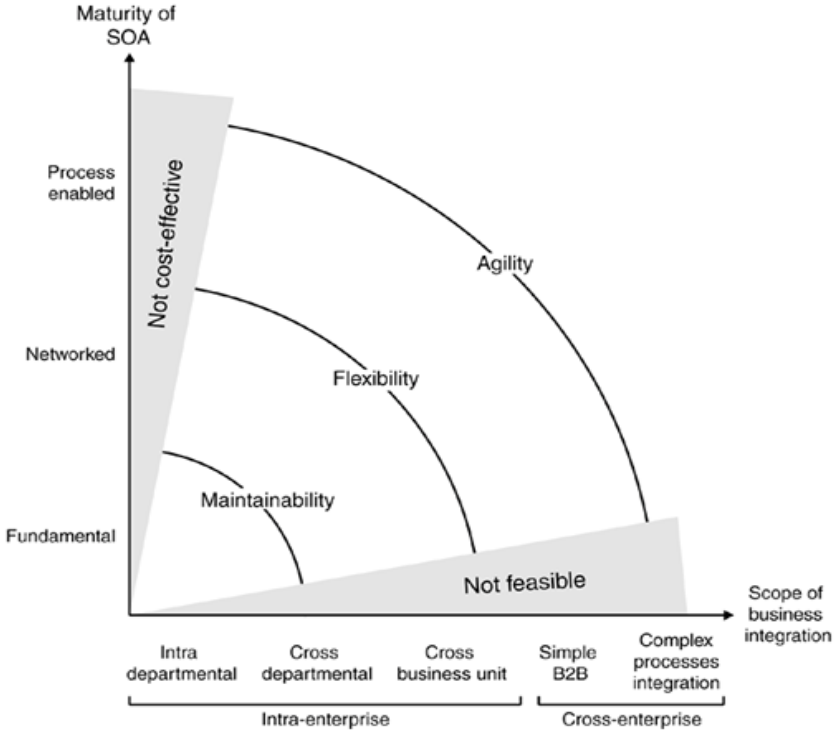
SOA 아키텍처의 단계적 발전 구조

SOA 아키텍처는 시스템의 규모와 업무적 요구 사항, 팀의 성숙도에 따라서 아래와 같이 3단계로 발전할 수 있다.

Fundamental SOA

Networked SOA

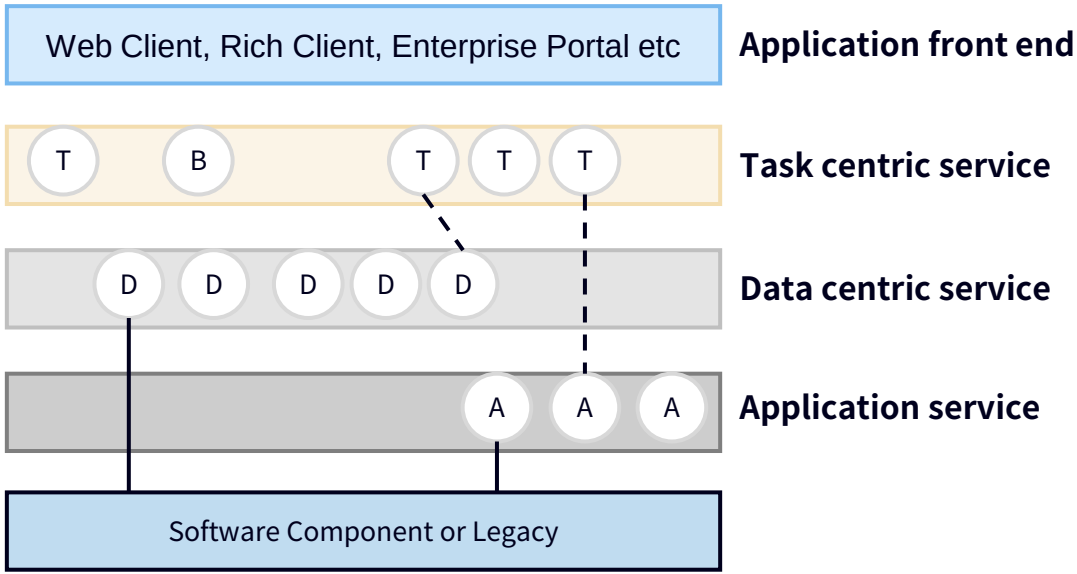
Process Oriented SOA



1단계 Fundamental SOA

기존의 시스템들을 서비스화하여 시스템을 통합하는 단계

- 서비스화에 목적이 맞춰져 있음
- 서비스에 대한 조합은 프론트엔드에서 담당
- 비지니스 서비스로만 구성됨
- IWay 아답터 처럼 레거시(Tuxedo, 메인프레임)을 웹서비스로 변환시켜주는 아답터를 이용
- 현대의 가장 기본적인 마이크로서비스 구조와 유사함



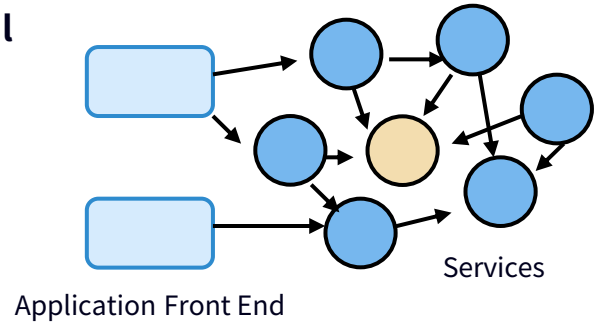
Fundamental SOA 개념도

2단계 Networked SOA

Fundamental SOA 구조의 문제점

- 서비스 to 서비스 연결이 P2P 연결이기 때문에, 서비스가 많아질 수록 매우 복잡해짐 (거미줄 모양)
- 시스템 구조의 유연성이 떨어짐

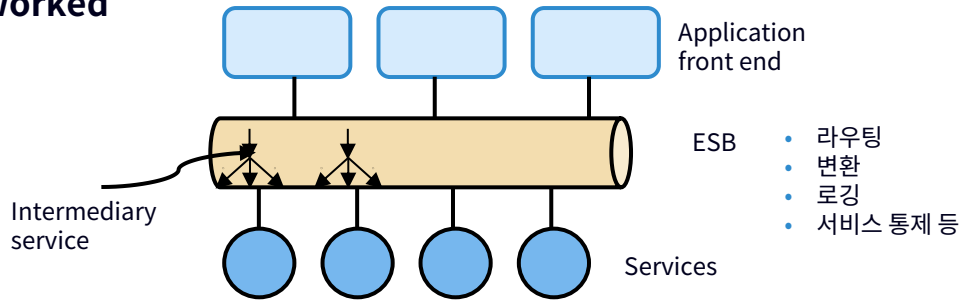
Fundamental SOA



Networked SOA

- Fundamental SOA의 문제를 보완하기 위해서 거미줄식의 연결을 중앙에 버스(Enterprise Service Bus aka ESB)를 넣어서 Hub & Spoke 형태의 구조를 제공함
- Intermediary 서비스를 ESB에 배포하여 아키텍쳐의 유연성을 확보함

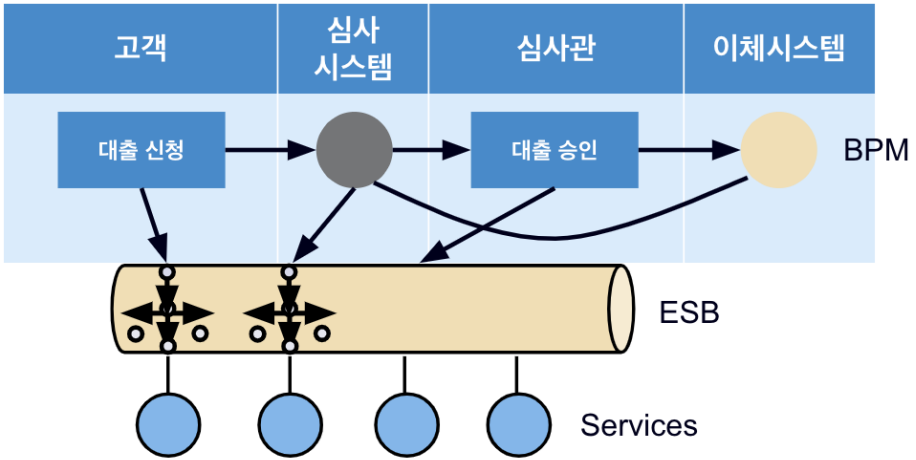
Networked SOA



3단계 Process Oriented SOA

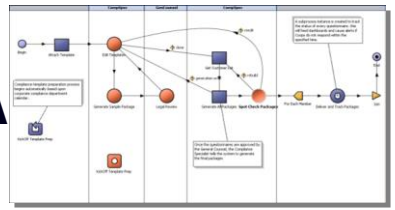
Process Oriented SOA

- 여러개의 서비스를 조합하여 하나의 복잡한 업무를 구현해야 하는 경우
- 서비스 조합 (오케스트레이션 : Orchestration)을 위한 BPM과 같은 워크 플로우 엔진을 별도로 도입



3단계 Process Oriented SOA

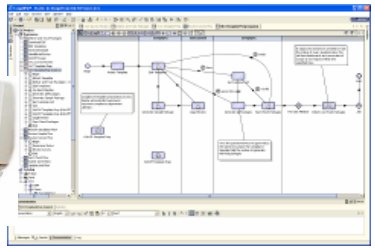
BPA



Process Modeling, Simulation and Documentation

Business Analysts

Process Analyst



Process Development and Systems Integration

BPM

Start

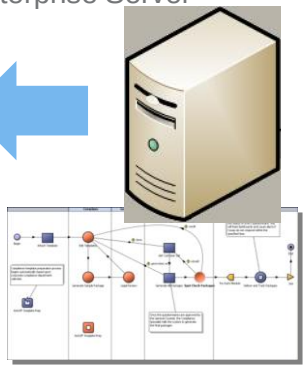
Business Owners

Process Management and Real-Time BAM

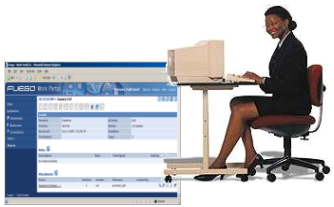
Historical & Trend Analysis Tools

BAM

BPM Enterprise Server



Process Server and Monitoring Repository



Process Participants

BPM Workspace

Business Systems

4. SOA 프로젝트 수행

수행 전략

- 기업의 장기적인 비즈니스 전략에 따라 IT 시스템을 전략 적용 단계에 맞춰서 개발
  - 기존 :비즈니스 각 전략 단계별로 각각의 독립된 시스템을 따로 개발
  - SOA :하나의 SOA시스템에 비즈니스 전략에 따라 해당 기능을 추가해 나감
- 서비스의 우선순위와 SOA화 범위를 비즈니스 전략의 실행단계에 맞춰서 정의

  * 기업 전략

  2004년 매출 증대

  2005년 고객 만족 실현

  2006년 브랜드 이미지 관리

  * SOA 전략

  2004년 매출 내용 전산화

  2005년 CRM 도입을 통한 고객 정보 수집과 매출 내용을 기반으로 고객 패턴 추출

  2006년 수집된 고객 정보를 토대로 마케팅 집중

레퍼런스 아키텍처

- 전체 기업 업무를 단일 시스템으로 운영하기 위한 플랫폼이 필요
- SOA 레퍼런스 아키텍쳐
    - Fundamental SOA
    - Networked SOA
    - Process Oriented SOA
    - Google과 Naver의 SOA전략
    - Adobe 기반의 SOA

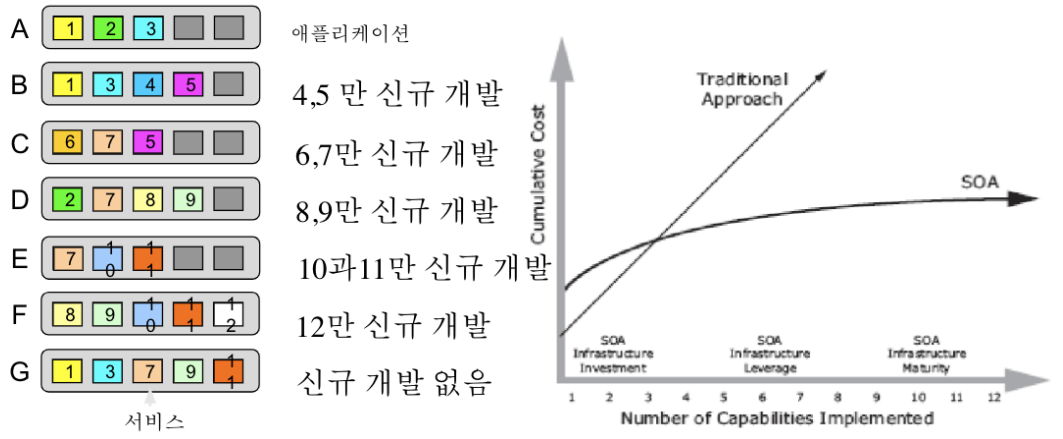
    + 보안 인증, Application Front end, Service repository
- SOA 시스템의 크기, 기업 시스템의 SOA화, 기업 비즈니스 전략에 따라서 지속적으로 레퍼런스 아키텍쳐(플랫폼)을 발전 시켜 나감

거버넌스

- 제어 통제가 필요한 이유
  - 전체 IT 시스템을 SOA화 함에 따라 장기적인 중앙 통제 그룹과 관리 도구가 필요함
- 통제 조직
  - SOA 시스템에 대한 정책 수립 및 표준화 - Standard
  - SOA 관련 기술 전파 및 가이드 - Evangelist
  - SOA 구축 계획 수립 및 실행 (로드맵)- Strategy
  - 자금 조달 및 집행 계획
  - 업무 분석 및 설계
  - 문화 변화 □ IT조직과 비즈니스 협업 조직의 협업문화 개발
  - 모범사례 수집과 배포

비용 통제

- 초기 플랫폼을 구축하는 데 비교적 많은 비용이 소요됨
- 서비스를 재사용 재조합하여 새로운 업무를 구현함으로써, 처음부터 개발하는 기존 시스템에 비해서 시스템이 성숙화 되어감에 따라 개발 비용이 감소함
- 중앙 통제와 제어를 통해서 유지보수 비용이 감소함

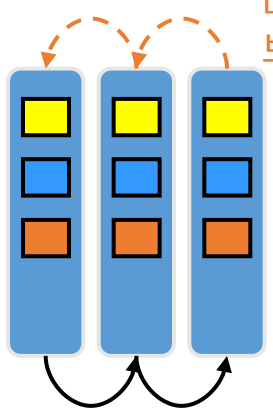


프로젝트 진행

- 반복적 개발 모델
  - 업무를 개발해서 운영 하면서 점진적으로 업무에 맞도록 개선 및 변경해나가면서 시스템의 성숙도를 늘려감
    BPA→BPM→BAM
  - 각 단위 서비스 업그레이드를 통해서 가능
- Thin Thread Model

  CF. 수직적 분할에 따른 개발

한 업무에 대한 기능을
모델,비즈니스,프리젠테이션을
포함해서 모두 개발
- 미리 검증 가능(파일럿)
- 경험 축척



[반복적 개발 모델]

[Thin Thread Model]

5. SOA 프로젝트 실패 요인

복잡한 기술 표준

SOAP/XML

- XML/WebService는 모든 기능을 표준화 스펙화하려고 하였음
- 결과적으로 복잡도가 증가하고, 학습 비용이 매우 증가함
- 일반 SI 개발 모델에서, 일반 수준의 개발자가 SOAP/XML의 특성을 잘 이해하여 최적화된 코드를 만들기 어려움

REST/JSON

- REST/JSON으로, 별도의 표준 없이 직관적이고 쉬운 인터페이스를 제공함
- 누구나 쉽게 이해하고 사용할 수 있도록 하여, 전체적인 학습 비용을 낮춤

Enterprise Service Bus

Enterprise Service Bus

- 개념은 좋지만 SPOF로 작용함
- XML/SOAP 파싱 비용이 너무 높아서 실제 성능 병목이 됨
- 너무 많이 Intermediary service 로직 (메시지 변환, 라우팅등)

API gateway & Service Mesh

- REST/JSON으로 경량화하고, 최소한의 기능만 유지 (라우팅, 메시지 변환 제거)
- 외부 API 서비스 컨트롤 및 내부간 서비스 컨트롤을 API Gateway와 Service Mesh로 나눔
- Service Mesh는 중앙 집중화된 SPOF 구조가 아니라 분산 구조

Event Driven Architecture

- ESB 아키텍처 자체는 좋은 구조이나 난이도가 높음
- 좋은 구조이기 때문에 Netflix등에서 사용 (EDA)

중앙 집중형 거버넌스 모델

중앙 집중형 거버넌스 모델

- 조직마다 다른 이해관계 (영업팀, 마케팅팀, 제조팀등 필요한 기능이 다르며, 예산 문제)
- 중앙 집중형 모델에 따른 커뮤니케이션 비용 증가 (결재)
- 시스템이 복잡해짐에 따라, 중앙 통제가 어려워짐

분산형 거버넌스

- 각 조직/사업부에 맞도록 서비스를 나누고, 팀을 분리하여 민첩성 증가
- 각 서비스별로 사업부와 이해관계를 정리
- 복잡한 시스템을 중앙에서 통제하지 않고, 각 서비스팀이 개발,운영을 할 수 있는 자치권을 부여 (Devops)

감사합니다.